RACK: a time-based fast loss recovery
draft-ietf-tcpm-rack-01

Yuchung Cheng
Neal Cardwell
Nandita Dukkipati

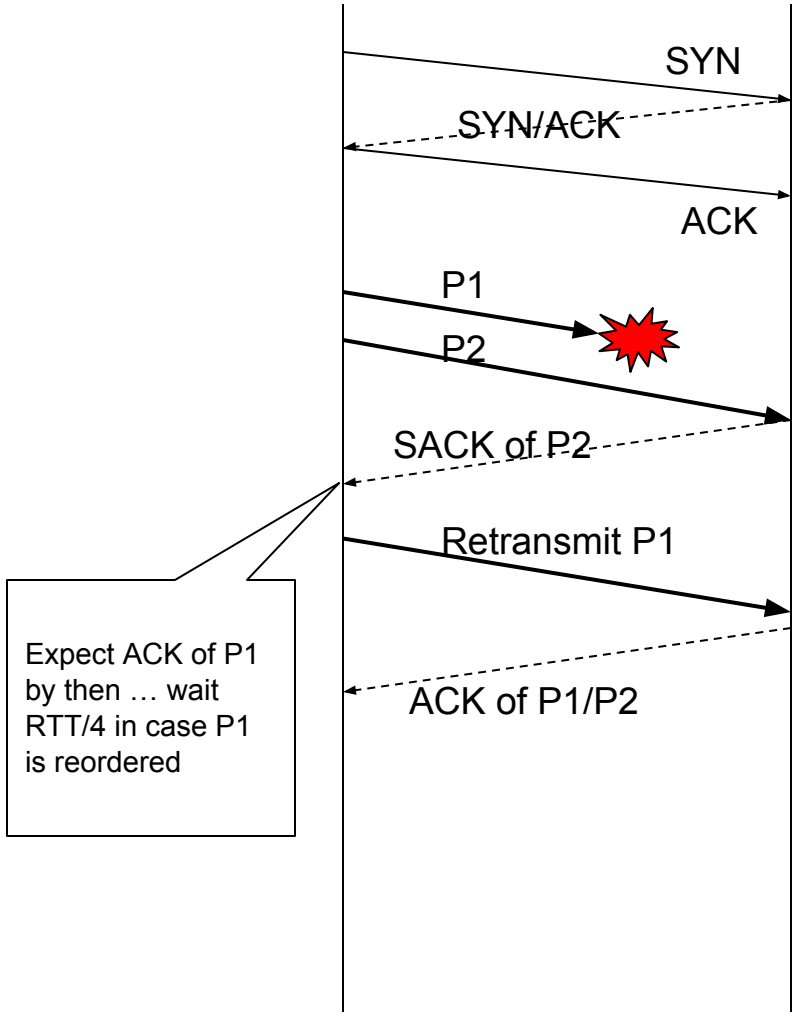
Google

IETF97: Seoul, Nov 2016

What's RACK (Recent ACK)?

Key Idea: time-based loss inferences (not packet or sequence counting)

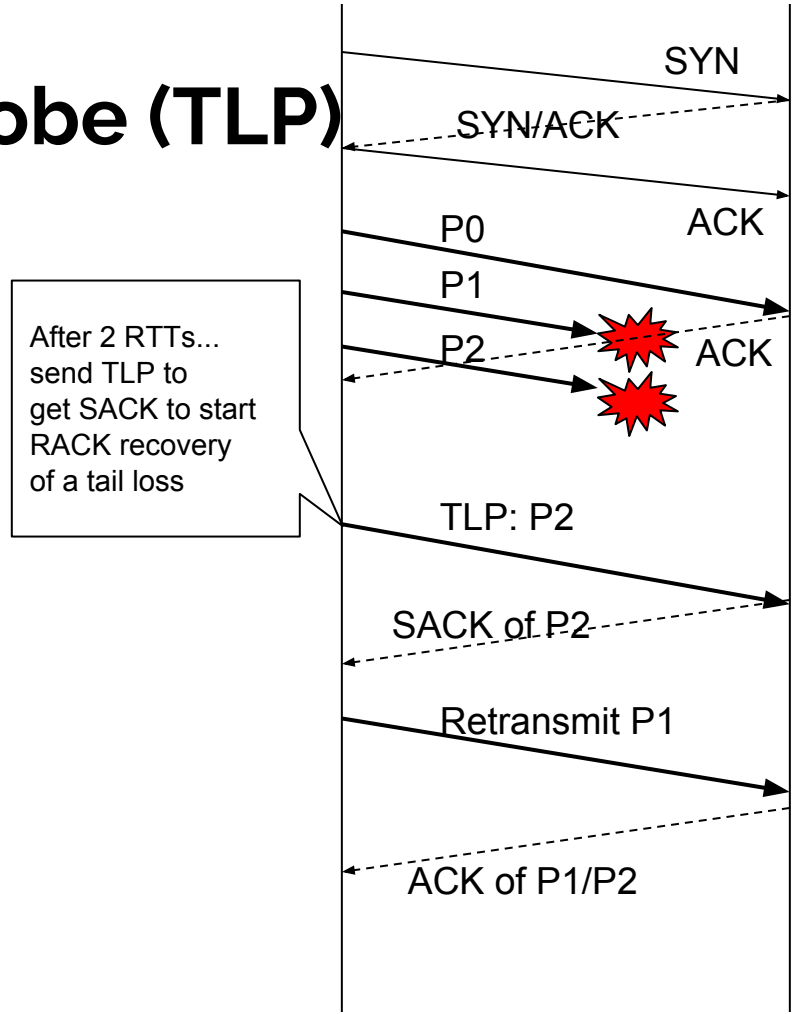
- If a packet is delivered out of order, then packets sent chronologically before it are either lost or reordered
- Wait RTT/4 before retransmitting in case the unacked packet is just delayed. RTT/4 is empirically determined
- Conceptually RACK arms a (virtual) timer on every packet sent. The timers are updated by the latest RTT measurement.



New in RACK: Tail Loss Probe (TLP)

- Problem
  - Tails drops are common on request response traffic
  - Tail drops lead to timeouts which is often 10x longer than fast recovery
  - 70% of losses on Google.com recovered via timeouts
- Goal
  - Reduce tail latency of request response transactions
- Approach
  - Convert RTOs to fast recovery
  - Retransmit the last packet in 2 RTTs to trigger RACK-based Fast Recovery

- draft-dukkipati-tcpm-tcp-loss-probe (expired 2013)
  - Past presentations @ IETF 87 86 85 84
  - Previously depended on non-standard FACK



Why RACK + TLP?

Problems in existing recovery (e.g., wait for 3 dupacks to start the repair process)

1. Poor performance
    - Losses on short flows, tail losses, lost retransmit often resort to timeouts
    - Work poorly with common reordering scenarios
        - e.g. Last pkt is delivered before the first N-1 pkts are delivered. Dupack threshold == N-1
2. Complex
    - Many additional heuristics case-by-case
    - RFC5681, RFC6675, RFC5827, RFC4653, RFC5682, FACK, thin-dupack (Linux has all!)

