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

Yuchung Cheng
Neal Cardwell
Nandita Dukkipati
Priyaranjan Jha
Google

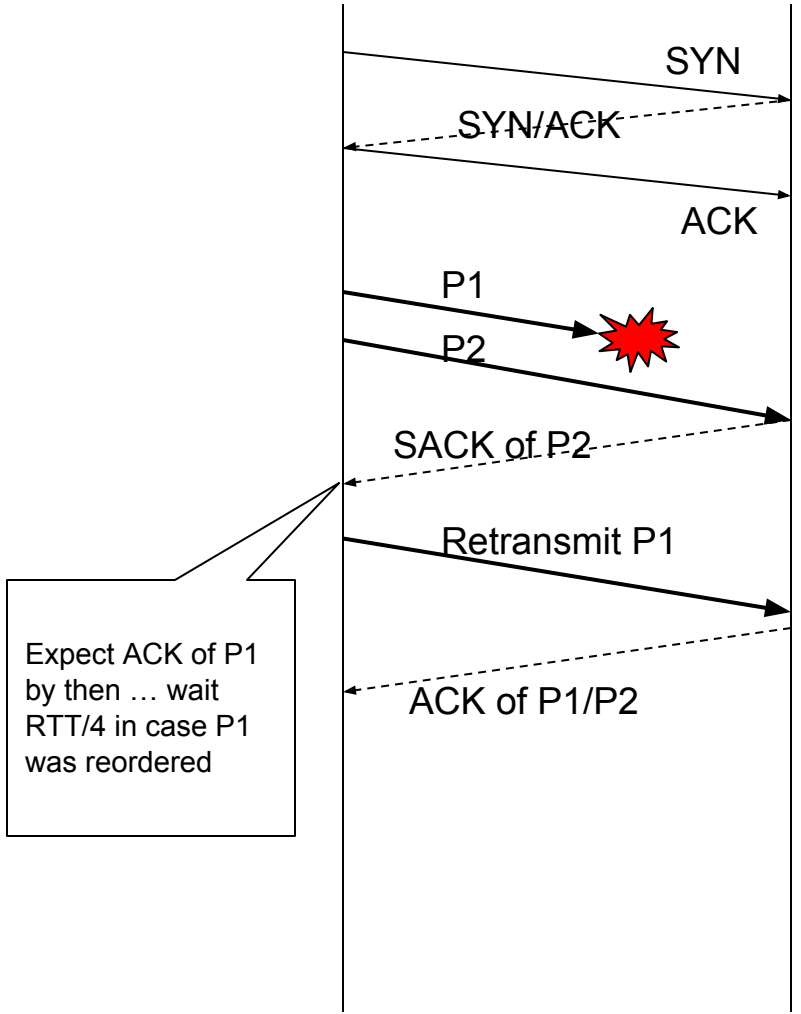
IETF 100: Singapore, Nov 2017

What's RACK (Recent ACK)?

Time-based loss inferences instead packet or sequence counting

- Conceptually every sent packet has a timer
- All timers are constantly adjusted based on most recent RTT sample
- A packet is retransmitted after RTT + reo_wnd

