QUIC
Loss Detection & Congestion Control

draft-ietf-quic-recovery

TCPM, IETF 103

Goals

Discuss QUIC loss recovery mechanisms

No slides on congestion control, but can discuss

(it's just NewReno)

Learn about egregious errors and blind spots

TCPM has the right experts

Increase engagement with TCPM

can do an update again at the next IETF

QUIC

Non-Goals (for the next hour)

Re-design mechanisms

Re-litigate constants

Re-litigate QUIC’s use of TCP standards

- 6298 and 5681 are non-normative references

… these things can be done, just don’t do them right now

QUIC

Overview

Some relevant QUIC details

Recovery mechanisms

Potential improvements / Open questions

QUIC

QUIC Packet Numbers

Monotonically increasing 62-bit *packet numbers*

(caveat: multiple PN spaces during connection setup)

Packet number DOES NOT indicate delivery ordering

QUIC

QUIC Acknowledgements

ACK frame is encrypted and carried within QUIC packets

ACK frame contains:

- largest acked
- one or more ack ranges
- “ack delay”: T(ack send) - T(largest acked packet received)
- 3 ECN counts: #ECT(0), #ECT(1), #CE

QUIC

Generating ACKs

SHOULD ACK every other packet

subject to 25ms delayed ack timer

SHOULD ACK immediately if:

Received packet number != largest received + 1

CE codepoint received

MAY process more packets before ACK

allows less frequent acking

QUIC

Notation

Packet: PN X

packet with Packet Number X

Ack Frame: A X(K-L)(M-N)

largest acked of X

ack ranges K-L and M-N

(Note: X > K, L, M, N)

Same, but different

Loss Detection

- fast retransmit, early retransmit
- tail loss probe, RTO
- spurious RTO detection

Congestion Control

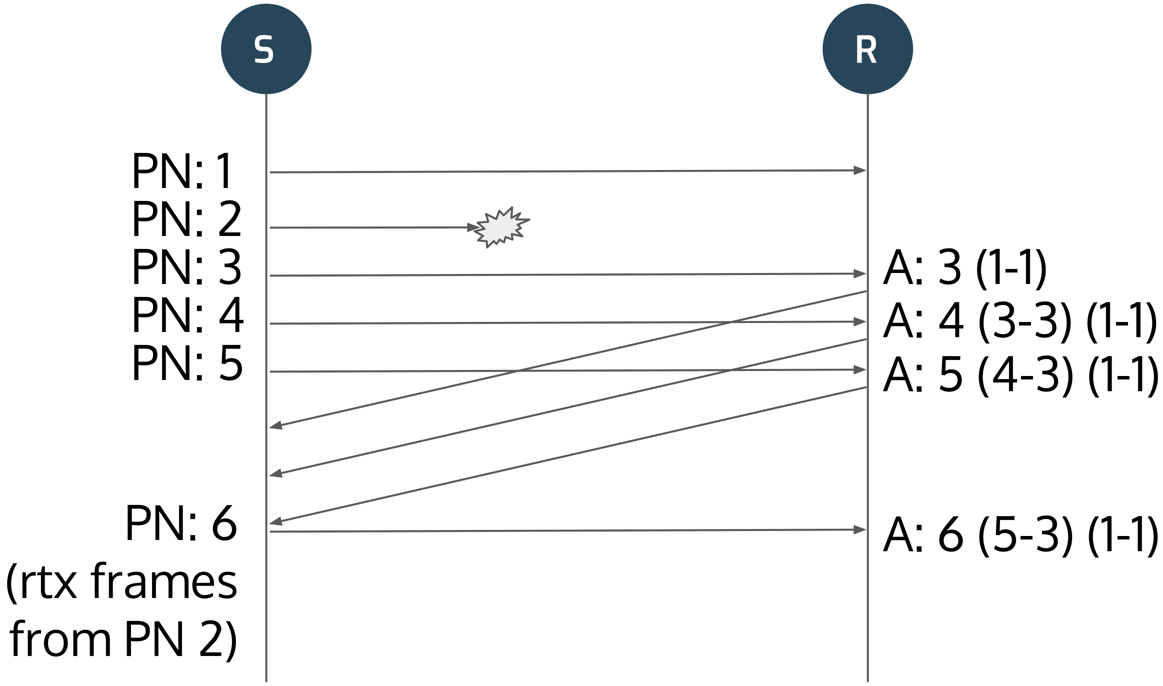
- NewReno, but largest_acked ends recovery period