RACK + TLP's goal is to solve both problems: performant and simple recovery!

Performance impact

A/B test on Google.com in Western-Europe for 3 days in Oct 2016

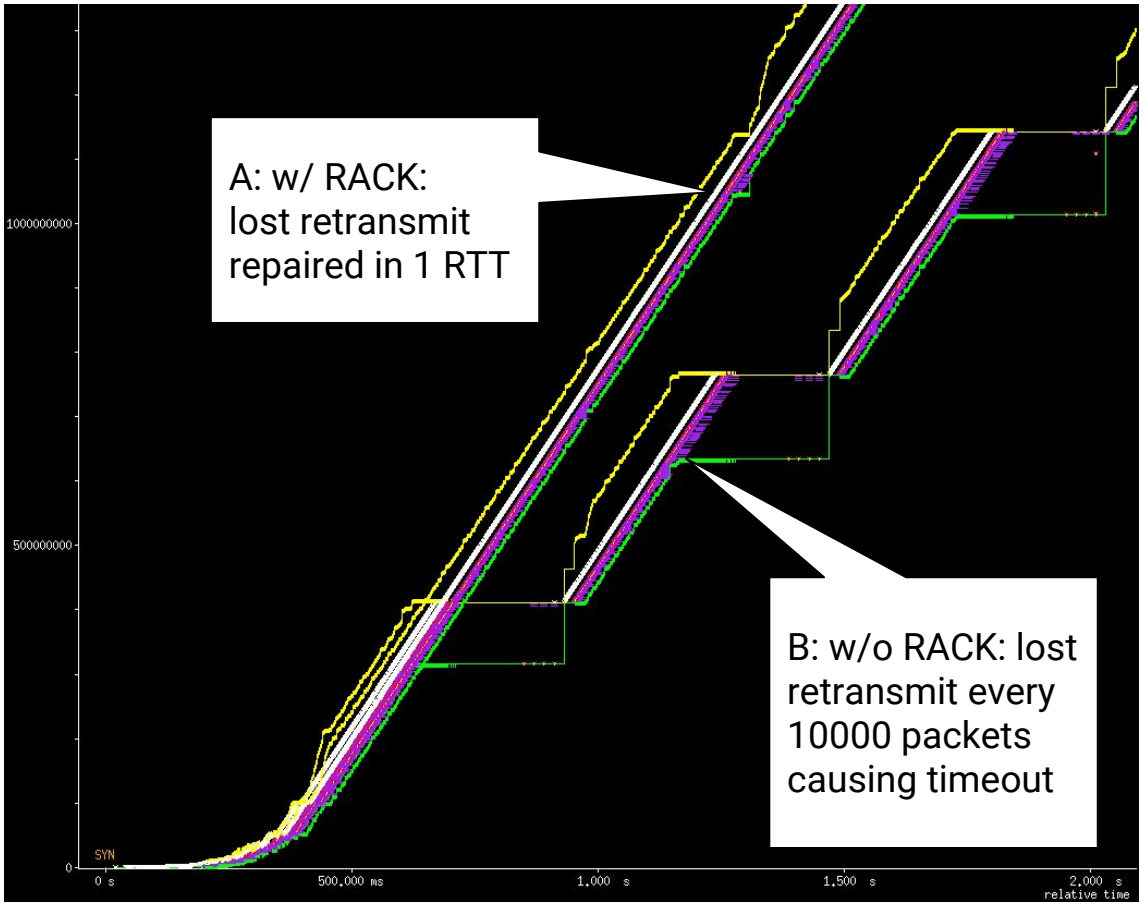
- Short flows: timeout-driven repair is ~3.6x ack-driven repairs
- A: RFC3517 (conserv. sack recovery) + RFC5827 (early retransmit) + F-RTO
- B: RACK + TLP + F-RTO

Impact

- -41% RTO-triggered recoveries
- -23% time in recovery, mostly benefited from TLP
- +2.6% data packets (TLP packets)
    - >30% TLP are spurious as indicated by DSACK

TODO: poor connectivity regions. Compare w/ RACK + TLP only

Timeouts can destroy throughput



20ms RTT, 10Gbps, 1% random drop, BBR congestion control

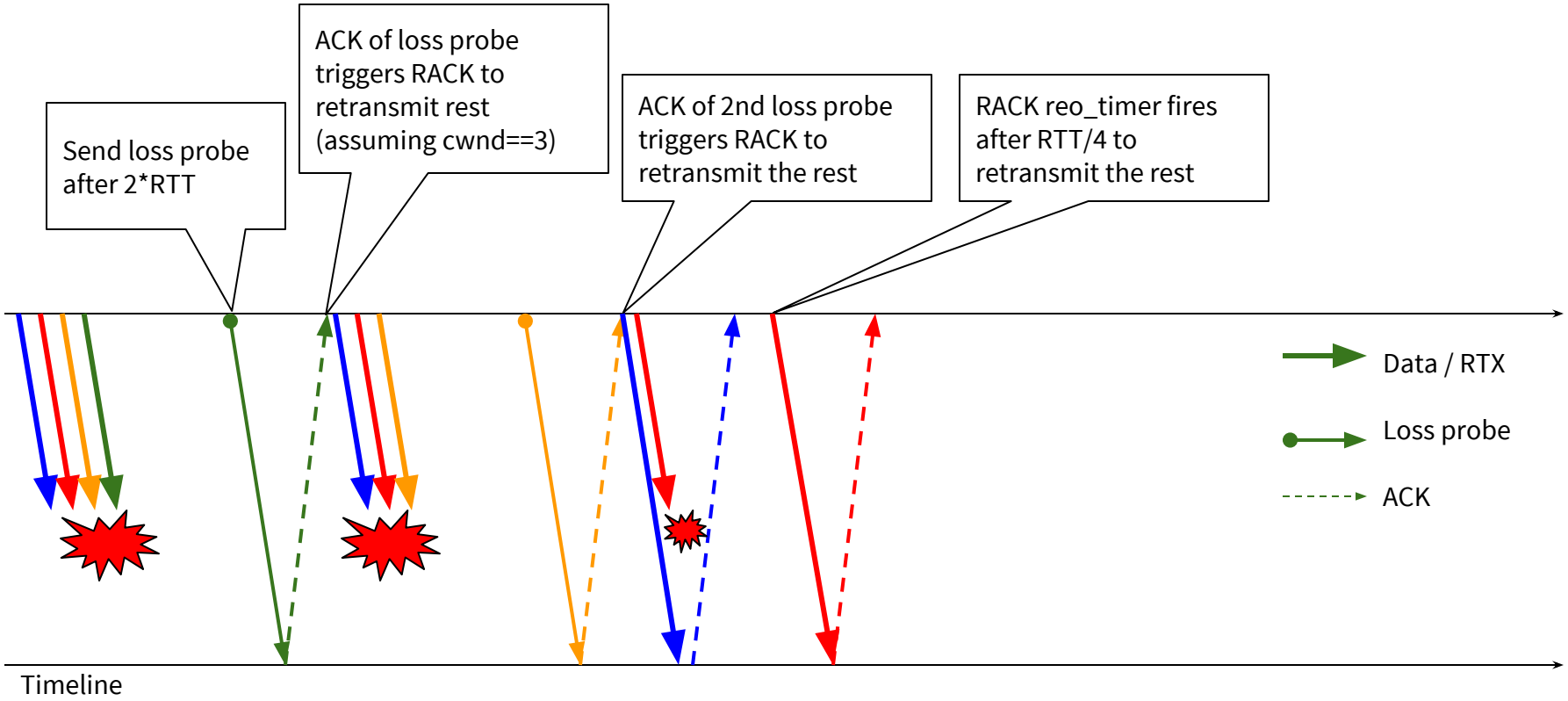
Two tests overlaid:

