BBR Congestion Control

Neal Cardwell, Yuchung Cheng,

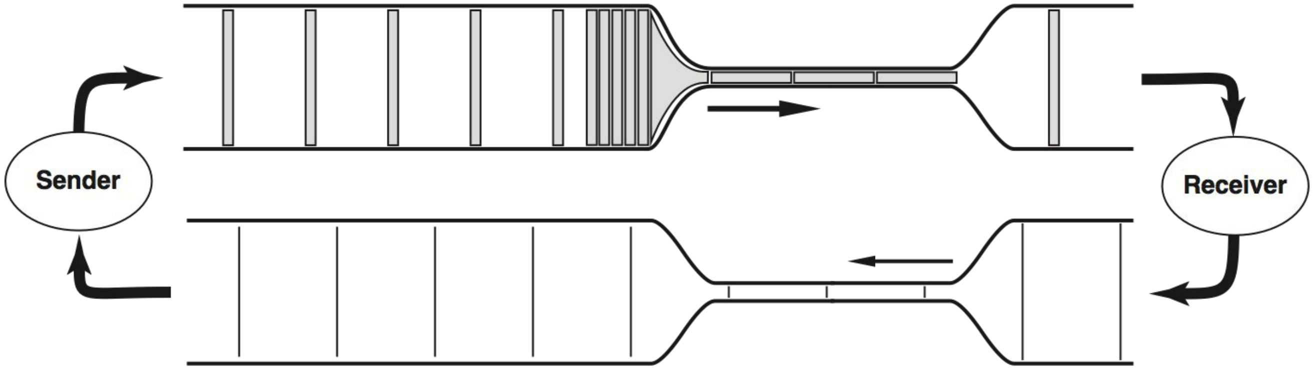
C. Stephen Gunn, Soheil Hassas Yeganeh,

Van Jacobson

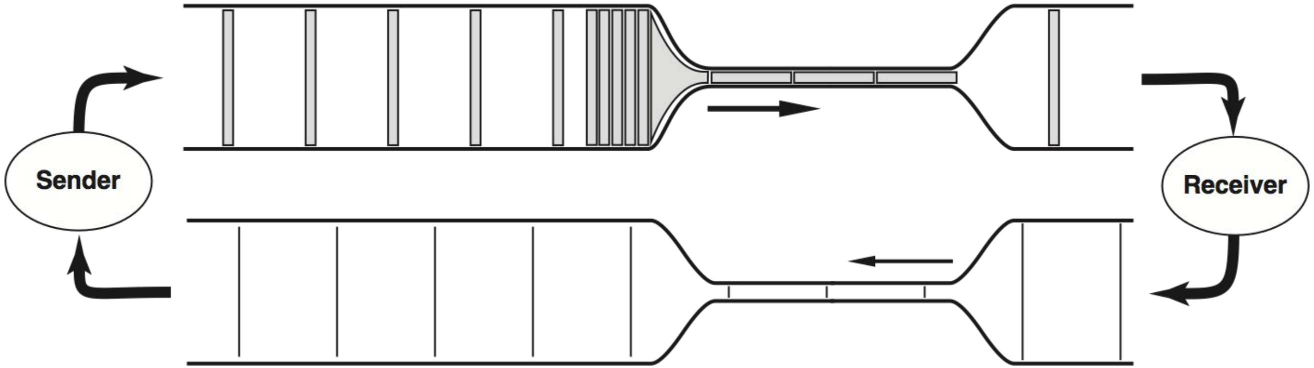
Google

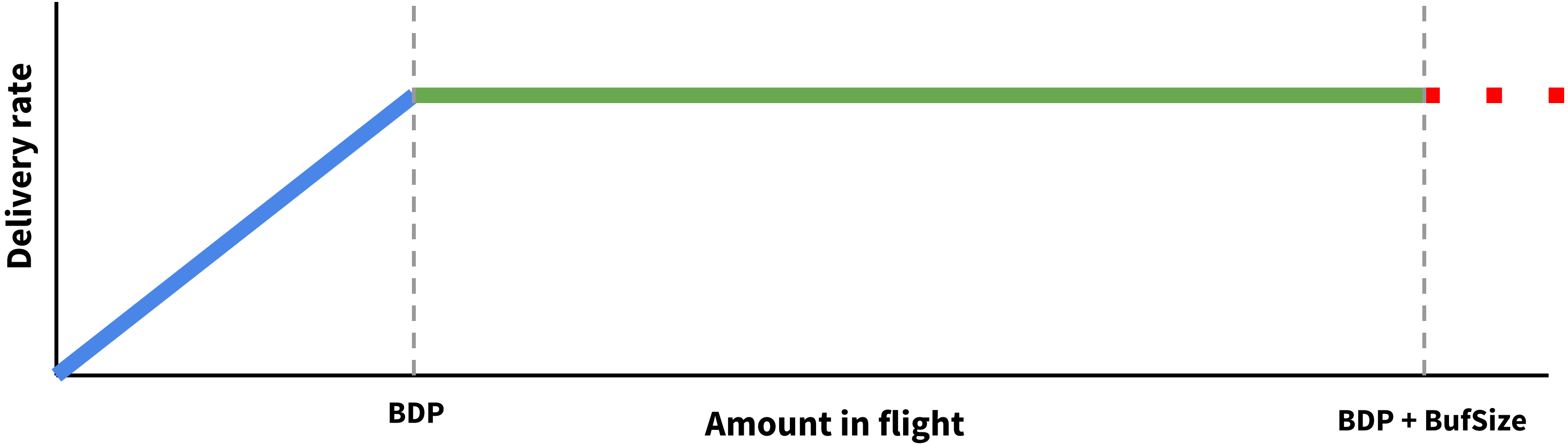
