

BBR Congestion Control

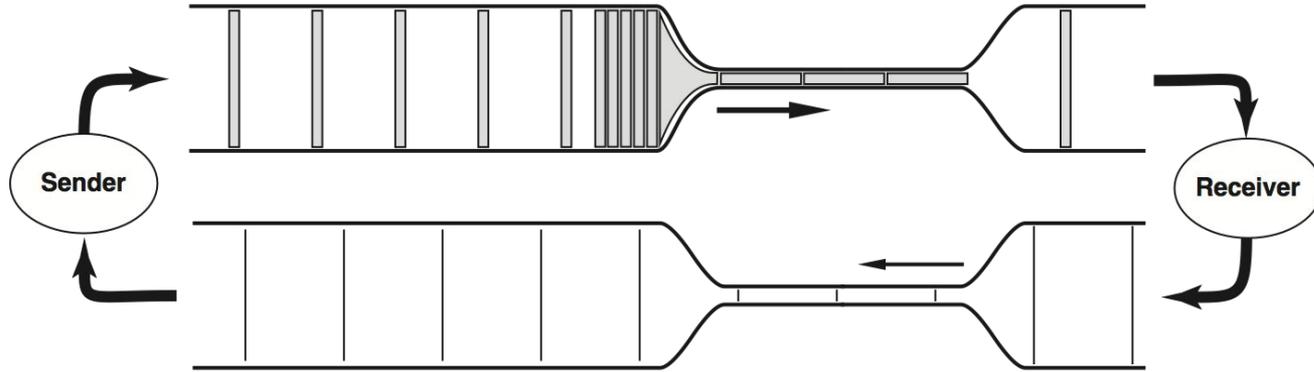
Neal Cardwell, Yuchung Cheng,

C. Stephen Gunn, Soheil Hassas Yeganeh,

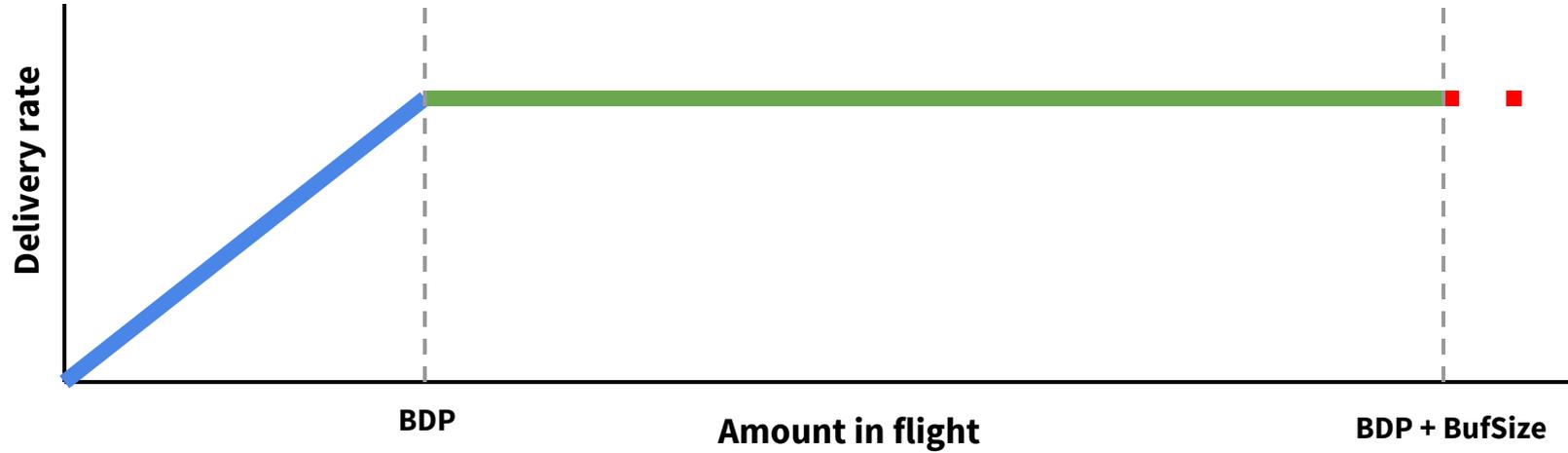
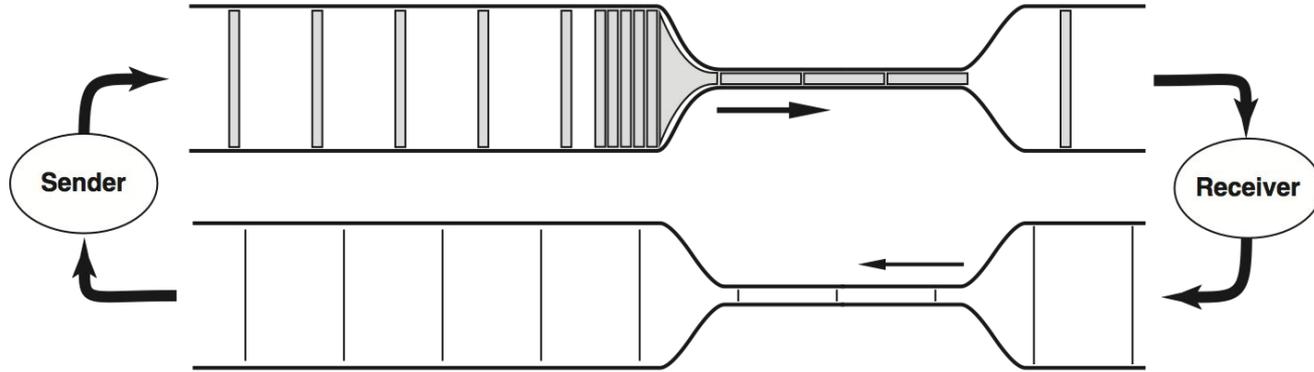
Van Jacobson

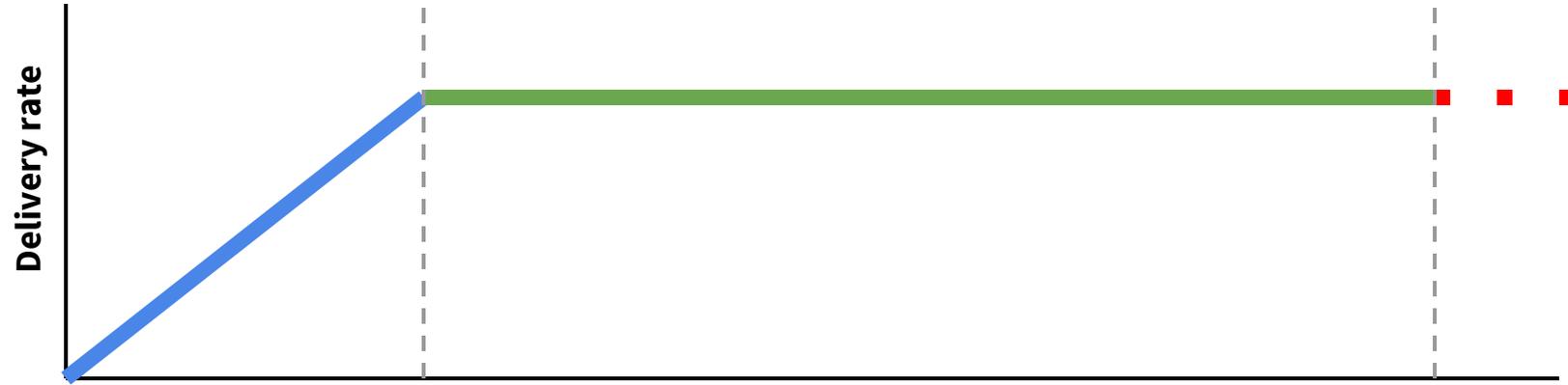
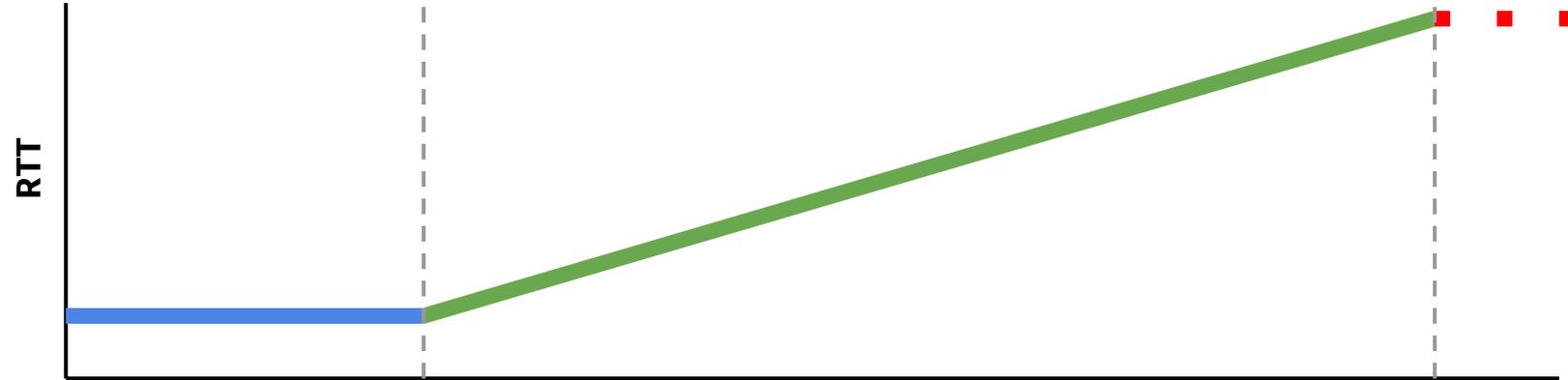


