CC Response While Application-Limited

Matt Mathis

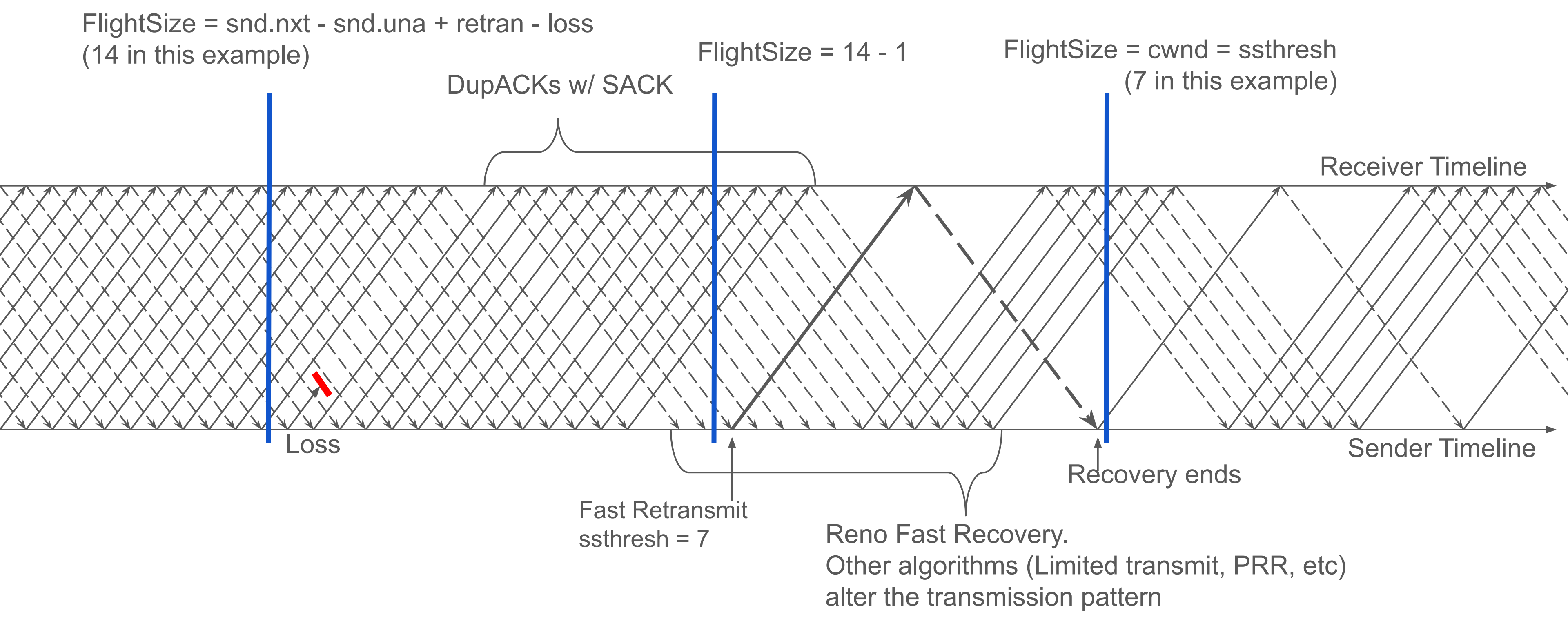
IETF 120, July 2024

Application-Limited aka Stalls

- Stalls cause FlightSize to fall below cwnd
    - Otherwise FlightSize tracks cwnd
- Example causes of stalls
    - Sender has insufficient data to fully use cwnd
        - e.g. CPU bound, waiting on other i/o or has competed a transaction
    - Receiver is announcing receive window less than cwnd
    - Sender deferral optimizations: TSO, Nagle, Silly Window, pacing, etc
        - Defer transmissions in anticipation of batching with a future transmission
        - These are ignored in the analysis here (they don't change the overall results)
- For non-paced stacks, short stalls typically cause line rate bursts to catch up
    - They typically send cwnd-FlightSize as a burst when the stall ends