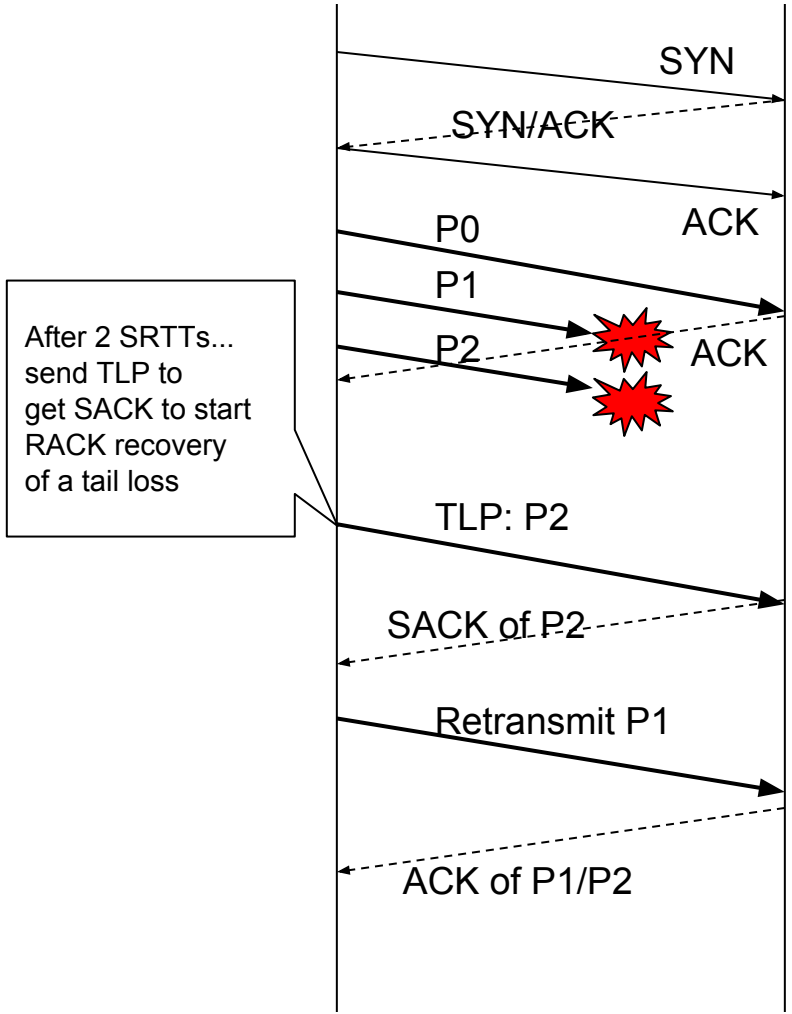
- RACK is about implementing this w/ one timer per connection and ACK events



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



Status updates

Deployments

- Linux, Google, NetFlix(BSD), Windows use RACK/TLP by default

Major changes since IETF 98

- Optimize paths with large BDP
- Optimize paths with frequent reorderings
- Fix a stalling issue due to middle-boxes

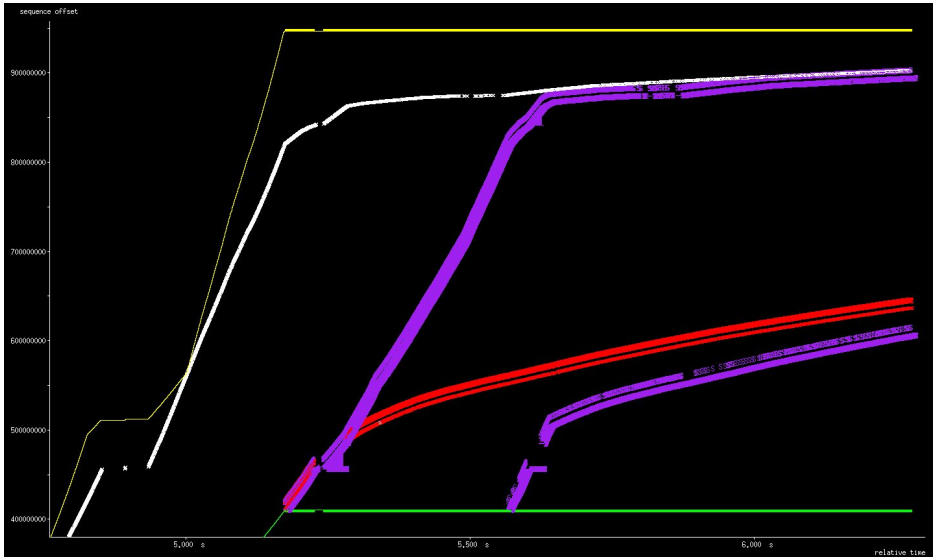
Large BDP paths

ACKs of doom on a long distance Google cloud transfer

- BDP = 10Gbps x 308ms = 385MB =~ 257K packets
- Both Linux SACK and RACK processing were O(n)
  - Write queue is a linked list w/ hint pointers
  - Worst case scans the entire queue
  - 4ms per ACK processing time

- Poor CPU efficiency and loss recovery performance b/c CPU is saturated by ACK processing:

```
Profile:
  29.89%  [kernel]        [k] tcp_rack_detect_loss
  24.57%  perf            [.] 0x0000000000045199
   4.64%  libc-2.19.so    [.] 0x0000000004c17ef1
   2.09%  [kernel]        [k] copy_user_enhanced_fast_string
...
```



Paced TCP CUBIC on a 10Gbps WAN path with RTT=308ms. Fast recovery at end of initial slow start.

