RACK: a time-based fast loss recovery
draft-ietf-tcpm-rack-02

Yuchung Cheng
Neal Cardwell
Nandita Dukkipati

Google

IETF98: Chicago, March 2017

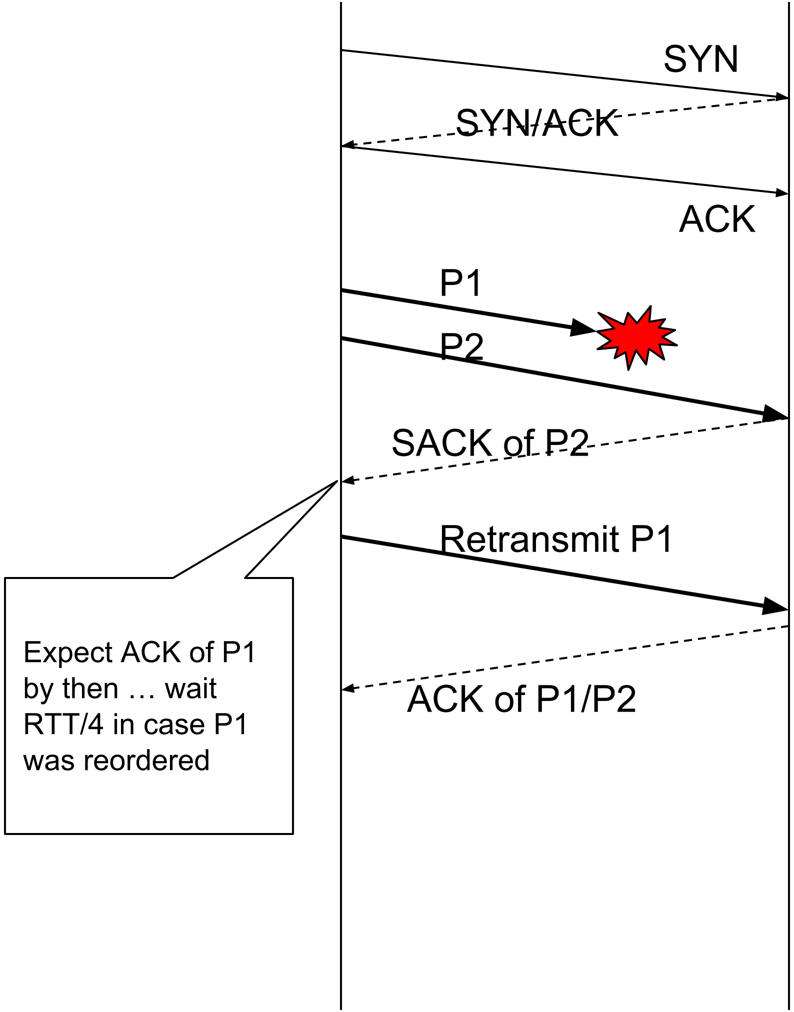
What's RACK (Recent ACK)?

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 (more later on making it adaptive)

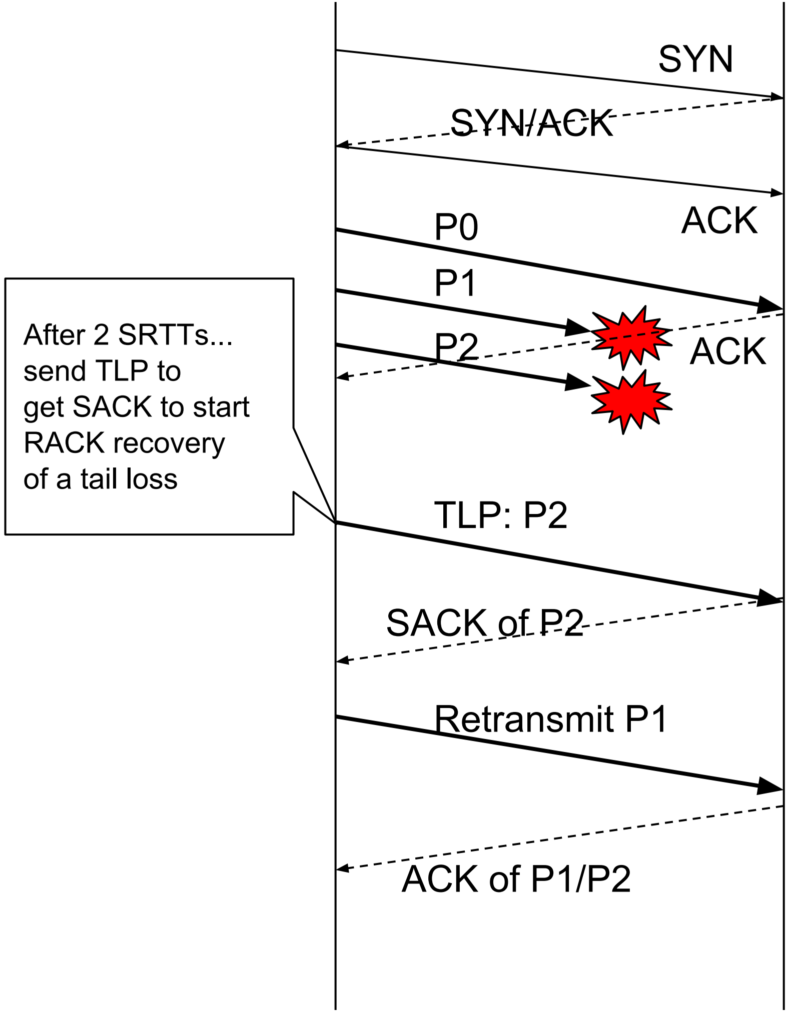
- Conceptually RACK arms a (virtual) timer on every packet sent. The timers are updated by the latest RTT measurement.