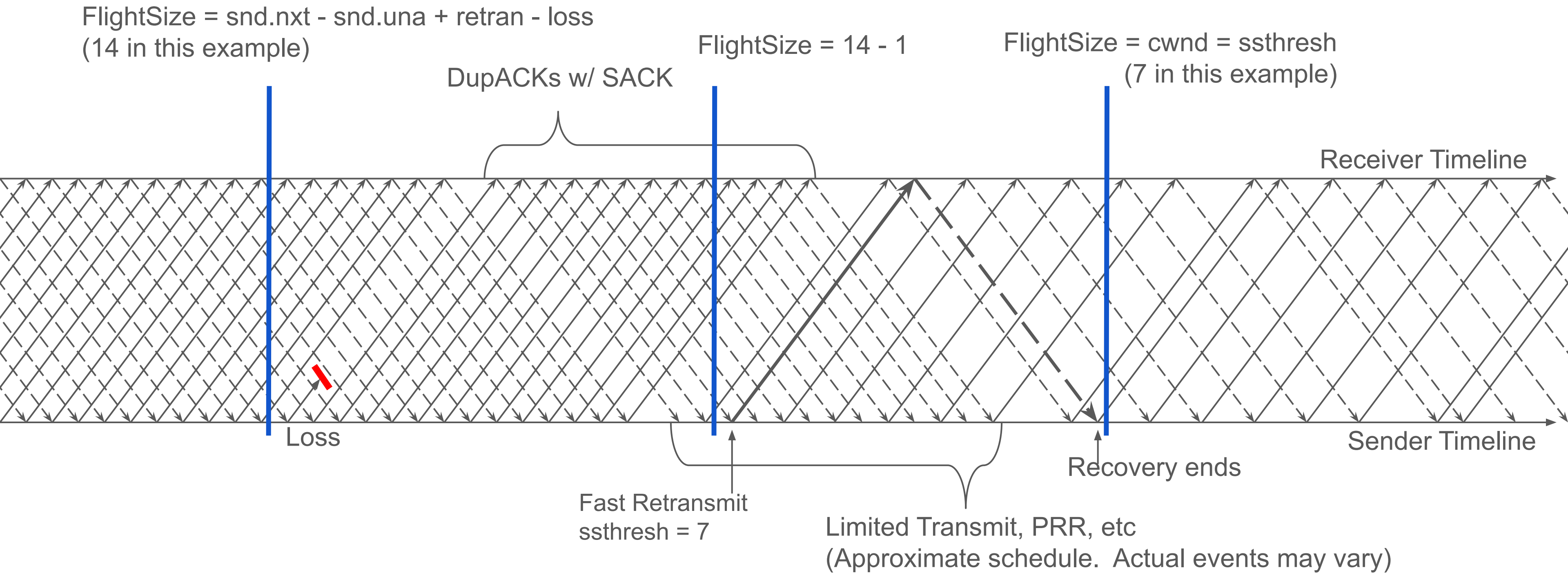
Simple Fast Retransmit + Recovery (classic Reno)

Van Jacobson's original Reno (when entering recovery, ssthresh is set to cwnd/2)
Additive Increase is also suppressed for clarity
Observation: Recovery cuts cwnd from 14 to 7
RFC 2581 Reno looks the same, because FlightSize == cwnd



