Stream0 Summary

QUIC, Stockholm, July 2018

Overview

- Issues and Goals
- Changes
    - QUIC Record Layer for TLS
    - Separate Packet Number Spaces
    - QUIC Stateless Rejects

TLDR: This solves issues raised in London and others

“Stream 0” Issues (Data)

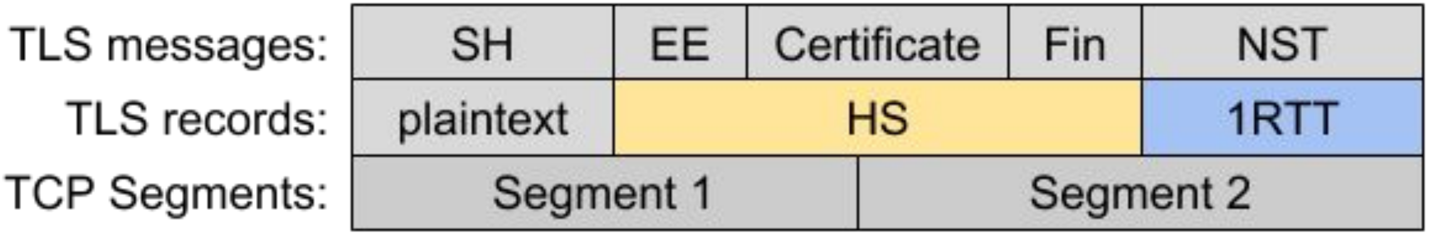
- Partly encrypted, partly not
    - Retransmission must maintain original encryption level
- Tight coupling with the TLS stack
    - Boundaries between flights
    - Is this an SH or an HRR (or a stateful versus stateless HRR go)
- Exempt from flow control during the handshake only
- Mismatch between QUIC and TLS 1.3 notions of 0-RTT boundaries

“Stream 0” Issues (ACKs)

- Holes from unencrypted packets being ACKed in encrypted packets #1018
- Complicated ACK rules
- Contradictions between ACKs and handshake state
  - SFIN means CFIN received but might not contain ACKs
- Reliability for the CFIN #1242
- Optimistic ACK attacks on handshake required address verification

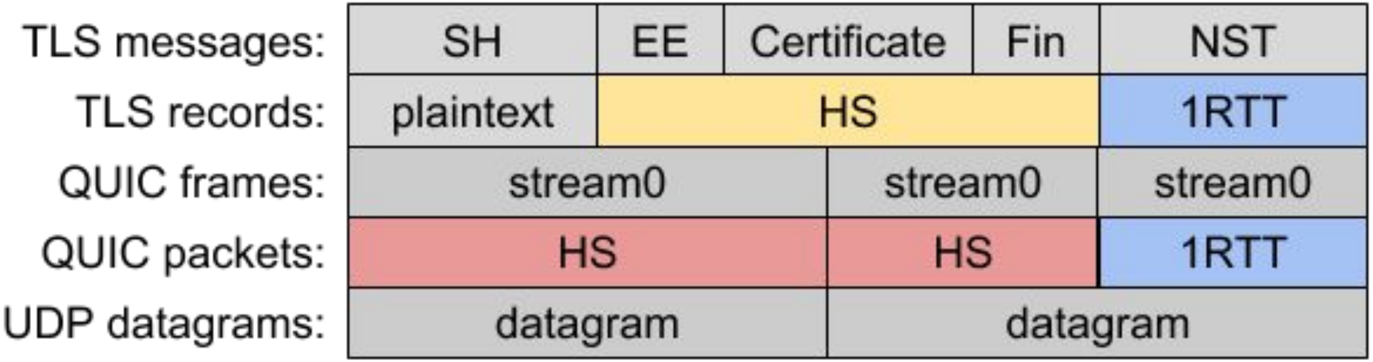
QUIC

Background: TLS 1.3 over TCP



- TLS handshake messages are carried in TLS records
- TLS records
  - Basic unit of encryption
  - Typed (handshake, application data, etc.)
- Records are carried over TCP

QUIC draft-12



- TLS records carried in QUIC stream 0
- Stream frames then carried in QUIC packets
  - These packets are always encrypted
  - TLS encryption boundaries match QUIC encryption boundaries (theoretically)

QUIC draft-13

<table>
<tr><td>TLS messages:</td><td>SH</td><td>EE</td><td colspan="2">Certificate</td><td>Fin</td><td>NST</td></tr>
<tr><td>QUIC frames:</td><td>CRYPTO_HS</td><td colspan="2">CRYPTO_HS</td><td colspan="2">CRYPTO_HS</td><td>CRYPTO_HS</td></tr>
<tr><td>QUIC packets:</td><td>Initial</td><td colspan="2">HS</td><td colspan="2">HS</td><td>1RTT</td></tr>
<tr><td>UDP datagrams:</td><td colspan="3">datagram</td><td colspan="3">datagram</td></tr>
</table>