QUIC

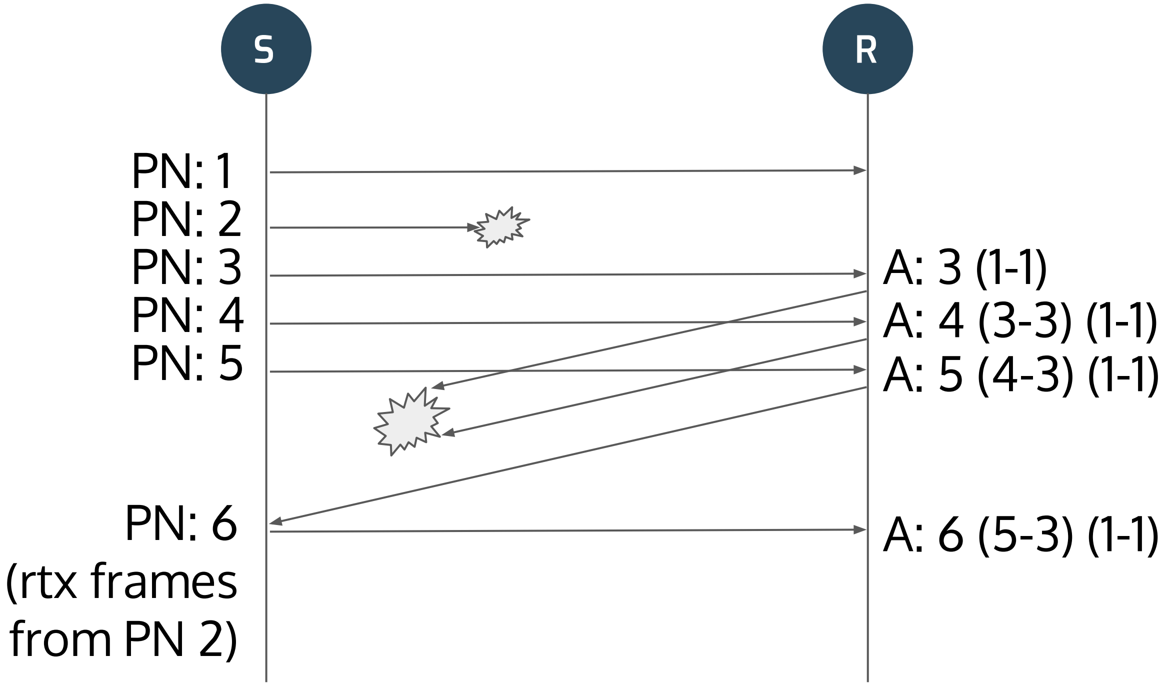
Recovery Mechanisms

QUIC

Fast Retransmit (Packet threshold)