Fast Retransmit with modern recovery

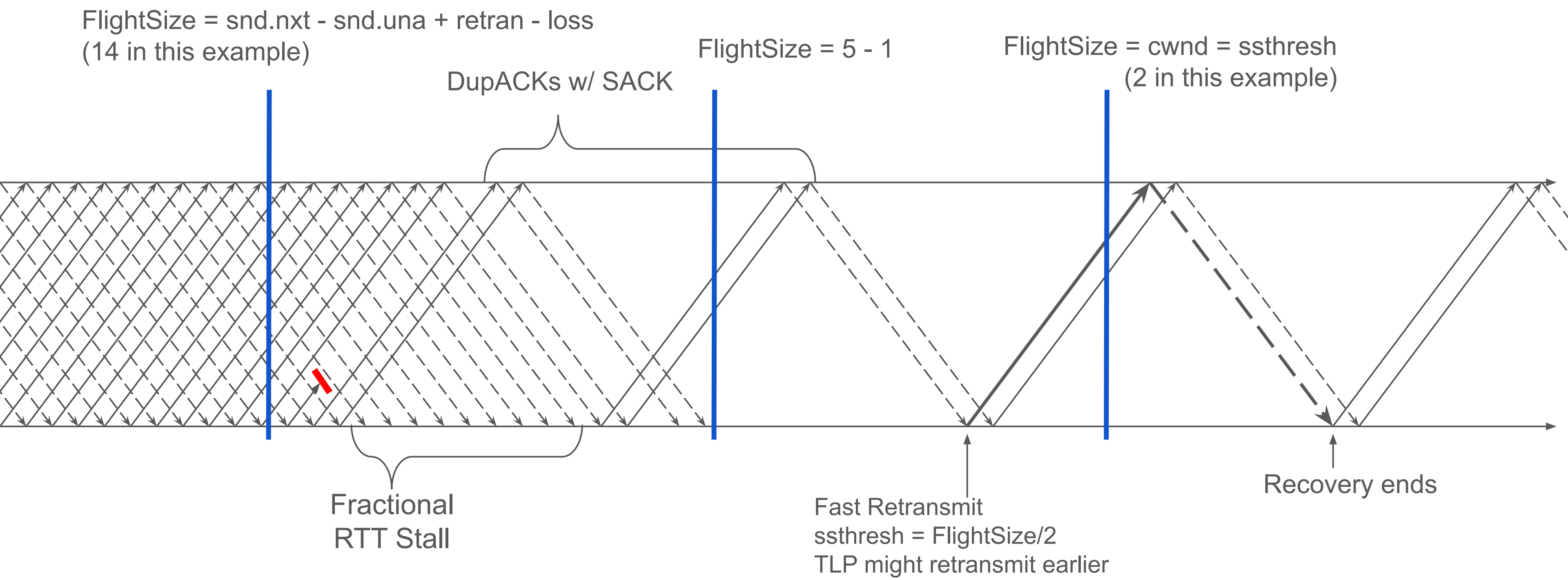
Same as the prior slide plus Limited Transmit and PRR
Observation: The only change is which ACKs trigger **new** data.
The net outcome is the same, cwnd = ssthresh = 7.



Although this is a better representation of modern stacks,
the added complexity does not improve clarity.

Fast Retransmit & Recovery with Stall

RFC 2581 Reno, using ssthresh = FlightSize/2
Observation: With a stall, ssthresh = FlightSize/2 causes Recovery to cut cwnd from 14 to 2.
This is nonsensical, given the stall happened after the lost packet.



Why does a stall **after** the loss change the CC outcome?
Ssthresh was computed from FlightSize, not cwnd

This problem was first observed in 1998

- Disk to disk transfer from PSC to SDSC
  - Unbuffered disks with slight pauses at track and cylinder boundaries
  - "Resonance" between track size and network pipe size
  - Inordinately likely for loss just before a stall
  - Unexpectedly poor performance
    - Reno AIMD cycle time was >90 seconds
  - Long predated modern tcptrace analysis tools
- Today this situation is pervasive
  - Vast majority of flows are partially application limited
    - Either transactional or streaming

RFC 2581 specified ssthresh = FlightSize/2

- "Editorial change" in draft-ietf-tcpimpl-cong-control-01 (Nov 1998)
  - ID diff is here
  - I disagreed at the time but did not have the tools to document the problems
  - Justified in the final document because "cwnd might rise above FlightSize"
- Many other documents follow
  - Obsolete: RFC 2582, RFC 3517
  - Experimental: RFC 4138, RFC 4653
  - Standards track: RFC 5681, RFC 6675, RFC 8511, RFC 9438
- But widely deployed stacks suppress cwnd growth while application limited
  - And compute ssthresh from cwnd, contrary to the advice in RFC 2581 and RFC 5681
    - Linux
    - FreeBSD
    - Others?
  - The justification in RFC 5681 is somewhat moot
- Which stacks do not suppress cwnd growth while application limited?

Downsides to the status quo

- Standard RFC 2581 / RFC 5681 implementations have:
  - Chronic poor performance for transactional and streaming workloads (web, rpc's etc)
  - Erratic performance for bulk transport when losses and stalls are correlated
- Know differences between "as implemented" and standards cause:
  - Double work for people validating standards
  - Different people having different mental models for how the protocols work
- For example in PRR:
  - For RFC 2581, PRR is always entered with ssthresh < FlightSize
  - For Linux, PRR is often entered with ssthresh > FlightSize
- It hurts IETF credibility
  - Mature, well tested and widely deployed implementations that differ from standards
  - We need to reverse engineer: why others made different choices?
  - If they are justified the IETF should consider "harmonization"

What next?

- Survey the extent to which stacks already approximate RFC 7661
  - Specifically avoid cwnd growth while application limited
  - See draft-welzl-ccwg-ratelimited-increase
- Survey how stacks compute ssthresh in fast recovery
- Inventory pro and cons of each approach
- Consider paths to harmonization