IETF 97: Seoul, Nov 2016

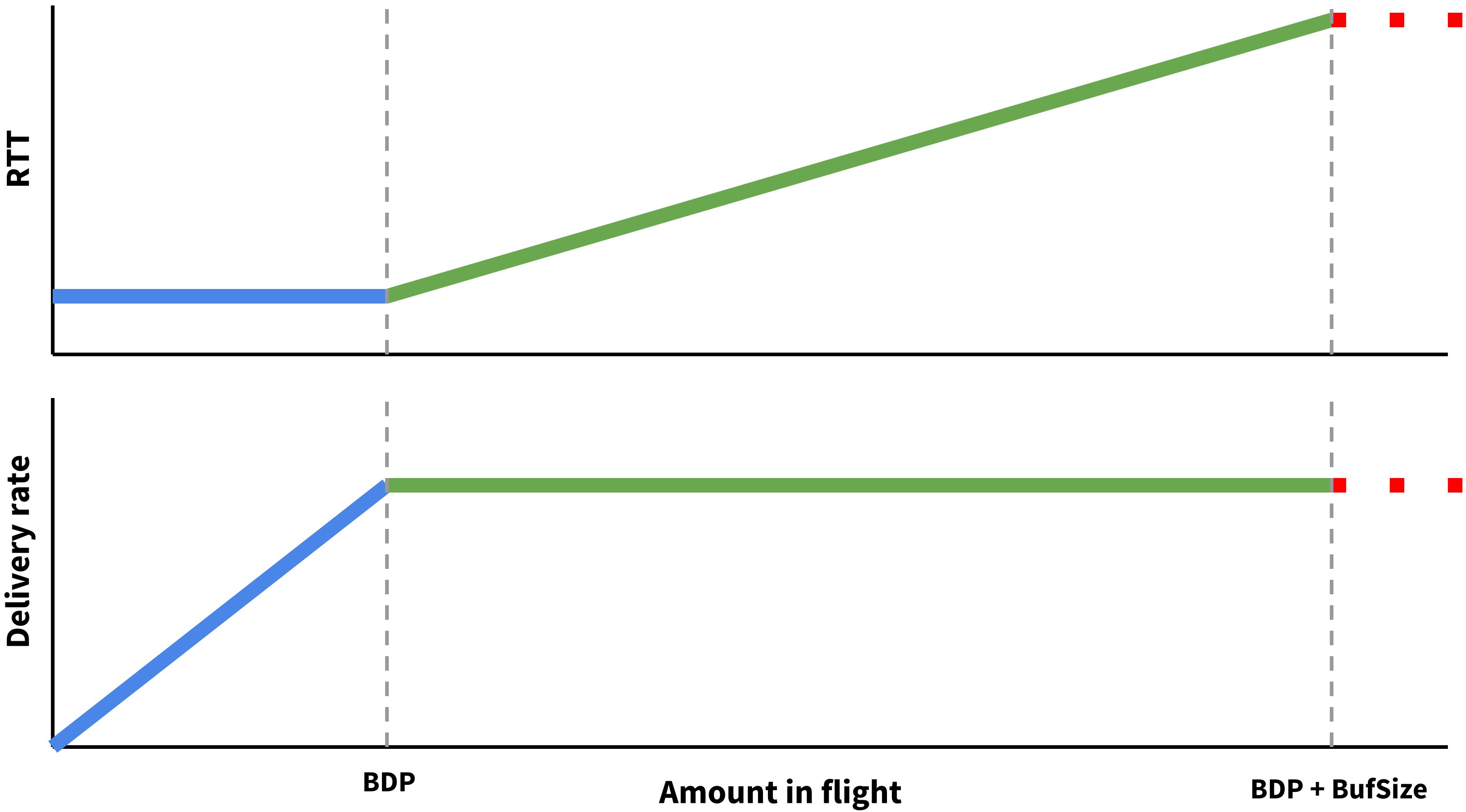
Congestion and bottlenecks

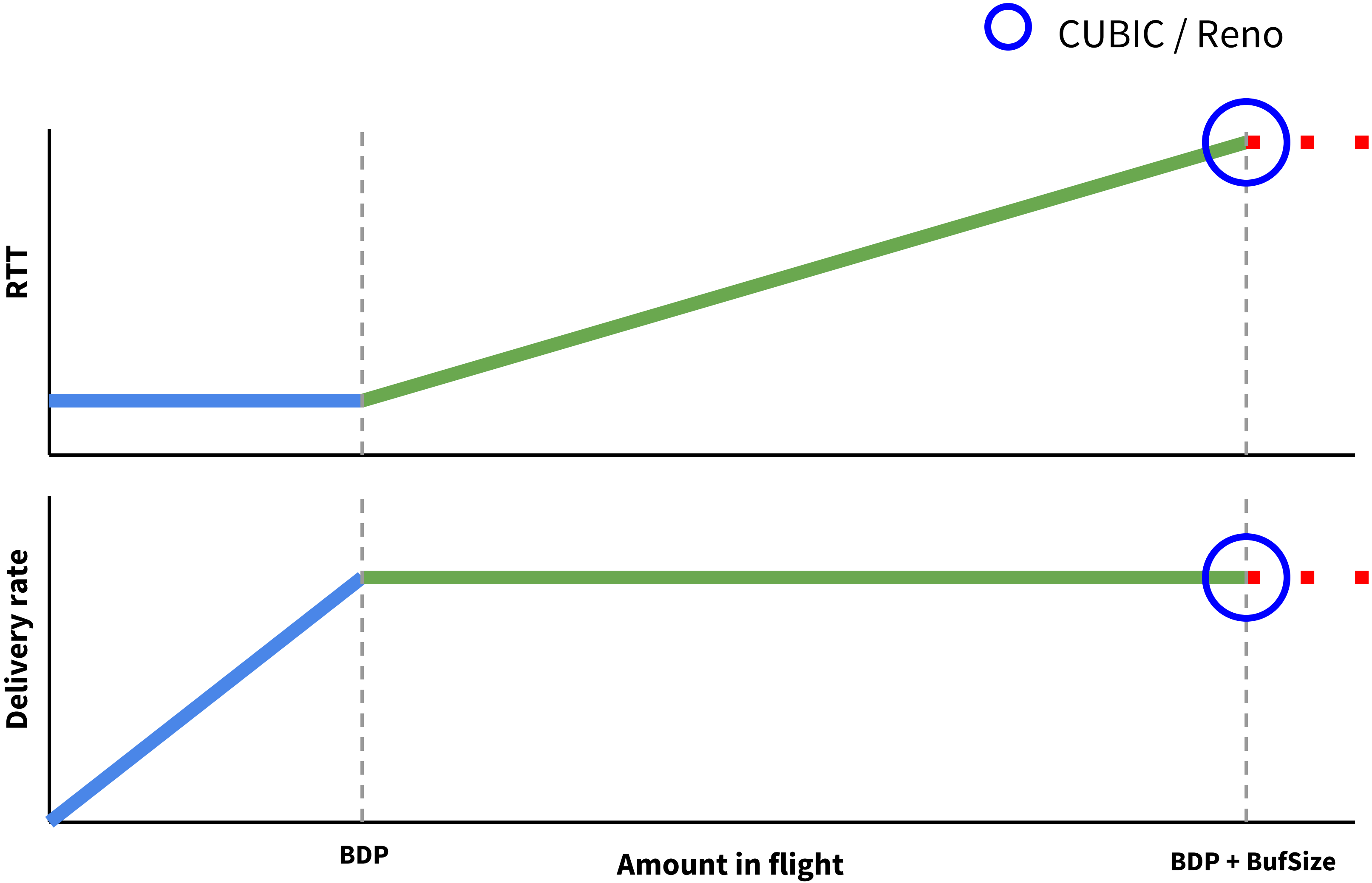


Congestion and bottlenecks



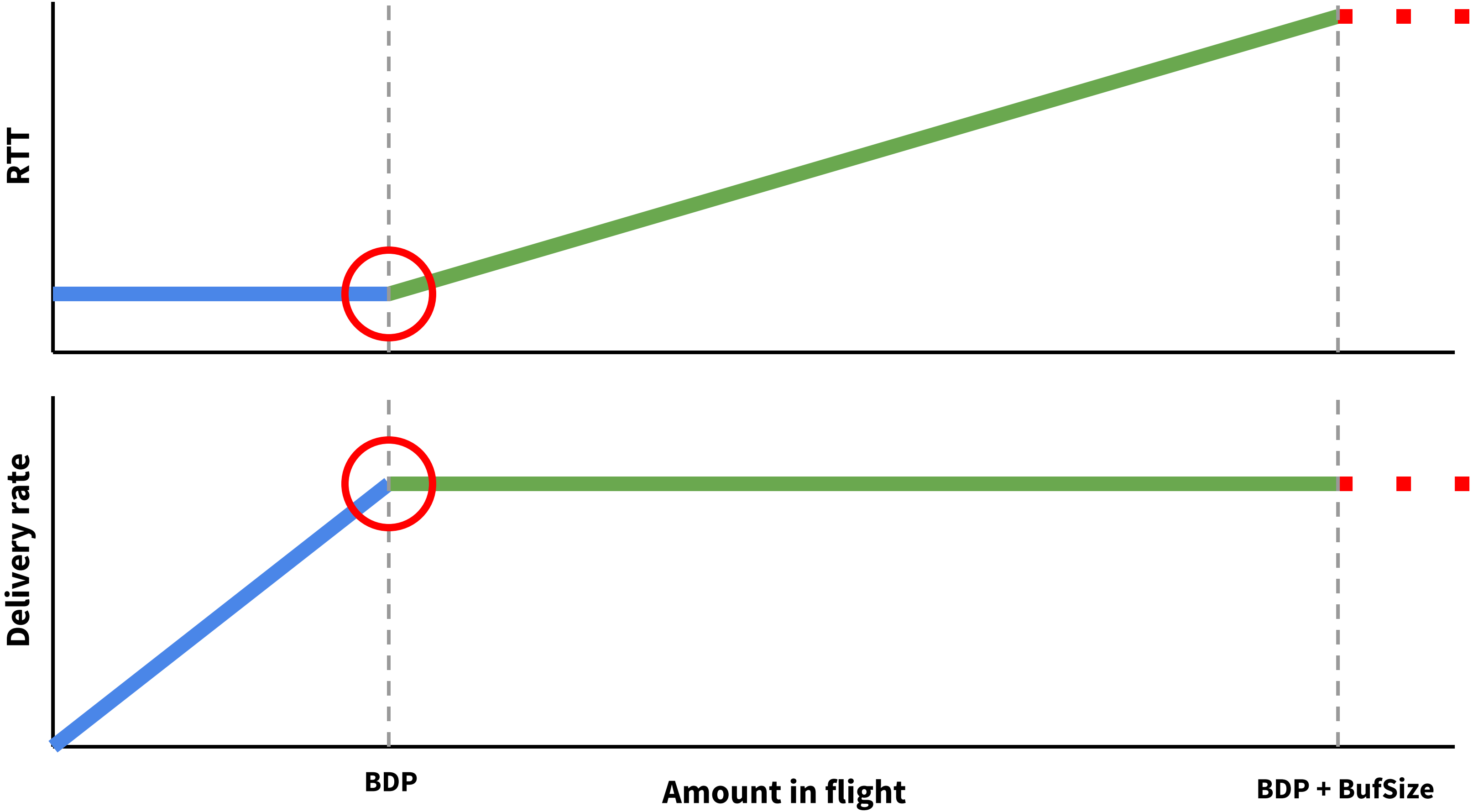



RTT
Delivery rate
BDP