Solution: better data structures

SACK processing

- s/linked list/rb tree/ for O(log n) worst case

RACK processing

- Sender keeps a new list for (re)transmitted packets
  - Ordered by packet's last tx tstamp
  - A packet is removed from the list if S/ACKed or deemed lost
  - O(1) upkeep
- For every ACK, checks only packets sent before the most recently acked
  - Fastest possible

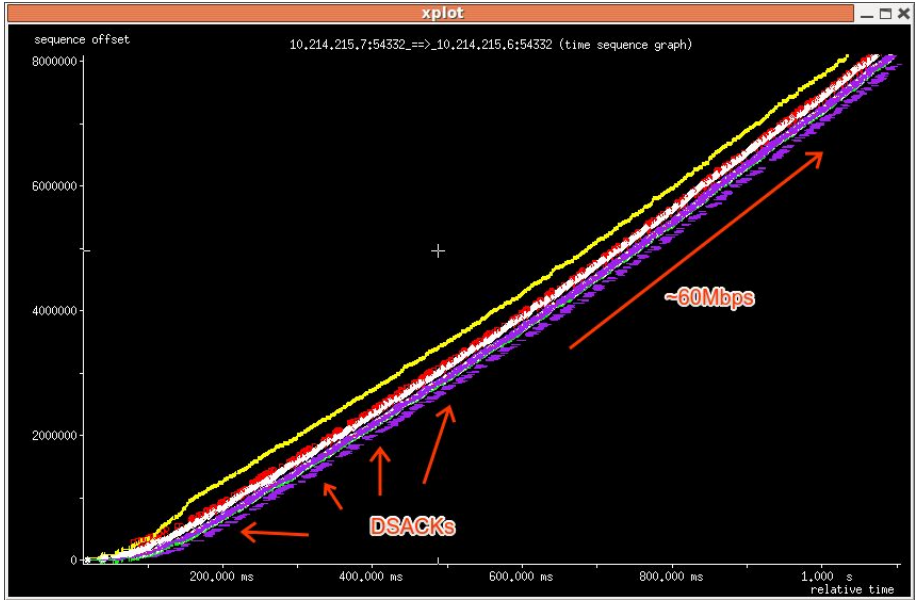
- Both improvements are in Linux 4.15
  - Reduce per SACK processing by two orders of magnitude on large BDP networks
  - TODO: update RACK draft

Paths with frequent reorderings

RACK uses static reo_wnd (min_RTT/4)

On a path that has frequent higher degree of reordering

- Frequent false recoveries causing C.C. to slow down
- Reverting cwnd upon detecting spurious recovery (TCP Eifel) can’t help much: sender enters another false recovery right after the cwnd revert



TCP-BBR on 100Mbps w/ rand[15ms,25ms] RTT.
TCP is constantly in (false) recoveries (**R** is retransmission, (D)SACK is purple)

