RACK: a time-based fast loss recovery
Draft-ietf-tcpm-rack-04 updates

Yuchung Cheng
Neal Cardwell
Nandita Dukkipati
Priyaranjan Jha
Google

tcpm IETF 102, July 2018

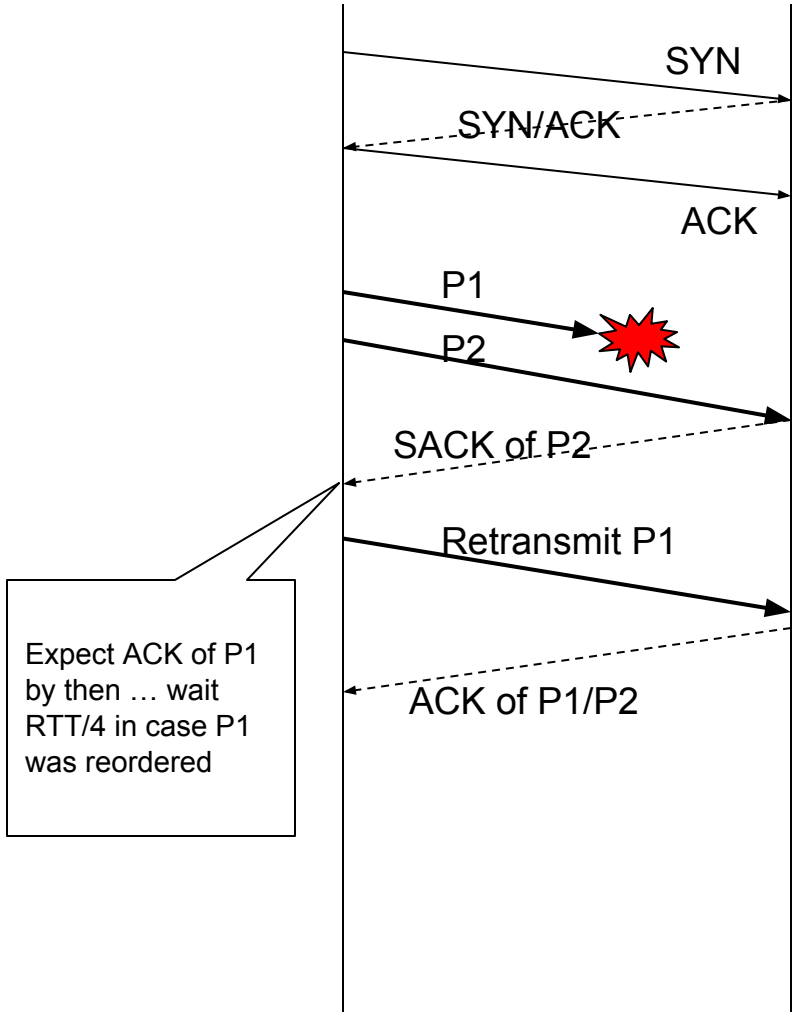
What's RACK (Recent ACK)?

Time-based loss inferences instead of 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



New section: reordering detection

Key heuristic: ACKs that indicate out-of-order data sequence delivery, e.g. D/SACK

```
RACK.fack // highest sequence s/acked (Forward ACK)

RACK_detect_reordering():
       For each Packet newly acknowledged cumulatively or selectively:
           If Packet.end_seq > RACK.fack:
               RACK.fack = Packet.end_seq
           Else if Packet.end_seq < RACK.fack AND
                   Packet.retransmitted is FALSE:
               RACK.reord = TRUE
       For each Packet covered by the DSACK option:
           If Packet.retransmitted is TRUE:
               RACK.reord = TRUE
```

New section: design rationale for reordering tolerance

Last meeting: big concerns about better TCP reordering tolerance allowing reckless network reordering

Excessive reordering hurts end to end performance:

1. Host stack: high CPU cost by breaking GRO and increasing #ACKs
2. Congestion control: assumes feedbacks from same bottleneck
3. Loss recovery: large reordering window causes slower loss recovery

RACK is designed to tolerate small reordering on slightly diverse paths (router parallelism or L2 retransmission)

RACK reordering window mandates

<verbatim>
To accomplish this RACK places the following mandates on the reordering window:

1. The initial RACK reordering window SHOULD be set to a small fraction of the round-trip time.

2. If no reordering has been observed, then RACK SHOULD honor the classic 3-DUPACK rule for initiating fast recovery. One simple way to implement this is to temporarily override the reorder window to 0.

3. The RACK reordering window SHOULD leverage Duplicate Selective Acknowledgement (DSACK) information [RFC3708] to adaptively estimate the duration of reordering events.

4. The RACK reordering window MUST be bounded and this bound SHOULD be one round trip.
</verbatim>

RACK reordering window computation

Respects mandates, to adapt to observed level of reordering (within careful bounds).

```
If RACK.reord is FALSE:
    If in loss recovery:  /* If in fast or timeout recovery */
        RACK.reo_wnd = 0
        Return
    Else if RACK.pkts_sacked >= RACK.dupthresh:
        RACK.reo_wnd = 0
        return
RACK.reo_wnd = RACK.min_RTT / 4 * RACK.reo_wnd_incr
RACK.reo_wnd = min(RACK.reo_wnd, SRTT)
```

(Section 6.2, Step 4)

Progress in Linux implementation

Linux 4.18 fully implements RACK/TLP

1. On by default
2. [RFC6675] (dupthresh-based) recovery is disabled

Linux loss recovery heuristics reduced from 10 to 2 (RACK/TLP, F/RTO)

RACK/TLP algorithm development is now concluded (no major work planned)