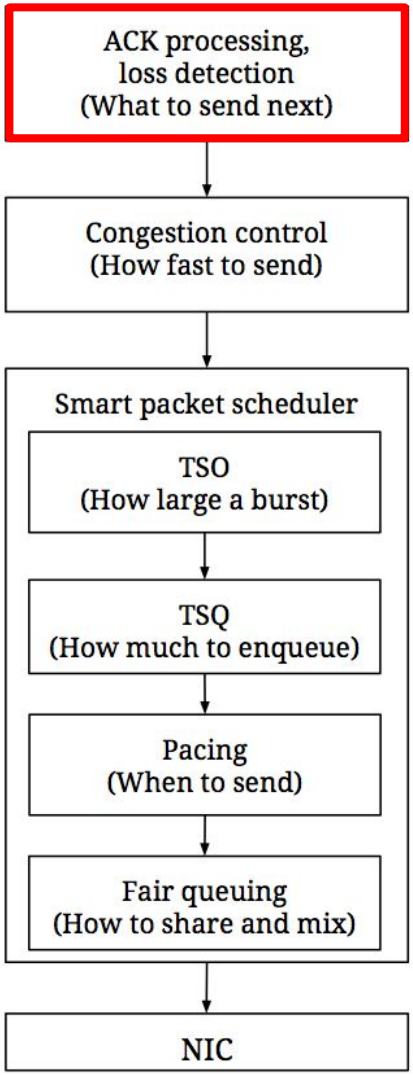
Making Linux TCP Fast

Yuchung Cheng
Neal Cardwell

netdev 1.2 Tokyo, October, 2016

Once upon a time, there was a TCP ACK...

Here is the a story of what happened next...



RACK: detect losses by packets' send time

Monitors the delivery process of every (re)transmission. E.x.

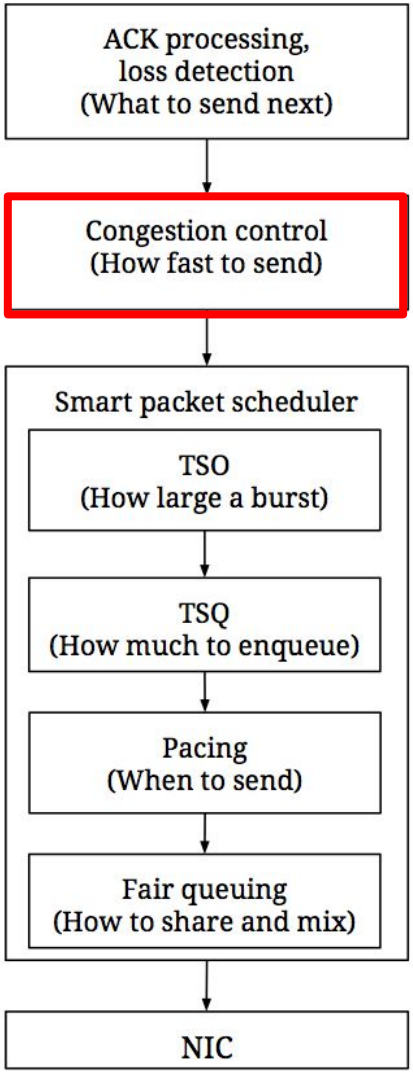
Sent packets P1 and P2

Receives a SACK of P2

=> P1 is lost if sent more than $RTT + $reo_wnd ago[1]

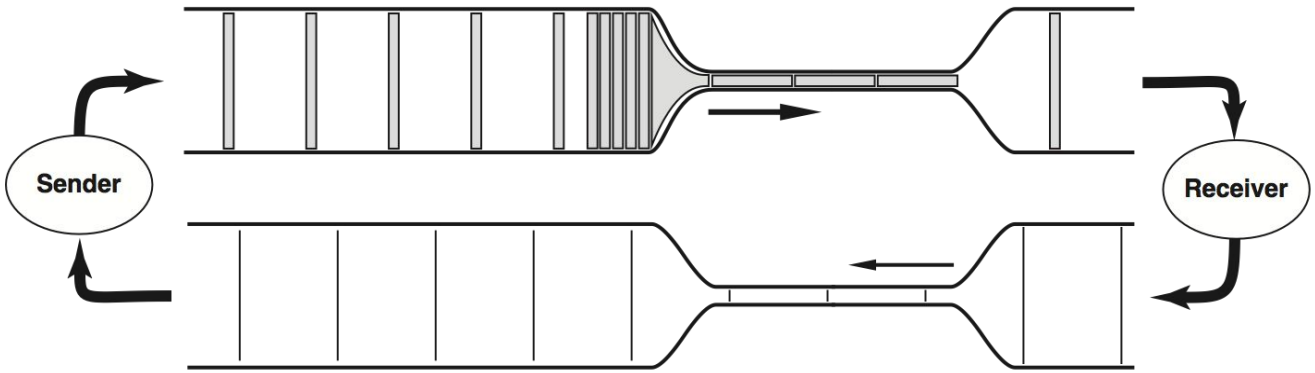
Reduce timeouts in Disorder state by 80% on Google.com

[1] RACK draft-ietf-tcpm-rack-00 since Linux 4.4

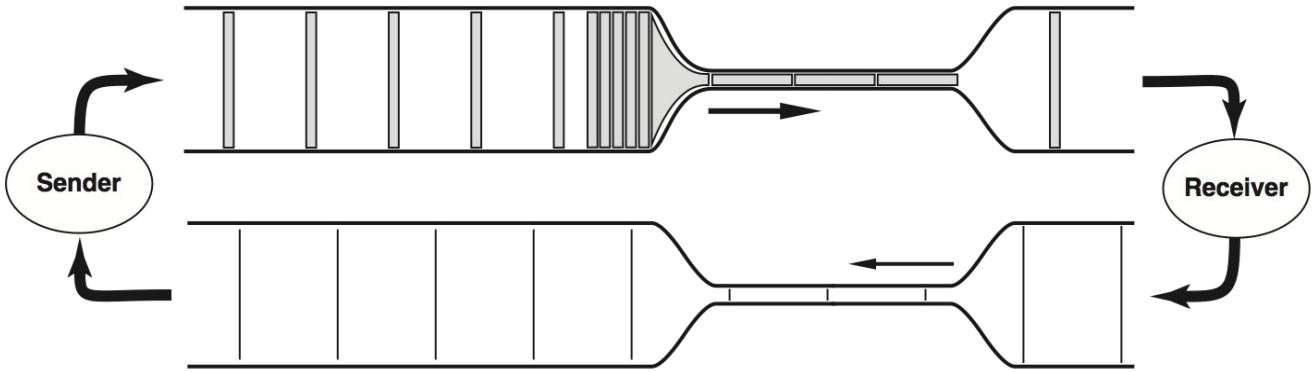


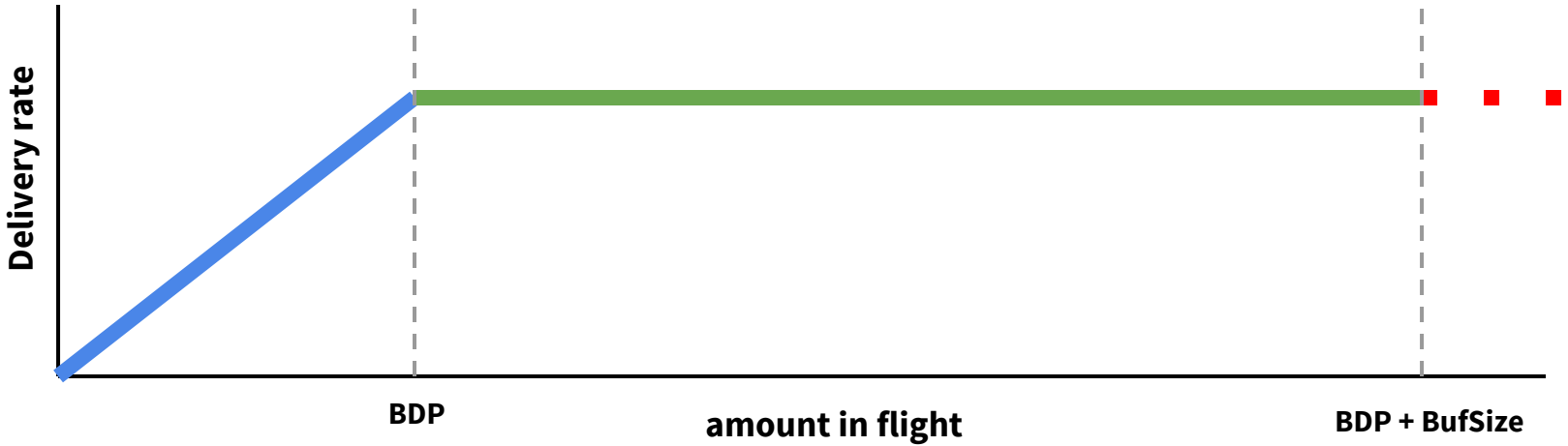
congestion control: how fast to send?

