QUIC
Internet-Draft
Intended status: Standards Track
Expires: June 1, 2017

J. Iyengar, Ed.
I. Swett, Ed.
Google

November 28, 2016

QUIC Loss Detection and Congestion Control
draft-ietf-quic-recovery-00

Abstract

QUIC is a new multiplexed and secure transport atop UDP. QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. QUIC implements the spirit of known TCP loss detection mechanisms, described in RFCs, various Internet-drafts, and also those prevalent in the Linux TCP implementation. This document describes QUIC loss detection and congestion control, and attributes the TCP equivalent in RFCs, Internet-drafts, academic papers, and TCP implementations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

QUIC is a new multiplexed and secure transport atop UDP. QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. The QUIC protocol is described in [QUIC-TRANSPORT].

QUIC implements the spirit of known TCP loss recovery mechanisms, described in RFCs, various Internet-drafts, and also those prevalent in the Linux TCP implementation. This document describes QUIC congestion control and loss recovery, and where applicable, attributes the TCP equivalent in RFCs, Internet-drafts, academic papers, and/or TCP implementations.

This document first describes pre-requisite parts of the QUIC transmission machinery, then discusses QUIC's default congestion control and loss detection mechanisms, and finally lists the various TCP mechanisms that QUIC loss detection implements (in spirit.)

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

2. Design of the QUIC Transmission Machinery

All transmissions in QUIC are sent with a packet-level header, which includes a packet sequence number (referred to below as a packet number). These packet numbers never repeat in the lifetime of a connection, and are monotonically increasing, which makes duplicate detection trivial. This fundamental design decision obviates the need for disambiguating between transmissions and retransmissions and eliminates significant complexity from QUIC's interpretation of TCP loss detection mechanisms.

Every packet may contain several frames. We outline the frames that are important to the loss detection and congestion control machinery below.

o Retransmittable frames are frames requiring reliable delivery. The most common are STREAM frames, which typically contain application data.

o Crypto handshake data is also sent as STREAM data, and uses the reliability machinery of QUIC underneath.

o ACK frames contain acknowledgment information. QUIC uses a SACK-based scheme, where acks express up to 256 ranges. The ACK frame also includes a receive timestamp for each packet newly acked.

2.1. Relevant Differences Between QUIC and TCP

There are some notable differences between QUIC and TCP which are important for reasoning about the differences between the loss recovery mechanisms employed by the two protocols. We briefly describe these differences below.

2.1.1. Monotonically Increasing Packet Numbers

TCP conflates transmission sequence number at the sender with delivery sequence number at the receiver, which results in retransmissions of the same data carrying the same sequence number, and consequently to problems caused by "retransmission ambiguity". QUIC separates the two: QUIC uses a packet sequence number (referred to as the "packet number") for transmissions, and any data that is to be delivered to the receiving application(s) is sent in one or more streams, with stream offsets encoded within STREAM frames inside of packets that determine delivery order.

QUIC's packet number is strictly increasing, and directly encodes transmission order. A higher QUIC packet number signifies that the packet was sent later, and a lower QUIC packet number signifies that the packet was sent earlier. When a packet containing frames is deemed lost, QUIC rebundles necessary frames in a new packet with a new packet number, removing ambiguity about which packet is acknowledged when an ACK is received. Consequently, more accurate RTT measurements can be made, spurious retransmissions are trivially detected, and mechanisms such as Fast Retransmit can be applied universally, based only on packet number.

This design point significantly simplifies loss detection mechanisms for QUIC. Most TCP mechanisms implicitly attempt to infer transmission ordering based on TCP sequence numbers - a non-trivial task, especially when TCP timestamps are not available.

2.1.2. No Reneging

QUIC ACKs contain information that is equivalent to TCP SACK, but QUIC does not allow any acked packet to be reneged, greatly simplifying implementations on both sides and reducing memory pressure on the sender.

2.1.3. More ACK Ranges

QUIC supports up to 256 ACK ranges, opposed to TCP's 3 SACK ranges. In high loss environments, this speeds recovery.

2.1.4. Explicit Correction For Delayed Acks

QUIC ACKs explicitly encode the delay incurred at the receiver between when a packet is received and when the corresponding ACK is sent. This allows the receiver of the ACK to adjust for receiver delays, specifically the delayed ack timer, when estimating the path RTT. This mechanism also allows a receiver to measure and report the delay from when a packet was received by the OS kernel, which is useful in receivers which may incur delays such as context-switch latency before a userspace QUIC receiver processes a received packet.