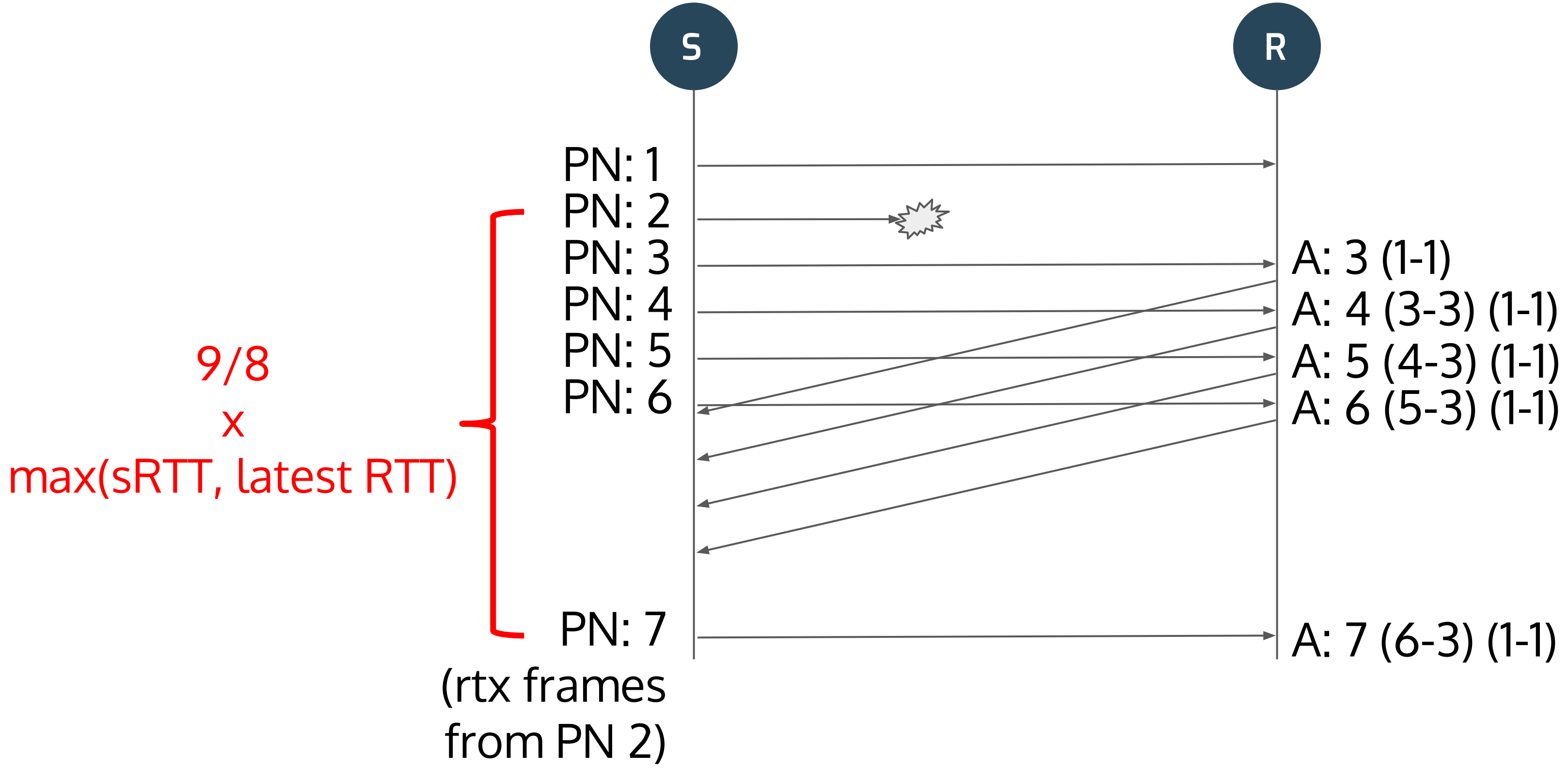


Fast Retransmit (FACK)