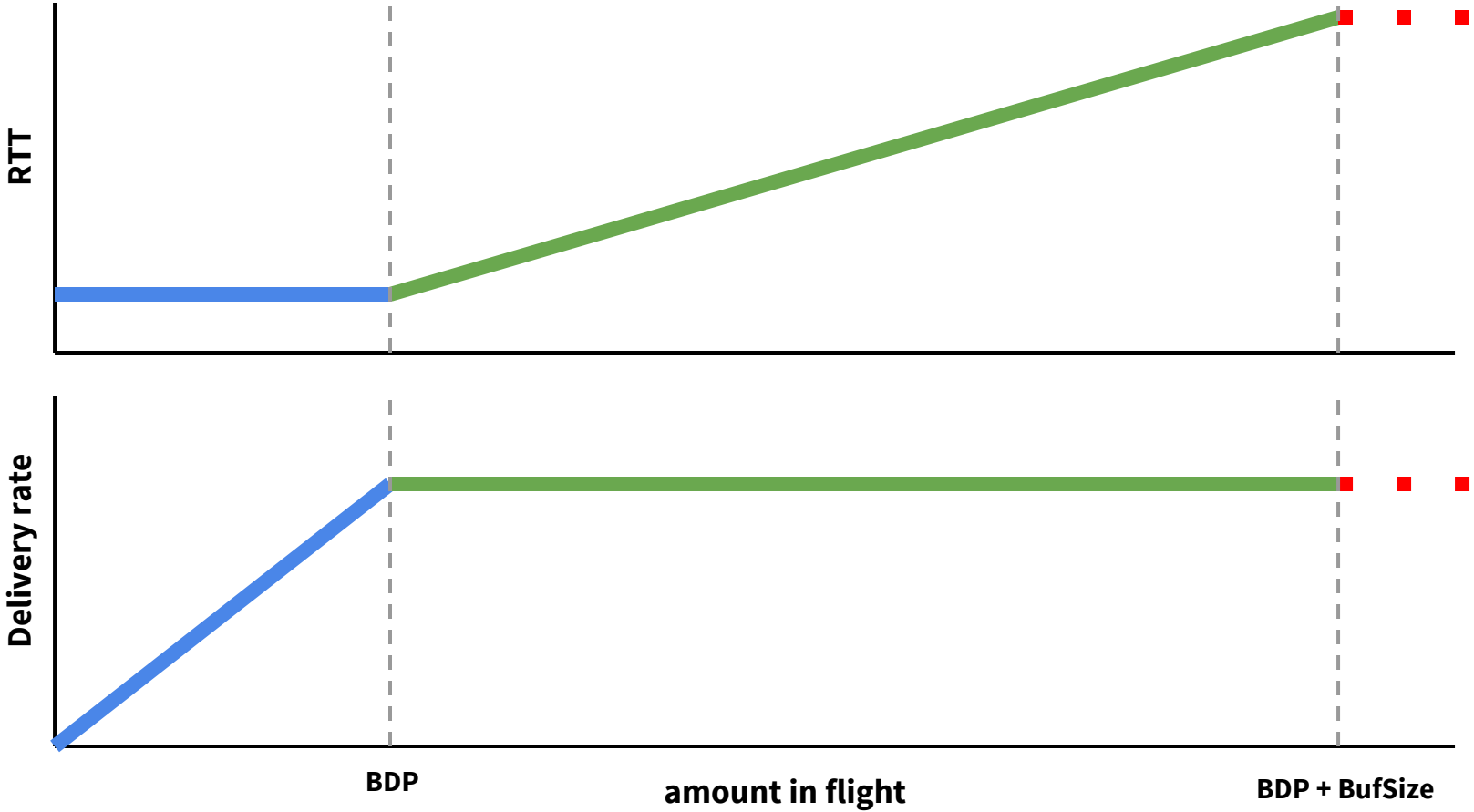
Congestion and bottlenecks

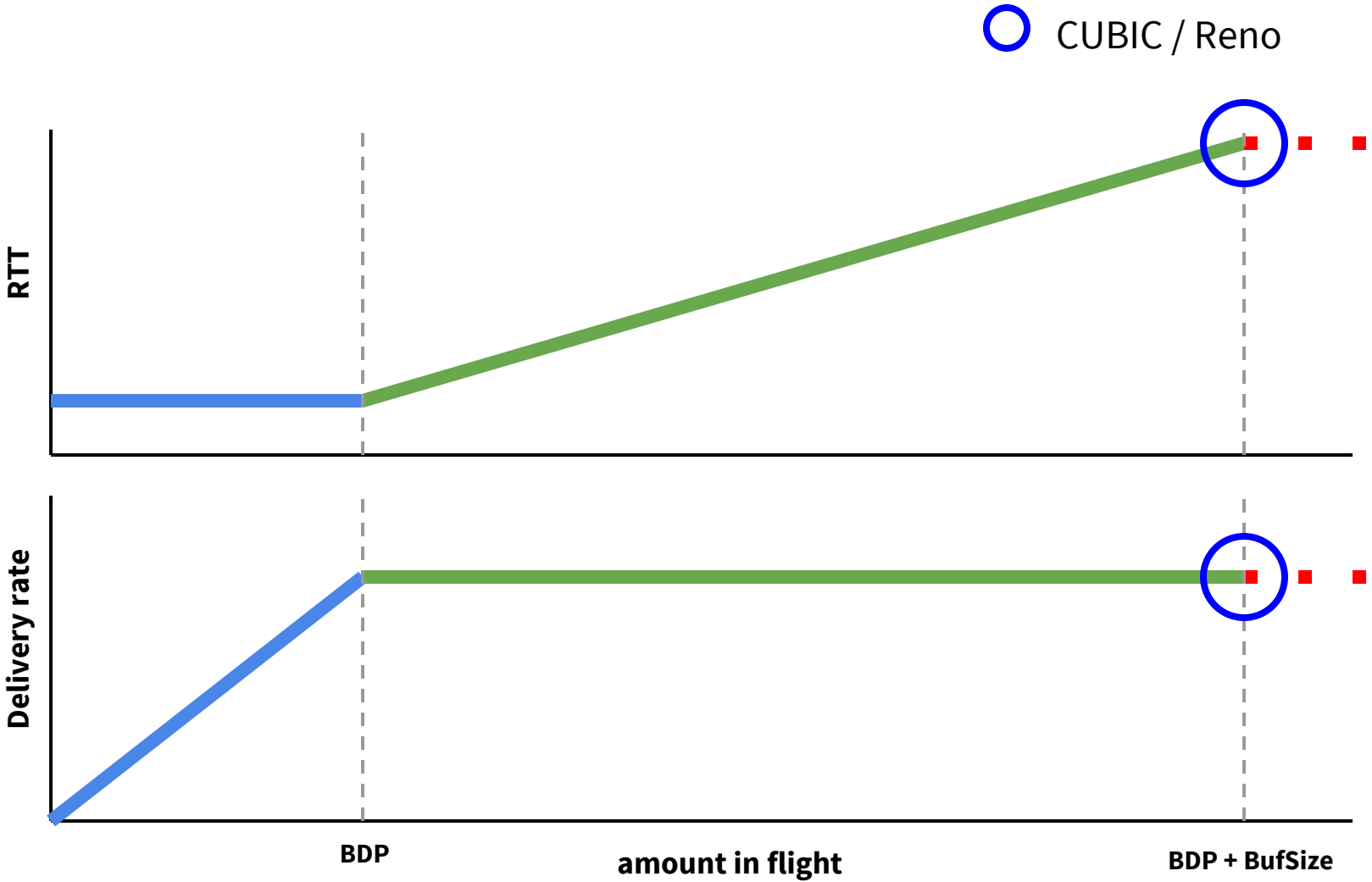


Congestion and bottlenecks

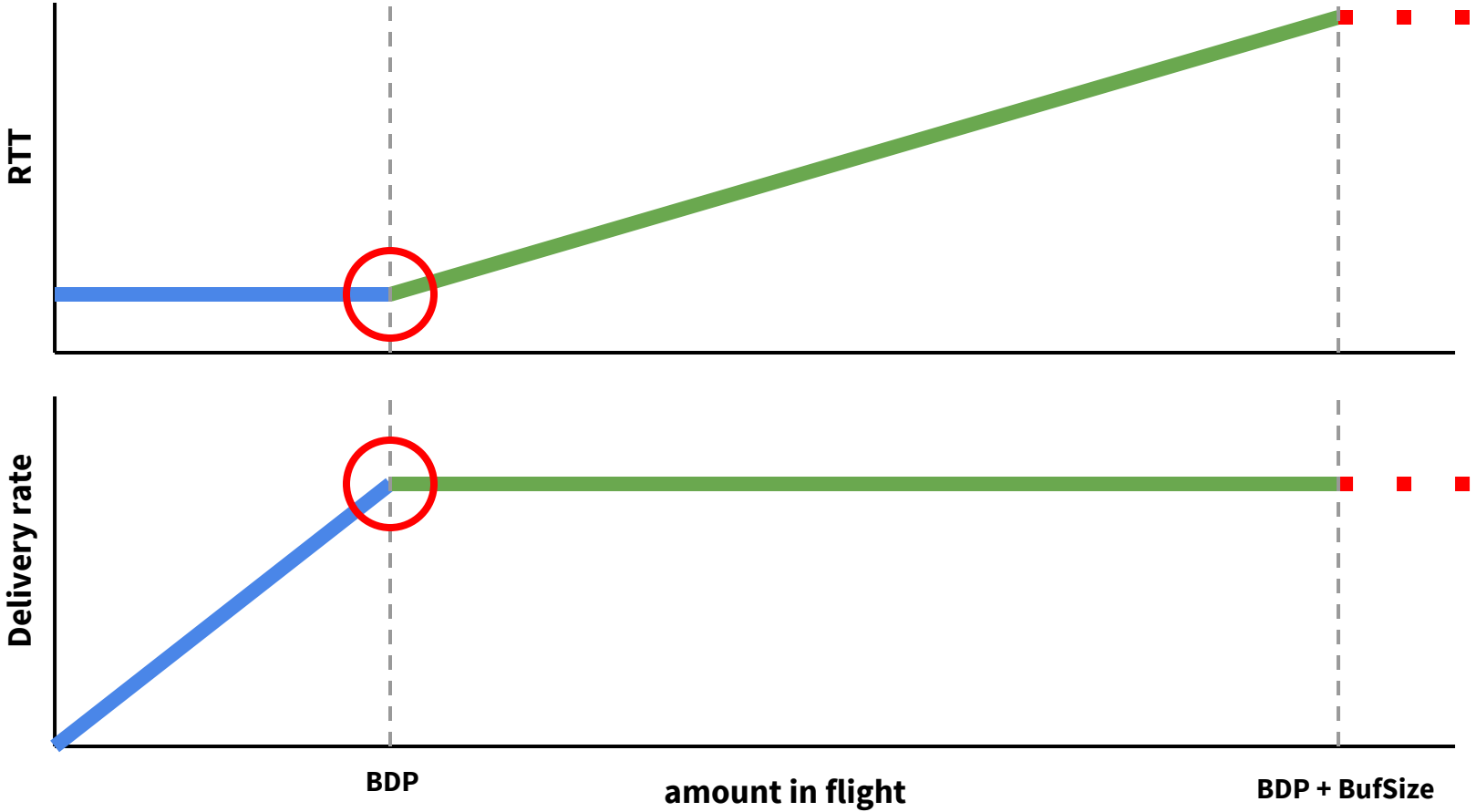




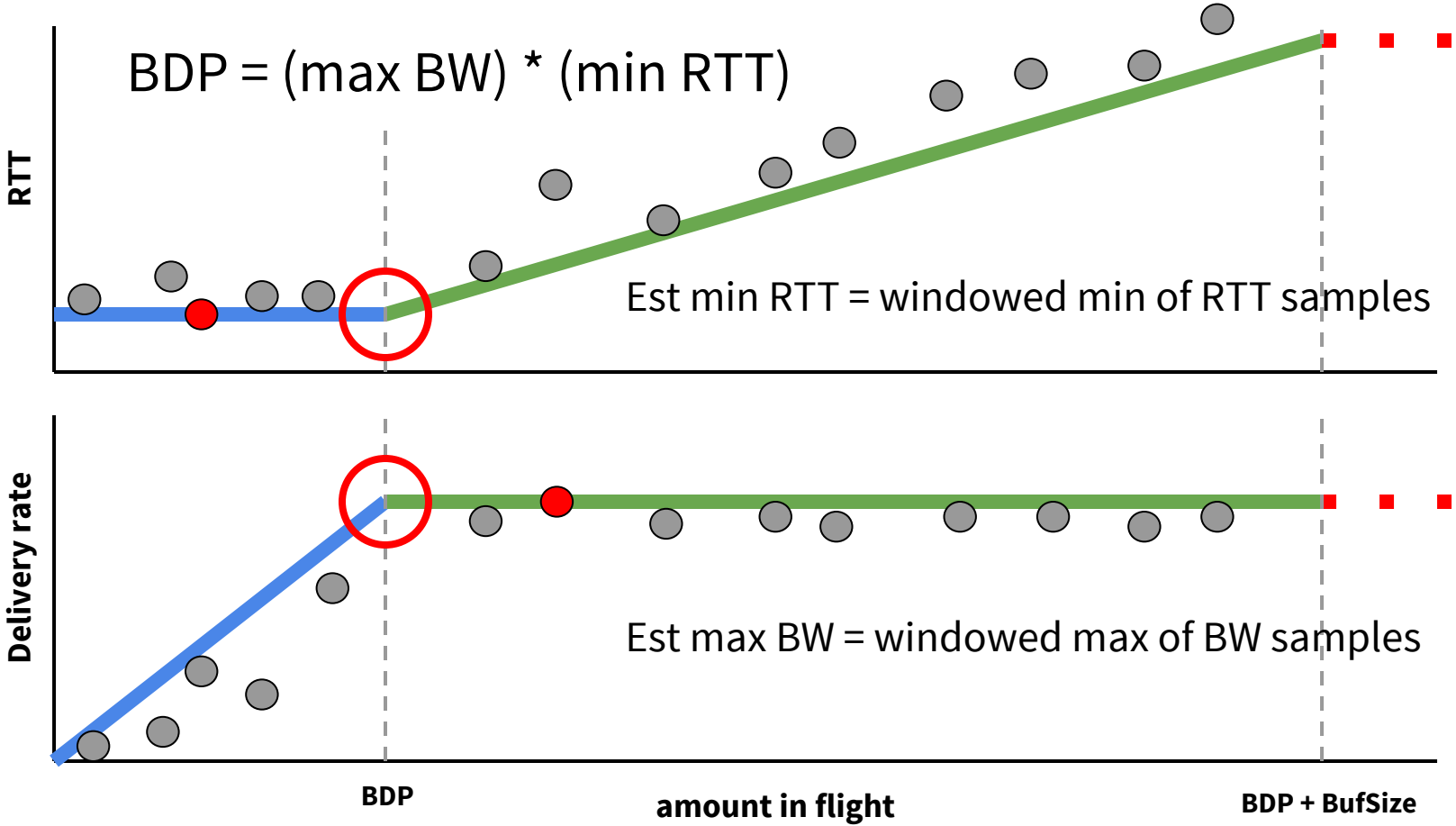

RTT
Delivery rate
BDP
amount in flight
BDP + BufSize


CUBIC / Reno
RTT
Delivery rate
BDP