Congestion and bottlenecks



Congestion and bottlenecks



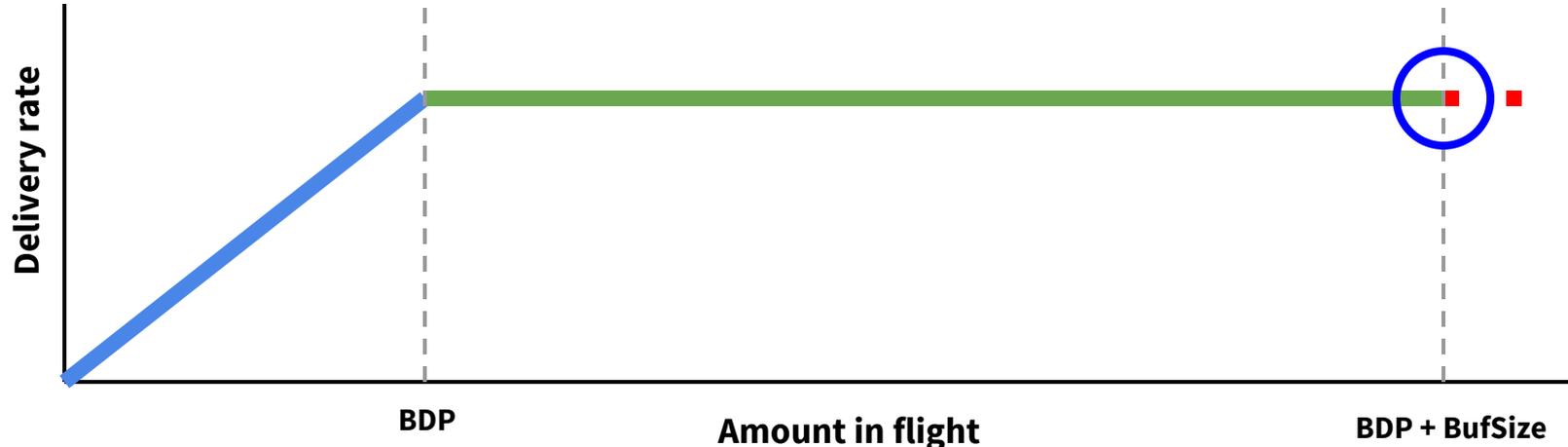
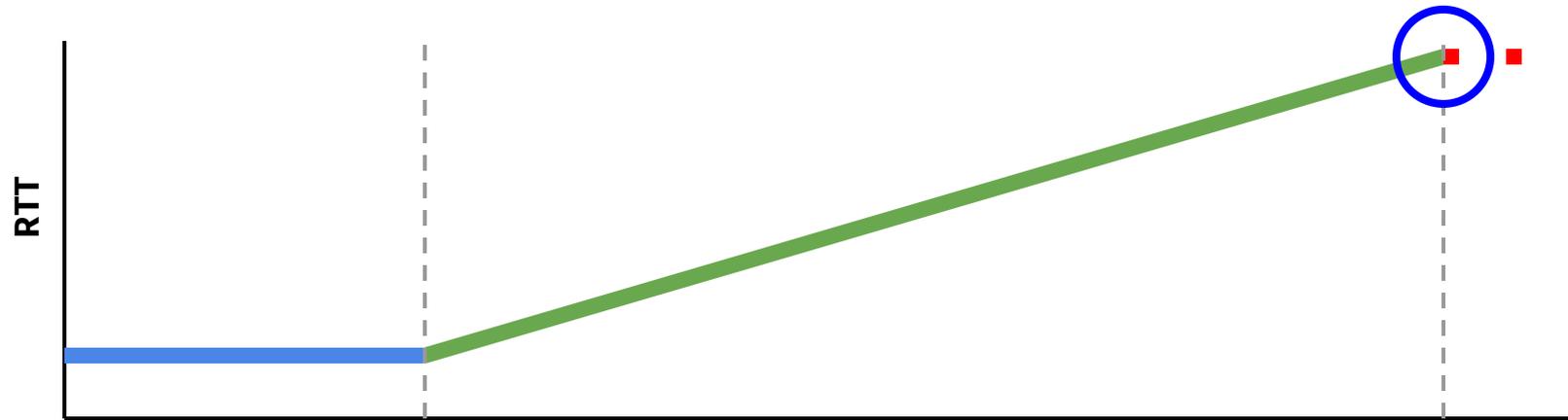


