April 20, 2016

# Packet loss detection based on recent acknowledgement (RACK)

Yuchung Cheng

Neal Cardwell

# Packet loss detection based on recent acknowledgement (RACK)

## ABSTRACT

This disclosure describes techniques for packet loss detection in networks based on Recent ACKnowledgement (RACK). RACK technique uses the notion of time, instead of conventional approaches for packet loss detection such as packet or sequence counting. Packets are deemed lost if a packet that was sent sufficiently later has been cumulatively or selectively acknowledged. In example implementations, a sender that implements RACK technique records packet transmission times and infers losses using cumulative or selective acknowledgements.

## KEYWORDS

- packet loss detection
- loss recovery
- packet reordering
- TCP

## BACKGROUND

Packet loss detection techniques are employed in packet-switching communication networks, such as networks that utilize the Transmission Control Protocol (TCP). Conventional packet loss detection techniques may not perform well for networks that experience modern traffic patterns or underlying network changes. For example, the prevalence of interactive request-response traffic means that TCP is often application-limited. Further, wide deployment of traffic policers can result in frequent lost retransmissions and losses at the tail of transactions. Additionally, mobile/wireless and router load-balancing can cause relatively frequent occurrences of small degrees of reordering.

Such factors make existing approaches such as packet or sequence counting inefficient. Mechanisms that are based purely on counting packets in sequence order can either detect packet loss quickly or accurately. However, it is difficult to achieve both speed and accuracy when the sender is application-limited, or when packet reordering is unpredictable.

One heuristic approach is to mark a retransmission as lost, if it was sent before a limited transmit (e.g., new data packet) is acknowledged in recovery, since the acknowledgement implies that at least one round trip time has elapsed. However, such approach has several limitations - it cannot detect tail drops (since it depends on limited transmit), it is disabled upon reordering, and it is only enabled in fast recovery, but not timeout recovery.

The techniques described in this disclosure enable quick and accurate packet loss detection. The techniques address the limitations of packet-counting based techniques and of the simple heuristic approach described above.

## DESCRIPTION

This disclosure describes packet loss detection techniques based on recent acknowledgements (“RACK”). RACK technique is implemented by a sender that sends packets over a network to a receiver. RACK can be implemented in a network with no changes on the receiver side.

A sender that implements RACK technique stores three factors.

1. In implementing RACK, the sender stores a selective acknowledgement SACK scoreboard. RACK presumes that the connection uses SACK options. In RACK implementations, the scoreboard is a data structure to store selective acknowledgement information on a per connection basis.
2. The sender stores its most recent transmission time at a fine granularity e.g., millisecond granularity. In certain implementations, e.g., for intra-datacenter communications, RACK technique can benefit from a sender maintaining such information at microsecond granularity.
3. For each packet, the sender stores whether the packet has been retransmitted or not.

### Example Implementation Environment

Fig. 1 shows an example environment in which RACK technique is implemented. Sender (110) is configured in communication with network (130) over which it can communicate (e.g., transmit packets to) with receiver (140). While only one sender and receiver are shown in Fig. 1, RACK technique can be employed in any size of network, with multiple senders or recipients. Sender and receiver may each be any type of device that is capable of communication over a packet-switched network, such as a server computer, a personal computer, a wireless device, a wearable device, a head mounted display, or such. Network (130) can include one or more intermediate devices such as routers, switches, gateways, hubs, etc. Network (130) can be a wired network, a wireless network, or a combination.

In the example implementation of RACK technique shown in Fig. 1, the sender stores values for a number of variables in memory (150).

1. Packet.xmit_time (112) is the time of the last transmission of a data packet from the sender, including any retransmissions. The sender records the transmission time for each packet sent that is not yet acknowledged. Packet.xmit_time is stored at a fine granularity of time e.g., at millisecond granularity or finer.
2. RACK.xmit_time (114) is the transmission time of the most recent packet from among all the packets from the sender that were delivered (e.g., cumulatively or selectively acknowledged) to the receiver on the connection.
3. RACK.RTT (116) is the associated round-trip time (RTT) measured when RACK.xmit_time is changed. It is the round-trip time of the most recently transmitted packet that has been delivered to the receiver on the connection.
4. RACK.reo_wnd (118) is a reordering window for the connection. The reordering window is computed in the same unit of time as that used to record packet transmission times. It is used to defer the moment at which RACK marks a packet as lost.
5. RACK.min_RTT (120) is the estimated minimum round-trip time of the connection.