amount in flight
BDP + BufSize

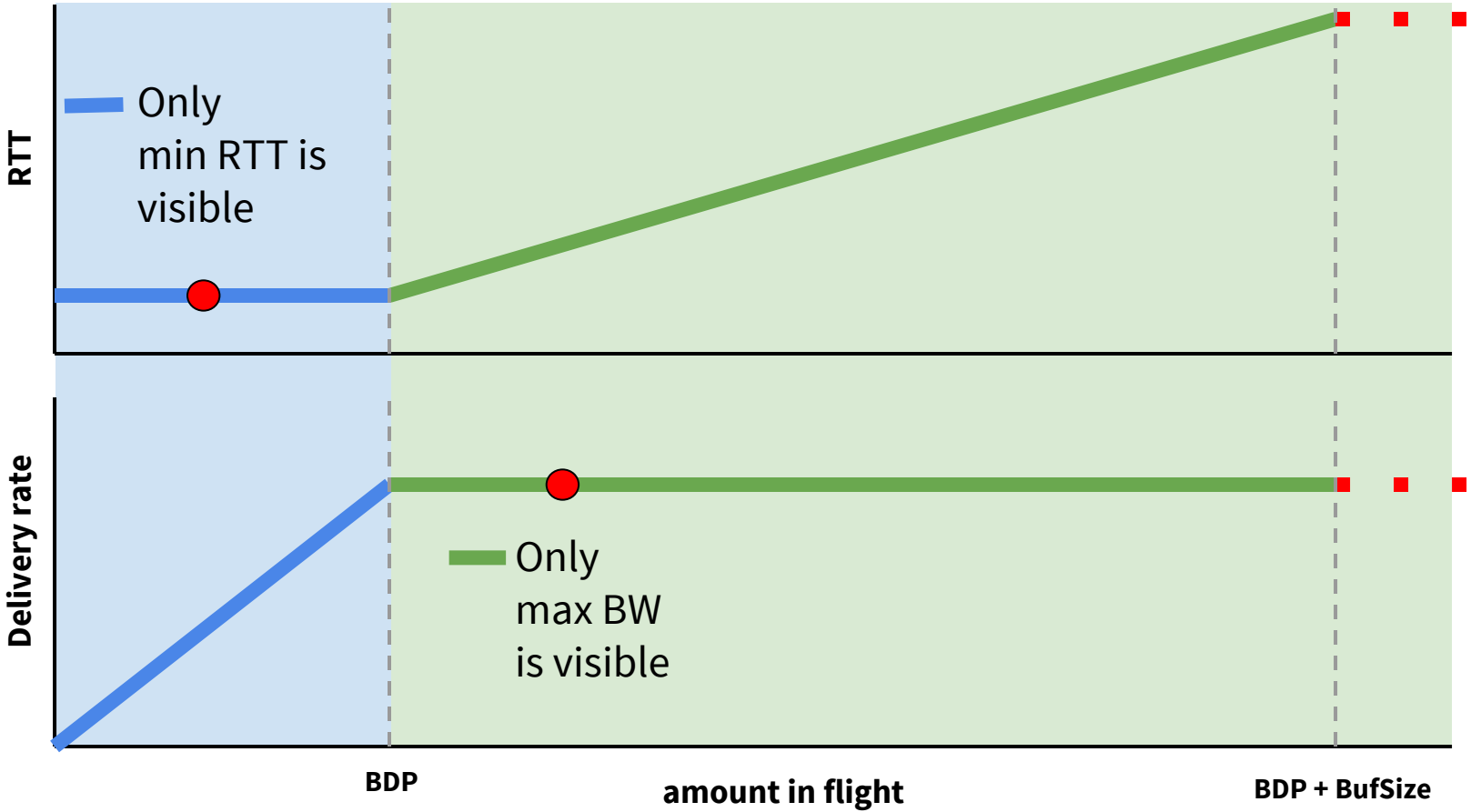
Optimal: max BW and min RTT (Gail & Kleinrock. 1981)



Estimating optimal point (max BW, min RTT)



But to see both max BW and min RTT,
must probe on both sides of BDP...



One way to stay near (max BW, min RTT) point:

Model network, update max BW and min RTT estimates on each ACK

Control sending based on the model, to...

Probe both max BW and min RTT, to feed the model samples

Pace near estimated BW, to reduce queues and loss

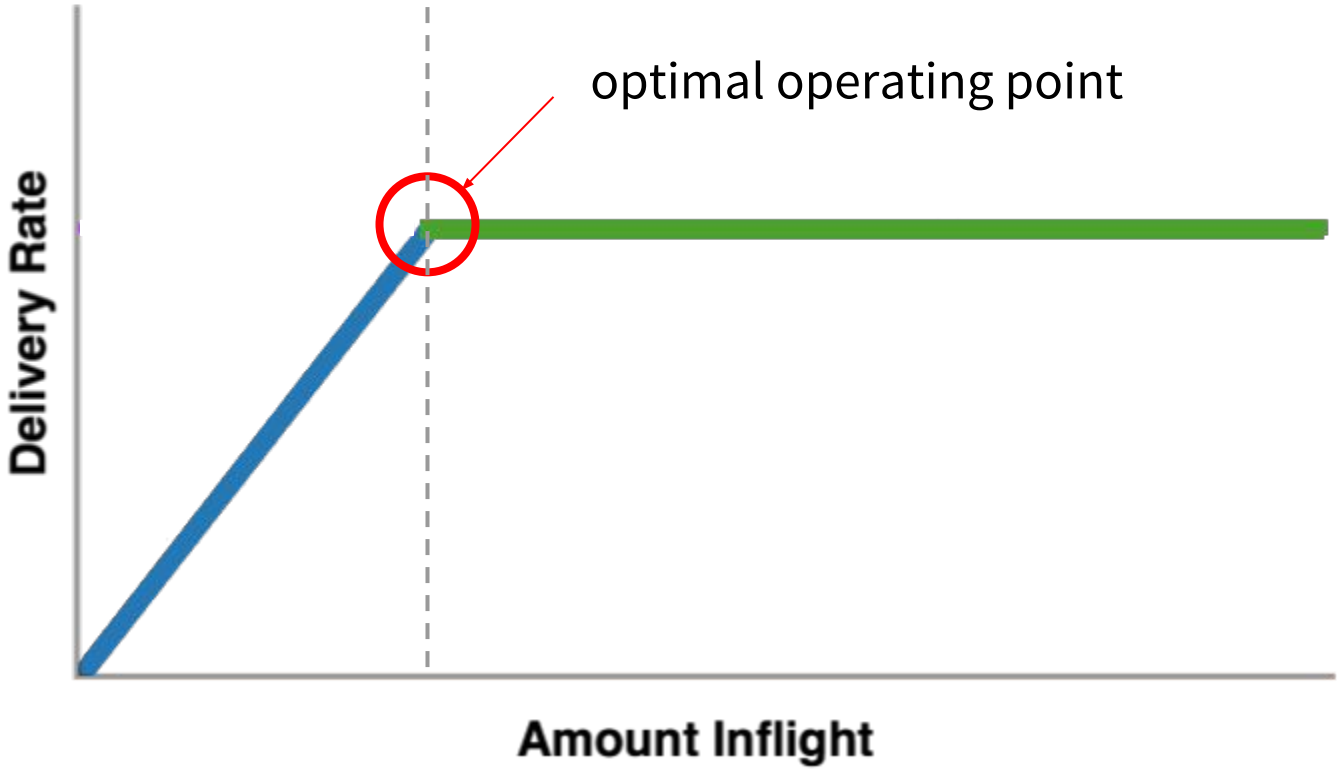
Vary pacing rate to keep inflight near BDP (for full pipe but small queue)