A: 9.6Gbps w/ RACK

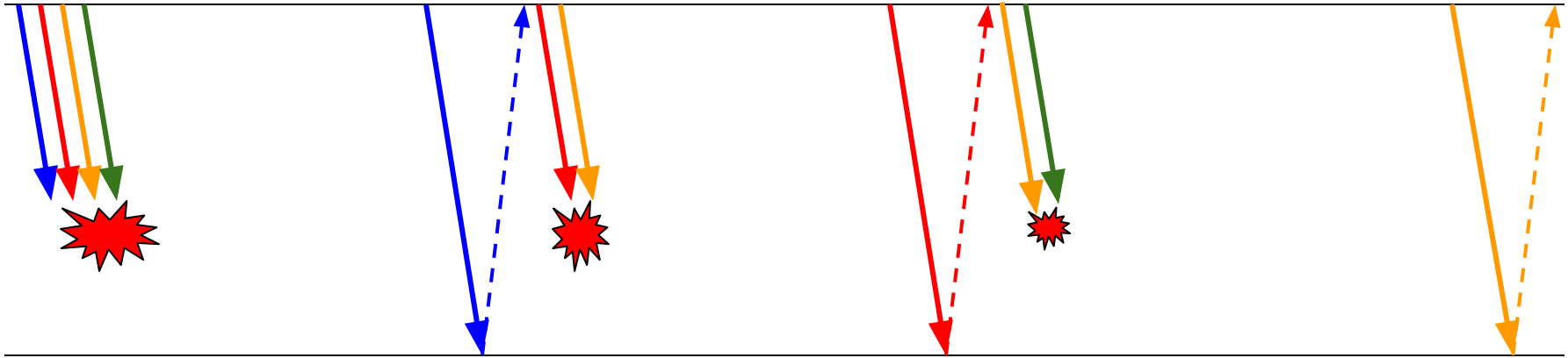
B: 5.4Gbps w/o RACK

Overlaid time-seq graphs of A & B
While line: sequence sent
Green line: cumulative ack received
Purple line: selective acknowledgements
Yellow line: highest receive window allows
Red dots: retransmission

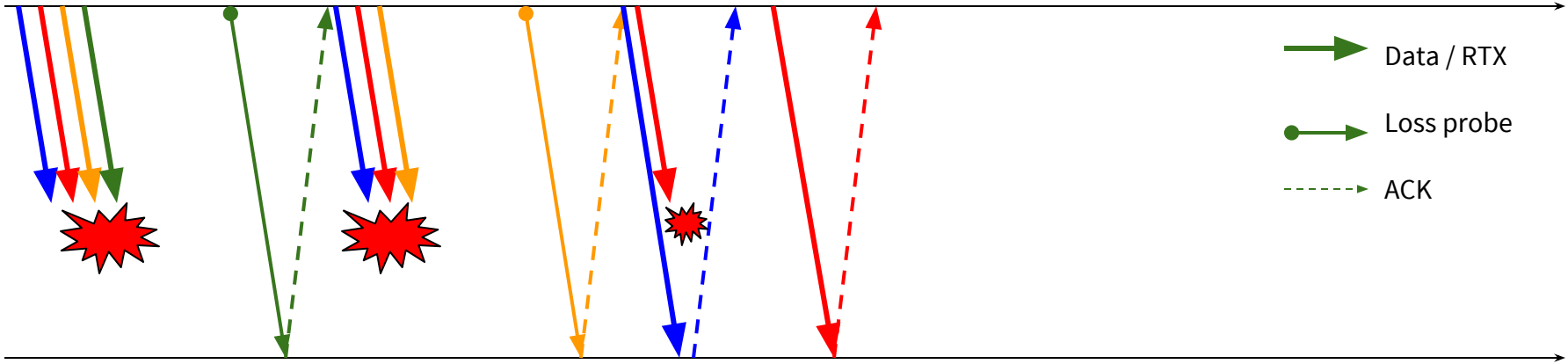
RACK + TLP fast loss recovery example



w/o RACK+TLP: slow repair by timeout (diagram assumes RTO=3*RTT for illustration)



w/ RACK + TLP (same from prev. slide)



TLP discussions

- Why retransmit the last packet instead of the first packet (SND.UNA)?

- When only one packet is in flight
  - Receiver may delayed the ACK: 2*RTT is too aggressive?
    - 1.5RTT + 200ms
  - TLP (retransmit the packet) may masquerade a loss event
    - Draft suggest a (slightly complicated) detection mechanism
    - Do we really care 1-pkt loss event?

- How many TLPs before RTO?
  - Draft uses 1, but more may help?
- Too many timers (RACK reo_timer, TLP timer, RTO)
  - Can easily implemen with one real timer b/c only one is active at any time