The sender stores Packet.xmit_time (112) for each packet in flight. The sender stores RACK.xmit_time (114), RACK.RTT (116), RACK.reo_wnd (118) and RACK.min_RTT (120) per connection.

## Example Method

Fig. 2 shows a flowchart of an example method to implement the RACK techniques of this disclosure. In an implementation, Sender (110) can implement the process of Fig. 2 to detect packet loss over a connection on the network.

Upon transmitting or retransmitting a packet, the sender records the transmission time in Packet.xmit_time. In this example, the sender stores the transmission time for each packet in flight. Upon receiving an acknowledgement (220), the sender updates (230) RACK.min_RTT. To estimate RACK.min_RTT, the sender uses round-trip time (RTT) measurements. For example, the sender tracks a simple global minimum of all RTT measurements from the connection (e.g., the connection with receiver 140 over network 130). In another example, the sender tracks a windowed minimum-filtered value of recent RTT measurements. Other approaches to estimate RACK.min_RTT can also be used.

The sender further updates RACK.reo_wnd (240). RACK.reo_wnd permits the sender to handle the prevalent small degree of reordering. RACK.reo_wnd serves as an allowance for settling time before the sender marks a packet as lost. In one example, RACK.reo_wnd may be set as a default value e.g., 1 millisecond. In another example, the sender implements reordering detection techniques to dynamically adjust the reordering window. For example, when the sender detects packet reordering, it may change RACK.reo_wnd to one-fourth of RACK.min_RTT.

The sender utilizes information provided in a received acknowledgement to mark each packet that has been acknowledged (ACKed) or selectively acknowledged (SACKed) as delivered. The sender then determines the most recent Packet.xmit_time from among all packets that have been acknowledged and advances (250) RACK.xmit_time (e.g., updates RACK.xmit_time to be equal to the most recent Packet.xmit_time), if the most recent Packet.xmit_time is greater than a current value of RACK.xmit_time.

In some examples, the sender does not update the RACK.xmit_time e.g., if the retransmission is considered as likely spurious. The sender ignores packets that are retransmitted in the determination of RACK.xmit_time if at least one of the below two conditions is true:

a) Timestamp Echo Reply field (TSecr) of the timestamp option of the ACK, if available, indicates the ACK was not an acknowledgement of the last retransmission of the packet
b) The packet was last retransmitted less than RACK.min_RTT ago.

If the RACK.xmit_time is changed (260) based on a particular ACK, the sender also records the RTT based on the ACK e.g., the sender sets RACK.RTT = (current time)- RACK.xmit_time.

If the RACK.xmit_time is not changed, the sender continues transmission of packets and measurement of the various parameters. If the RACK.xmit_time is changed, the sender detects losses (270).

## Loss Detection

### *Marking packets as lost*

For each packet that has not been acknowledged (e.g., fully SACKed), the sender determines if RACK.xmit_time is after Packet.xmit_time + RACK.reo_wnd. If RACK.xmit_time is after Packet.xmit_time + RACK.reo_wnd, the sender marks the packet (or its corresponding sequence range) as lost. In this example, the sender determines another packet that was sent later has been delivered, and the reordering window or "reordering settling time" has already passed, to conclude that the packet was likely lost.

### *Packets not yet lost*

The sender determines, for a given packet, that another packet that was sent later has been delivered. The sender further determines that the reordering window has not passed. Based on these determinations, the server concludes that the given packet is not lost as of the time of determination.

The sender waits for the next ACK to further advance RACK.xmit_time. However, in some implementations, this can risk a timeout (RTO) e.g., if no more ACKs come back (e.g., due to losses or application limit). In some implementations, the sender installs a "reordering settling" timer for timely loss detection. For example, the sender sets the timer to fire at the earliest moment at which it is safe to conclude that some packet is lost. In this example, the earliest moment is the time it takes to expire the reordering window of the earliest unacknowledged packet in flight, which is the minimum value of (Packet.xmit_time + RACK.RTT + RACK.reo_wnd + 1ms) across all unacknowledged packets.