BDP

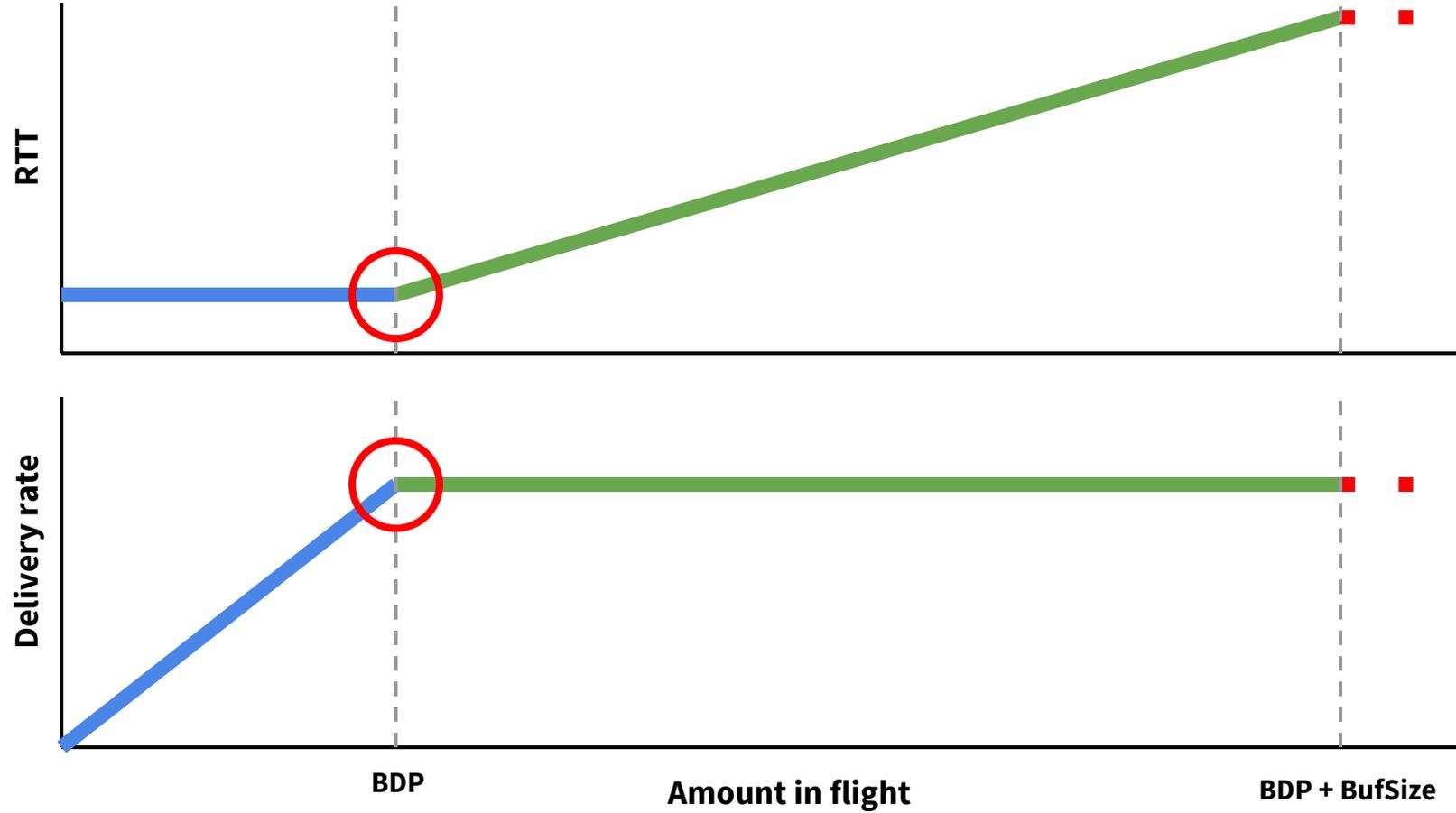
Amount in flight

BDP + BufSize

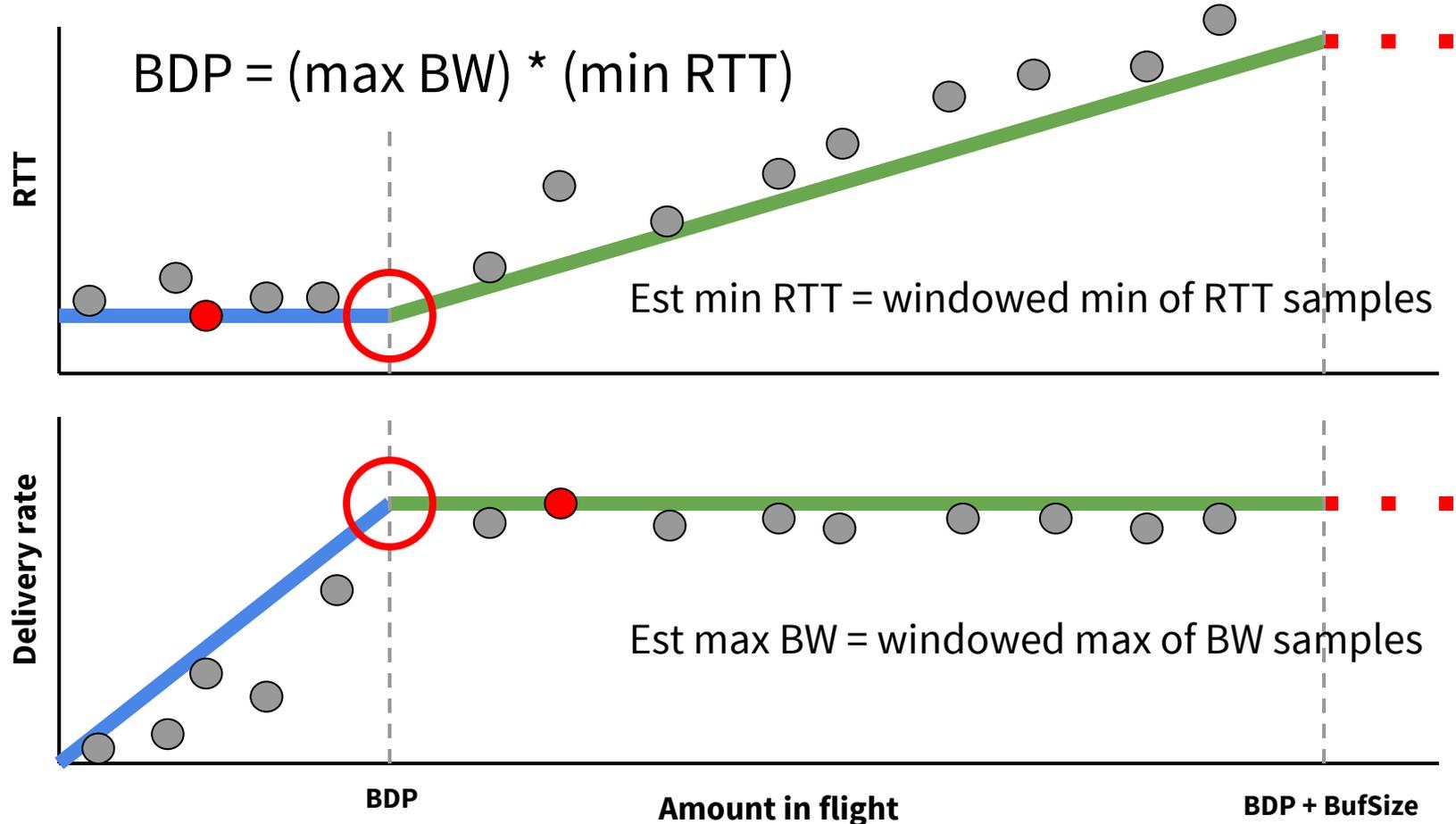
○ CUBIC / Reno



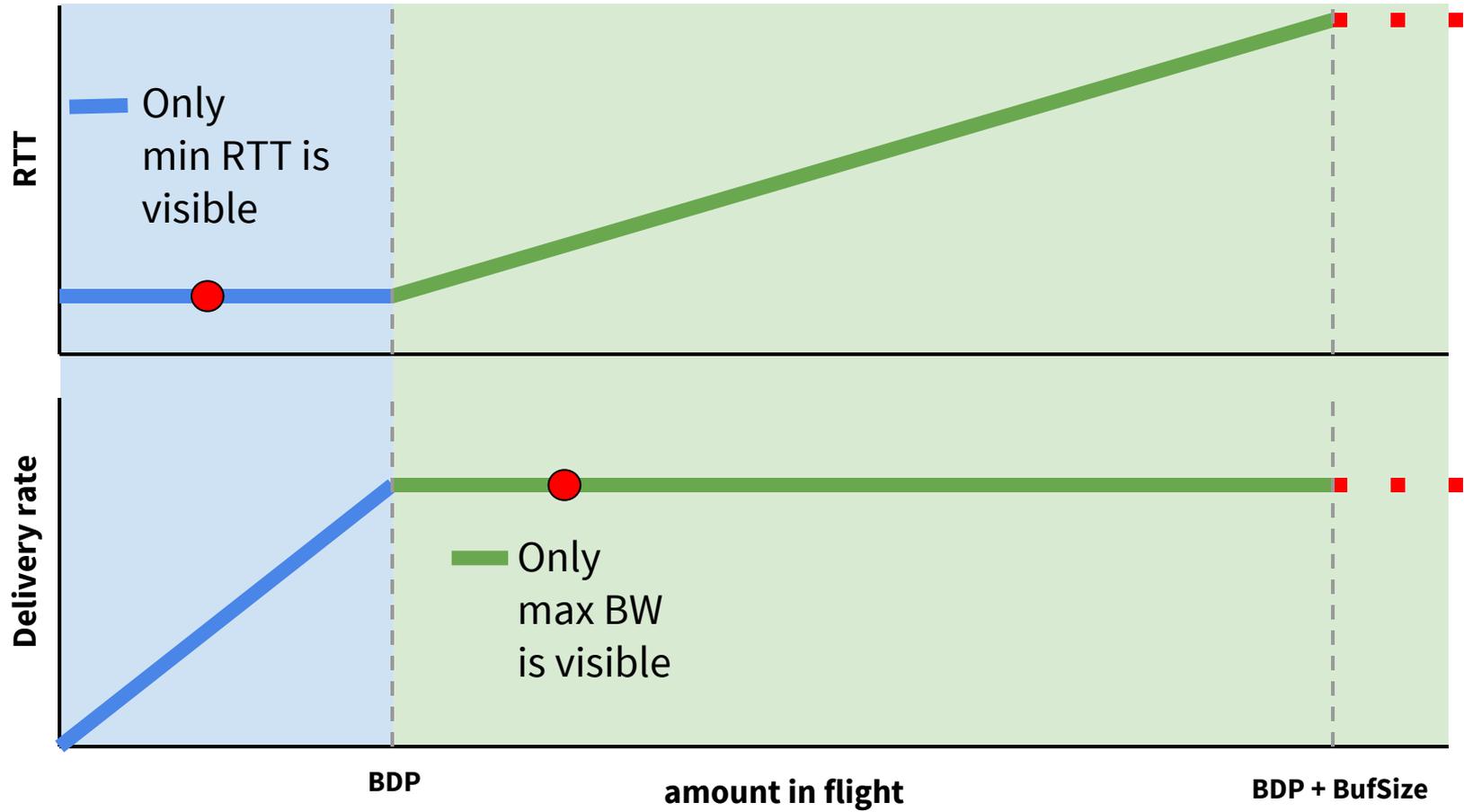
○ Optimal: max BW and min RTT (Gail & Kleinrock. 1981)



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 [move queue to sender]

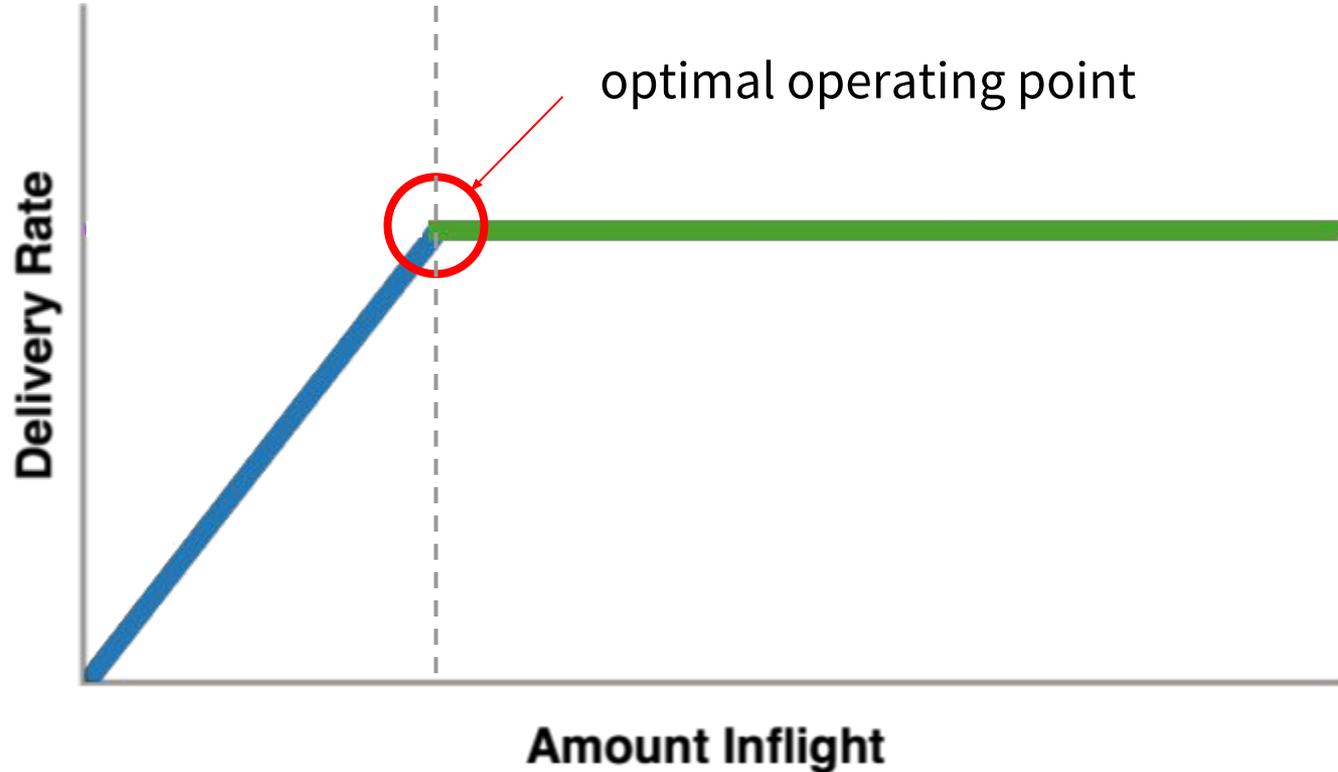
Vary pacing rate to keep inflight near BDP (for full pipe but small queue)

That's **BBR** congestion control:

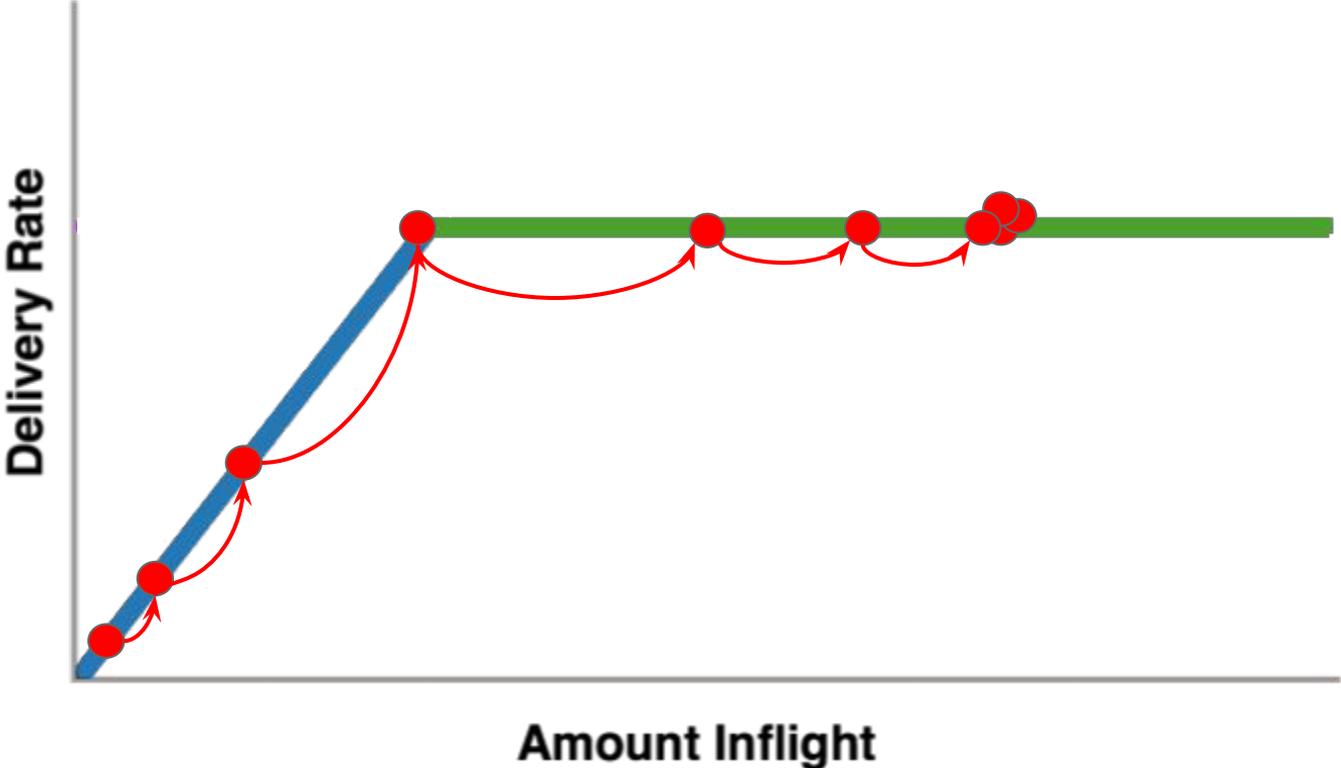
BBR = **B**ottleneck **B**andwidth and **R**ound-trip propagation time

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

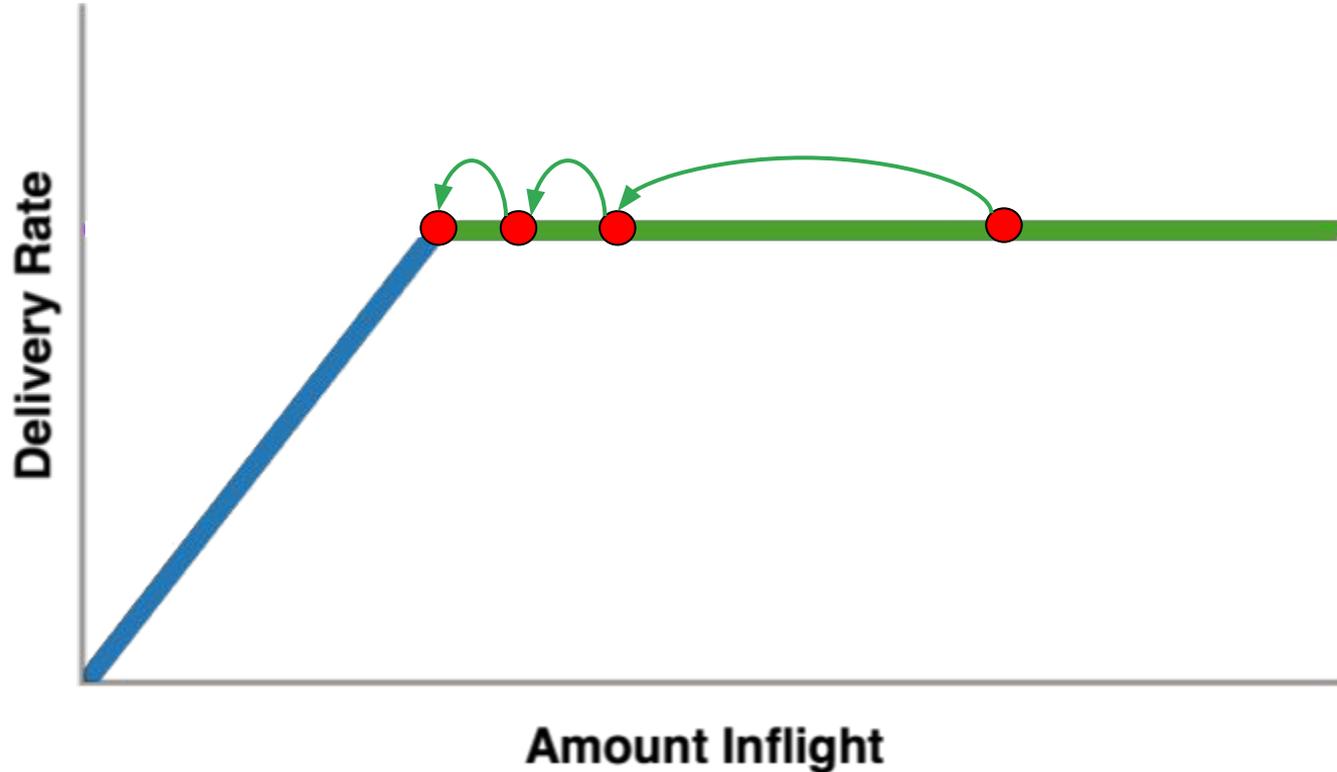
BBR: model-based walk toward max BW, min RTT