Amount in flight
BDP + BufSize

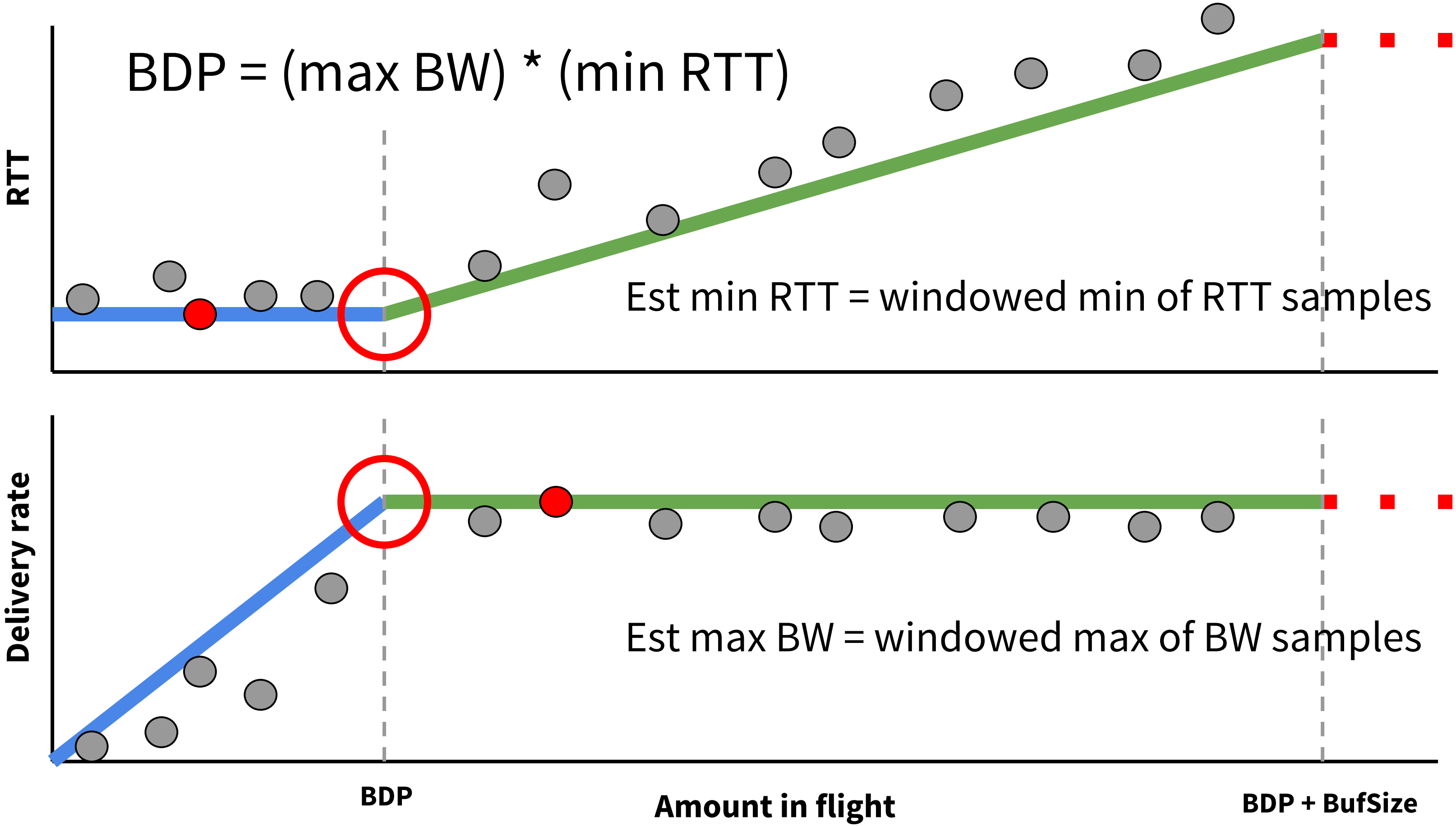


CUBIC / Reno
RTT
Delivery rate
BDP
Amount in flight
BDP + BufSize

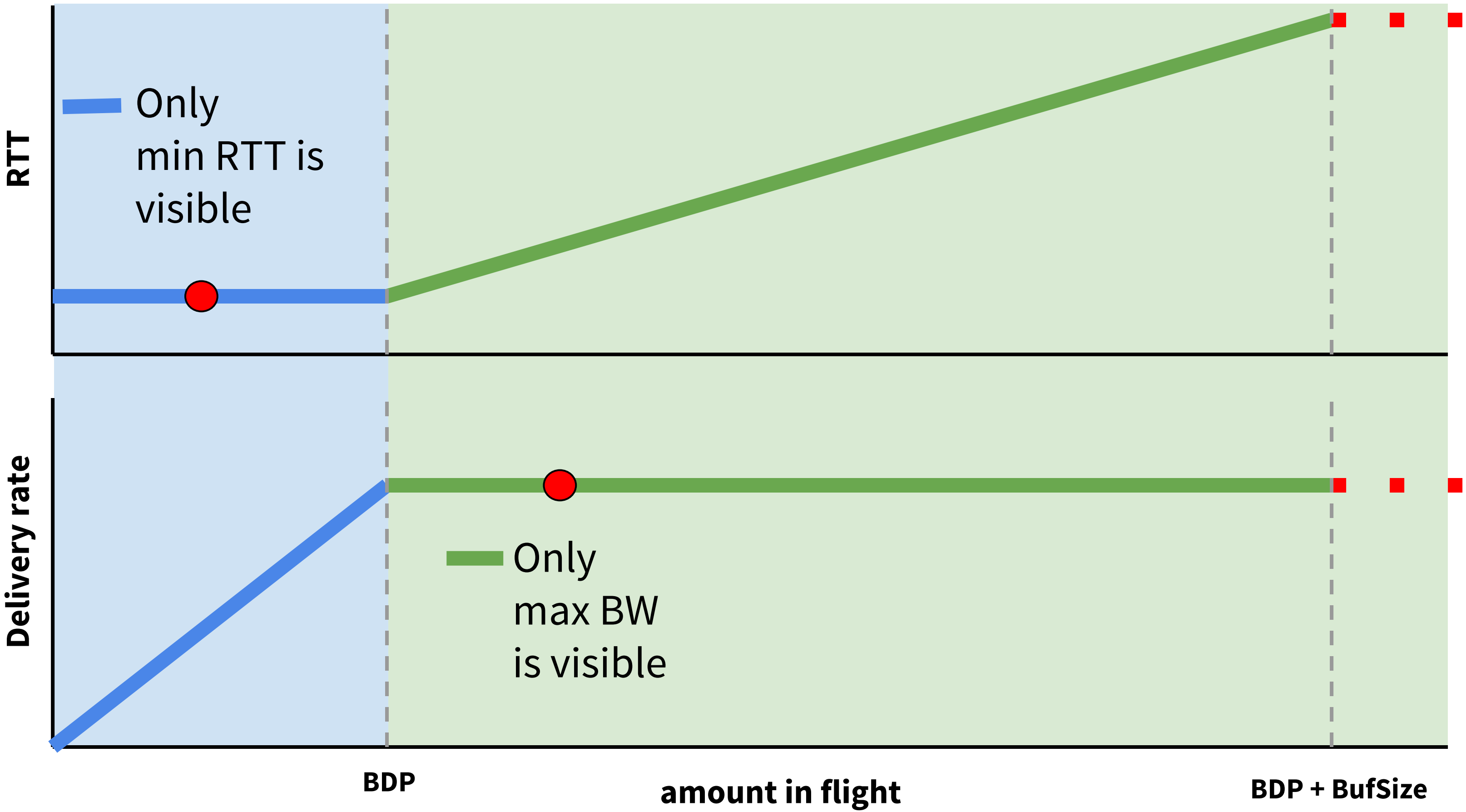
Optimal: max BW and min RTT (Gail & Kleinrock. 1981)