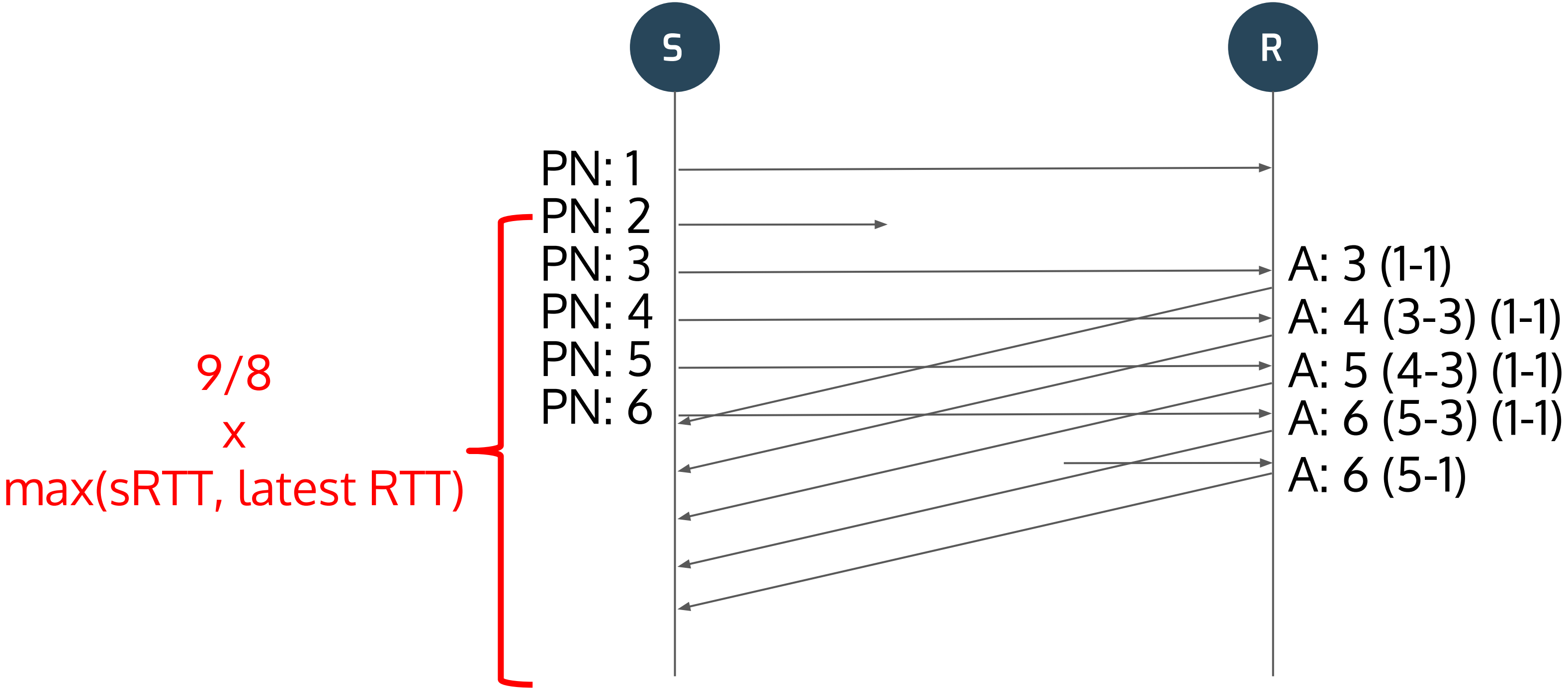
QUIC

Fast Retransmit (Time threshold)



QUIC

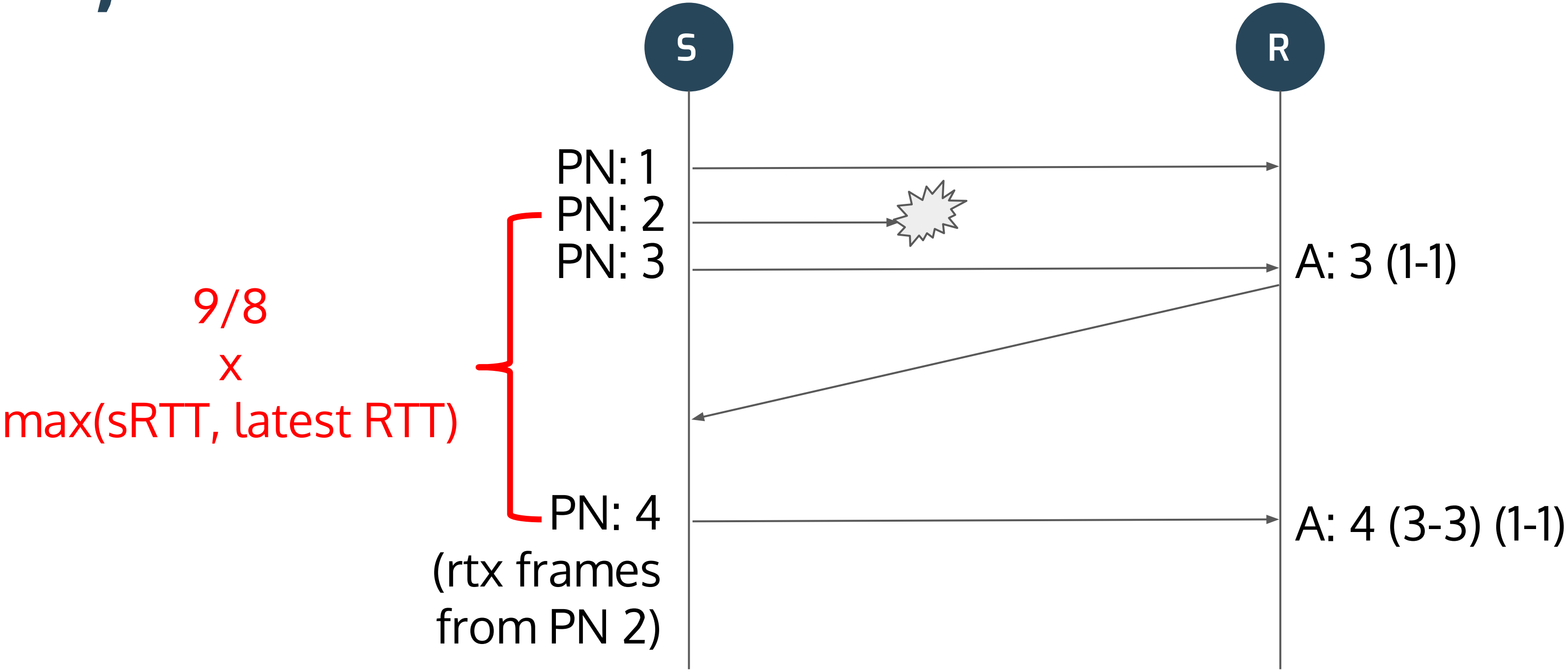
Fast Retransmit (Time threshold)