WIP: extend RACK + TLP to mitigating spurious RTO retransmission storm

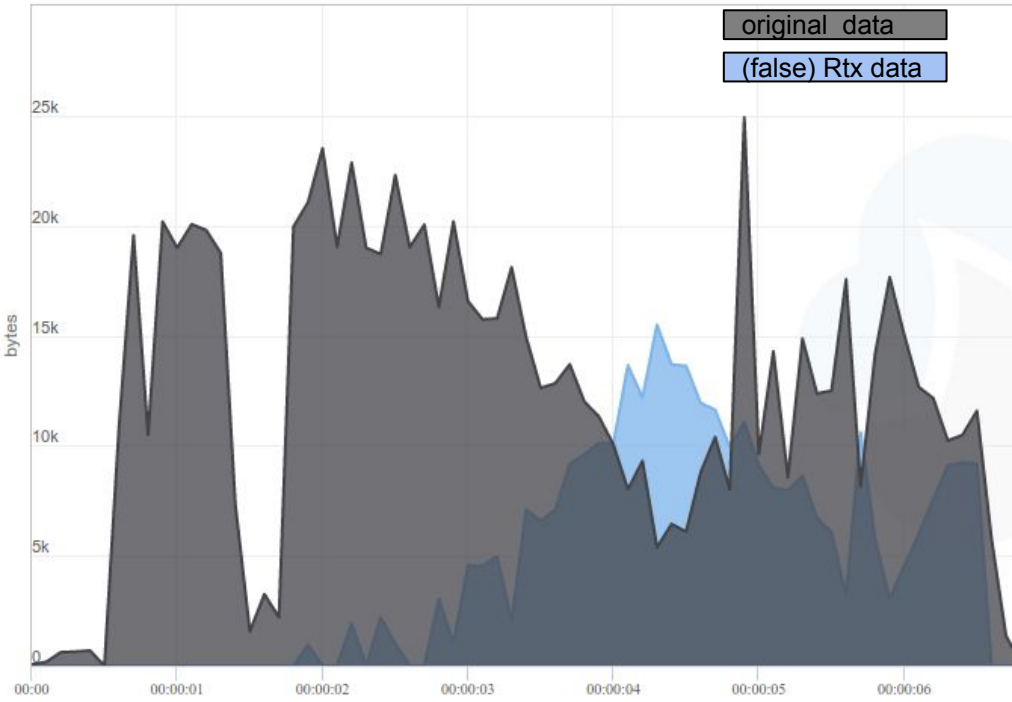
Retransmission storm induced by spurious RTO

1. (Spurious) timeout!
   Mark all packets (P1… P100)lost, retransmit P1
2. ACK of original P1, retransmit P2 P3 spuriously
3. ACK of original P2, retransmit P4 P5 spuriously
4. … End up spuriously retransmitting all
   a. Double the bloat and queue

Extend RACK + TLP to mitigating spurious RTO retransmission st

Retransmission storm induced by spurious RTO

1. (Spurious) timeout!
   Mark all packets (P1… P100)lost, retransmit P1
2. ACK of original P1, retransmit P2 P3 spuriously
3. ACK of original P2, retransmit P4 P5 spuriously
4. … End up spuriously retransmitting all
   a. Double the bloat and queue



Time-series of bytes received on Chrome loading many images in parallel from pinterests.com:
incast -> delay spikes -> false RTOs -> spurious RTX storms

Extend RACK + TLP to mitigating spurious RTO retransmission storm

Retransmission storm induced by spurious RTO

1. (Spurious) timeout!
   Mark all packets (P1… P100)lost, retransmit P1
2. ACK of original P1, retransmit P2 P3 spuriously
3. ACK of original P2, retransmit P4 P5 spuriously
4. … End up spuriously retransmitting all
   a. Double the bloat and queue

Extending RACK + TLP to RTOs could save this!

1. (Spurious) timeout!
   Mark first packet (P1) lost, retransmit P1
2. ACK of original P1, retransmit P99 and P100 (TLP)
3. ACK of original P2
   ==> never retransmitted P2 so stop!

(If the timeout is genuine, step 3 would receive ACK of P99 and P100, then RACK would repair P2 … P 98)

RACK + TLP as a new integrated recovery

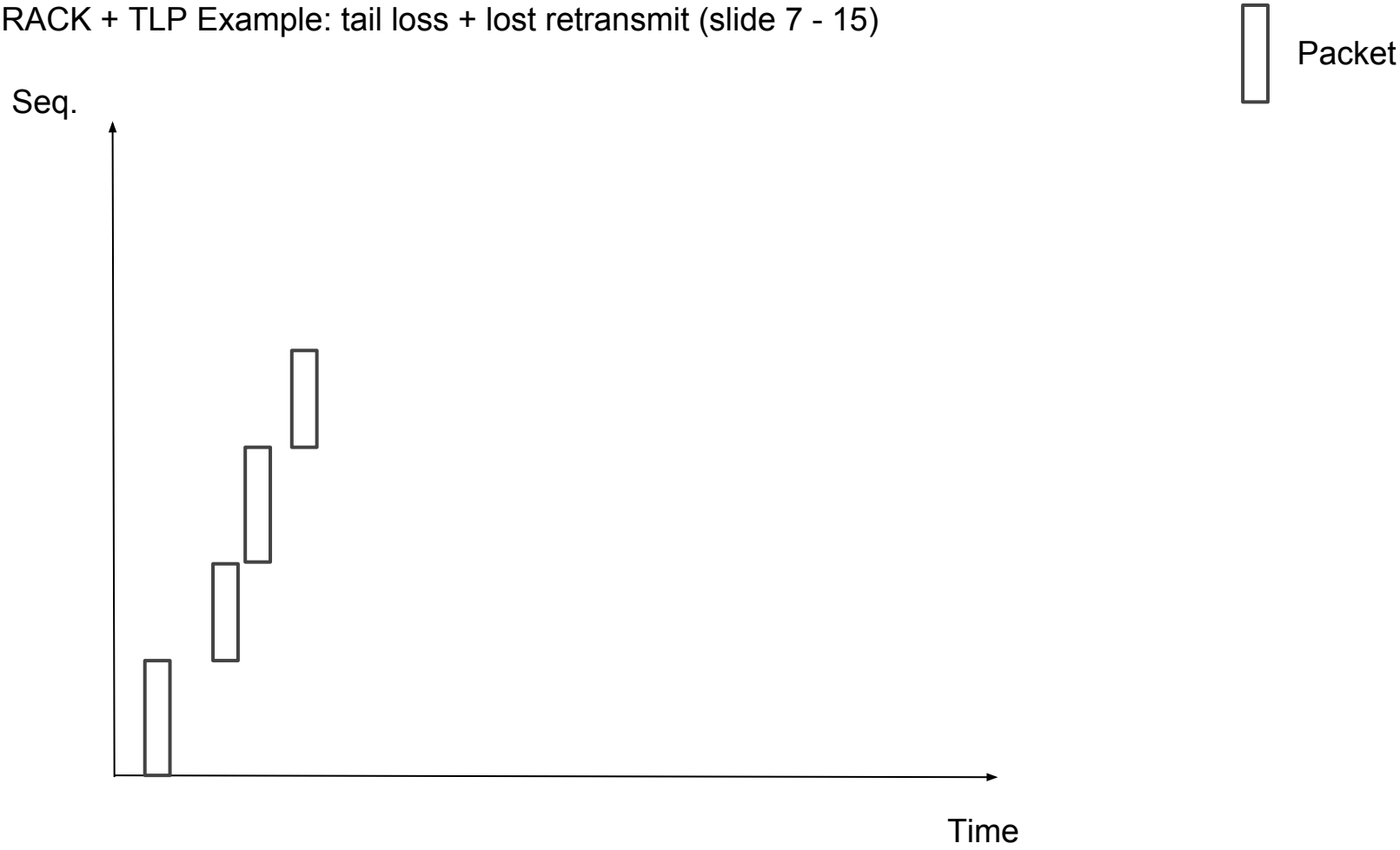
- Conceptually more intuitive (vs N dupacks mean loss)
- ACK-driven repairs as much as possible (even lost retransmits)
- Timeout-driven repairs as the last resort
  - Timeout can be long and conservative
  - End RTO tweaking game risking falsely resetting cwnd to 1
- Robust under common reordering (traversing slightly different paths or out-of-order delivery in wireless)