That's **BBR** congestion control (code in Linux v4.9; paper: ACM Queue, Oct 2016)

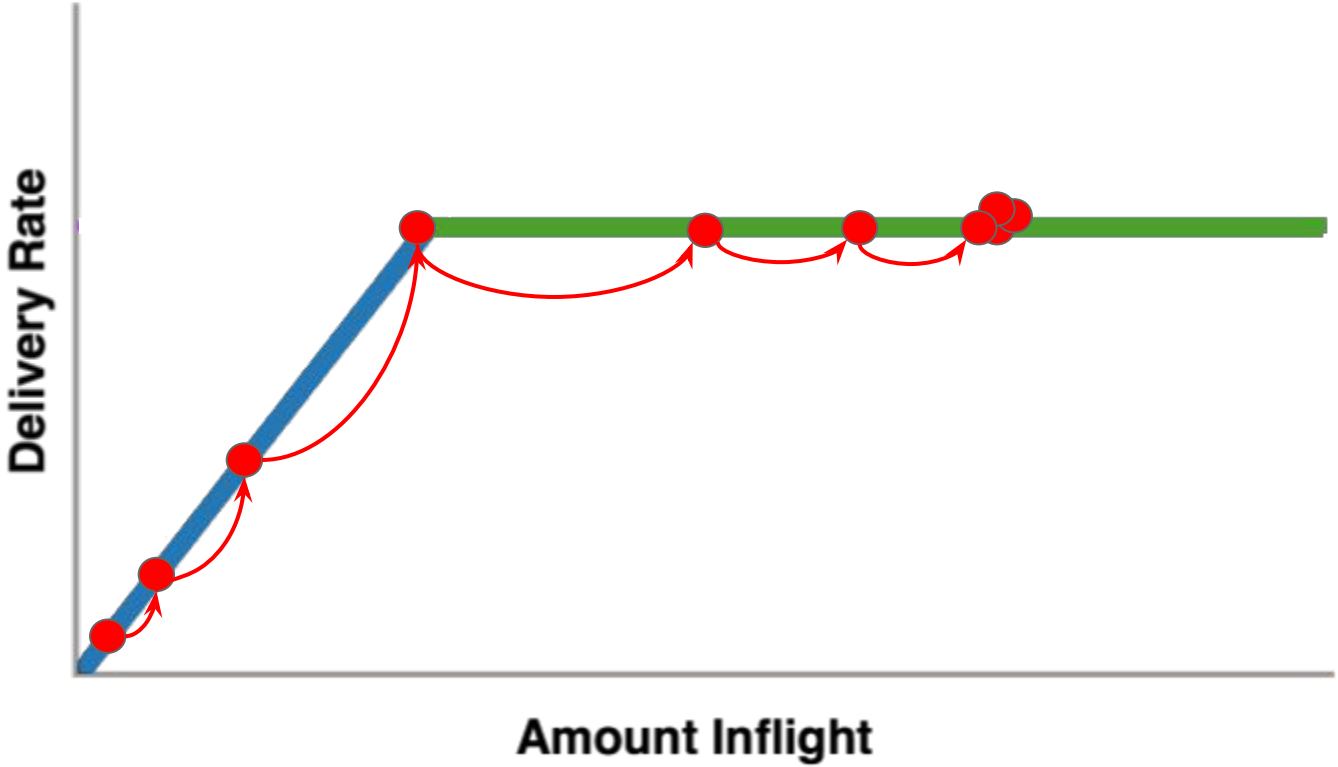
BBR = **B**ottleneck **B**andwidth and **R**ound-trip propagation time

BBR seeks high tput with small queue by probing BW and RTT *sequentially*

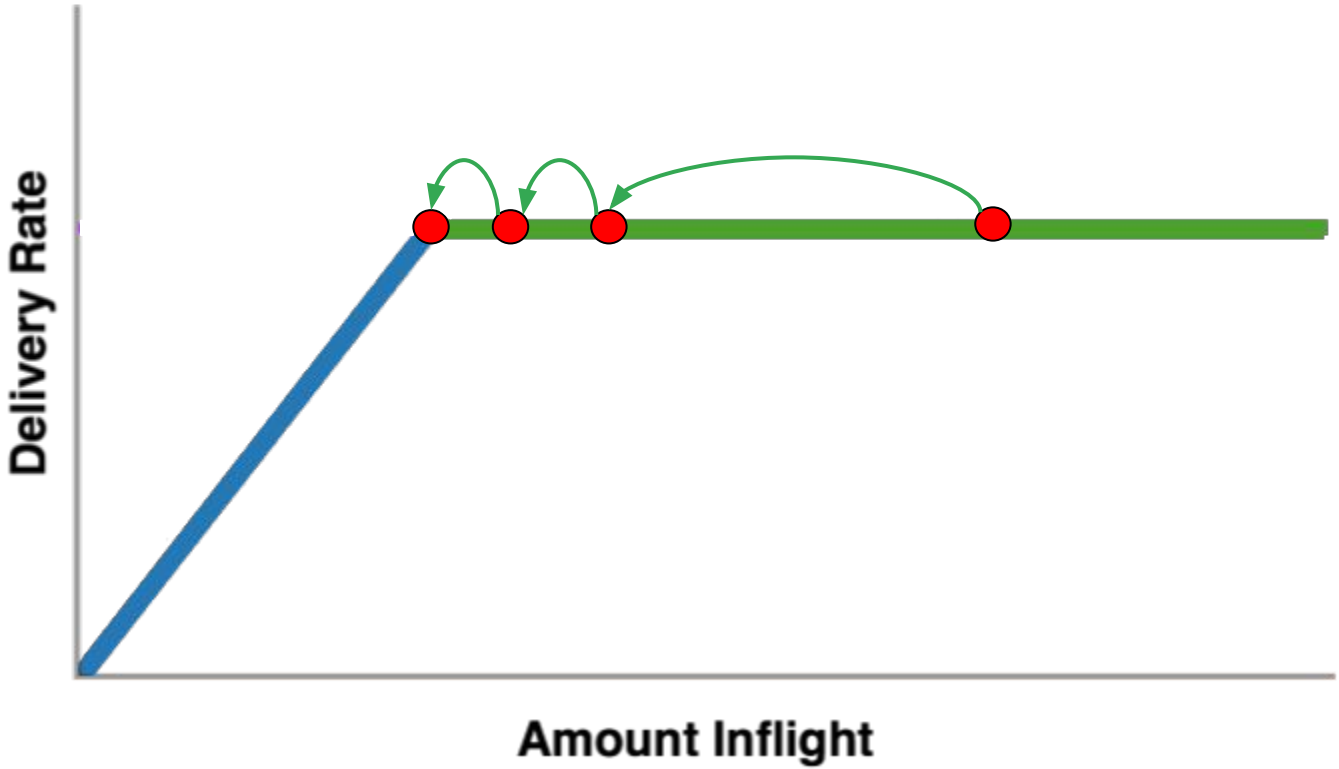
BBR: model-based walk toward max BW, min RTT



