Proportional Rate Reduction for TCP

draft-ietf-tcpm-proportional-rate-reduction-00.txt

IETF 82

16-Nov-2011

Matt Mathis, Nandita Dukkipati, Yuchung Cheng

Google

We want to improve TCP recovery

- Traces frequently show avoidable timeouts
  - TCP misses "obvious" opportunities to transmit
- Current implementation based in part on my prior work
  - Rate-Halving w/ bounding parameters
    - Send data on alternate ACKs during recovery
    - Incomplete ID and web pages from 1998
  - We abandoned it due to unsolved corner cases
  - Philosophy was to aim for cwnd=(FlightSize-losses)/2
    - Too conservative
    - Application stalls are treated like losses
  - Hard wired 50% cwnd reduction, even if CC does not
    - e.g. CUBIC uses only a 30% reduction

Standard TCP fast recovery (RFC3517)

FlightSize: outstanding (original) packets
pipe: estimated packets in the network

Entering recovery:

```
cwnd = ssthre□sh = FlightSize/2
retransmit_first_loss()
```

For every ACK during recovery:

```
pipe = update_scoreboard()
if cwnd > pipe
    transmit(cwnd - pipe)
```

Issues

- Half-RTT silence under light losses
- May (re)transmit large bursts under heavy losses
- Pipe can be wrong in the presence of reordering

Working from first principles

- Strictly packet conserving:
  - Arriving data triggers equal transmissions
  - Sender computes DeliveredData on each ACK
    - Well defined and robust even with reordering
    - Use DeliveredData as the recovery clock
    - Adjusted +/-1 to track cwnd/ssthresh
- Want recovery rate to be proportional to CC change
- Want final window to be chosen by CC
  - As it is with RFC 3517
  - Losses delay transmissions, but final window is the same
  - If losses exceed CC change, what action?

When losses exceed CC reduction

Three choices:

- Conservative bound (akin to rate halving)
  - Follow strict packet conservation during recovery
  - Window too small at the end of recovery
  - Slowstart after recovery
- Unlimited bound (follows 3517)
  - Allow full (ssthresh-pipe) bursts during recovery
- Slowstart bound
  - Relax conservative bound by 1 segment per ACK
  - Same total number of transmissions as 3517, but not in bursts

PRR with slowstart bound

Start of recovery:
ssthresh = CongCtrlAlg() // Target cwnd after recovery.
RecoverFS = snd.nxt - snd.una // FlightSize.
prr_delivered = prr_out = 0 // Accounting.

On each ACK in recovery, compute:
// DeliveredData: #pkts newly delivered to receiver.
DeliveredData = delta(snd.una) + delta(SACKd)
// Total pkts delivered in recovery.
prr_delivered += **DeliveredData**
pipe = RFC 3517 pipe algorithm

Algorithm:
if (pipe > ssthresh) // PRR.
  sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
else // Slow start.
  limit = max(prr_delivered − prr_out, **DeliveredData**) + 1
  sndcnt = MIN(ssthresh - pipe, limit)
On any data transmission or retransmission:
prr_out += (data sent)

PRR properties

- Better (ACK) clocking
  - fewer timeouts
  - more accurate fast recovery in spite of reordering, stretch acks, etc
  - smoother transmissions during recovery
- Cwnd converges to ssthresh
  - Not effected by additional loss or application stalls

PRR results

- Performs better than Rate Halving
  - Avoids excess window reductions
  - 3-10% better transaction response times
- Performs better than 3517
  - Avoids consequences of sending bursts
    (45% loss episodes cause pipe <= ssthresh)
    - Fewer lost retransmissions
    - Fewer timeouts
- See full results in IMC11 paper (slides attached)

New results for youtube in India

- Similar configuration as the Web experiment
- 3 days in $DC_{youtube-India}$
- Average video response is 2.3MB

	Linux	Standard	PRR
Retransmission rate	5.0%	6.6%	5.6%
Retransmission lost	2.4%	16.4%	4.8%
Slow start after recovery	56%	1%	0%