- Experimentation: implemented as a supplemental loss detection
  - Progressively replace existing conventional approaches
  - In Linux 4.4, Windows 10/Server 2016, FreeBSD/NetFlix

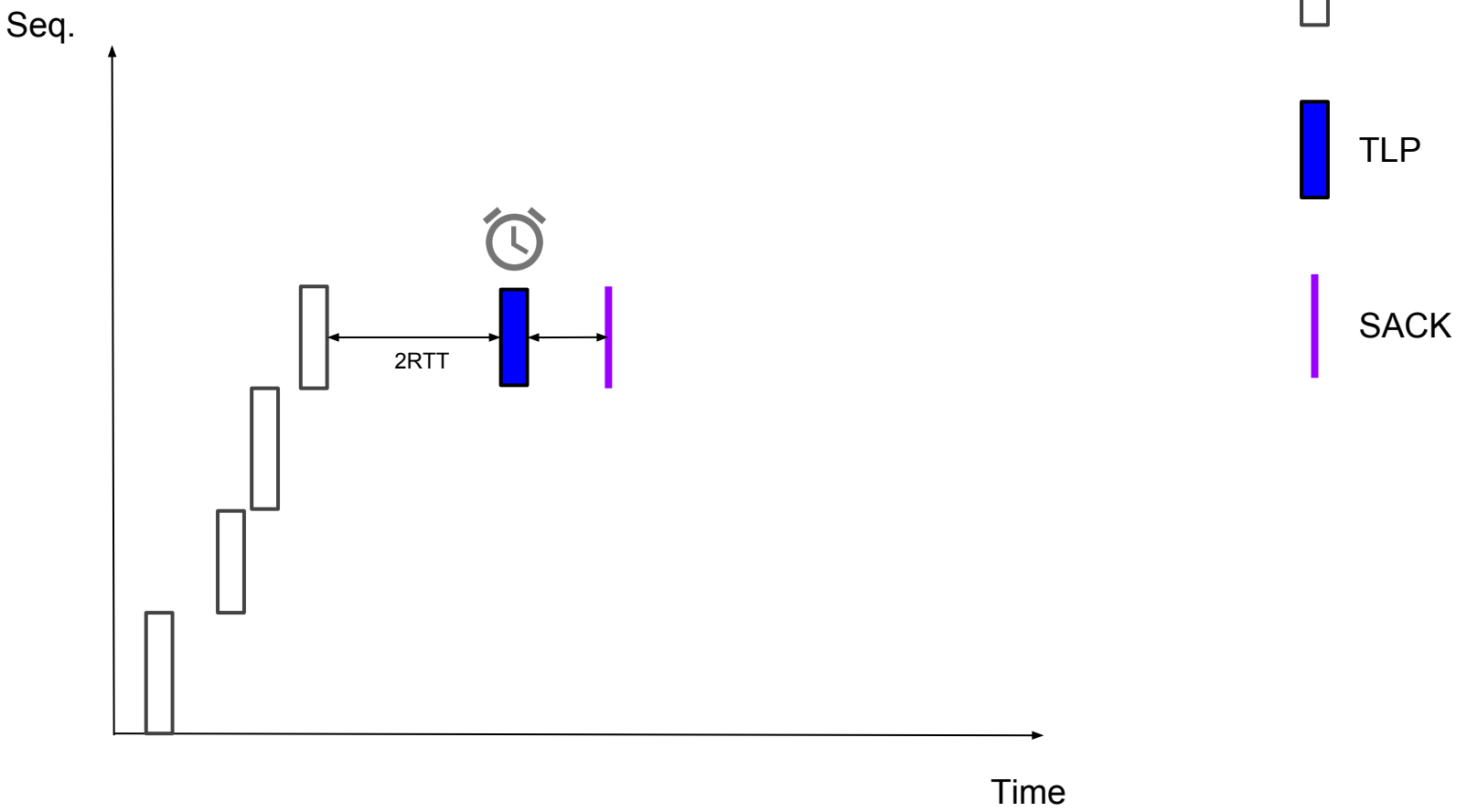
- **Please help review the draft and share any data and implementation experiences on tcpm list!**

Backup slides

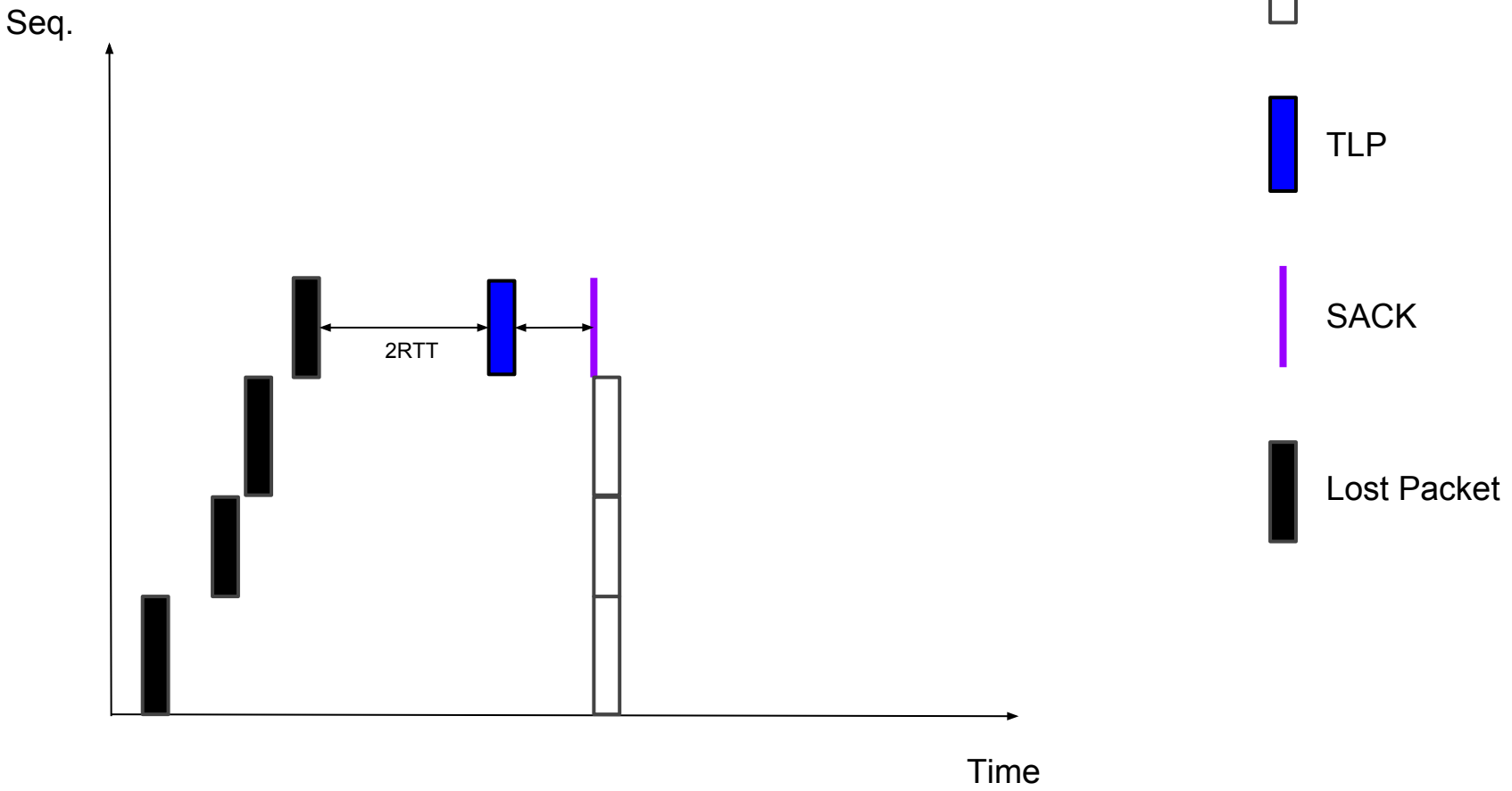
RACK + TLP Example: tail loss + lost retransmit (slide 7 - 15)