### *Example pseudocode for lost packet detection*

```
RACK_detect_loss():
     min_timeout = 0

     For each packet, Packet, in the scoreboard:

               If Packet is already SACKed, ACKed, or marked
               lost and not yet retransmitted:
                    Skip to the next packet

               If Packet.xmit_time > RACK.xmit_time:
                    Skip to the next packet

               timeout = Packet.xmit_time + RACK.RTT +
               RACK.reo_wnd + 1

               If now >= timeout
                    Mark Packet lost
               Else If (min_timeout == 0) or (timeout is
               before min_timeout):
                    min_timeout = timeout

     If min_timeout != 0
               Arm the RACK timer to call RACK_detect_loss()
          at the time min_timeout
```

## Advantages

One advantage of RACK technique is that it can utilize every data packet, original or retransmission, to detect losses of packets that were sent prior to it.

### *Example 1: Tail Drop*

Consider a sender that transmits a window of three data packets (P1, P2, P3), and P1 and P3 are lost. Suppose the transmission of each packet is at least RACK.reo_wnd after the transmission of the previous packet. RACK technique marks P1 as lost when the SACK of P2 is received, triggering the retransmission of P1 as R1. When R1 is cumulatively acknowledged, RACK technique marks P3 as lost and the sender retransmits P3 as R3. This example illustrates how RACK technique is able to repair certain drops at the tail of a transaction without any timer. Packet or sequence count based techniques cannot detect such losses.

*Example 2: Lost Retransmit*

Consider a window of three data packets (P1, P2, P3) that are sent; P1 and P2 are dropped. Suppose the transmission of each packet is at least RACK.reo_wnd after the transmission of the previous packet. When P3 is SACKed, RACK technique marks P1 and P2 lost and the sender retransmits these as R1 and R2. Suppose R1 is lost again (as a tail drop) but R2 is selectively acknowledged. RACK technique marks R1 lost for retransmission again. Conventional approaches cannot detect such losses. Such a lost retransmission is very common when TCP is being rate-limited e.g., by token bucket policers with large bucket depth and low rate limit. Retransmissions are often lost repeatedly because standard congestion control requires multiple round trips to reduce the rate below the policed rate.

*Example 3: Reordering*

Consider a common reordering event: a window of packets sent as (P1, P2, P3). P1 and P2 carry a full payload of MSS octets, but P3 has only a 1-octet payload due to application-limited behavior. Suppose the sender has detected reordering previously and RACK.reo_wnd is min_RTT/4. Now P3 is reordered and delivered first, before P1 and P2. As long as P1 and P2 are delivered within min_RTT/4, RACK technique does not consider P1 and P2 lost. But if P1 and P2 are delivered outside the reordering window, then RACK will still falsely mark P1 and P2 lost. RACK technique can improve performance in such situations by measuring the degree of reordering in time, instead of packet distances, e.g., by storing the delivery timestamp of each packet. Alternatively, RACK can use smoothed value of round-trip time.

The examples above show that RACK technique is particularly useful when the sender is limited by the application, which is common for interactive, request/response traffic. Similarly, RACK technique works when the sender is limited by the receive window, which is common for applications that use the receive window to throttle the sender. RACK technique decouples loss detection from congestion control. RACK technique is applicable for both fast recovery and recovery after a retransmission timeout (RTO). RACK is compatible with standard RTO techniques. RACK technique has no impact on the risk profile for TCP.