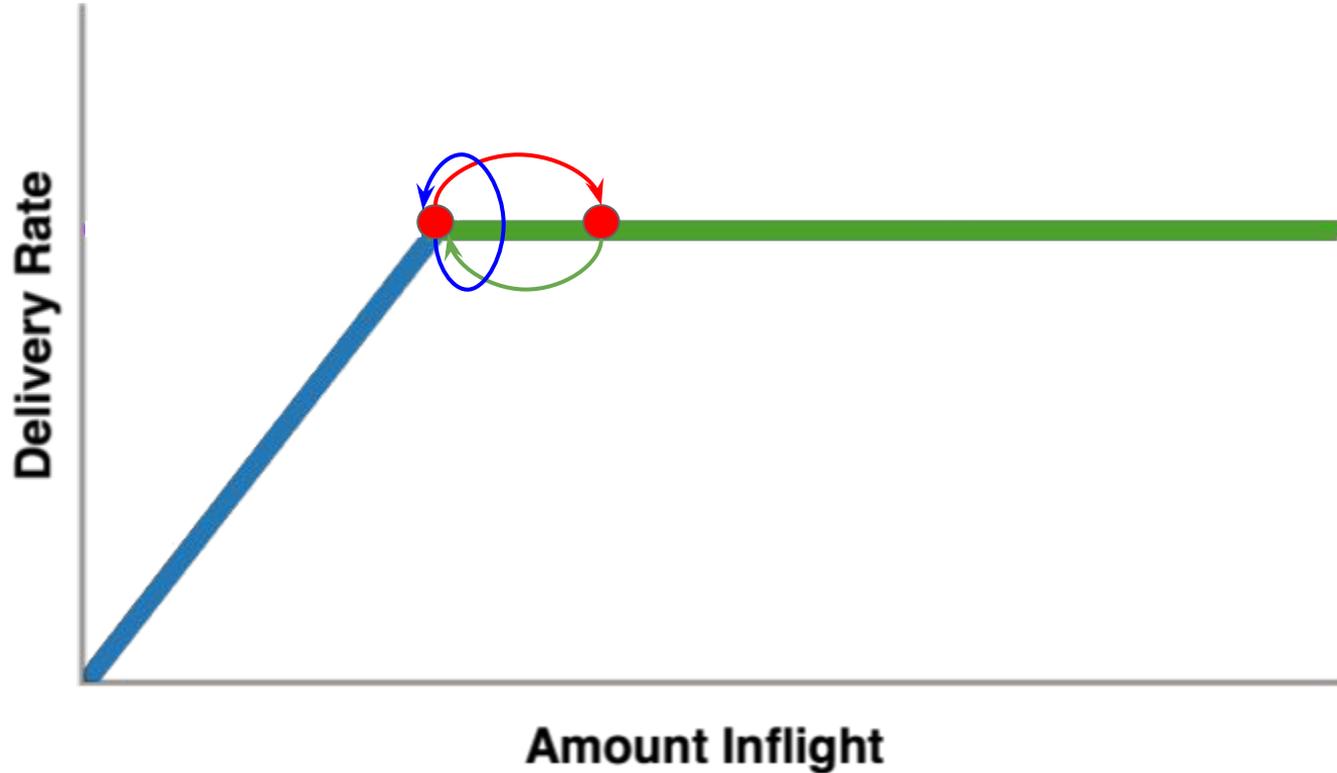
STARTUP: exponential BW search



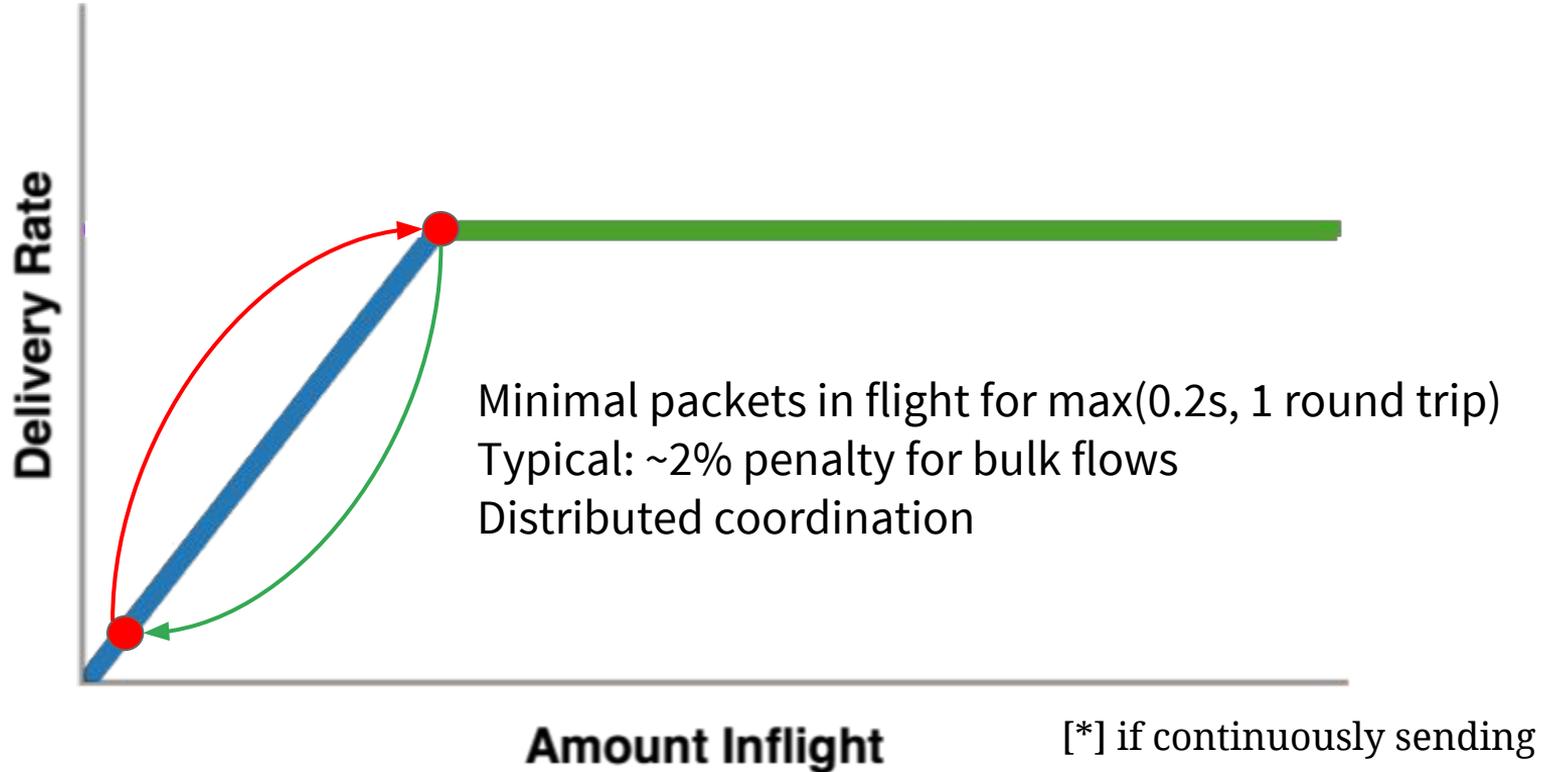
DRAIN: drain the queue created during startup



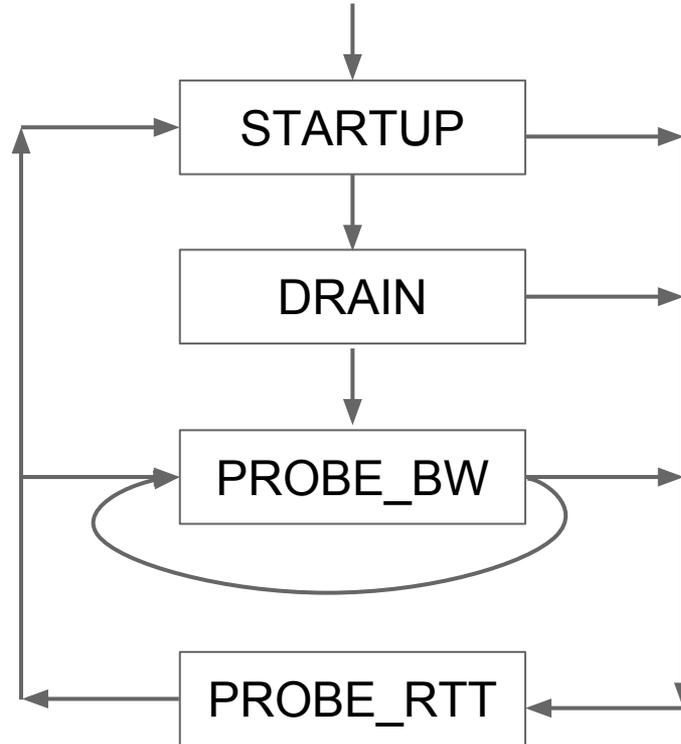
PROBE_BW: explore max BW, drain queue, cruise



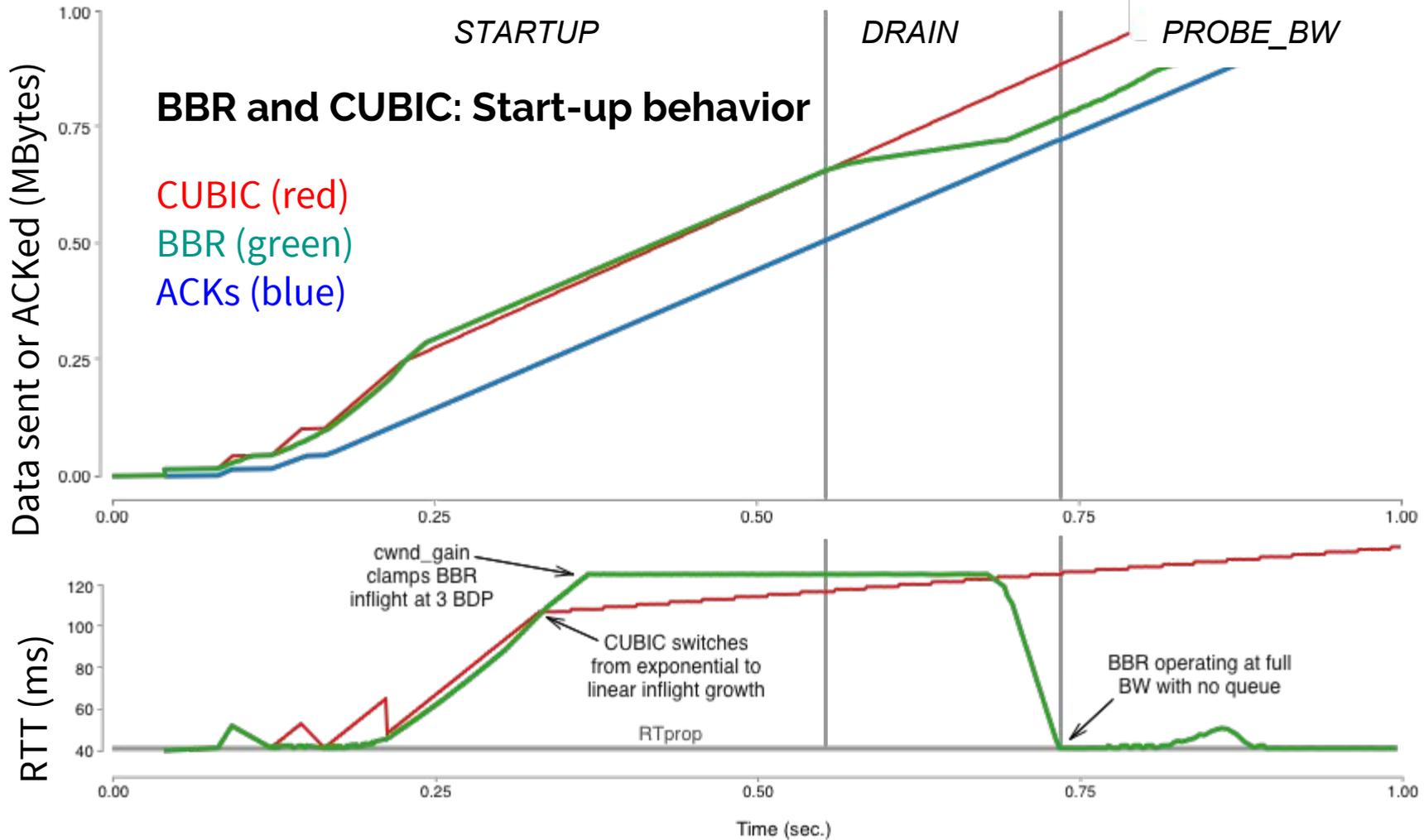
PROBE_RTT briefly if min RTT filter expires (=10s)*