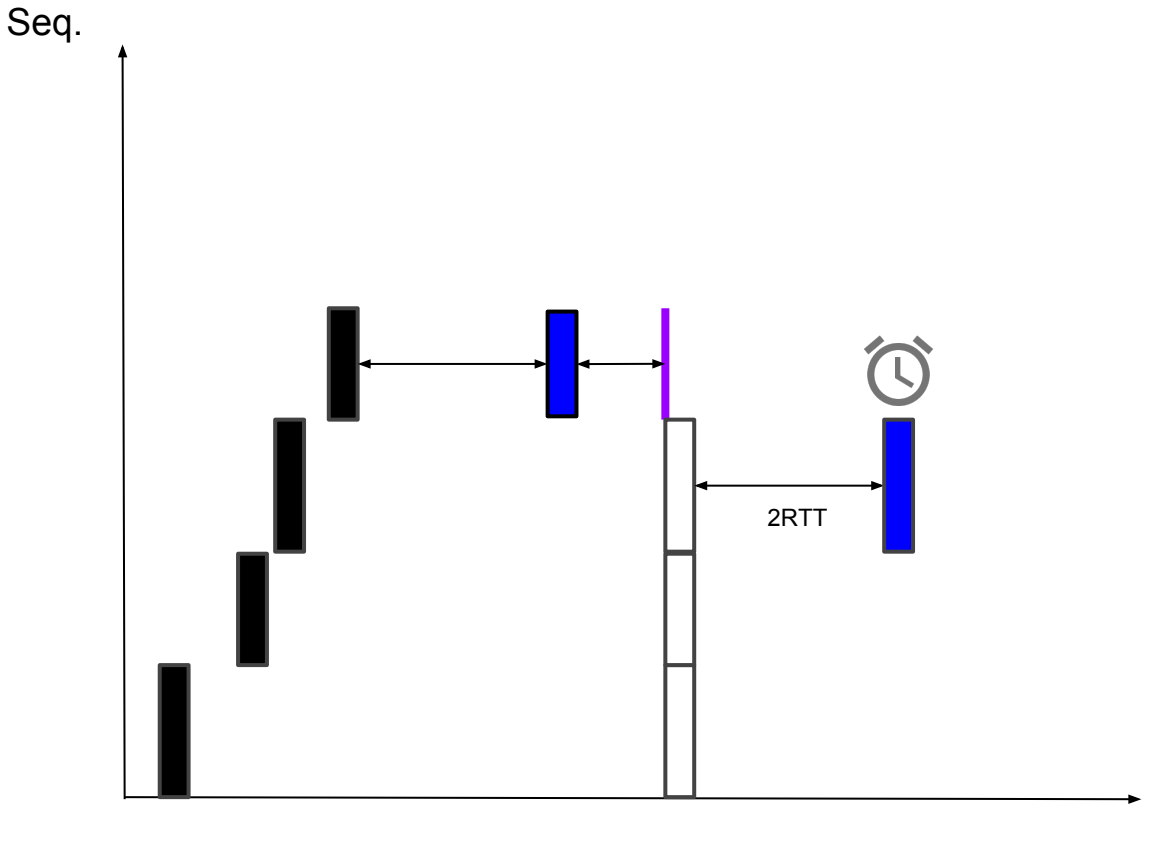
TLP retransmit the tail, soliciting an ACK/SACK