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 windowed 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]

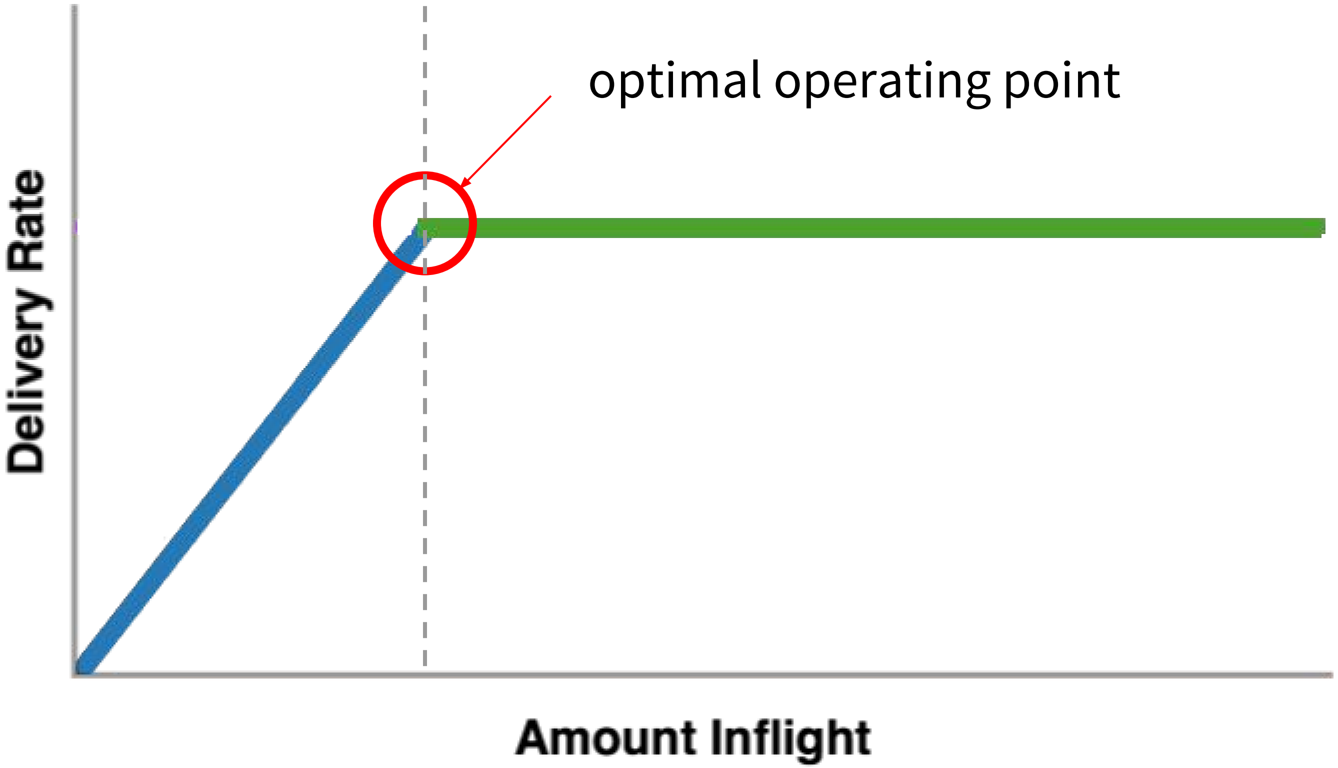
Vary pacing rate to keep inflight near BDP (for full pipe but small queue)