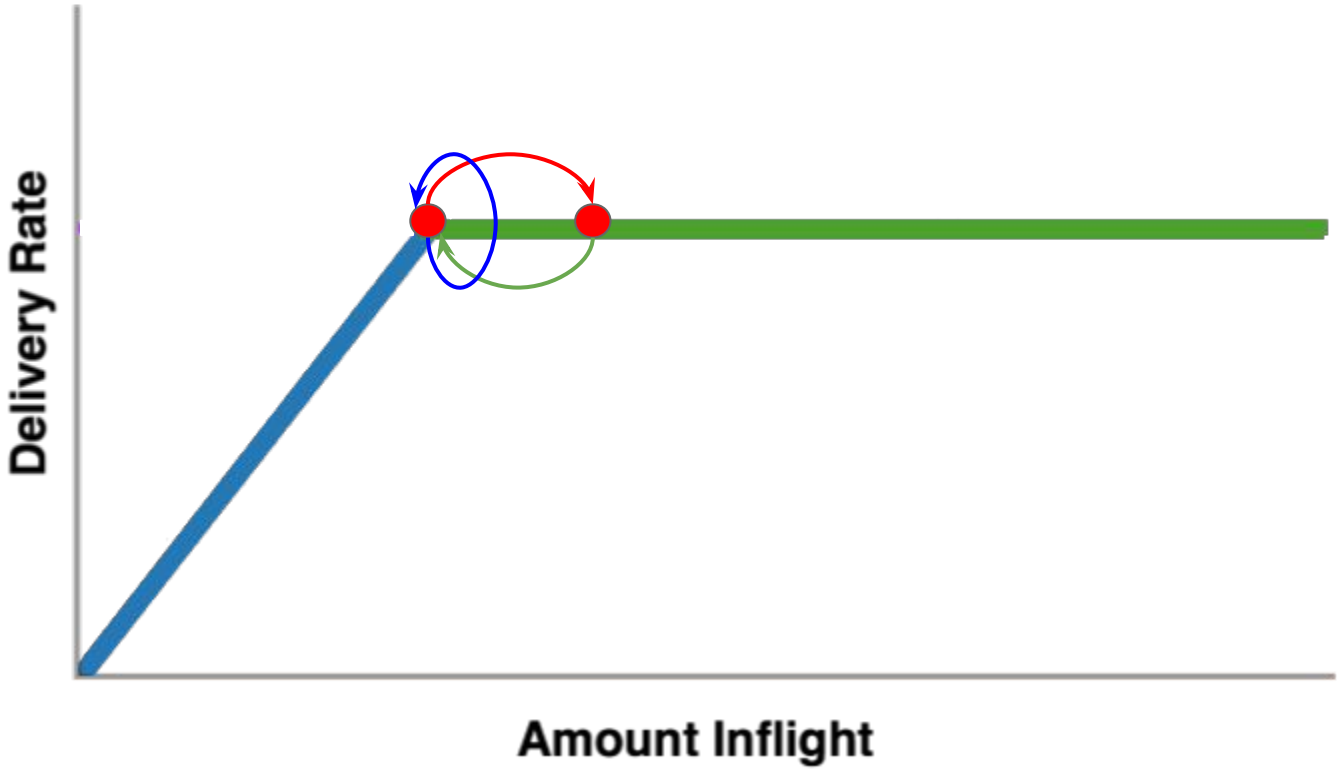
STARTUP: exponential BW search



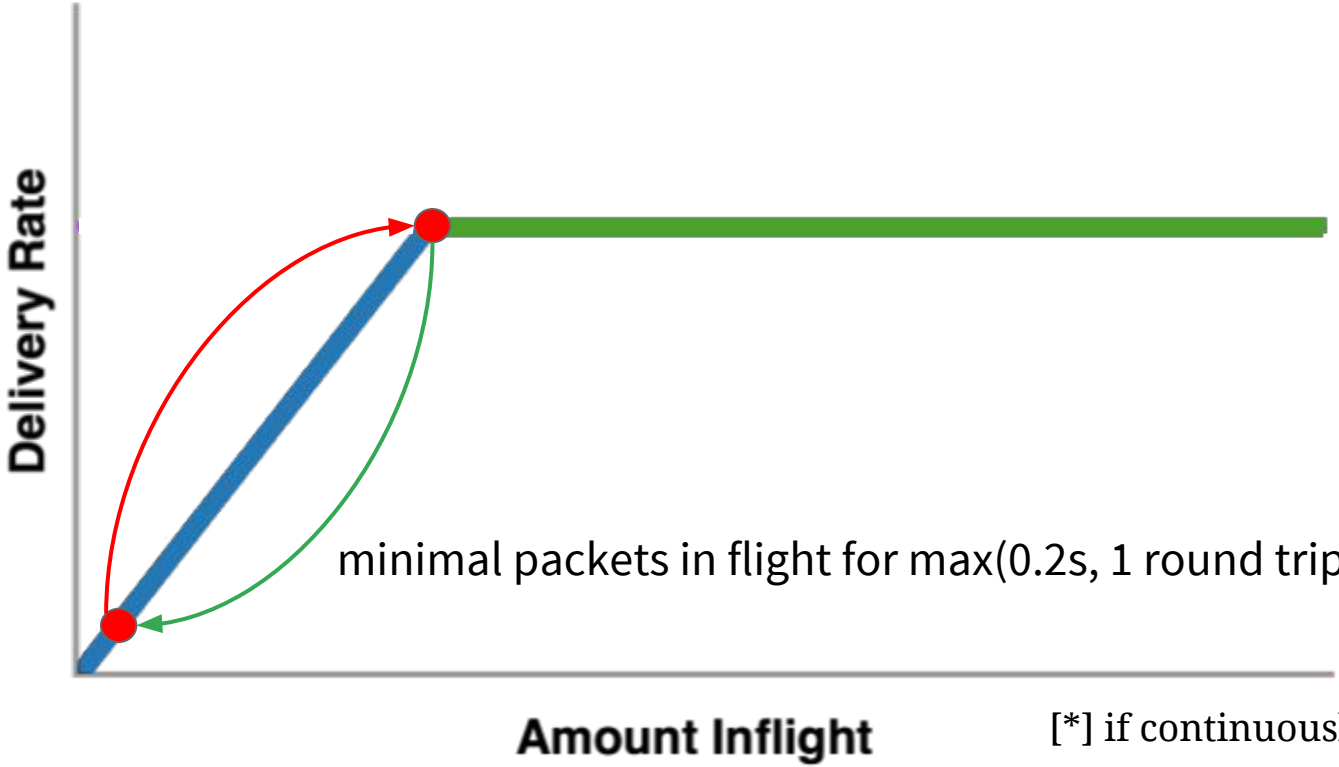
DRAIN: drain the queue created during startup



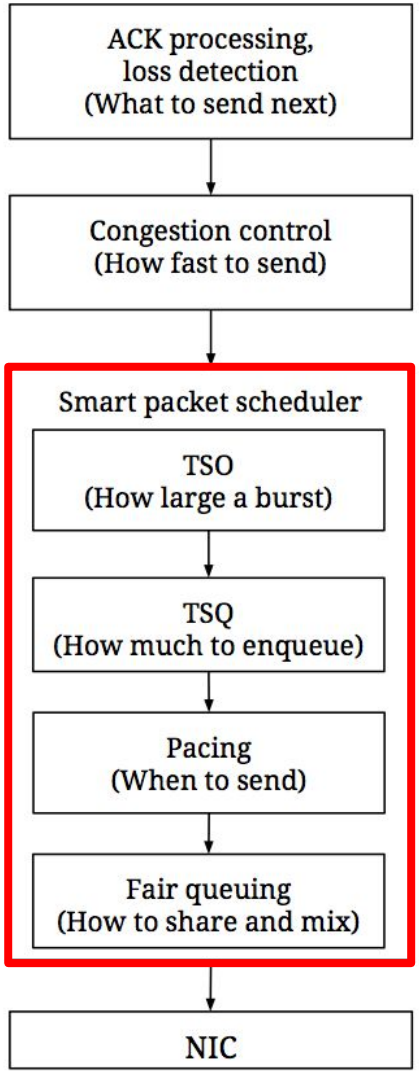
PROBE_BW: explore max BW, drain queue, cruise