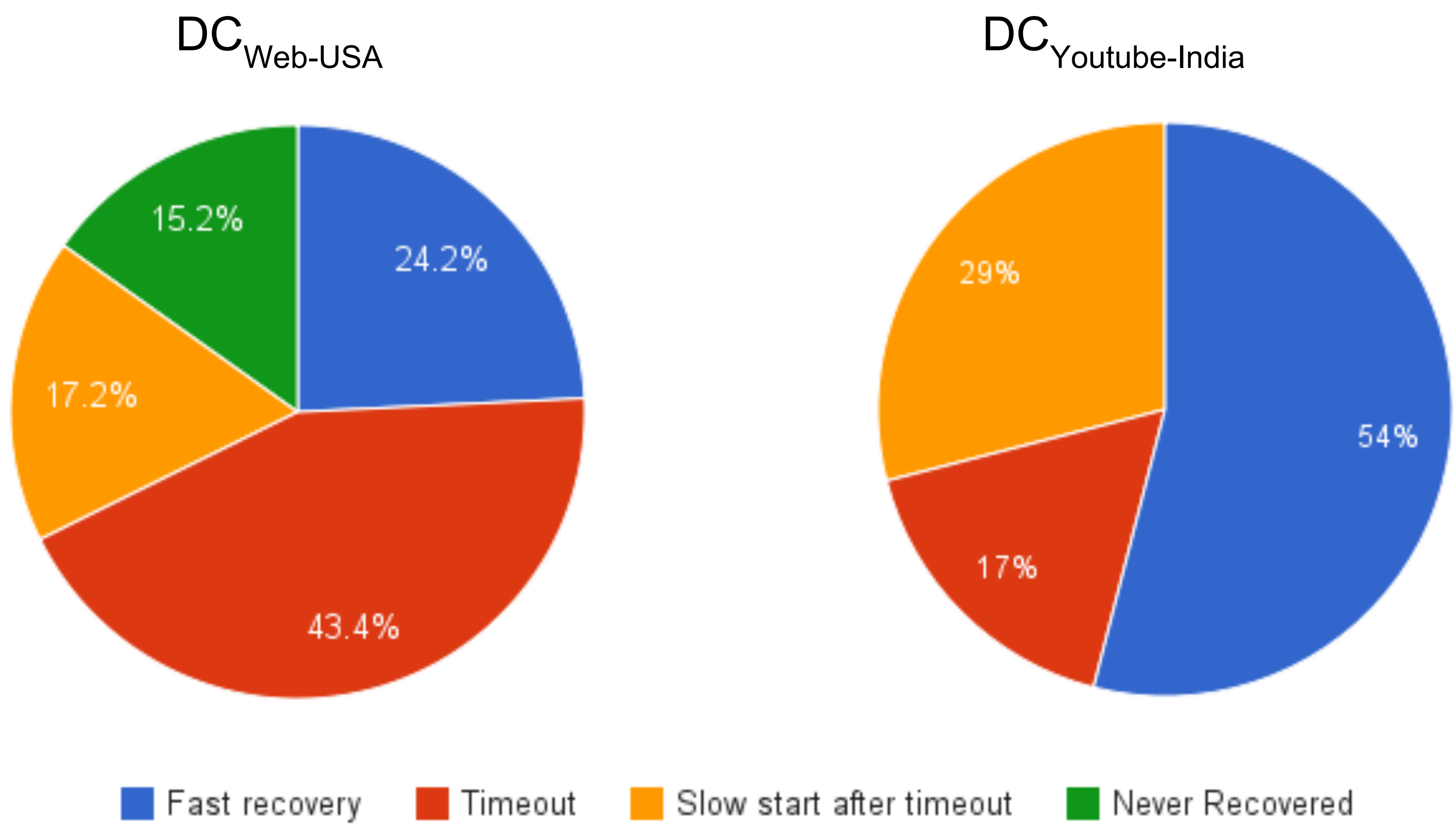
Standard TCP may cause high lost retransmission. PRR strikes the balance.

Onward

- Results are overwhelmingly good
- No substantiated downsides
- Already staged to Linux upstream

Post script: Total TCP retransmissions

in two Google data centers



15.2% USA retransmissions are for connections that NEVER recover! WHAT IS GOING ON?