That's **BBR** congestion control:

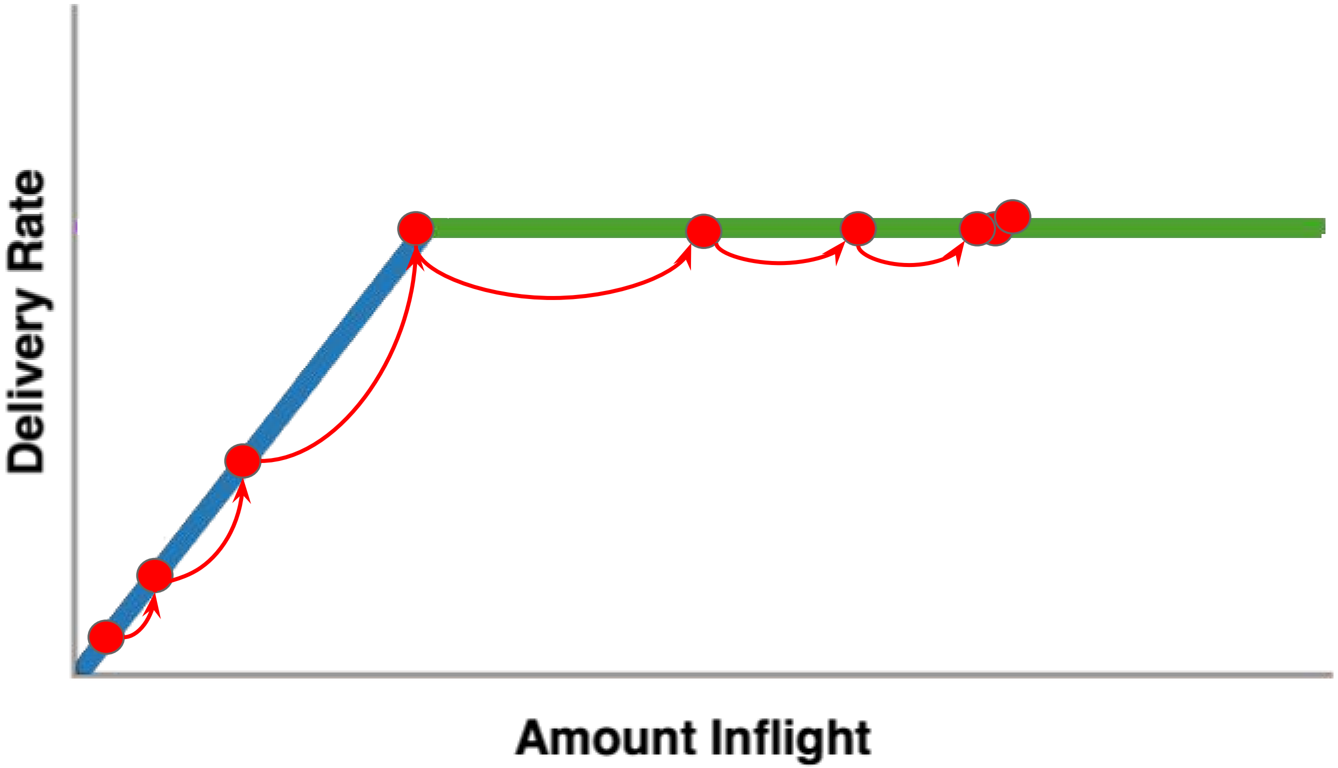
BBR = **B**ottleneck **B**andwidth and **R**ound-trip propagation time

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

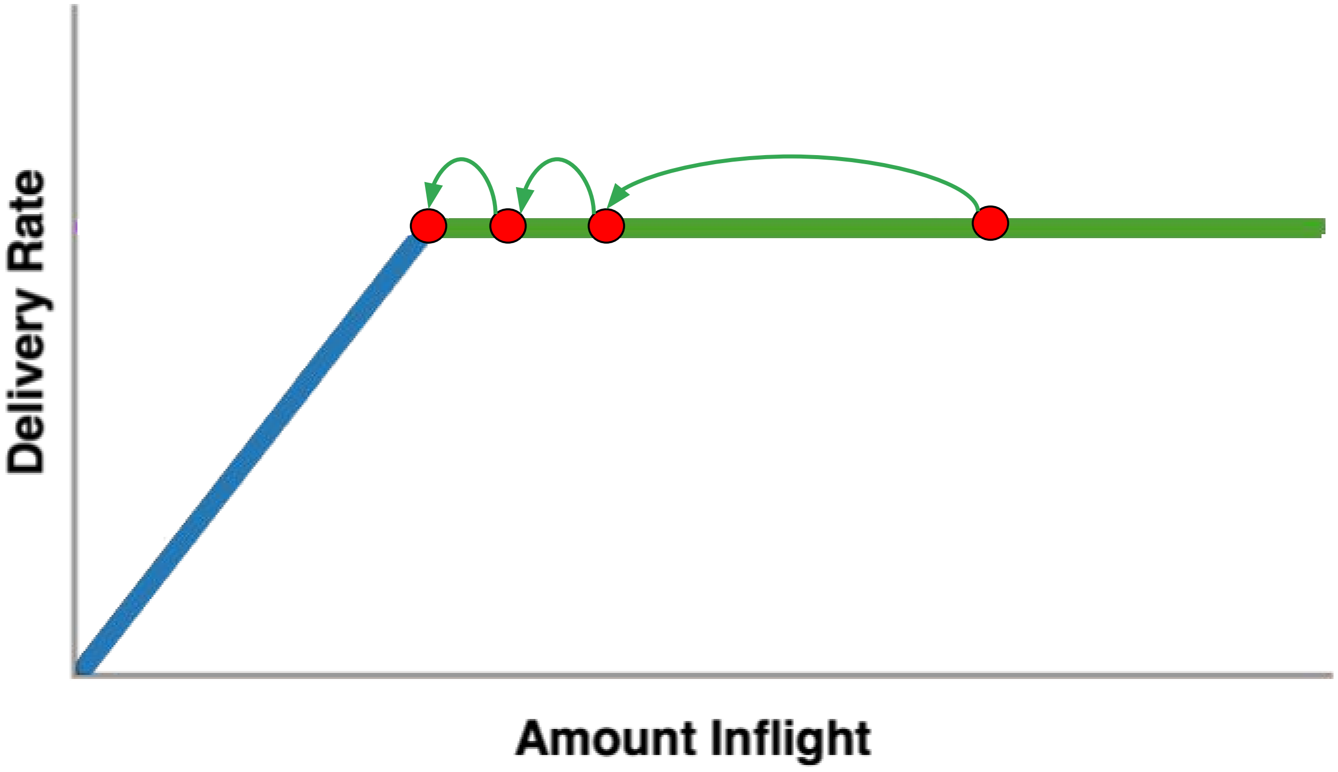
BBR: model-based walk toward max BW, min RTT