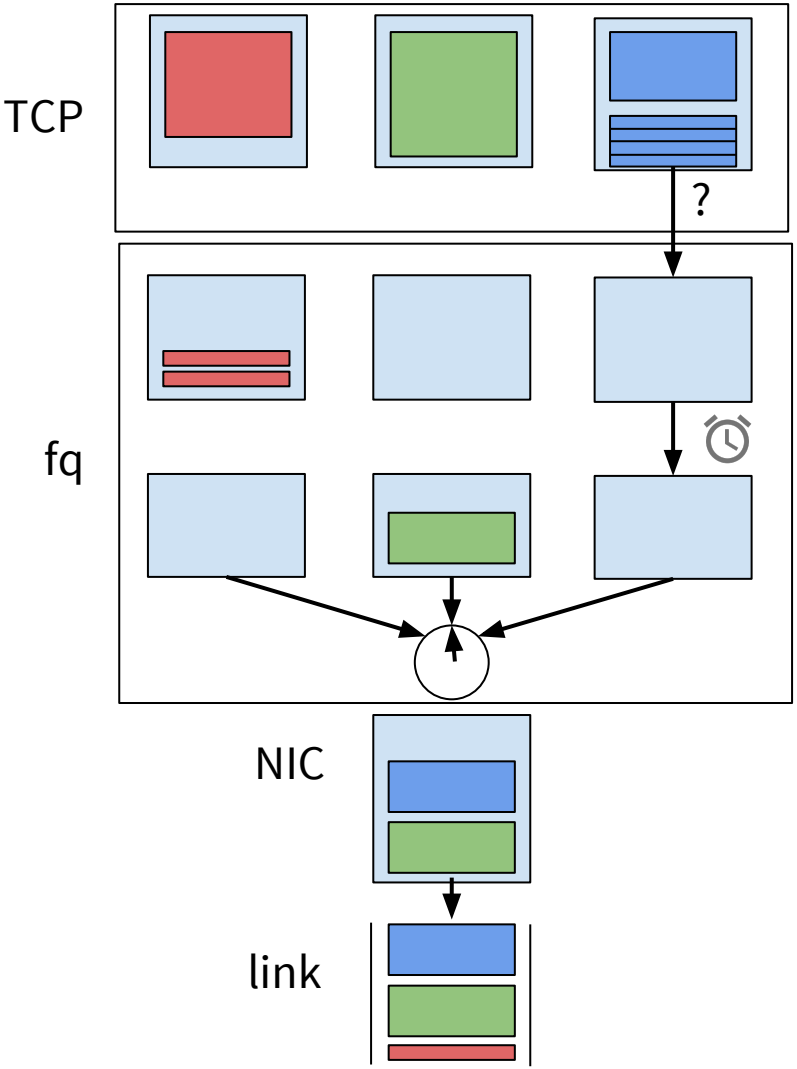
PROBE_RTT briefly if min RTT filter expires (=10s)*



[*] if continuously sending



Packet scheduling: when to send?



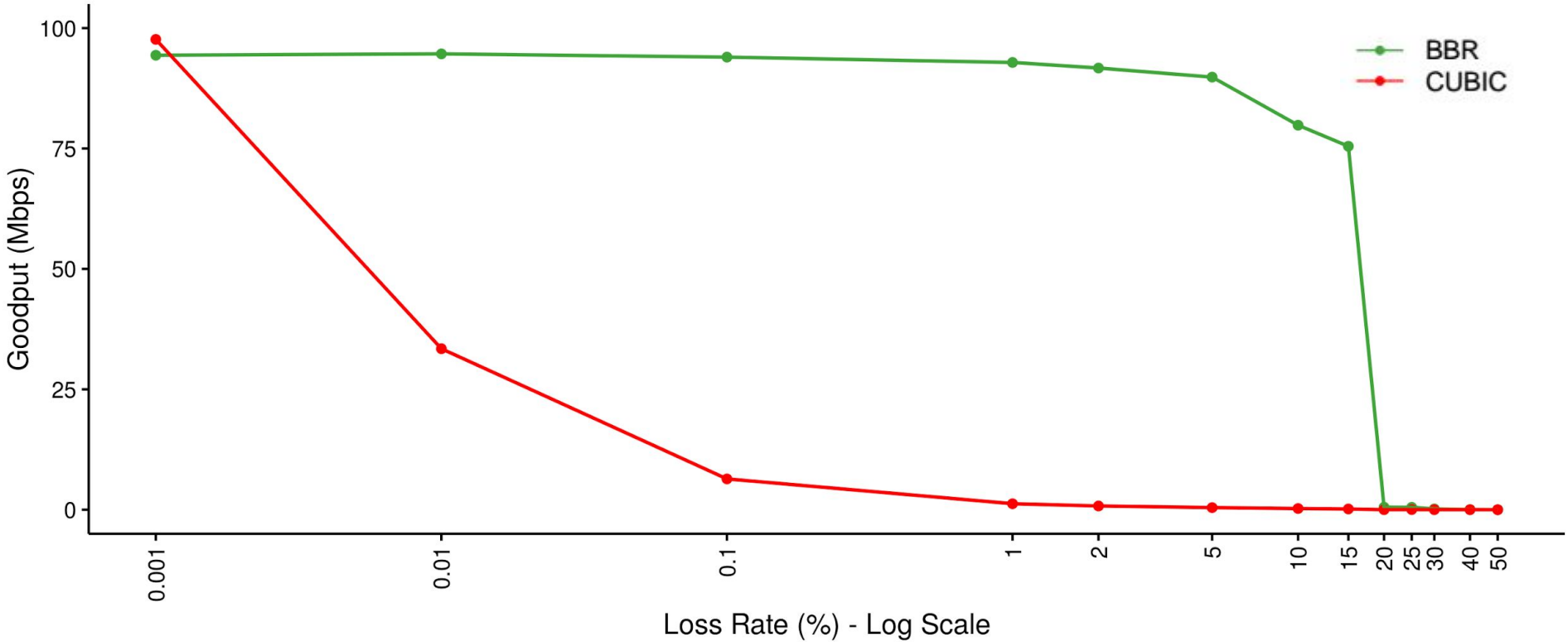
TSO autosizing
TCP Small Queues (TSQ)

Pacing

Fair queuing

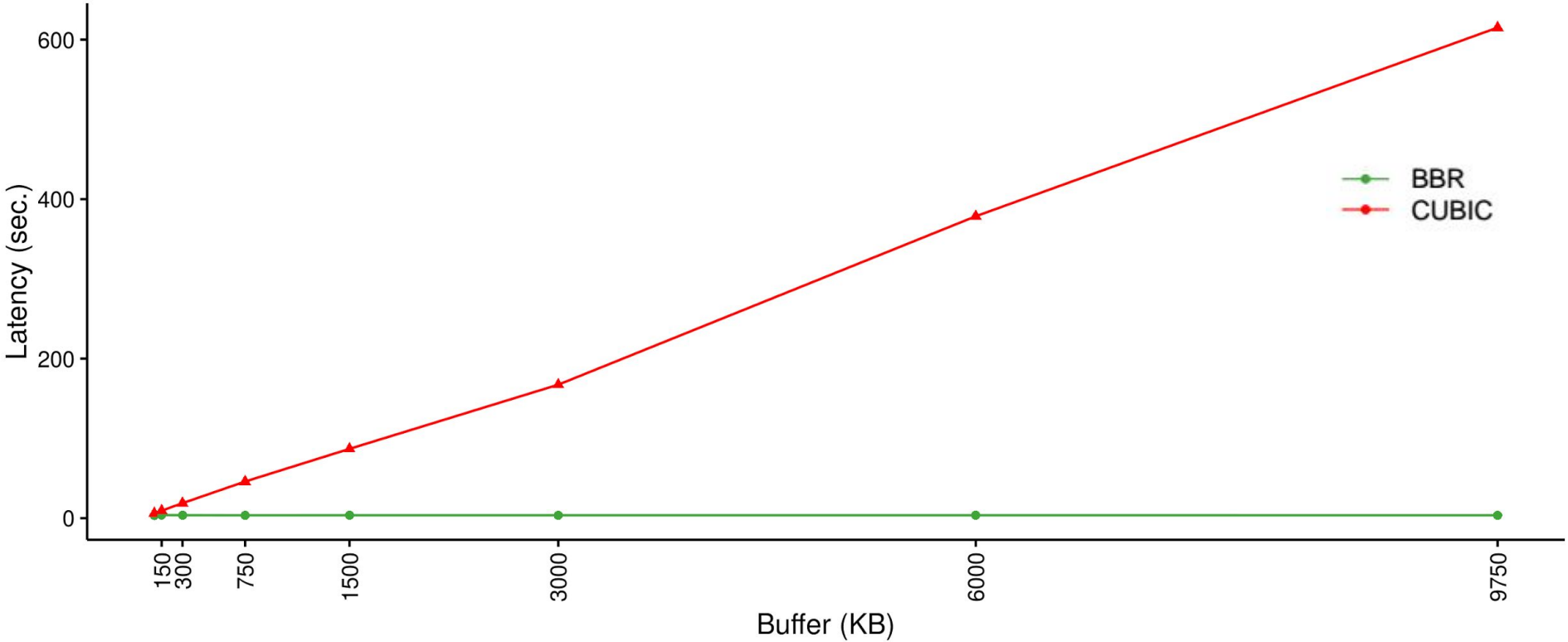
Performance results...