IMC11 presentation

- Below is Yuchung's full presentation to IMC11 (Internet Measurement Conference)

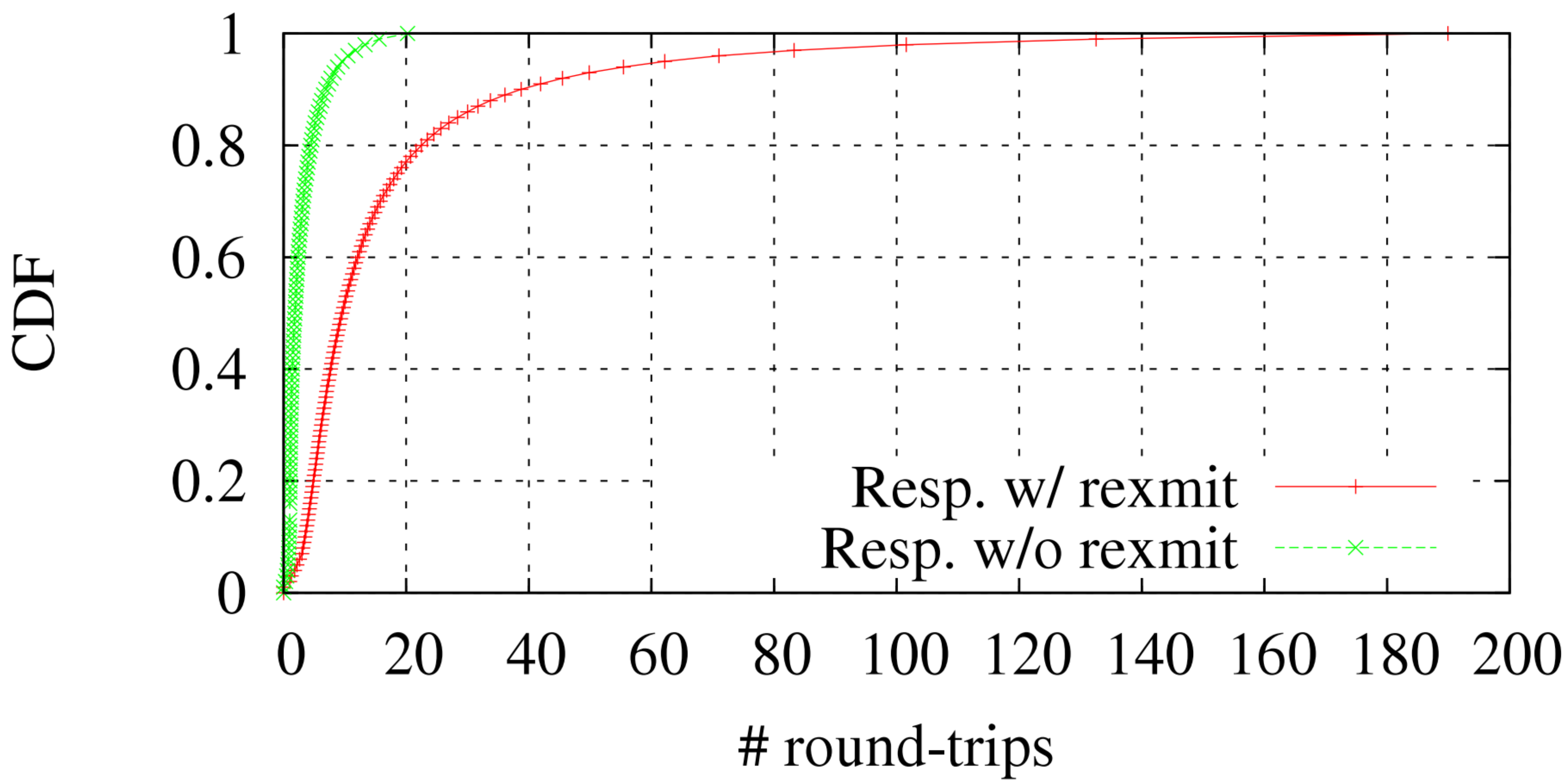
Proportional Rate Reduction for TCP

A fast and smooth loss recovery

Nandita Dukkipati, Matt Mathis, Yuchung Cheng, Monia Ghobadi