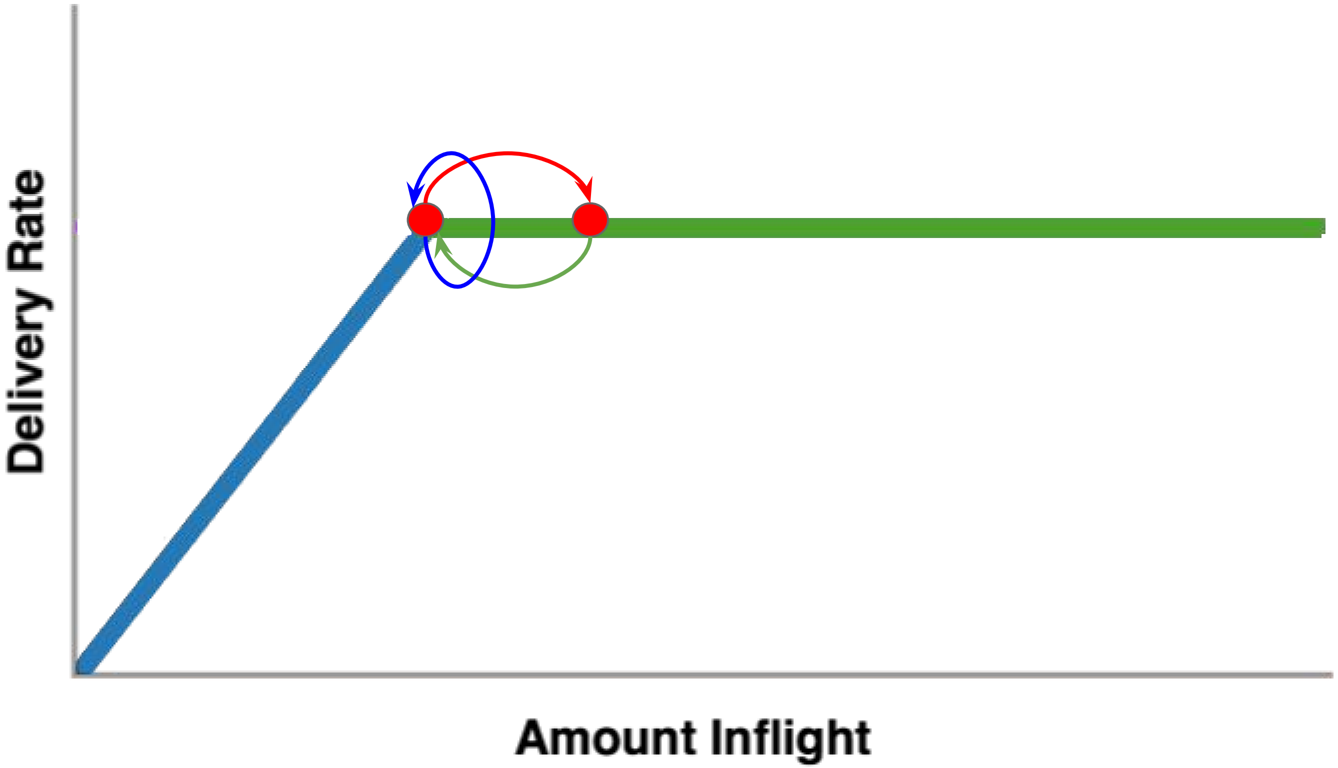
STARTUP: exponential BW search



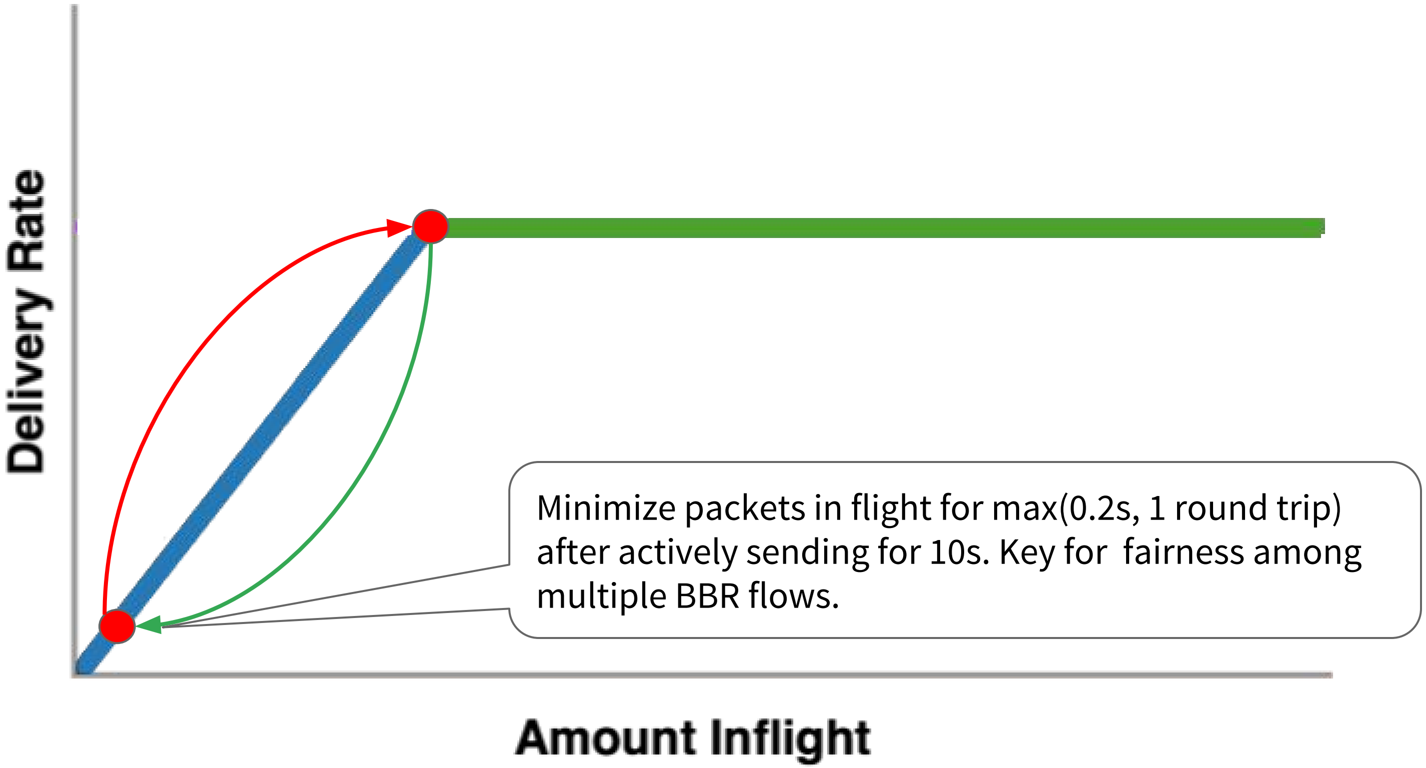
DRAIN: drain the queue created during startup