Time threshold allows reordering tolerance in packet space

QUIC

Early Retransmit



Small delay allows for some reordering

RTT and Timeouts

RTT is RFC 6298, except for RTT sample:

rtt = now - largest_acked.sent_time - ack.ack_delay

max_ack_delay

declared by both endpoints during handshake

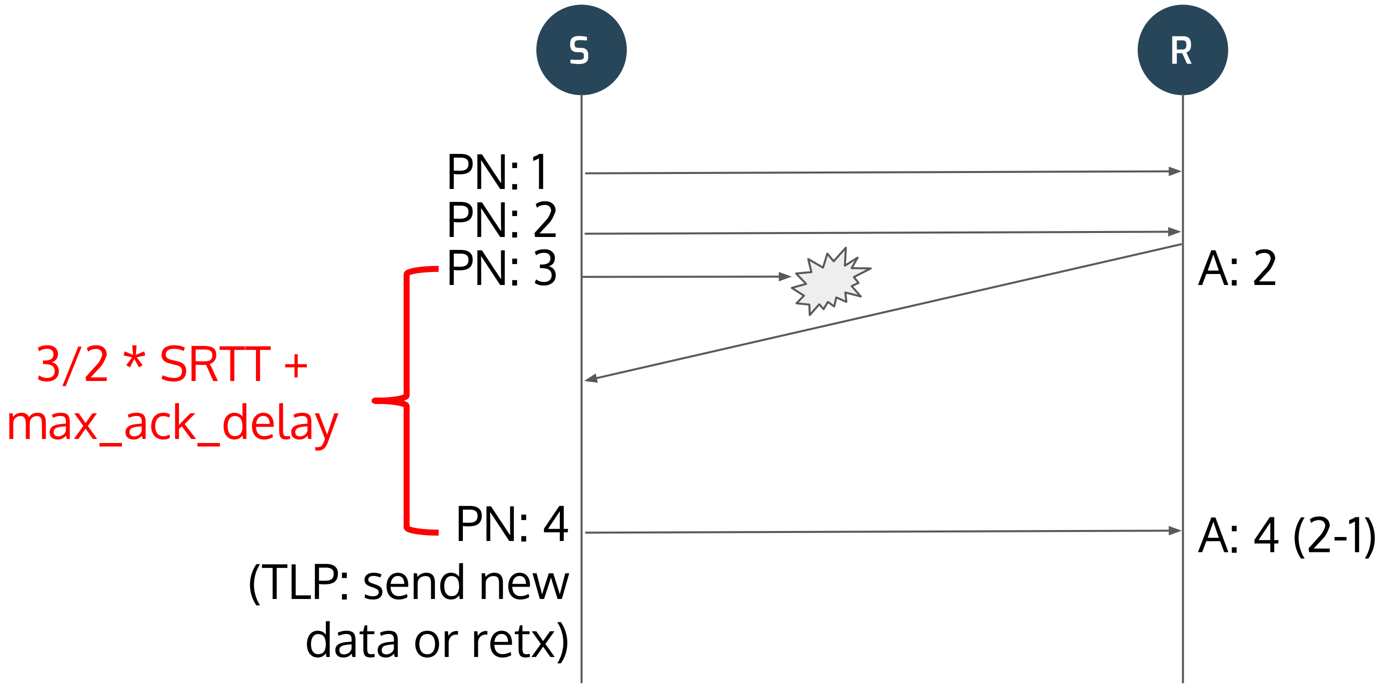
Timeouts:

RTO = srtt + 4 * rttvar + max_ack_delay (min: 200ms)

TLP = 1.5 * srtt + max_ack_delay (min: 10ms)