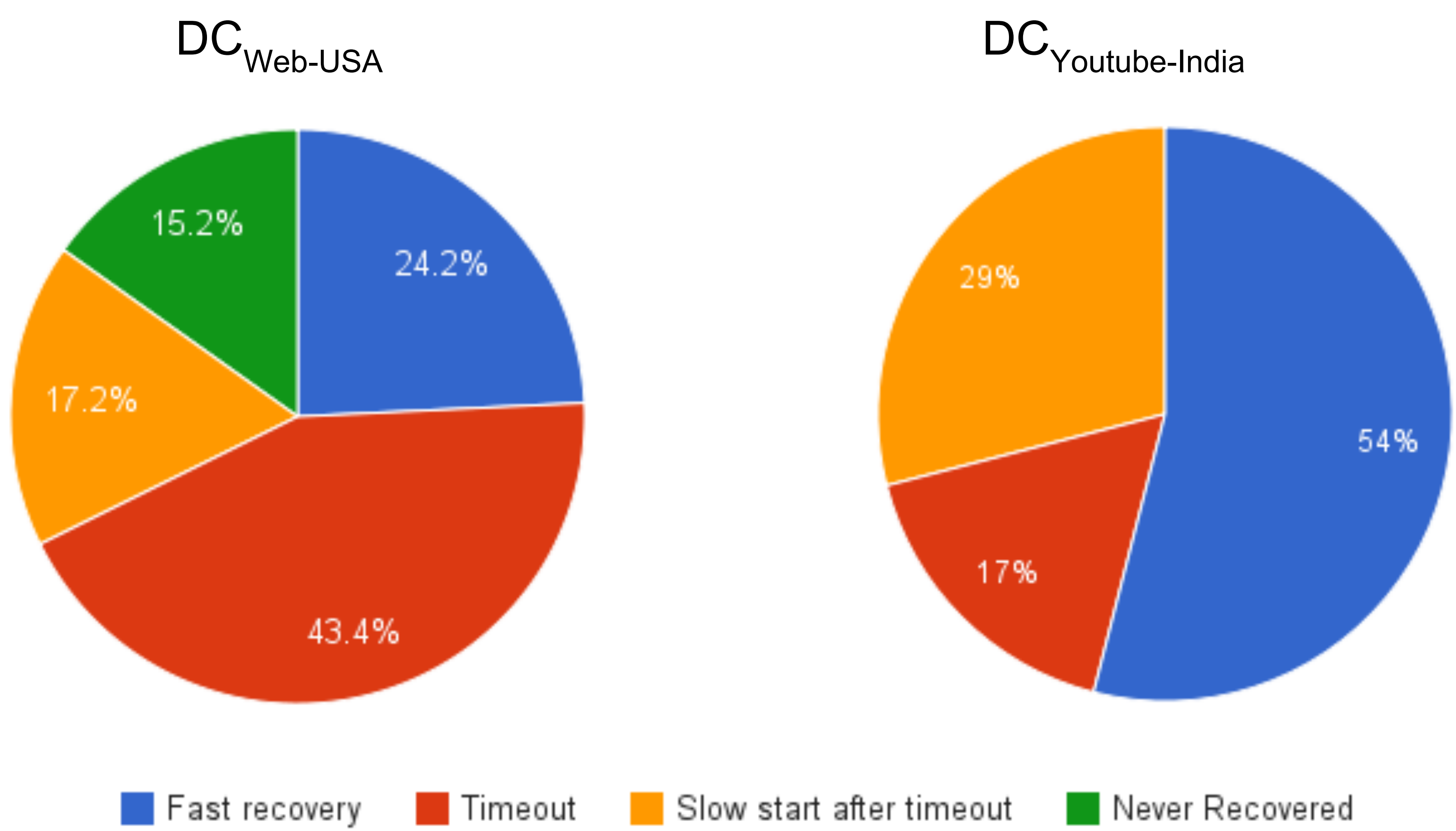
Google™

Losses hurt Web latency bad



Google HTTP responses. 6.1% experience losses.

How does TCP recover from losses?



TCP retransmission breakdown in two Google DCs. Over 96% connections support SACK.