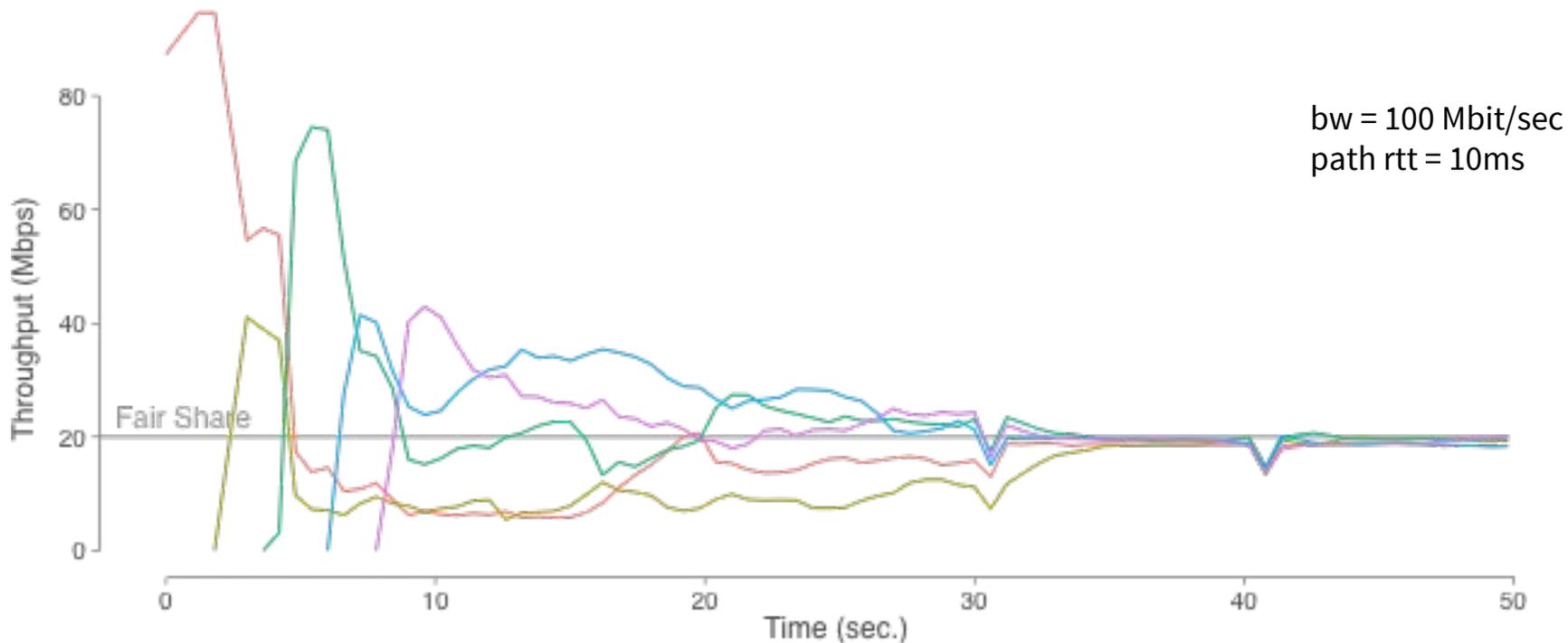
BBR state transition diagram



Performance results



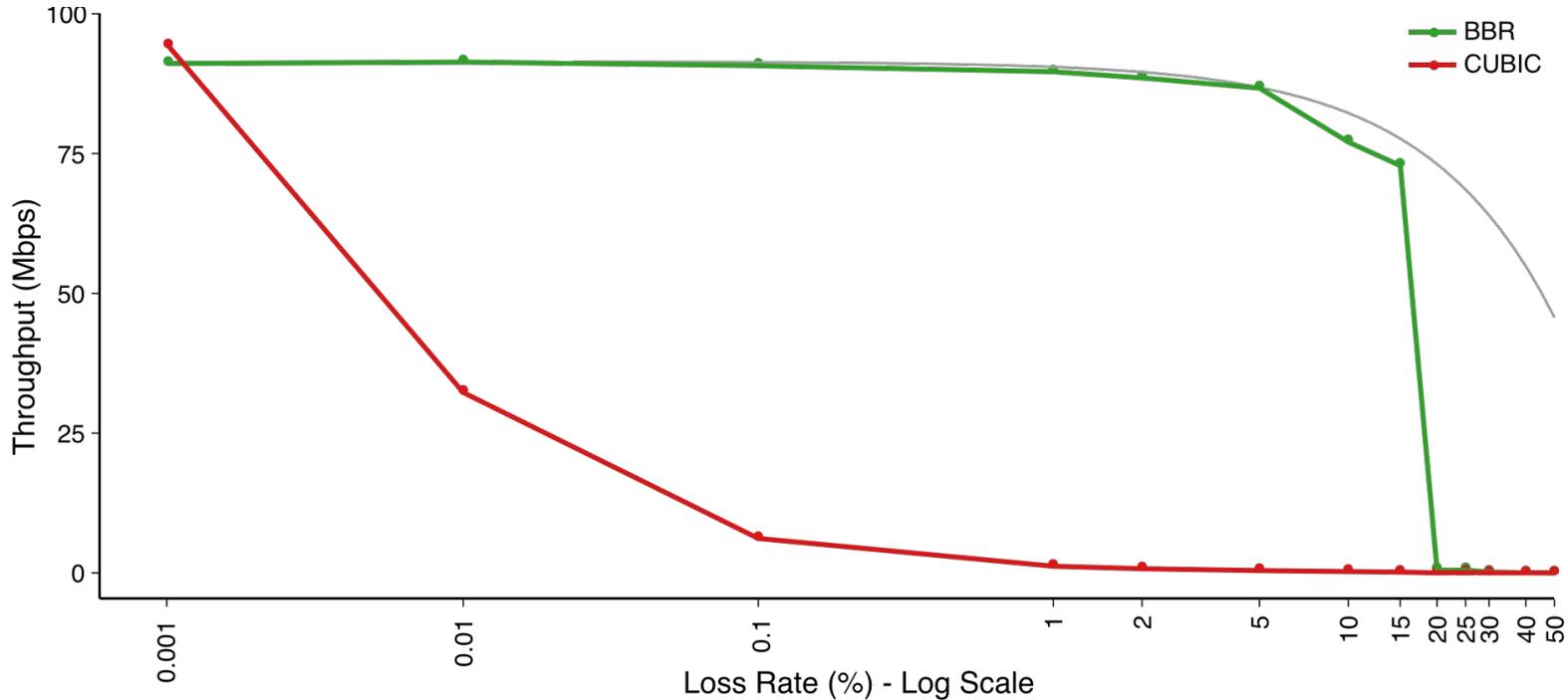
BBR multi-flow convergence dynamics



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

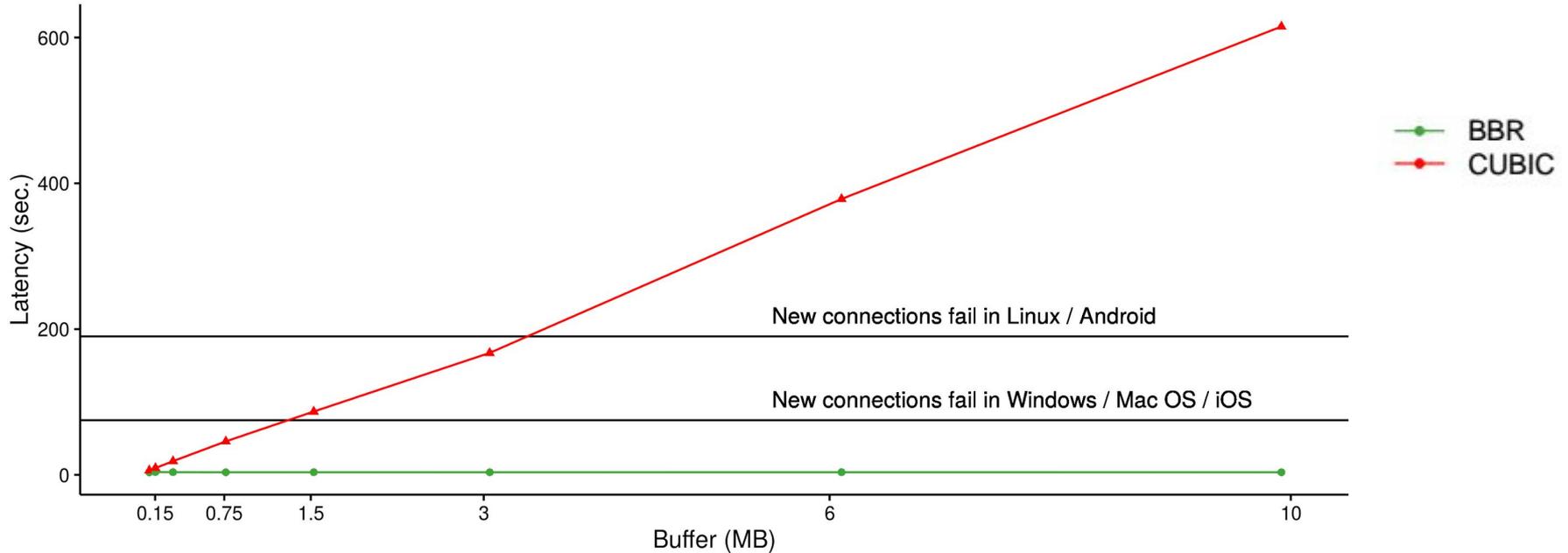
- Queue (RTT) reduction is observed by every (active) flow
- Elephants yield more (multiplicative decrease) to let mice grow: each flow learns its fair share