Tail Loss Probe (TLP)

- Problem
  - Tail drops are common on request/response traffic
  - Tail drops lead to timeouts, which are 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
  - Solicit a DUPACK by retransmitting the last packet in 2 SRTTs
  - Requires RACK to trigger fast recovery



What's new since last IETF in Nov.

1. Fully implemented in Linux 4.10
    a. On by default
    b. Reduced number of loss recovery heuristics from 9 to 4:
    RACK, TLP, F-RTO, DupThresh (RFC6675), ~~FACK, Early Retransmit (RFC5827), Thin-D~~
    ~~(RFC4653), Forward Retransmit~~
    c. Deployed in Google TCP

2. -02 draft
    a. New experiments on reordering window length and removing DupThresh
    b. New text on interacting with congestion control

Exp: RACK+TLP vs DupACK threshold

Diffs compared to RFC6675 (DupThresh)			
	RACK	RACK + TLP	RACK + TLP + RFC6675 (Linux)
Time in loss recovery	-0.5%	-24.0%	-24.1%
%RTO reduced	-5.4%	-25.8%	-23.7%
Retrans. Rate (including TLP)	1.3%	1.5%	1.5%

4-way experiment at one Google DC in Europe for a week in 2017. ~1.5B flows sampled

- RFC6675-only retransmit rate is 1.3%
- RACK + TLP reduces recovery latency by 24% and may replace DupThresh approach

How RACK interacts with congestion control

RACK influences congestion control indirectly

- Congestion control (Reno/RFC5681)
  - On fast recovery cwnd is reduced to ssthresh
  - On RTO cwnd resets to 1
- By reducing RTOs, RACK + TLP speeds up fast recovery and avoids cwnd resets
  - Rationale: cwnd should only reset if the entire flight is lost <u>and</u> the ack clock is also lost