3. Loss Detection

We now describe QUIC's loss detection as functions that should be called on packet transmission, when a packet is acked, and timer expiration events.

3.1. Variables of interest

We first describe the variables required to implement the loss detection mechanisms described in this section.

- loss_detection_alarm: Multi-modal alarm used for loss detection.

- alarm_mode: QUIC maintains a single loss detection alarm, which switches between various modes. This mode is used to determine the duration of the alarm.

- handshake_count: The number of times the handshake packets have been retransmitted without receiving an ack.

- tlp_count: The number of times a tail loss probe has been sent without receiving an ack.

- rto_count: The number of times an rto has been sent without receiving an ack.

- smoothed_rtt: The smoothed RTT of the connection, computed as described in [RFC6298]. TODO: Describe RTT computations.

- reordering_threshold: The largest delta between the largest acked retransmittable packet and a packet containing retransmittable frames before it's declared lost.

- time_loss: When true, loss detection operates solely based on reordering threshold in time, rather than in packet number gaps.

3.2. Initialization

At the beginning of the connection, initialize the loss detection variables as follows:

```
loss_detection_alarm.reset();
handshake_count = 0;
tlp_count = 0;
rto_count = 0;
smoothed_rtt = 0;
reordering_threshold = 3;
time_loss = false;
```

3.3. Setting the Loss Detection Alarm

QUIC loss detection uses a single alarm for all timer-based loss detection. The duration of the alarm is based on the alarm's mode, which is set in the packet and timer events further below. The function SetLossDetectionAlarm defined below shows how the single timer is set based on the alarm mode.

Pseudocode for SetLossDetectionAlarm follows:

```
 SetLossDetectionAlarm():
    if (retransmittable packets are not outstanding):
      loss_detection_alarm.cancel();
      return;
    if (handshake packets are outstanding):
      alarm_duration = max(1.5 * smoothed_rtt, 10ms) << handshake_count;
      handshake_count++;
    else if (largest sent packet is acked):
      // Set alarm based on short timer for early retransmit.
      alarm_duration = 0.25 x smoothed_rtt;
    else if (tlp_count < 2):
      if (retransmittable_packets_outstanding = 1):
        alarm_duration = max(1.5 x smoothed_rtt + delayed_ack_timer,
                             2 x smoothed_rtt);
      else:
        alarm_duration = max (10ms, 2 x smoothed_rtt);
      tlp_count++;
    else:
      if (rto_count = 0):
        alarm_duration = max(200ms, smoothed_rtt + 4 x rttvar);
      else:
        alarm_duration = loss_detection_alarm.get_delay() << 1;
      rto_count++;

    loss_detection_alarm.set(now + alarm_duration);
```

3.4. On Sending a Packet

After any packet is sent, be it a new transmission or a rebundled transmission, the following OnPacketSent function is called. The parameters to OnPacketSent are as follows:

- packet_number: The packet number of the sent packet.

- is_retransmittble: A boolean that indicates whether the packet contains at least one frame requiring reliable deliver. The retransmittability of various QUIC frames is described in [QUIC-TRANSPORT]. If false, it is still acceptable for an ack to be received for this packet. However, a caller MUST NOT set is_retransmittable to true if an ack is not expected.

Pseudocode for OnPacketSent follows:

```
 OnPacketSent(packet_number, is_retransmittable):
     if is_retransmittable:
       SetLossDetectionAlarm()
```

3.5. On Packet Acknowledgment

When a packet is acked for the first time, the following OnPacketAcked function is called. Note that a single ACK frame may newly acknowledge several packets. OnPacketAcked must be called once for each of these newly acked packets.

OnPacketAcked takes one parameter, acked_packet, which is the packet number of the newly acked packet, and returns a list of packet numbers that are detected as lost.

Pseudocode for OnPacketAcked follows:

```
   OnPacketAcked(acked_packet):
     handshake_count = 0;
     tlp_count = 0;
     rto_count = 0;
     UpdateRtt(); // TODO: document RTT estimator.
     DetectLostPackets(acked_packet);
     SetLossDetectionAlarm();
```

3.6. On Alarm Firing

QUIC uses one loss recovery alarm, which when set, can be in one of several modes. When the alarm fires, the mode determines the action to be performed. OnAlarm returns a list of packet numbers that are detected as lost.

Pseudocode for OnAlarm follows:

```
OnAlarm(acked_packet):
  lost_packets = DetectLostPackets(acked_packet);
  MaybeRetransmitLostPackets();
  SetLossDetectionAlarm();
```

3.7. Detecting Lost Packets

Packets in QUIC are only considered lost once a larger packet number is acknowledged. DetectLostPackets is called every time there is a new largest packet or if the loss detection alarm fires the previous largest acked packet is supplied.

