Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 20, 2016

J. Iyengar
I. Swett
Google

December 18, 2015

QUIC Loss Recovery And Congestion Control
draft-tsvwg-quic-loss-recovery-01

Abstract

QUIC (Quick UDP Internet Connection) is a new multiplexed and secure transport atop UDP, designed from the ground up and optimized for HTTP/2 semantics. While built with HTTP/2 as the primary application protocol, 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 recovery mechanisms, described in RFCs, various Internet-drafts, and also those prevalent in the Linux TCP implementation. This document describes QUIC loss recovery, and where applicable, attributes the TCP equivalent in RFCs, Internet-drafts, academic papers, and/or TCP implementations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 20, 2016.

Copyright Notice

1. Introduction

QUIC (Quick UDP Internet Connection) is a new multiplexed and secure transport atop UDP, designed from the ground up and optimized for HTTP/2 semantics. While built with HTTP/2 as the primary application protocol, 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 [draft-tsvwg-quic-protocol].

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 loss recovery, and where applicable, attributes the TCP equivalent in RFCs, Internet-drafts, academic papers, and/or TCP implementations.

This document first describes parts of the QUIC transmission machinery that are necessary to describe the loss recovery mechanisms. The document then describes QUIC's loss recovery, followed by a list of the various TCP mechanisms that QUIC re-interprets.

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 can contain several frames; we outline the frames that are important to the loss detection and congestion control machinery below.

- STREAM frames contain application data. Crypto handshake data is also sent as STREAM data, and uses the reliability machinery of QUIC underneath.

- ACK frames contain acknowledgment information. QUIC uses a NACK-based scheme, where the largest_observed packet number is reported, and packets with sequence numbers lesser than the largest_observed not yet seen are reported as NACK 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 Sequence 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 transmission number (referred to as the "sequence 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 sequence number is strictly increasing, and directly encodes transmission order. A higher QUIC sequence number signifies that the packet was sent later, and a lower QUIC sequence number signifies that the packet was sent earlier.

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

QUIC resends lost packets with new packet sequence numbers when retransmission is necessary, 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 sequence number.

2.1.2. No SACK 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 NACK Ranges

QUIC supports up to 255 NACK 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. An Overview of QUIC Loss Recovery

We briefly describe QUIC's actions on packet transmission, ack reception, and timer expiration events.

3.1. On Sending a Packet

A retransmission timer may be set based on the mode:

- If the handshake has not completed, start a handshake timer.

    * 1.5x the SRTT, with exponential backoff.

- If there are outstanding packets which have been NACKed, possibly set the loss timer
    * Depends on the loss detection implementation, default is 0.25RTT in the case of Early Retransmit.
- If fewer than 2 TLPs have been sent, compute and restart TLP timer.
    * Timer is set for max(10ms, 2*SRTT) if there are multiple packets in flight
    * Timer is set to max(1.5*SRTT + delayed ack timer, 2*SRTT) if there is only one packet in flight.
- If 2 TLPs have been sent, set the RTO timer.

    * Timer is set to max(200ms, SRTT+4*RTTVAR) with exponential backoff after the first RTO.

3.2. On Receiving an ACK

The following steps are performed when an ACK is received:

- Validate the ack, including ignoring any out of order acks.
- Update RTT measurements.
- Sender marks unacked packets lower than the largest_observed and not NACKed in this ACK frame as ACKED.
- Packets with packet number lesser than the largest_observed that are NACKed have missing_reports incremented based on FACK.(largest_observed - missing packet number)
- Threshold is set to 3 by default.
- Packets with missing_reports > threshold are marked for retransmission. This logic implements Fast Retransmission and

FACK-based retransmission together.

- If nacked packets are outstanding and the largest observed is the largest sent packet, the retransmission timer will be set to 0.25SRTT, implementing Early Retransmit with timer.

- Stop timers if no packets are outstanding.

3.3. On Timer Expiration

QUIC uses one loss recovery timer, which when set, can be in one of several states. When the timer expires, the state determines the action to be performed. (TODO: describe when the timers are set)

- Handshake state:

  * Retransmit any outstanding handshake packets.

- Loss timer state:

  * Lose the outstanding packets which have been NACKed so far.

  * Report the loss to the congestion controller.

  * Retransmit as many as the congestion controller allows.

- TLP state:

  * Retransmit the smallest outstanding packet which is retransmittable.

  * Do not mark any packets as lost until an ACK arrives.

  * Restart timer for a TLP or RTO.

- RTO state:

  * Retransmit the two smallest outstanding packets which are retransmittable.

  * Do not collapse the congestion window (ie: set to 1 packet)

until an ack arrives and confirms that the RTO was not spurious. Note that this step obviates the need to implement FRTO.

* Restart the timer for next RTO (with exponential backoff.)

4. 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.

4.1. 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.

4.2. 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 sequence number to measure the FACK reordering threshold.

Currently QUIC does not implement an adaptive threshold as many TCP implementations(ie: the Linux kernel) do.

4.3. 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.

4.4. 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.

4.5. 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.

4.6. 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.

4.7. RFC 6937 (Proportional Rate Reduction)

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

4.8. 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.

4.9. 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 ⅛ of the connection min_rtt.

5. References

5.1. Normative References

[RFC2119] Bradner, S., "Key Words for use in RFCs to Indicate Requirement Levels", March 1997.

5.2. Informative References

[draft-tsvwg-quic-protocol] Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport For HTTP/2", July 2015.

Authors' Addresses

Janardhan Iyengar
Google

Email: jri@google.com

Ian Swett
Google

Email: ianswett@google.com