## Examples of use

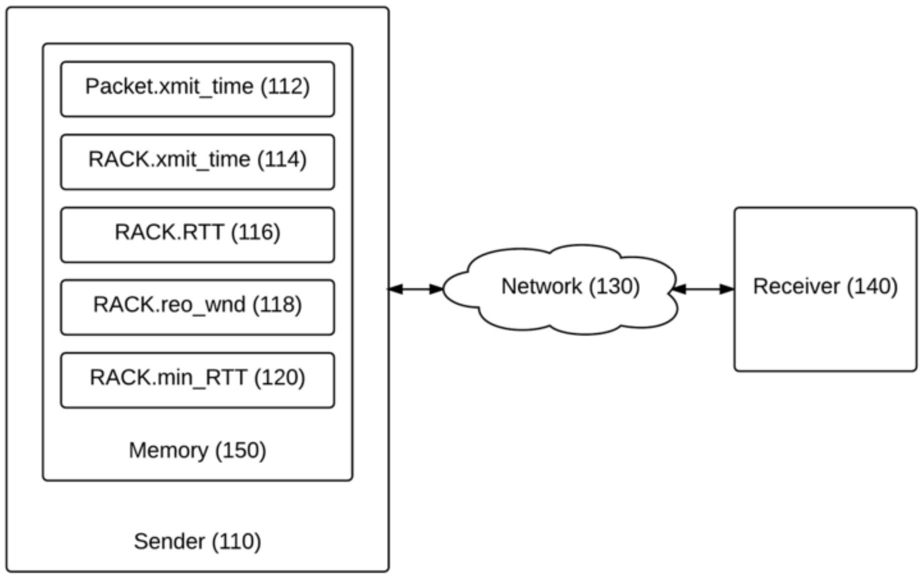
The techniques described in this disclosure can be implemented for packet loss detection in a variety of contexts. For example, the techniques may be implemented in operating systems (e.g., in OS kernels) to detect TCP losses. The techniques can be used to detect packet loss within a data center (e.g., intra-datacenter traffic), between data centers, or edge serving. The techniques can also be implemented in UDP-based protocols, and used for packet loss detection during communications between a data center (e.g., that provides Internet-based applications) and client internet browsers or applications.

## CONCLUSION

The techniques described in this disclosure permit accurate and timely detection of packet loss for networks. The techniques can be particularly useful for modern traffic patterns (e.g., interactive request-response traffic), in networks that have wide deployment of traffic policers, and in the presence of mobile/wireless and router load balancing.

By utilizing the time sequence instead of the data sequence of packets, the techniques of this disclosure can detect tail drops when a later retransmission is acknowledged (or selectively acknowledged). The use of a dynamically adjusted reordering window can reduce false positives even in the presence of small degree of reordering. The techniques perform well in the presence of unpredictable or frequent reordering of packets.

<u>FIGURES</u>



<u>Fig. 1</u>

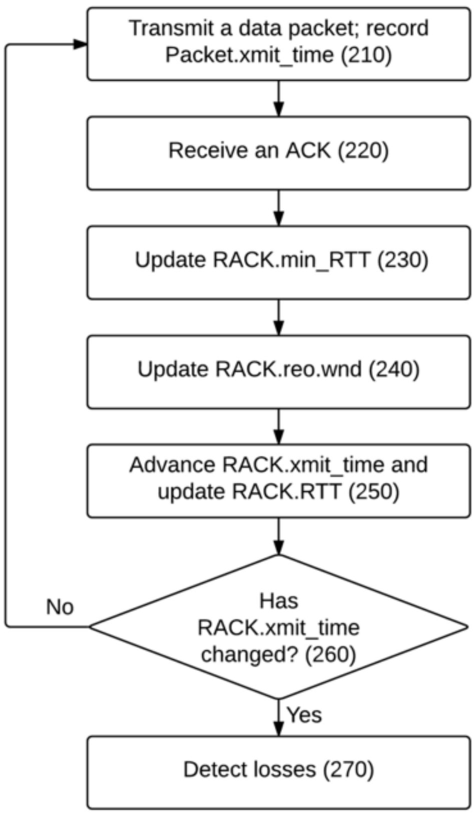

Transmit a data packet; record Packet.xmit_time (210)
Receive an ACK (220)
Update RACK.min_RTT (230)
Update RACK.reo.wnd (240)
Advance RACK.xmit_time and update RACK.RTT (250)
Has RACK.xmit_time changed? (260)
No
Yes
Detect losses (270)

Fig. 2