RACK detects first 3 packets are lost from the ACK/SACK, and retransmits



After 2RTT send a TLP again



Packet

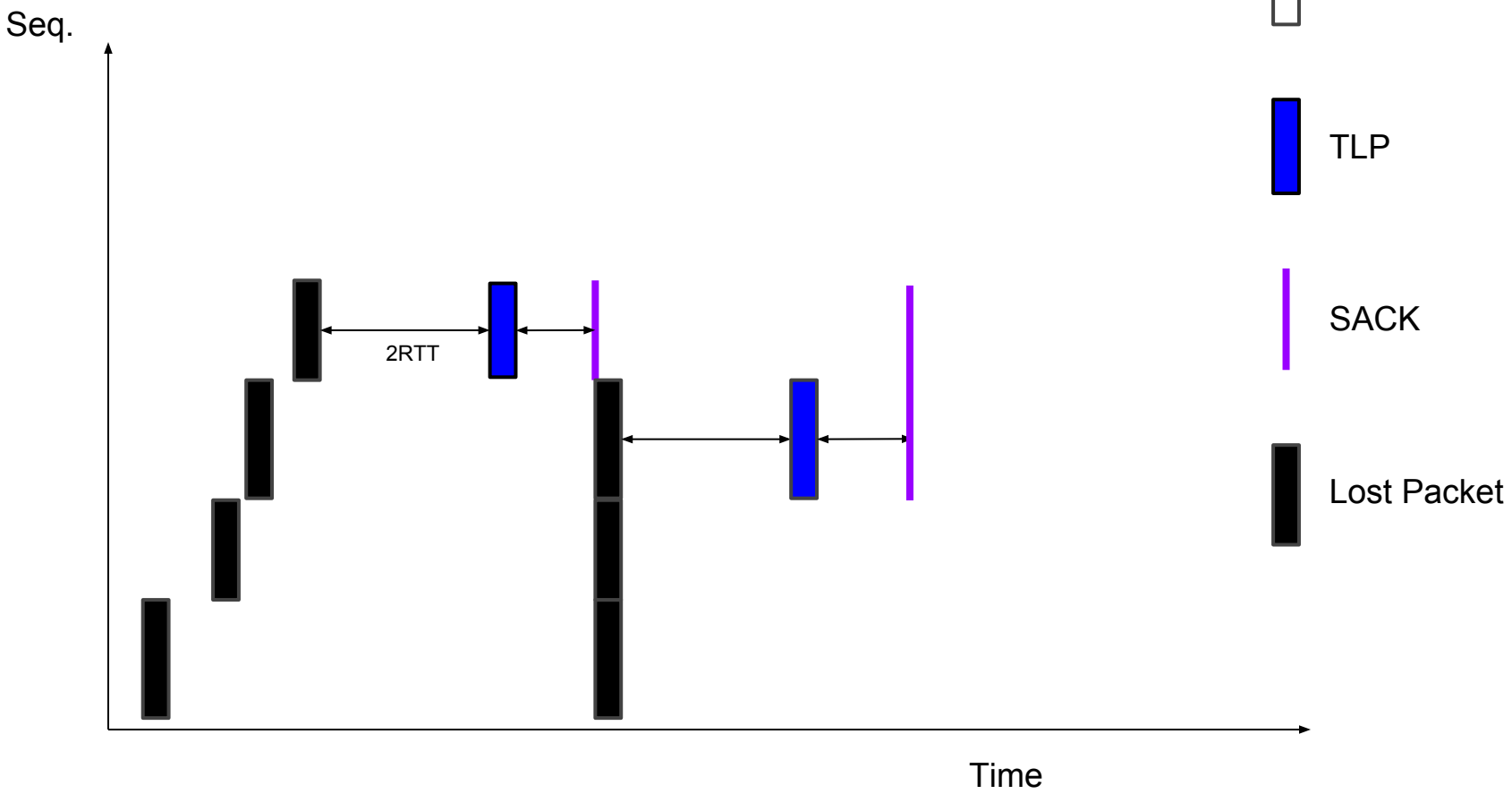
TLP

SACK

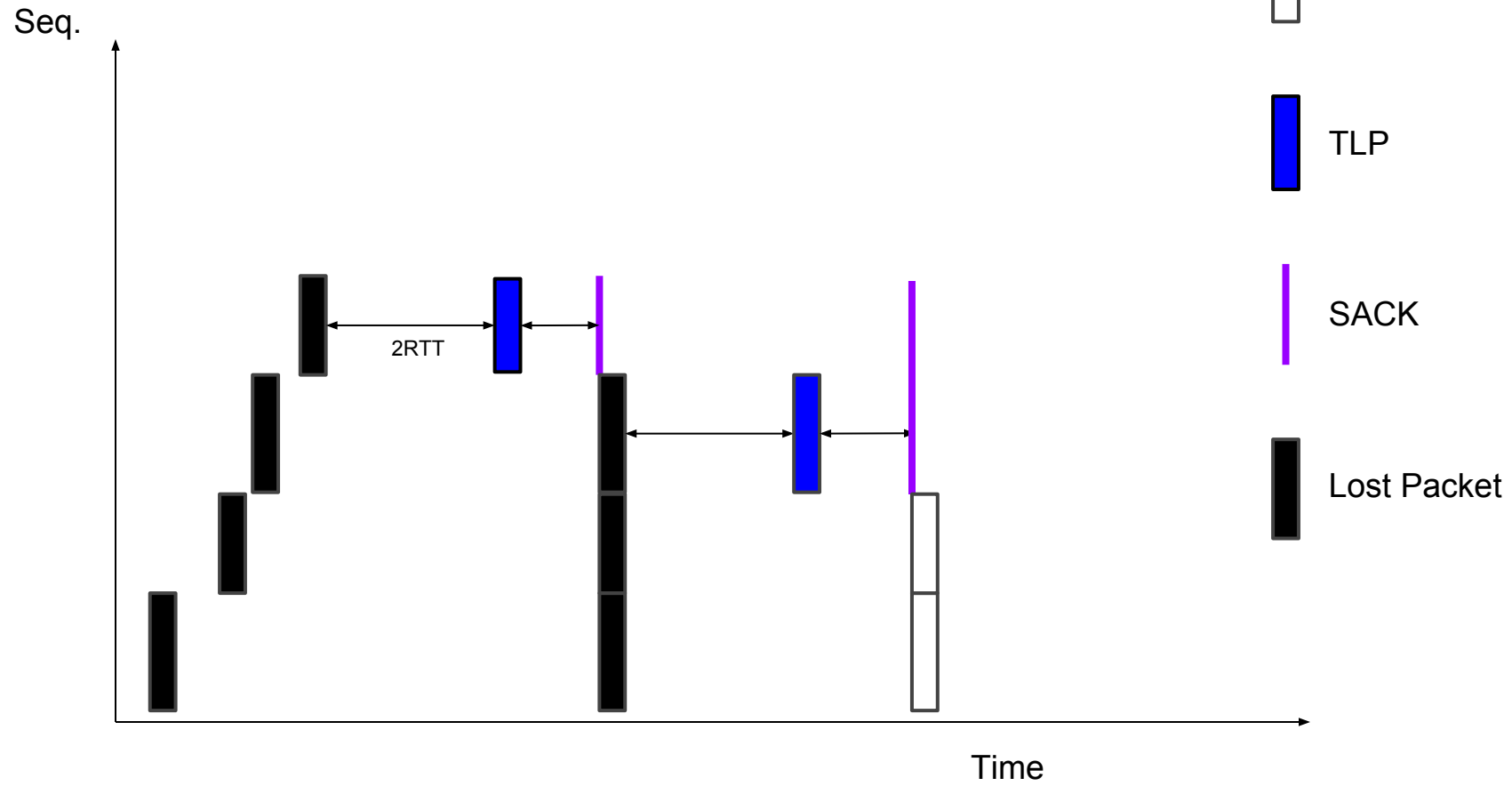
Lost Packet

(Need to update draft-02 to probe in recovery)