PROBE_BW: explore max BW, drain queue, cruise

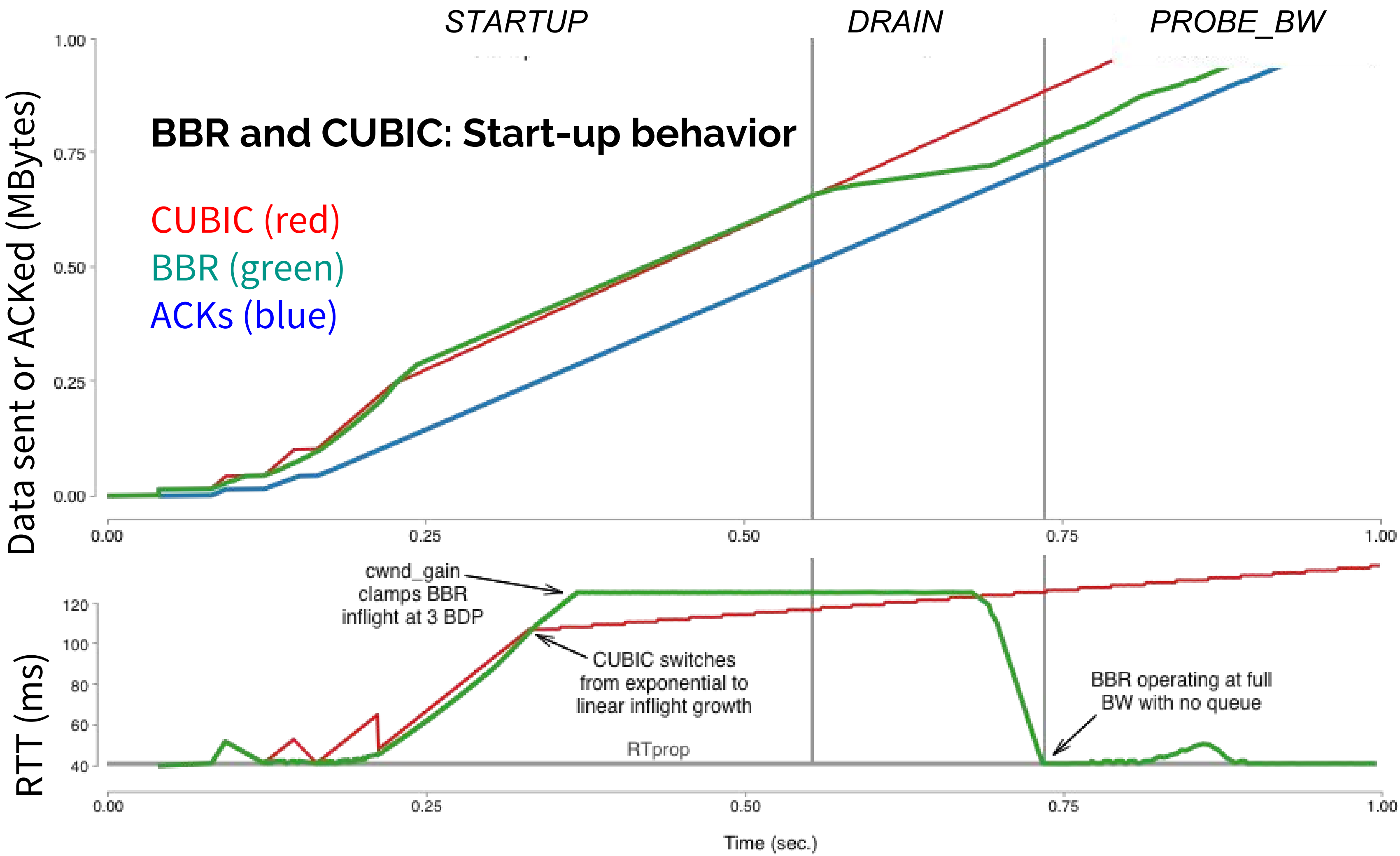


PROBE_RTT drains queue to refresh min_RTT

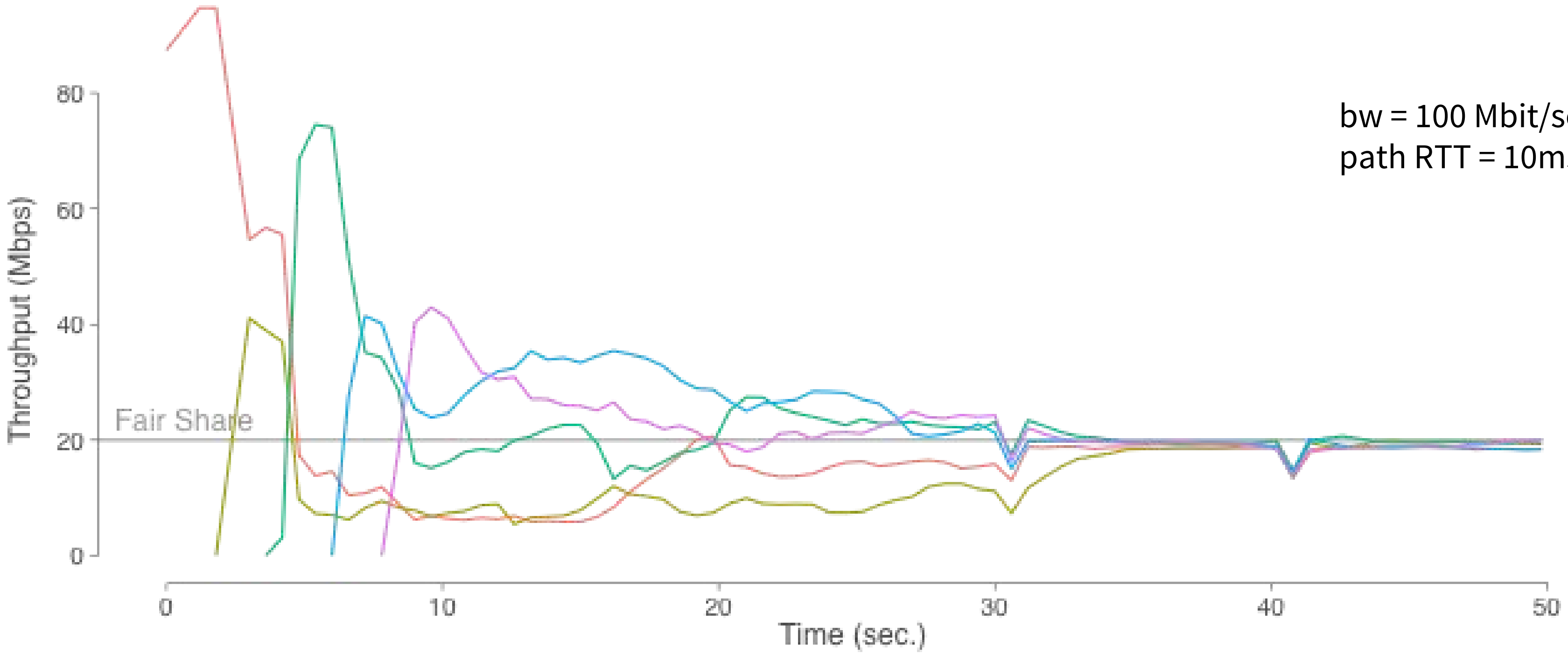


Performance results

BBR and CUBIC: Start-up behavior



BBR multi-flow convergence dynamics

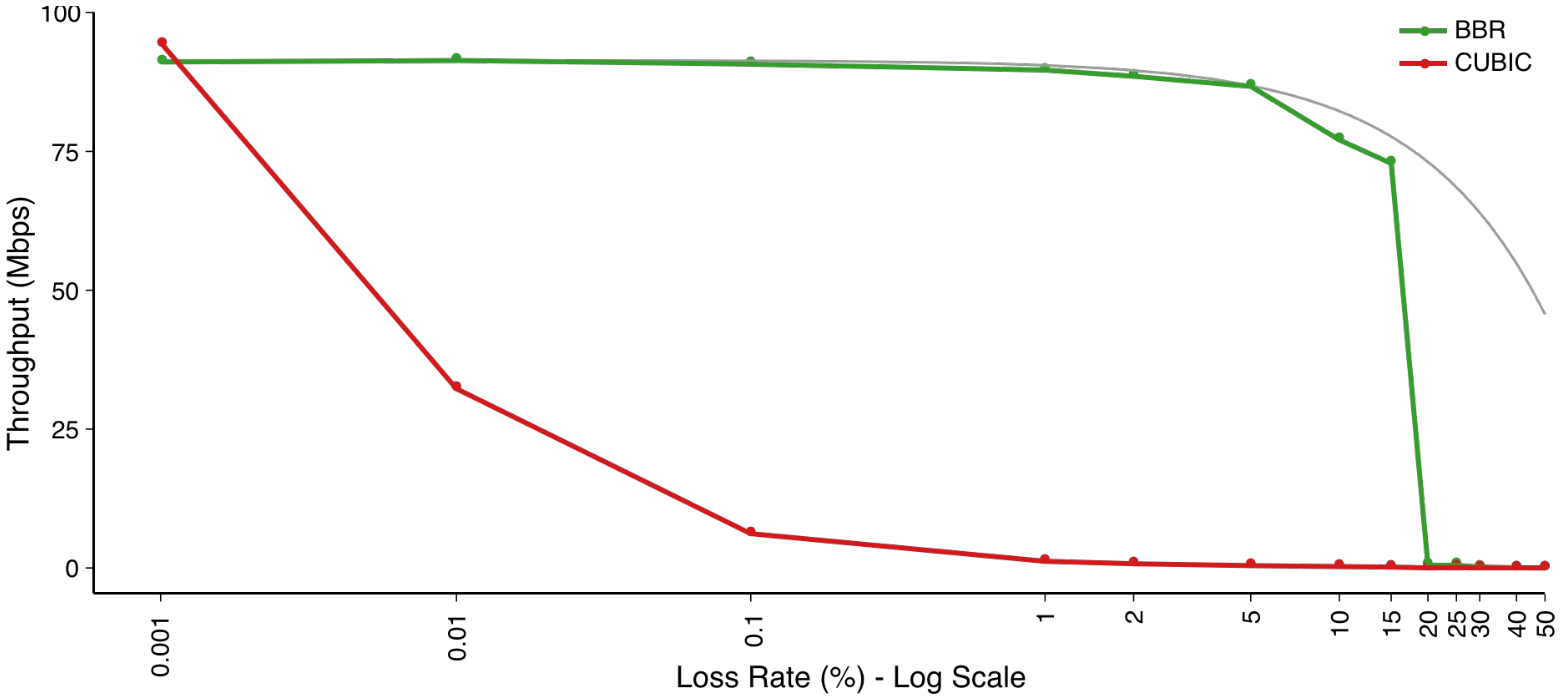


bw = 100 Mbit/sec