Standard TCP fast recovery (RFC3517)

FlightSize: outstanding (original) packets
pipe: estimated packets in the network

```
Entering recovery:
    cwnd = ssthre□sh = FlightSize/2
    retransmit_first_loss()

For every ACK during recovery:
    pipe = update_scoreboard()
    if cwnd > pipe
        transmit(cwnd - pipe)
```

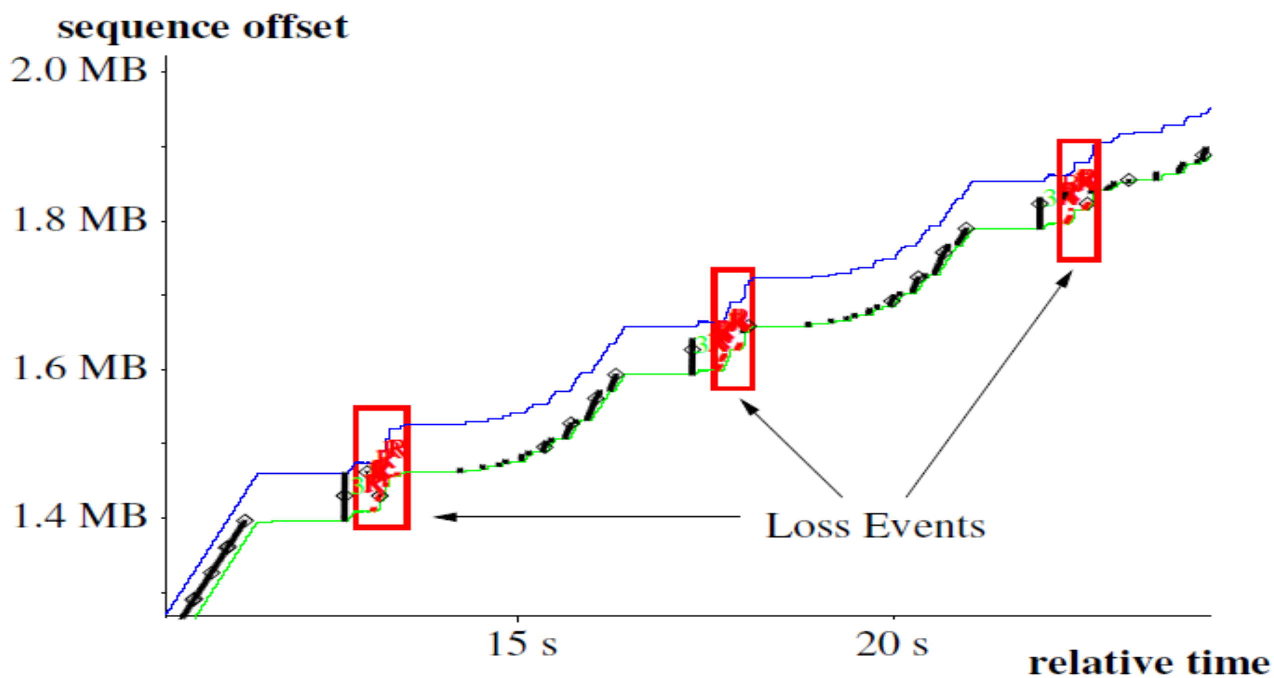
Issues

- Half-RTT silence under light losses
- May (re)transmit large burst under heavy losses

Linux TCP fast recovery

- Rate-halving: send one packet every other ACK
  - Too conservative under heavy losses

- cwnd moderation: $\text{cwnd} = \text{pipe}+1$ exiting recovery
  - Often slow start w/ $\text{cwnd} == 2$



Courtesy of "Application Flow Control in YouTube Video Streams", CCR, Apr., 2011

Proportional rate reduction (PRR)