DetectLostPackets takes one parameter, acked_packet, which is the packet number of the largest acked packet, and returns a list of packet numbers detected as lost.

Pseudocode for DetectLostPackets follows:

```
DetectLostPackets(acked_packet):
  lost_packets = {};
  foreach (unacked_packet less than acked_packet):
      if (unacked_packet.time_sent <
          acked_packet.time_sent - 1/8 * smoothed_rtt):
        lost_packets.insert(unacked_packet.packet_number);
    else if (unacked_packet.packet_number <
             acked_packet.packet_number - reordering_threshold)
      lost_packets.insert(unacked_packet.packet_number);
  return lost_packets;
```

4. Congestion Control

(describe NewReno-style congestion control for QUIC.)

5. TCP mechanisms in QUIC

QUIC implements the spirit of a variety of RFCs, Internet drafts, and other well-known TCP loss recovery mechanisms, though the implementation details differ from the TCP implementations.

5.1. RFC 6298 (RTO computation)

QUIC calculates SRTT and RTTVAR according to the standard formulas. An RTT sample is only taken if the delayed ack correction is smaller than the measured RTT (otherwise a negative RTT would result), and the ack's contains a new, larger largest observed packet number.

min_rtt is only based on the observed RTT, but SRTT uses the delayed ack correction delta.

As described above, QUIC implements RTO with the standard timeout and CWND reduction. However, QUIC retransmits the earliest outstanding packets rather than the latest, because QUIC doesn't have retransmission ambiguity. QUIC uses the commonly accepted min RTO of 200ms instead of the 1s the RFC specifies.

5.2. FACK Loss Recovery (paper)

QUIC implements the algorithm for early loss recovery described in the FACK paper (and implemented in the Linux kernel.) QUIC uses the packet number to measure the FACK reordering threshold. Currently QUIC does not implement an adaptive threshold as many TCP implementations (i.e., the Linux kernel) do.

5.3. RFC 3782, RFC 6582 (NewReno Fast Recovery)

QUIC only reduces its CWND once per congestion window, in keeping with the NewReno RFC. It tracks the largest outstanding packet at the time the loss is declared and any losses which occur before that packet number are considered part of the same loss event. It's worth noting that some TCP implementations may do this on a sequence number basis, and hence consider multiple losses of the same packet a single loss event.

5.4. TLP (draft)

QUIC always sends two tail loss probes before RTO is triggered. QUIC invokes tail loss probe even when a loss is outstanding, which is different than some TCP implementations.

5.5. RFC 5827 (Early Retransmit) with Delay Timer

QUIC implements early retransmit with a timer in order to minimize spurious retransmits. The timer is set to 1/4 SRTT after the final outstanding packet is acked.

5.6. RFC 5827 (F-RTO)

QUIC implements F-RTO by not reducing the CWND and SSThresh until a subsequent ack is received and it's sure the RTO was not spurious. Conceptually this is similar, but it makes for a much cleaner implementation with fewer edge cases.

5.7. RFC 6937 (Proportional Rate Reduction)

PRR-SSRB is implemented by QUIC in the epoch when recovering from a loss.

5.8. TCP Cubic (draft) with optional RFC 5681 (Reno)

TCP Cubic is the default congestion control algorithm in QUIC. Reno is also an easily available option which may be requested via connection options and is fully implemented.

5.9. Hybrid Slow Start (paper)

QUIC implements hybrid slow start, but disables ack train detection, because it has shown to falsely trigger when coupled with packet pacing, which is also on by default in QUIC. Currently the minimum delay increase is 4ms, the maximum is 16ms, and within that range QUIC exits slow start if the min_rtt within a round increases by more than one eighth of the connection mi

5.10. RACK (draft)

QUIC's loss detection is by it's time-ordered nature, very similar to RACK. Though QUIC defaults to loss detection based on reordering threshold in packets, it could just as easily be based on fractions of an rtt, as RACK does.

6. IANA Considerations

This document has no IANA actions. Yet.

7. Normative References

[QUIC-TLS]
Thomson, M., Ed. and S. Turner, Ed, Ed., "Using Transport Layer Security (TLS) to Secure QUIC", November 2016.

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", November 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <http://www.rfc-editor.org/info/rfc6298>.

Appendix A. Acknowledgments

Authors' Addresses

Jana Iyengar (editor)
Google

Email: jri@google.com

Ian Swett (editor)
Google

Email: ianswett@google.com