Fully use bandwidth, despite high loss

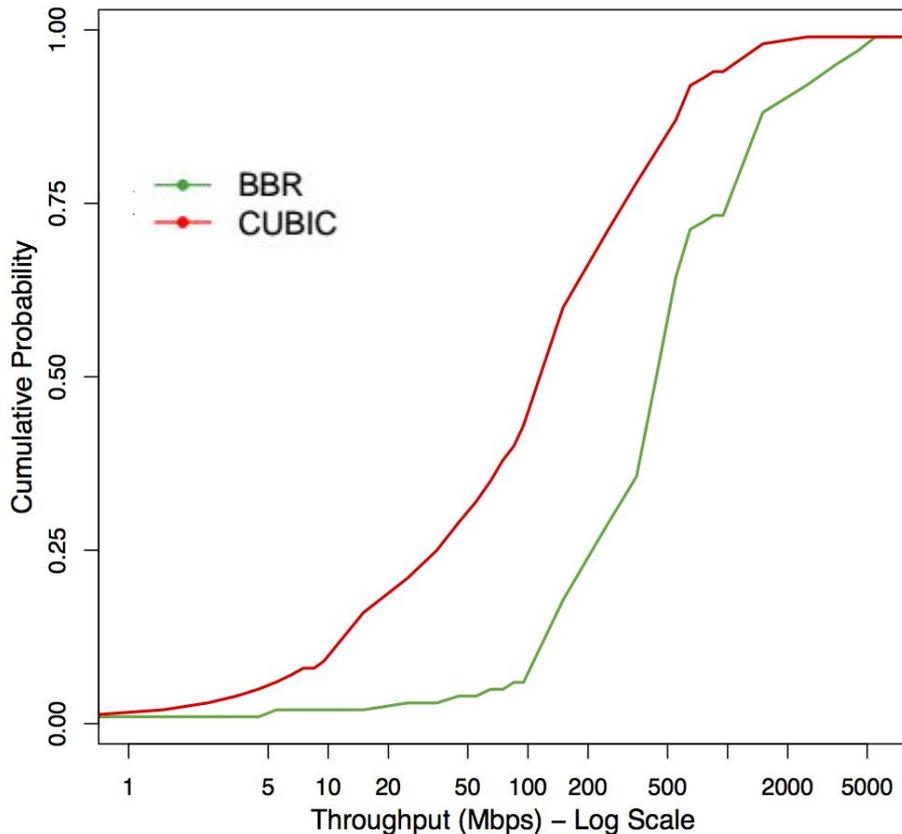


BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms

Low queue delay, despite bloated buffers



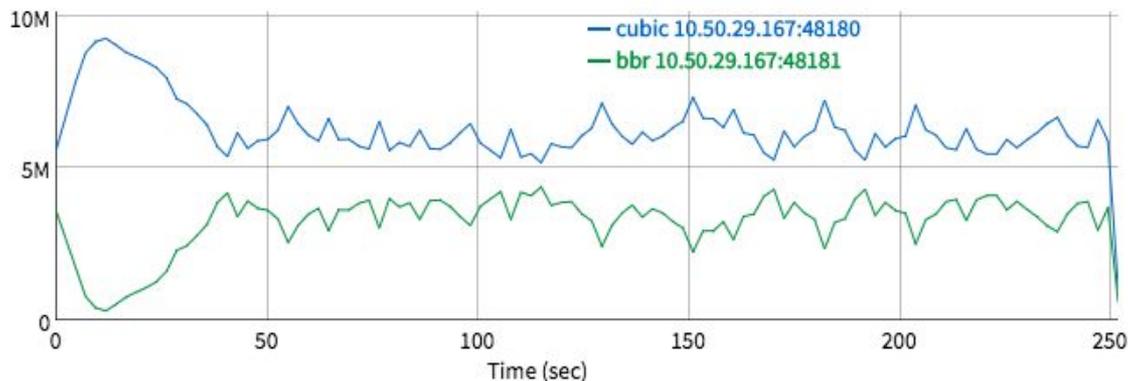
BBR is 2-20x faster on Google WAN



- BBR used for all TCP on Google B4
- Most BBR flows so far rwin-limited
 - max RWIN here was 8MB
 - $10 \text{ Gbps} \times 100\text{ms} = 125\text{MB BDP}$
- after lifting rwin limit:
 - BBR 133x faster than CUBIC

Deep dives & implementation

Sharing deep buffers with loss-based CC



At first CUBIC/Reno gains an advantage by filling deep buffers

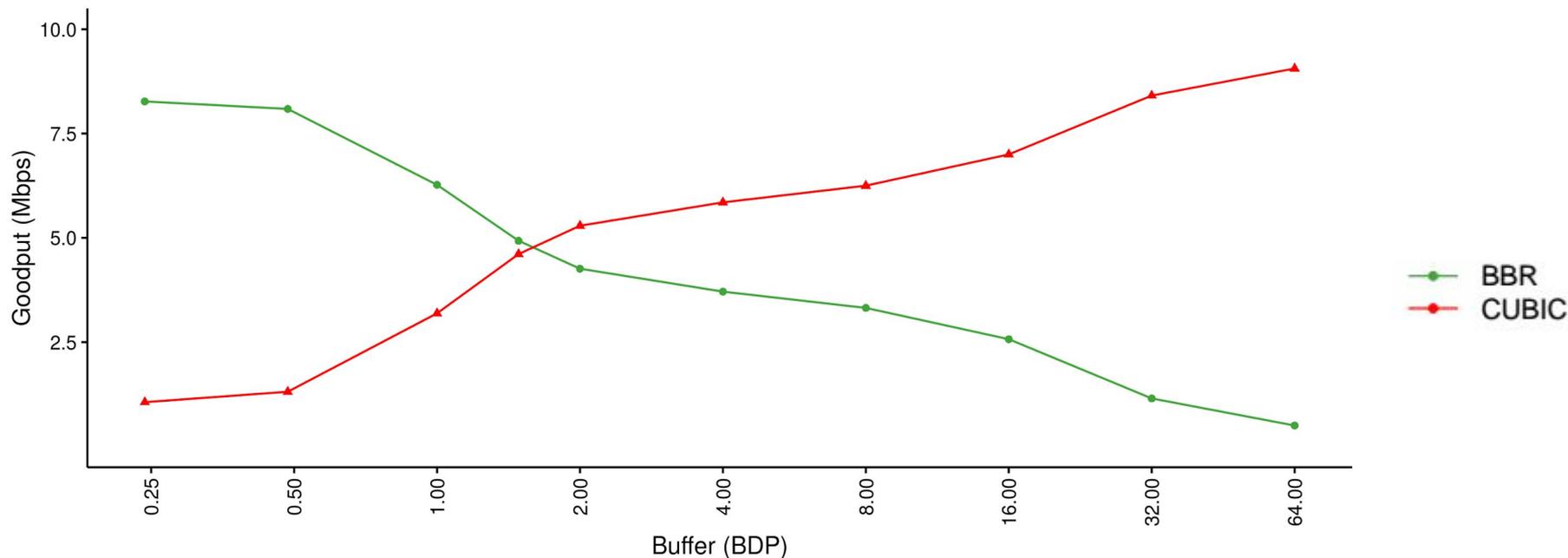
But BBR does not collapse; it adapts: BBR's bw and RTT probing tends to drive system toward fairness

Deep buffer data point: 8*BDP case: bw = 10Mbps, RTT = 40ms, buffer = 8 * BDP

-> CUBIC: 6.31 Mbps vs BBR: 3.26 Mbps

Current dynamics w/ with loss-based CC

CUBIC vs BBR bw: bw = 10Mbps, RTT = 40ms, 4 min. bulk xfer, varying buffer sizes

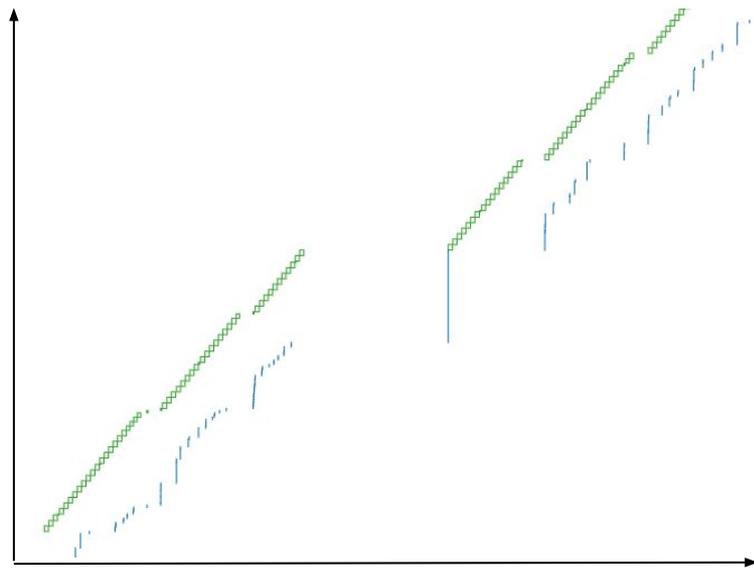


Top priority: reducing queue usage

- Current active work for BBR
- Motivation:
 - Further reduce delay and packet loss
 - Better fairness w/ loss-based CC in shallow buffers
 - Better fairness w/ higher-RTT BBR flows
 - Lower tail latency for cross-traffic
- Mechanisms:
 - Draining queue more often
 - Drain inflight down to BDP each gain cycle
 - Estimate available buffer; modulate probing magnitude/frequency
 - In shallow buffers, BBR bw probing makes loss-based CC back off

Common real-world issues

- ACK compression
 - One TCP ACK for +200 packets
 - Particularly wireless & cable networks
 - Cap inflight $\leq 2 \cdot \text{BDP}$
- Application idles
 - Paces at BW restarting from idle
- Inappropriate receive window
 - Linux default 3MB \Rightarrow 240Mbps on 100ms RTT
- Token-bucket traffic policers
 - Explicit model policers
 - Details presented in maprg



Implementation status

- [Linux v4.9 TCP](#)
 - A congestion control module in GPL/BSD dual licence
 - Requires fq/pacing qdisc (BBR needs pacing support)
 - Fully deployed for WAN between Google datacenters
 - Being deployed on Google.com and YouTube
- [QUIC](#) implementation under way
 - {vasilvv,ianswett,jri}@google.com
- FreeBSD implementation under way
 - rrs@netflix.com

BBR FAQ

Is BBR fair to Cubic/Reno?	Buffer $\geq 1.5 \cdot \text{BDP}$: Yes; Else: WIP
Is BBR $1/\sqrt{p}$?	No
Is BBR {delay loss ECN AIMD}-based?	No, No, No, No
Is BBR ack-clocked?	No
Does BBR require pacing?	Yes
Does BBR require FQ scheduler?	No, but it helps
Does BBR require receiver or network changes	No
Does BBR improve latency on short flows?	Yes, if there is packet loss

Conclusion

BBR: model-based congestion control

- Goal is to maximize bandwidth then minimize queue
- Orders of magnitude higher bandwidth and lower latency

BBR will continue to evolve as we gain more experience

- Help us make it better! <https://groups.google.com/forum/#!forum/bbr-dev>
- "[BBR: Congestion-based Congestion Control](#)", ACM Queue, Oct 2016
Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson

Backup slides...