Design principles

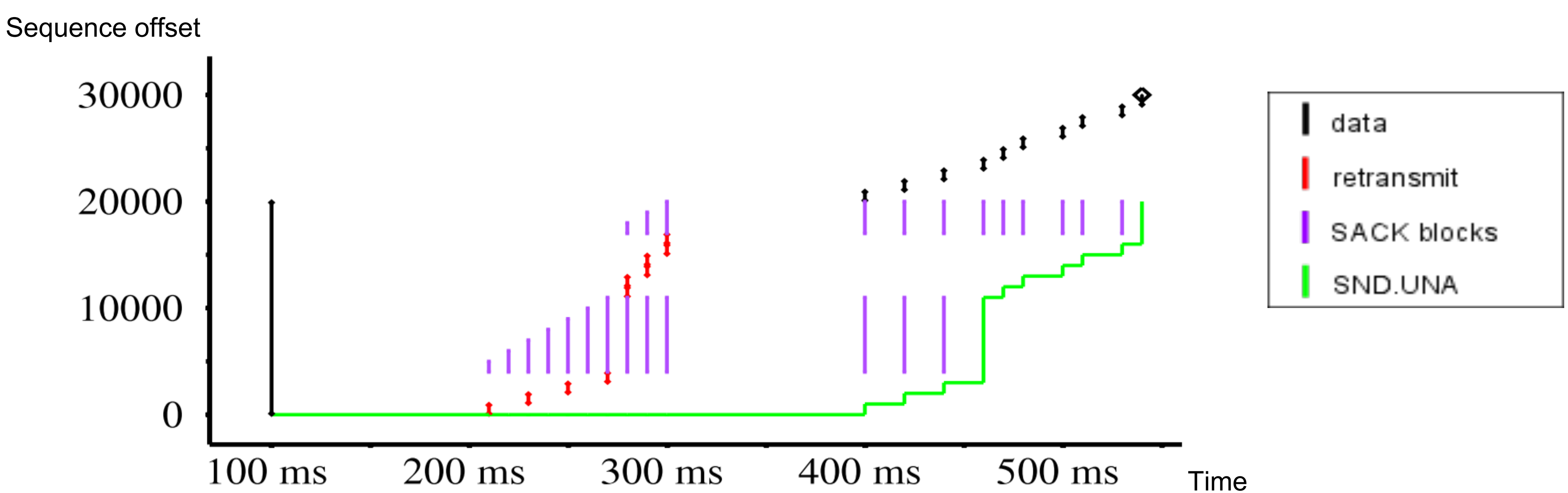
- VJ's packet conservation principle
- Decouples loss detection and window adjustment
  - Loss detection
    - *dupack_thresh, FACK, lost-retrans, etc*.
  - Window adjustment
    - Gradually reduces cwnd across acks
    - *pipe* converges to *ssthresh*
    - Works with different congestion controls

Proportional Rate Reduction (PRR)

Entering recovery: P = ssthresh / cwnd

For every ACK received:

- pipe > ssthresh
  - Reduce cwnd every P packets delivered
  - Transmit rate = P * delivery_rate
- pipe <= ssthresh
  - Slow start to bring pipe to ssthresh