- TLS handshake messages carried directly over QUIC packets
  - In special CRYPTO{_HS} frames
  - TLS records replaced with QUIC packets
- QUIC packets encrypted using keys from TLS key schedule*

* Potentially with key separation

CRYPTO_HS frame

CRYPTO_HS is similar to a STREAM frame

- Not FIN-able
- No StreamID
- Each encryption level re-starts at offset 0
- Not flow controlled

Benefits of new approach

- Clear rules about where every handshake message is sent
  - These match the TLS rules
  - Trivial to enforce
- QUIC doesn’t need to know TLS handshake state
- No double encryption
- Built-in path validation
  - ACKs encrypted with handshake keys prove on-path

Costs

- New API to expose TLS key schedule to QUIC
  - Prototype implementations in: PicoTLS, Mint, BoringSSL
  - Successful interop between Quicly (PicoTLS) and Minq (Mint)

Separate Packet Number Spaces: Issues

- Fixing packet ‘shadowing’ attack requires knowing encryption level of packets being acknowledged #1018
  - An attacker may inject an unprotected packet that causes the sender to incorrectly believe its packet has been delivered.

- Acknowledgement of packets at one level should not detect loss of packets at a higher encryption level #1413
  - Loss recovery will spuriously retransmit undecryptable packets

Separate Packet Number Spaces

ACK frames apply only to the packet number space they're in

- A packet number could be present in multiple spaces
- 0-RTT and 1-RTT packets are in a single space
    - The transition to 1-RTT is more analogous to a key phase change
    - Acknowledgement of 1-RTT packets can declare 0-RTT packets lost

Separate Packet Number Spaces: Benefits

- Solves the packet shadowing attack
- Corrects loss detection to deal with encryption level
- Clarifies what level an ACK can be sent at
- Easy to handle encryption level in incoming acks
- Dense ACK frames
- Removes temptation to implement implementation-dependent recovery optimizations
- Simplifies implementation (each space is just separate)

Separate Packet Number Spaces: Costs

- May require a sent_packets datastructure per encryption level
- Must store an ACK datastructure per encryption level during the handshake
- More coalesced packets

QUIC Transport Retry: Motivation

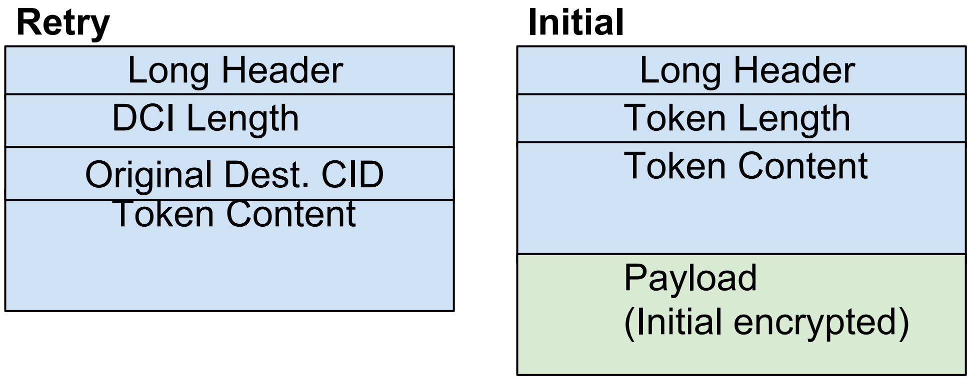
Current Retry complicates TLS interaction #1094, #1233

Generating a Retry requires cleverness in TLS to preserve the handshake transcript

Ideally DDoS mitigation is as cheap as possible

=> Move Retry into the transport

QUIC Transport Retry



Client uses token to prove source address for 0RTT or Retry

Server supplies a short-lived token in a Retry packet

Server supplies a longer-lived token in NEW_TOKEN frame

QUIC

QUIC Transport Retry: Benefits

- Minimize CPU by not protecting Retry
  - Similar to SYN cookies
- No need to consult a TLS stack to generate Retry
- No need to know TLS handshake state
  - Things automatically end up in the right packet type
  - HRR is only used for KeyShare correction

QUIC

QUIC Transport Retry: Issues

- Lots of errors in the initial description
- Client's Initial DCID is unauthenticated #1486
- Looping with Retry Packets #1451

Martin will talk about these later...

QUIC