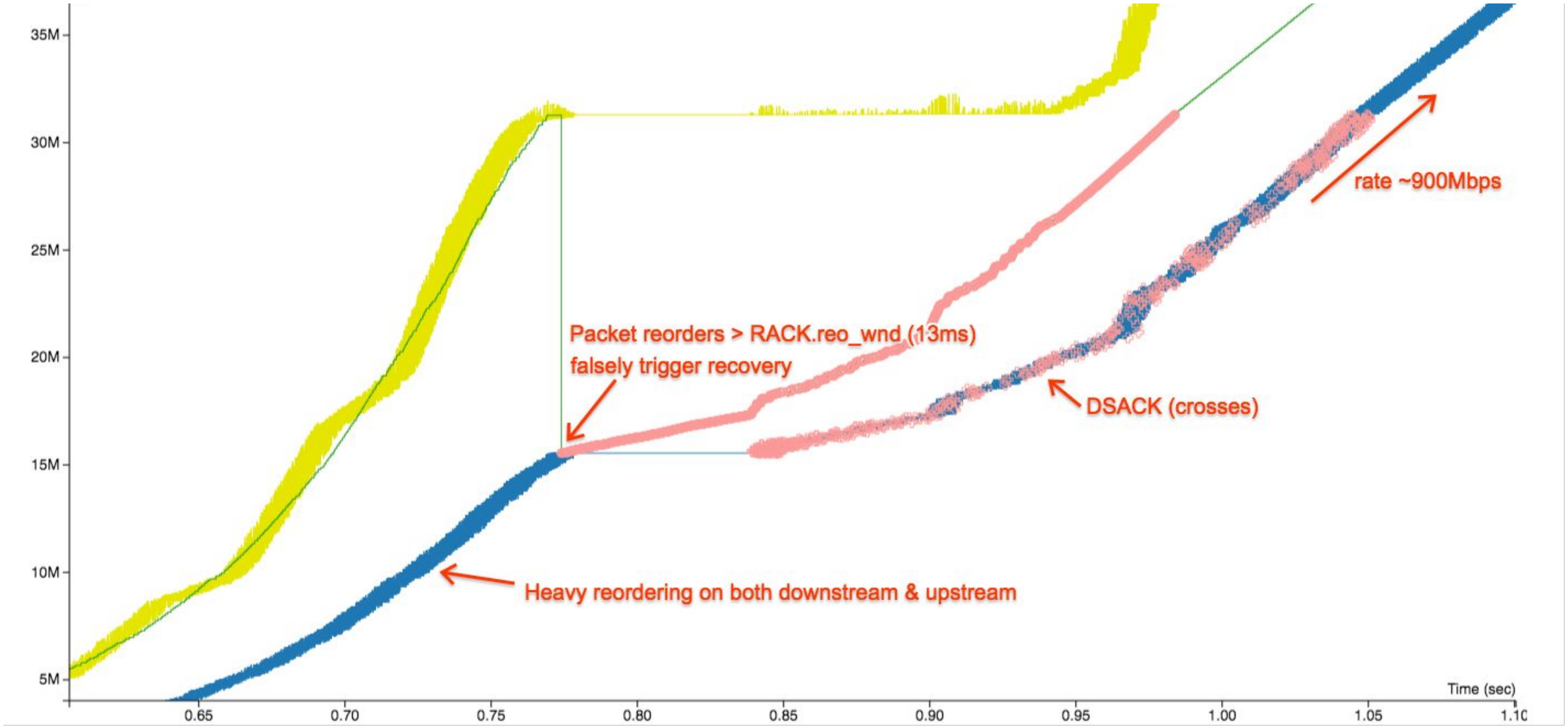
Example: Reno C.C. w/ cwnd=20. Send 10 packets, which are all dropped.			
	Events	Recovery Time	cwnd upon recovery ends
Standard	RTO then slow start	RTO + 4*RTT	1*2*2 = 4
RACK + TLP	Fast recovery (PRR slow start)	2*RTT + 4*RTT	20/2 = 10

Next: smarter reordering window

Current window is max(1ms, min_RTT/4)

- Too low: high spurious retransmit if reordering exceeds the window
- Too high: 1ms is too high inside a data-center (RTT < 100us)
    - But <1ms timer has high cost
- WIP: adaptive reordering window
    - Measure reordering degree in time
        - reor_deg = (last_out_of_order_delivery_time - last_inorder_delivery_time)
        - reo_wnd = K * reordering_degree
    - Reduce K if recovery finished w/o signs of reordering
    - Increase K if DSACKs or timestamps indicate reo_wnd was too small
    - Stress test on low-latency (<100us) and high-reordering (multi-path) environments

BBR/RACK on emulated 1Gbps, 53ms RTT and random packet delay jitter [0, 10ms]



RACK may cause excessive spurious retransmits if reordering > RACK.reo_wnd

Next: one loss recovery (RACK)

Linux still uses both RACK and RFC6675 (DupThresh)

- Runs both algorithms on each ACK
- Recovery starts when either algorithm marks a packet lost

Goal: RACK + TLP as the omnipotent recovery

- a/b experiment disabling RFC6675 on Google
- Experiment w/ DupThresh-triggered start to fast recovery
  - reo_wnd = 0 if not in recovery and #DupAcks >= DupThresh
  - Progressively phase out DupThresh approach

Conclusion

Vision: making TCP resilient and efficient to reordering and loss with <u>one</u> algorithm

- Better load-balancing (e.g. multi-paths, flowlets)
- Faster forwarding (e.g. parallel forwarding, wireless link layer optimization)
- Simpler transport with time-based approach

RACK is now the key loss recovery in Linux

Work-in-progress

1. Optimize reordering window for high reordering and/or low RTT
2. Pure time-based recovery by completely retiring DupThresh approach