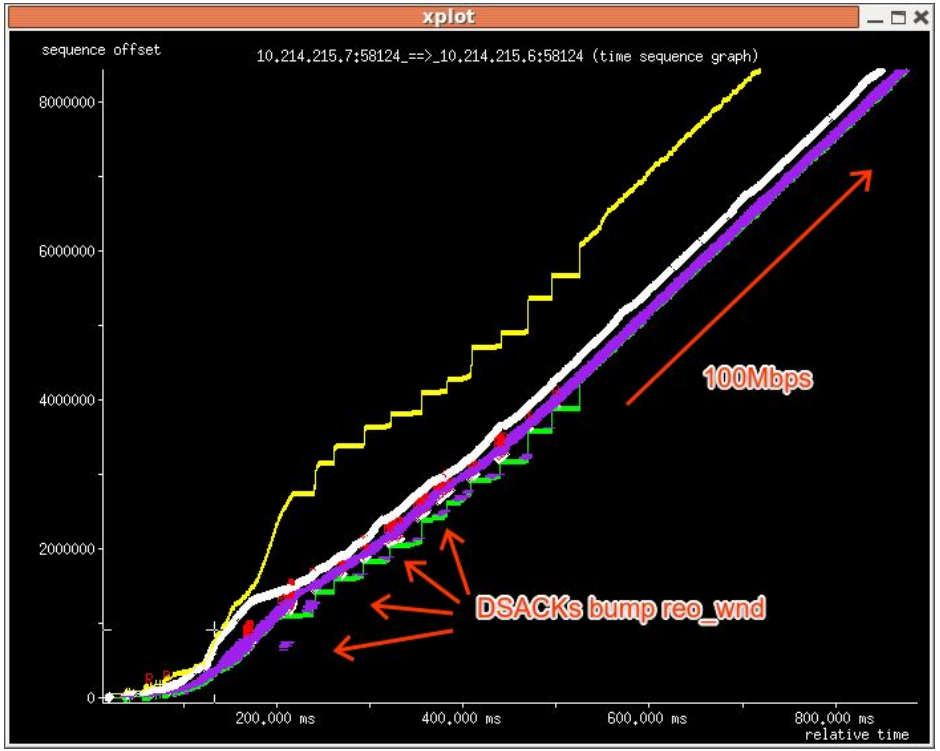
Adapting reordering window with DSACK

Use DSACK as feedback on window under-estimation

- Receivers return a DSACK [RFC2883] upon receiving a spurious retransmission
- Supported by Linux, MacOS/iOS, Windows

Init: reo_wnd = min_RTT/4
For every round trip w/ DSACK(s)
    reo_wnd += min_RTT/4
    reo_wnd = min(reo_wnd, SRTT)
Re-init reo_wnd after 16 DSACK-free recoveries

Q: why not measure reordering degree in time directly?
A: difficult in Linux b/c it merges SACK'd packets



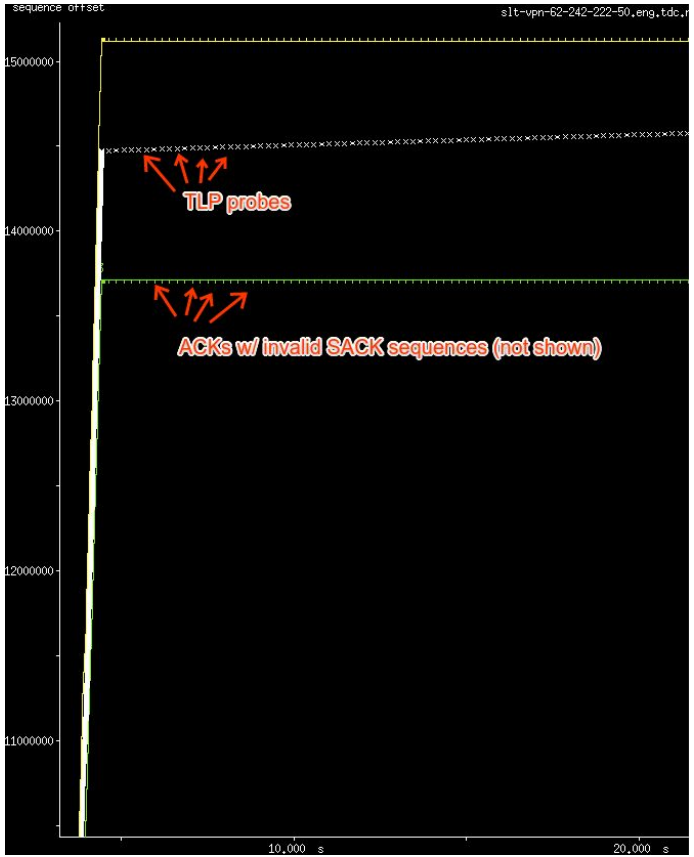
Same test with adaptive reo_wnd

Mitigating broken middle-boxes for TLP

Some middle-boxes rewrote TCP header sequences but not sequences in SACK options

- TLP rearmed TLP timer on any ACK
- The sender loops forever
  - TLP timer fires and send a probe packet
  - SACK of the probe arrives
  - SACK sequences are invalid but causes TLP timer to be rearmed

Fixed in Linux 4.13+: only rearms TLP timer if the S/ACK after the probe packet acknowledges new data or the probe packet



Next steps

Vision: making TCP resilient and efficient to reordering and loss with one algorithm

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

Work-in-progress

1. ~~Optimize for frequent reordering and high BDP path~~
2. Demonstrate RACK/TLP can be standalone by default to retire DUPACK threshold approach. I.e. one heuristic in TCP based on time.
3. Update the (expired) draft ...