BBR: control logic details

Controls sending based on the model, to move toward network's best operating point:

- send rate near available bandwidth (primary):
 - Pacing rate = $\text{pacing_gain} * \text{BtlBw}$
- volume of data in flight near BDP (secondary):
 - Max inflight = $\text{cwnd_gain} * \text{BDP} = \text{cwnd_gain} * \text{BtlBw} * \text{RTprop}$

BBR: state machine details

STARTUP: exponential growth to quickly fill pipe (like slow-start)

- stop growth when bw estimate plateaus, not on loss or delay (Hystart)
- `pacing_gain = 2.89, cwnd_gain = 2.89`

DRAIN: drain the queue created in *STARTUP*

- `pacing_gain = 0.35 = 1/2.89, cwnd_gain = 2.89`

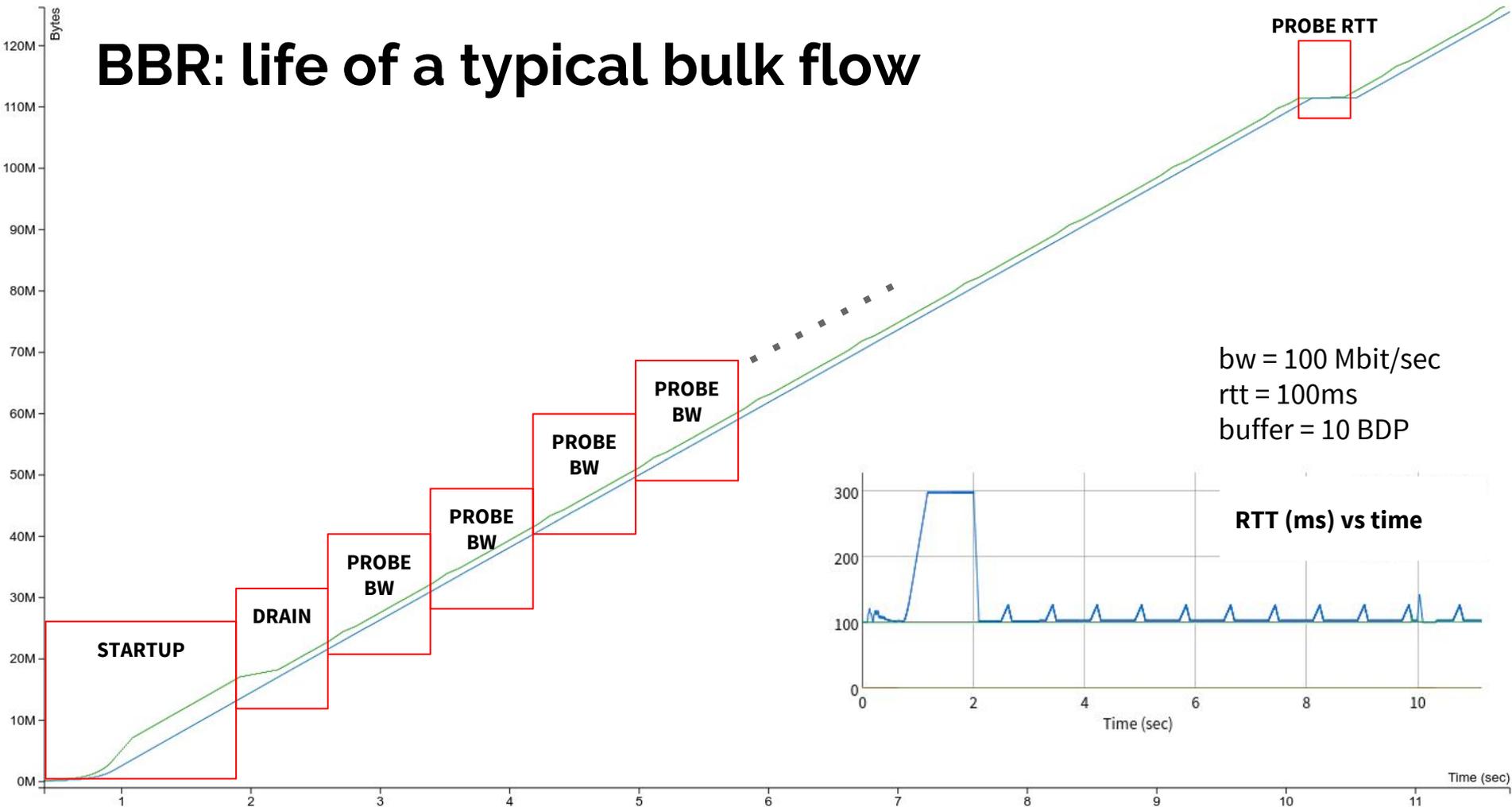
PROBE_BW: cycle `pacing_gain` to explore and fairly share bandwidth (`cwnd_gain = 2` in all phases):

- [`1.25, 0.75, 1, 1, 1, 1, 1, 1`] (1 phase per min RTT)
- `Pacing_gain = 1.25` => probe for more bw
- `Pacing_gain = 0.75` => drain queue and yield bw to other flows
- `Pacing_gain = 1.0` => cruise with full utilization and low, bounded queue

PROBE_RTT: if needed, occasionally send slower to probe min RTT

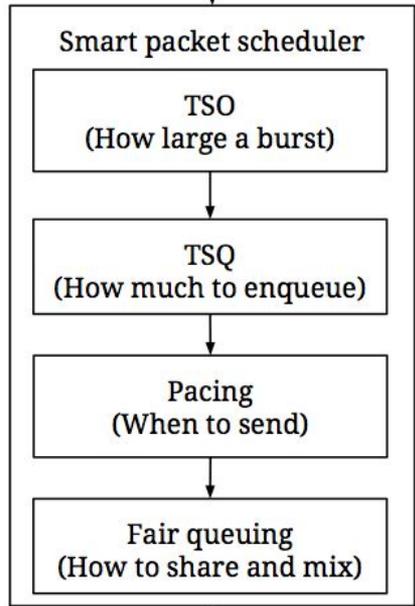
- Maintain inflight of 4 for at least $\max(1 \text{ round trip}, 0.2 \text{ sec})$; `pacing_gain = 1.0`

BBR: life of a typical bulk flow



ACK processing,
loss detection
(What to send next)

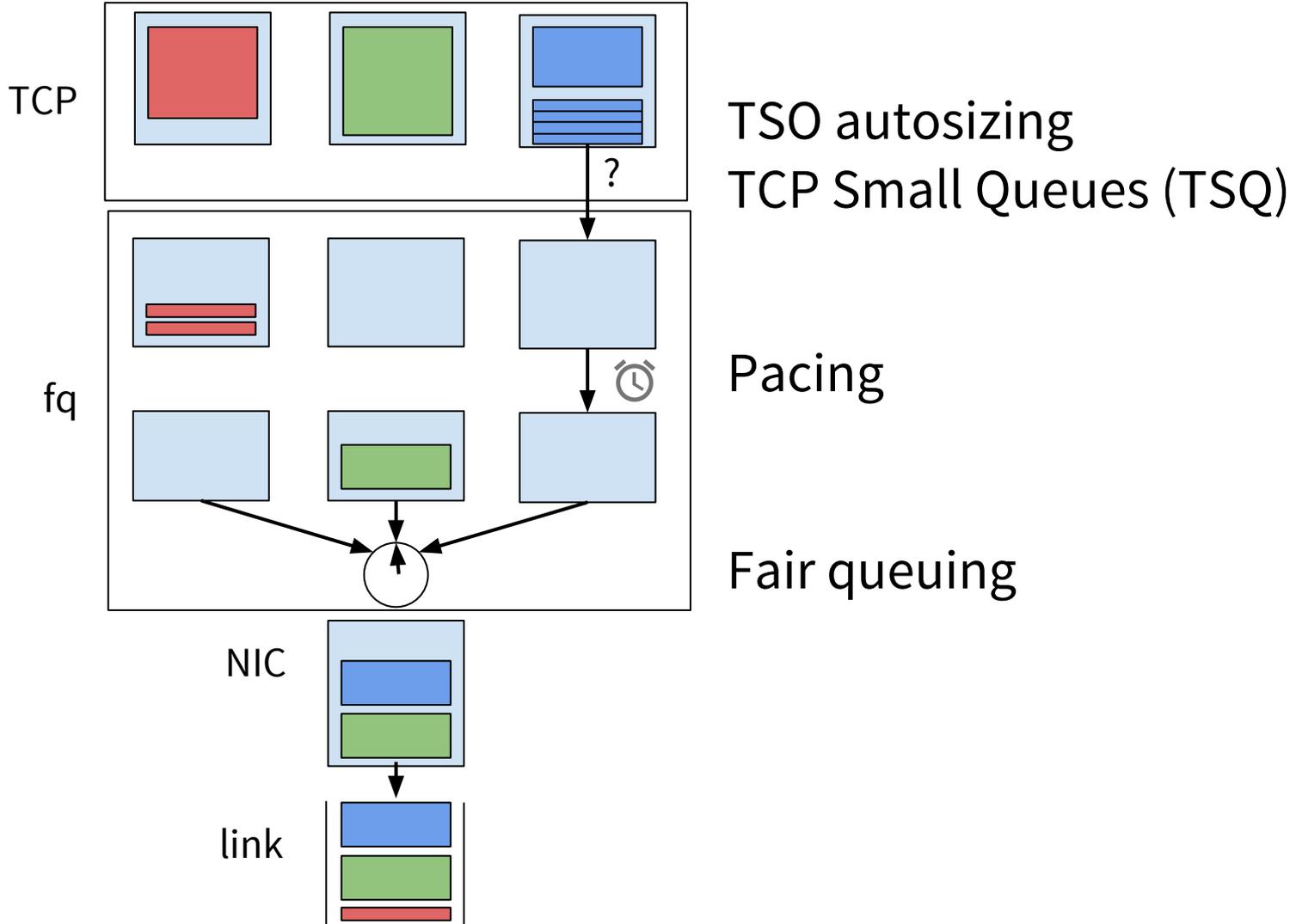
Congestion control
(How fast to send)



NIC

How BBR Fits into Transport Stacks

- ACK processing, loss detection - What to send
- Congestion control - How fast to send
- Smart packet scheduler - When to send



Fully use bandwidth, despite high loss