Fully use bandwidth, despite high loss



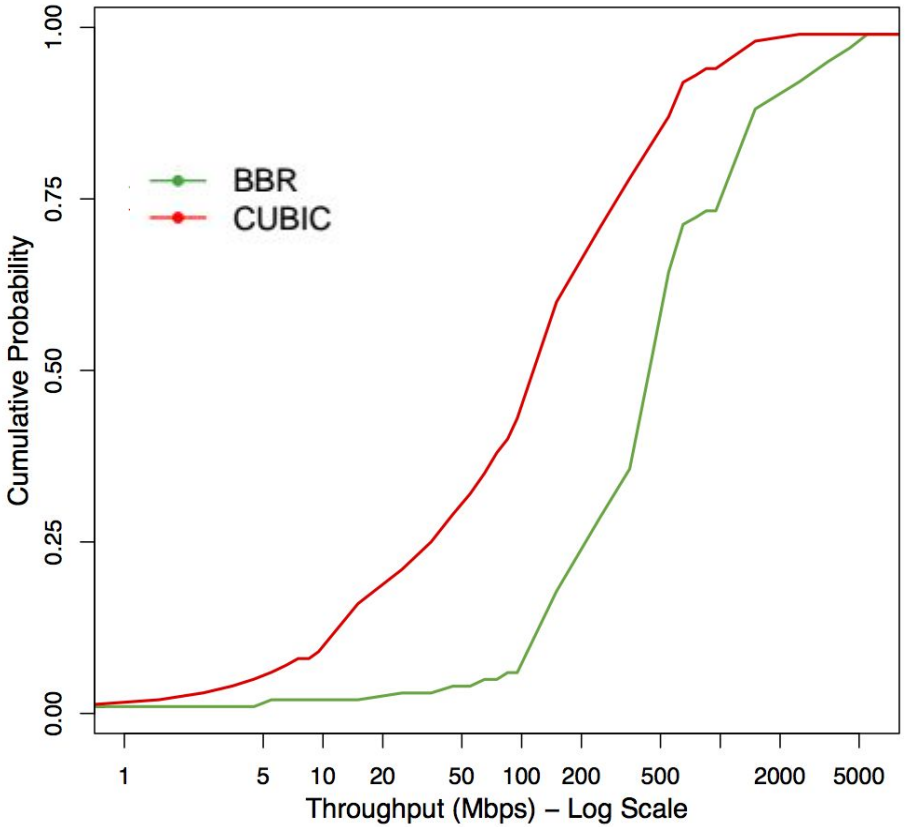
BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms

Low queue delay, despite bloated buffers



BBR vs CUBIC: synthetic bulk TCP test with 8 flows, bottleneck_bw=128kbps, RTT=40ms

BBR is 2-20x faster on Google WAN



- BBR used for all TCP on Google B4
- Most BBR flows so far rwin-limited
  - max RWIN here was 8MB (tcp_rmem[2])
  - 10 Gbps x 100ms = 125MB BDP
- after lifting rwin limit:
  - BBR 133x faster than CUBIC

Conclusion

Algorithms and architecture in Linux TCP have evolved

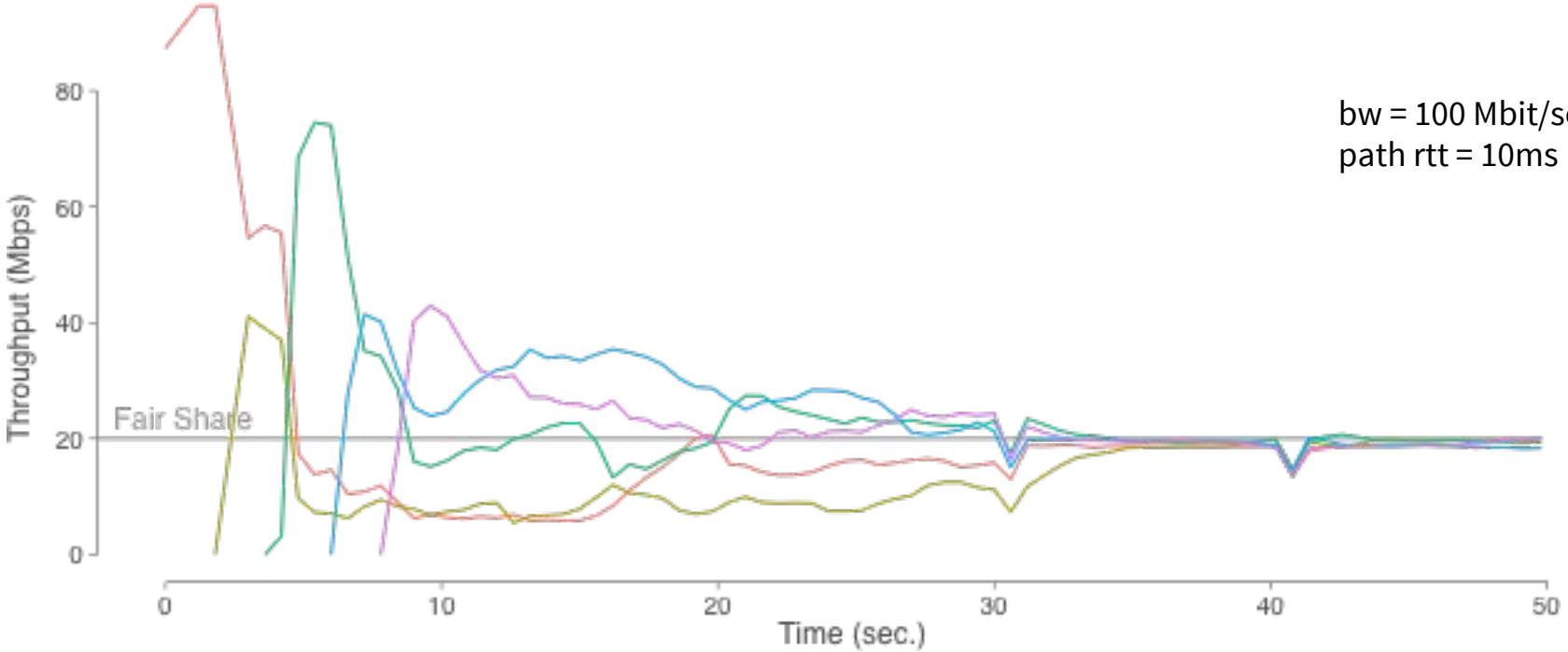
- Maximizing BW, minimizing queue, and one-RTT recovery (BBR, RACK)
- Based on groundwork of a high-performance packet scheduler (fq/pacing/tsq/tso-autosizing)
- Orders of magnitude higher bandwidth and lower latency

Next: Google, YouTube, and... the Internet?

- Help us make them better! https://groups.google.com/forum/#!forum/bbr-dev

Backup slides...

BBR convergence dynamics



Converge by sync'd PROBE_RTT + randomized cycling phases in PROBE_BW

- Queue (RTT) reduction is observed by every (active) flow
- Elephants yield more (multiplicative decrease) to let mice grow