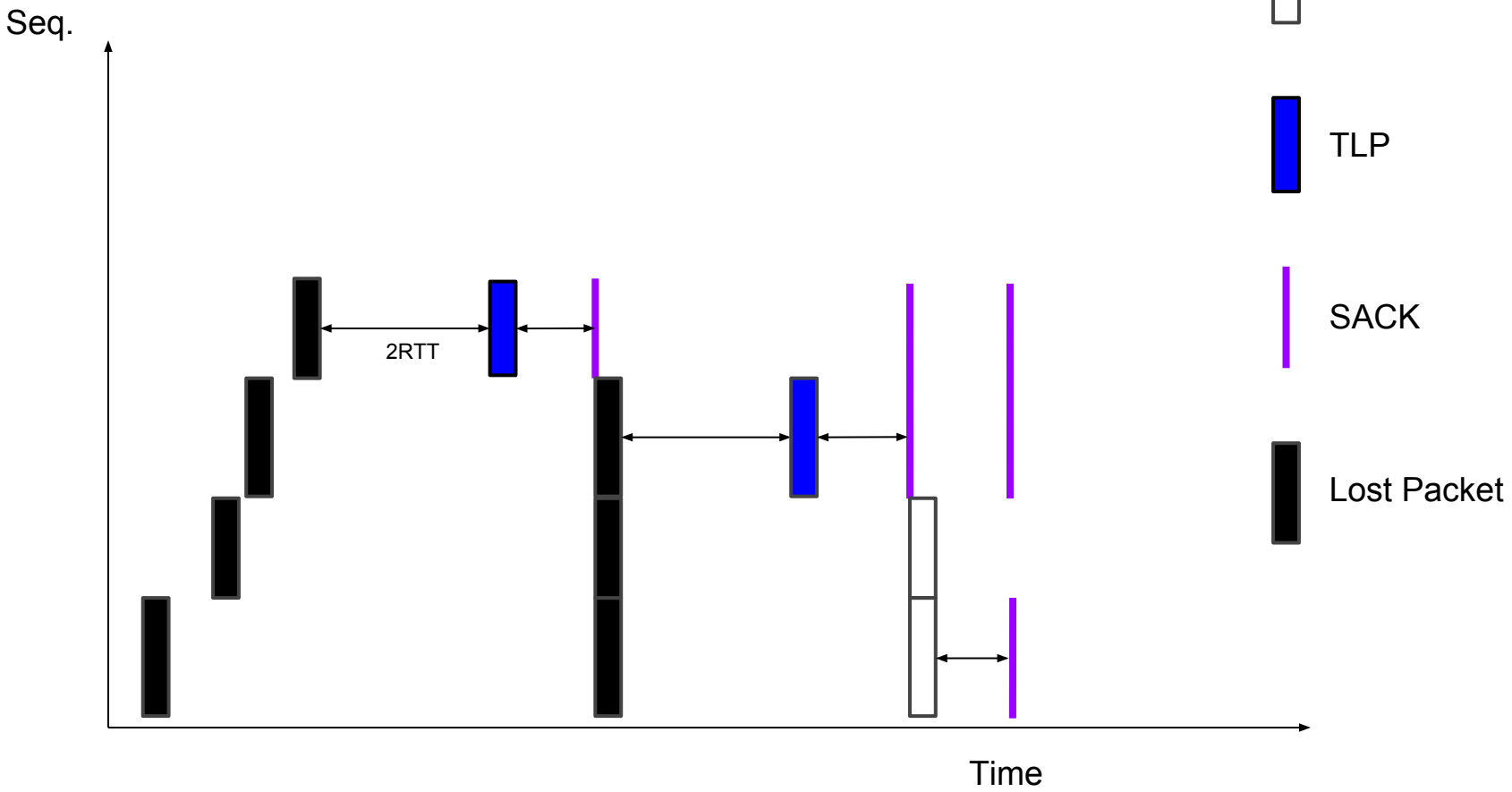
The TLP solicits another ACK/SACK



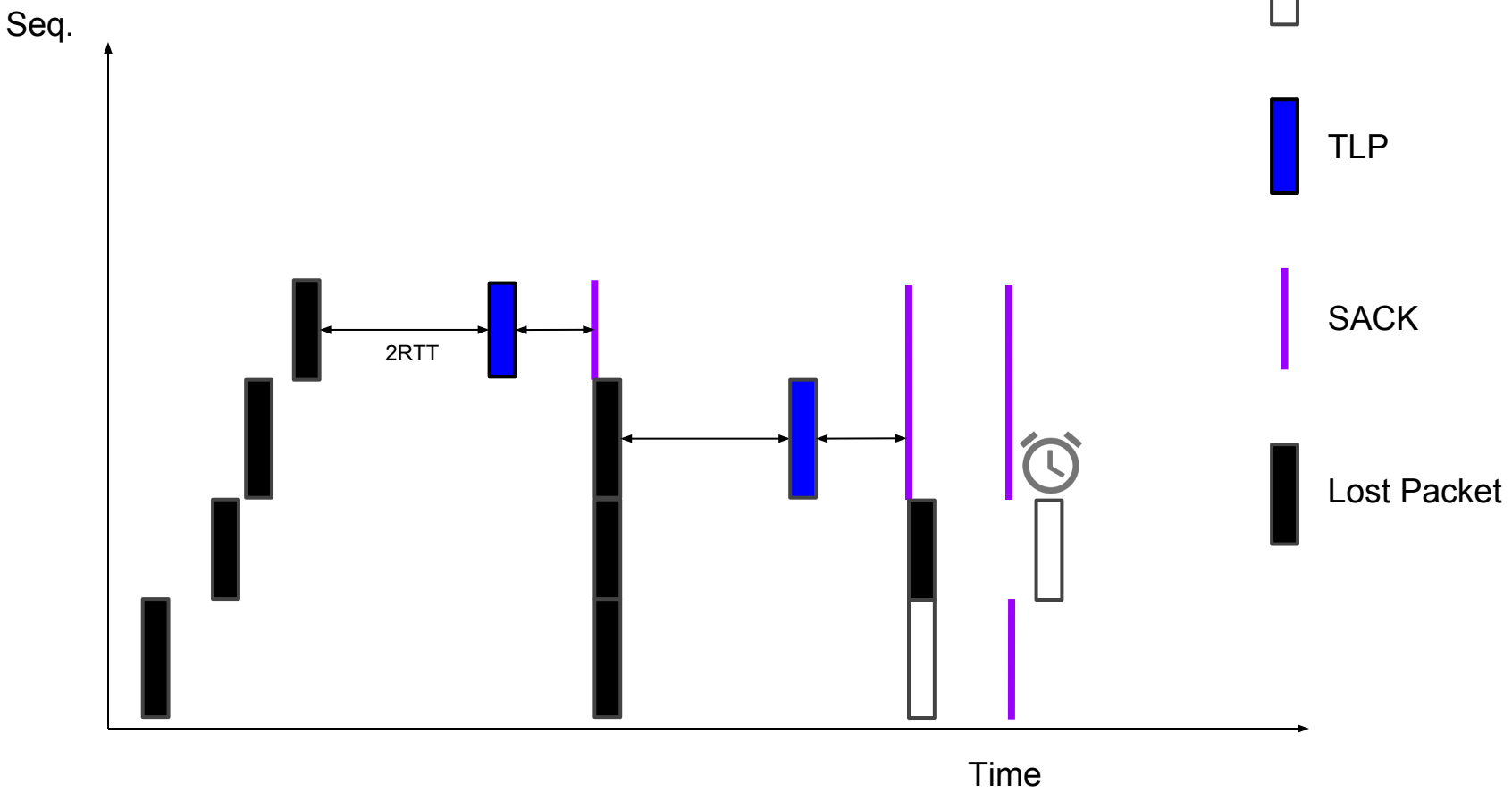
The ACK/SACK let RACK detect first two retransmits are lost and retransmit them (again)



The new ACK/SACK indicates 1st packet is lost for the 3rd time



After waiting, RACK detects the lost retransmission and retransmits again



All acked and repaired: loss rate = 8/4 = 200%!