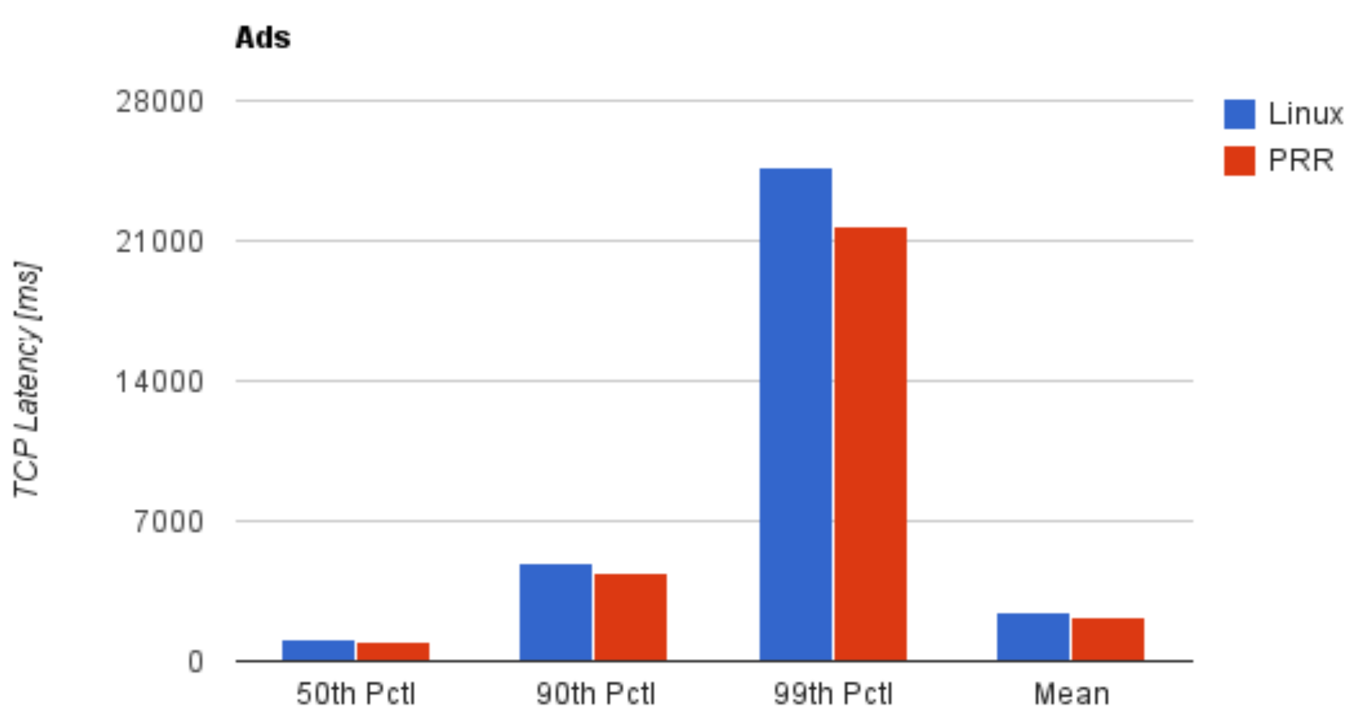


PRR properties

- Maintain ACK clocking
- Adjust cwnd by data delivered
  - More robust against reordering, stretched acks, loss detection errors, esp. with SACK
- cwnd converges to ssthresh after recovery
- Bank sending opportunities during application stalls

Google Web server experiment in US

- Experiment
  - Linux 2.6 with FACK, Cubic
  - Split servers in 3 groups: Standard, Linux, PRR
  - 5 days in $DC_{web\text{-}usa}$
- PRR
  - 45% fast recoveries start with $pipe <= ssthresh$
  - Reduce average TCP latency by 3-10% vs. Linux



Youtube experiment in India

- Similar configuration as the Web experiment
- 3 days in $DC_{youtube\text{-}India}$
- Average video response is 2.3MB

	Linux	Standard	PRR
Retransmission rate	5.0%	6.6%	5.6%
Retransmission lost	2.4%	16.4%	4.8%
Slow start after recovery	56%	1%	0%

Standard TCP may cause high lost retransmission. PRR strikes the balance.

Early retransmit (RFC 5827)

- *dupack_thresh* = 1 or 2 if FlightSize = 2 or 3
  - Increase fast retransmit by 13%
  - 24% are spurious due to (small) network reordering
- Mitigation
  - Stop if reordering > 3
  - Delay RTT/4 before early retransmit
  - Reduce spurious retransmission rate to 6%

Percentile	Linux	ER w/ mitigation	Improvement
10th	319 ms	301 ms	-5.6%
50th	1084 ms	997 ms	-8.0%
90th	4223 ms	4084 ms	-3.3%

TCP latency of all responses except ones that has < 2 packets or do not experience losses

Conclusion

- Packet losses significantly increase Web latency

- PRR is a new TCP fast recovery algorithm
  - Recovers quickly and smoothly
  - Adopted by Linux upstream :-)
  - IETF RFC in progress

- Early retransmit (ER)
  - Useful but needs to mitigate reordering
  - Both PRR and ER are being deployed on all Google servers

- Ongoing efforts
  - Timeout recovery, mobile TCP, TCP Fast Open, TCP/video

PRR full algorithm

Start of recovery:
ssthresh = CongCtrlAlg() // Target cwnd after recovery.
RecoverFS = snd.nxt - snd.una // FlightSize.
prr_delivered = prr_out = 0 // Accounting.

On each ACK in recovery, compute:
// DeliveredData: #pkts newly delivered to receiver.
DeliveredData = delta(snd.una) + delta(SACKd)
// Total pkts delivered in recovery.
prr_delivered += **DeliveredData**
pipe = RFC 3517 pipe algorithm

Algorithm:
if (pipe > ssthresh) // PRR.
  sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
else // Slow start.
  ss_limit = max(prr delivered − prr out, **DeliveredData**) + 1
  sndcnt = MIN(ssthresh - pipe, ss_limit)
On any data transmission or retransmission:
prr_out += (data sent)