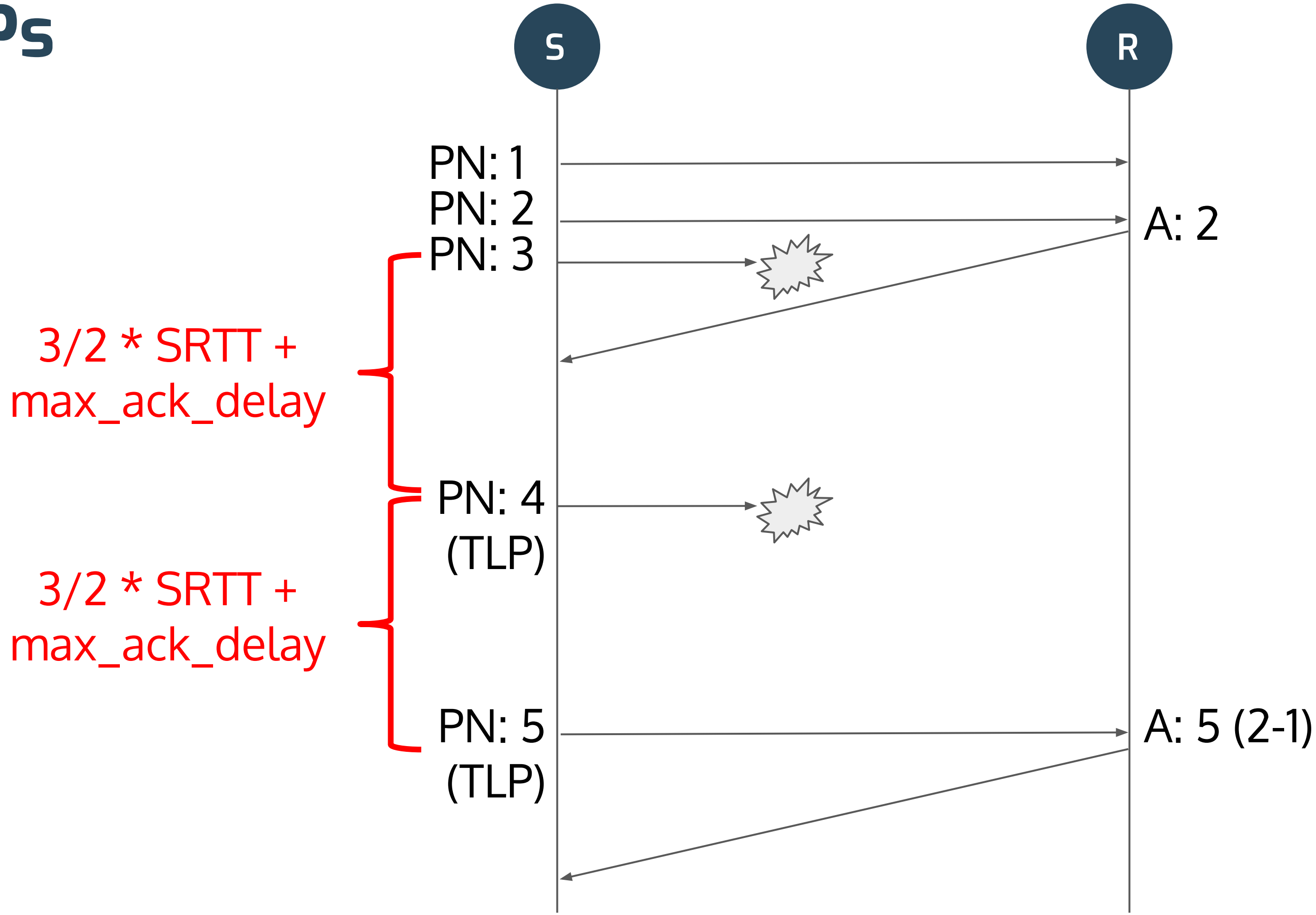
QUIC

TLP



TLP always includes max_ack_delay

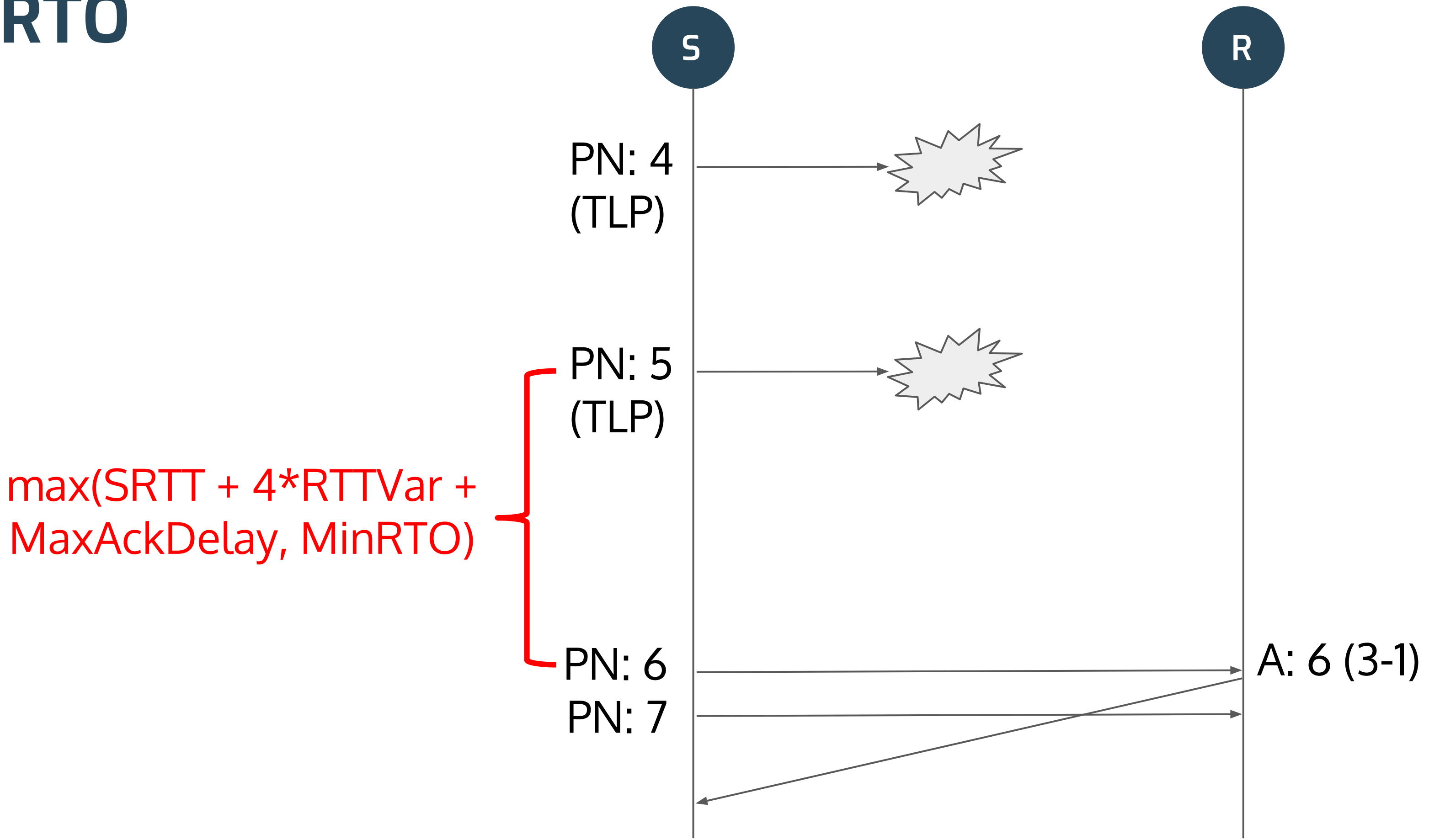
2 TLPs



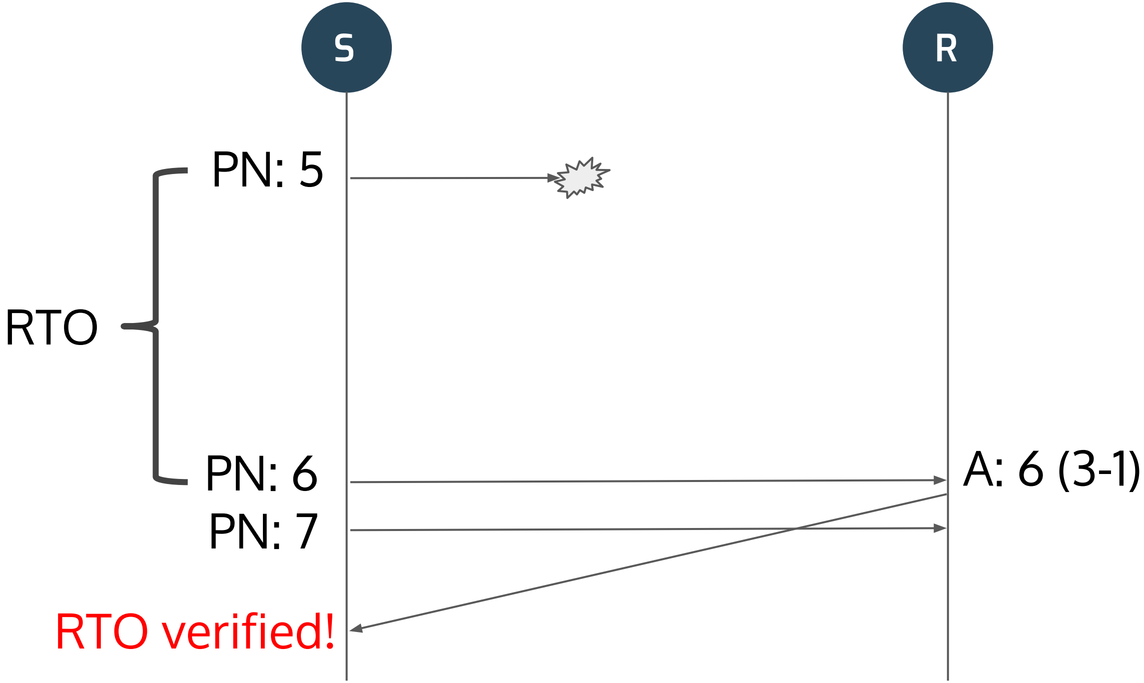
S
R
PN: 1
PN: 2
PN: 3
A: 2
3/2 * SRTT +
max_ack_delay
PN: 4
(TLP)
3/2 * SRTT +
max_ack_delay
PN: 5
(TLP)
A: 5 (2-1)

QUIC

RTO

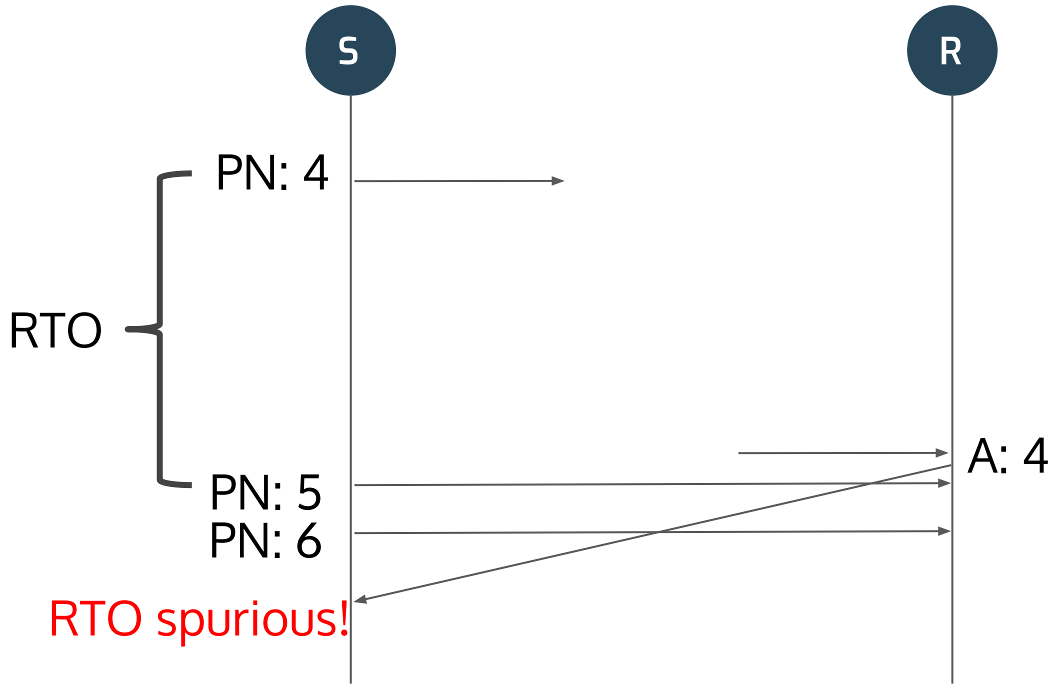

S
R
PN: 4
(TLP)
PN: 5
(TLP)
max(SRTT + 4*RTTVar +
MaxAckDelay, MinRTO)
PN: 6
PN: 7
A: 6 (3-1)

Spurious RTO Detection



QUIC

Spurious RTO Detection



Spurious RTO

No congestion control actions on RTO

If any packet sent prior to RTO is newly acked

declare RTO as spurious

nothing more to be done

If all packets acked are ones sent after RTO

declare RTO as verified

congestion control actions

(open issue: #1966)

Crypto Timeout

Set aggressively

- before RTT sample: 200 ms
- after RTT sample: 2 x smoothed RTT
- set to max(timeout, kMinTLPTimeout)

Exponential backoff on consequent timeouts

Retransmit all outstanding crypto packets on timeout

Potential Improvements (NOT IN DRAFT!)

Generating fewer ACKs

SHOULD be sent immediately upon receipt of a second packet

- wireless drivers, middleboxes compress TCP acks
- should QUIC generate acks less frequently *by default?*

Removing MinRTO (#1017)

MaxAckDelay is explicitly communicated in the handshake

TCP's minRTO was to avoid spurious RTOs (RFC 6298)

primary cost is bandwidth collapse when timer fires

spurious RTO detection eliminates this cost

QUIC could remove the MinRTO

since spurious RTOs have substantially lower cost

Potential Timeout Simplification

Combine TLP and RTO

both are similar, but no practical difference in QUIC

Issue

TLP is commonly spurious

Why different than TCP

cost of spurious RTO and TLP is low in QUIC

Fast retransmit

Should we do adaptive time thresholding?

How do we best use both packet and time thresholds?

- working on this now

QUIC