path RTT = 10ms

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

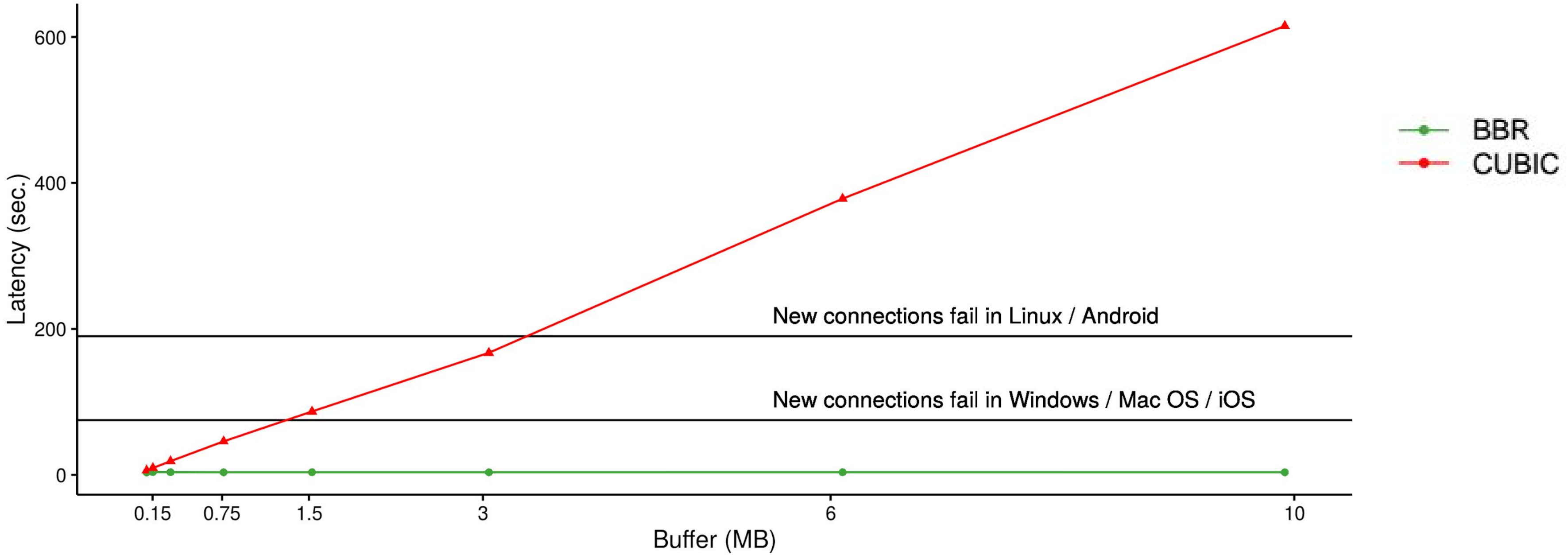
- Queue (RTT) reduction is observed by every (active) flow
- Elephants yield more (multiplicative decrease) to let mice grow: each flow learns its fair share

Fully use bandwidth, despite high loss



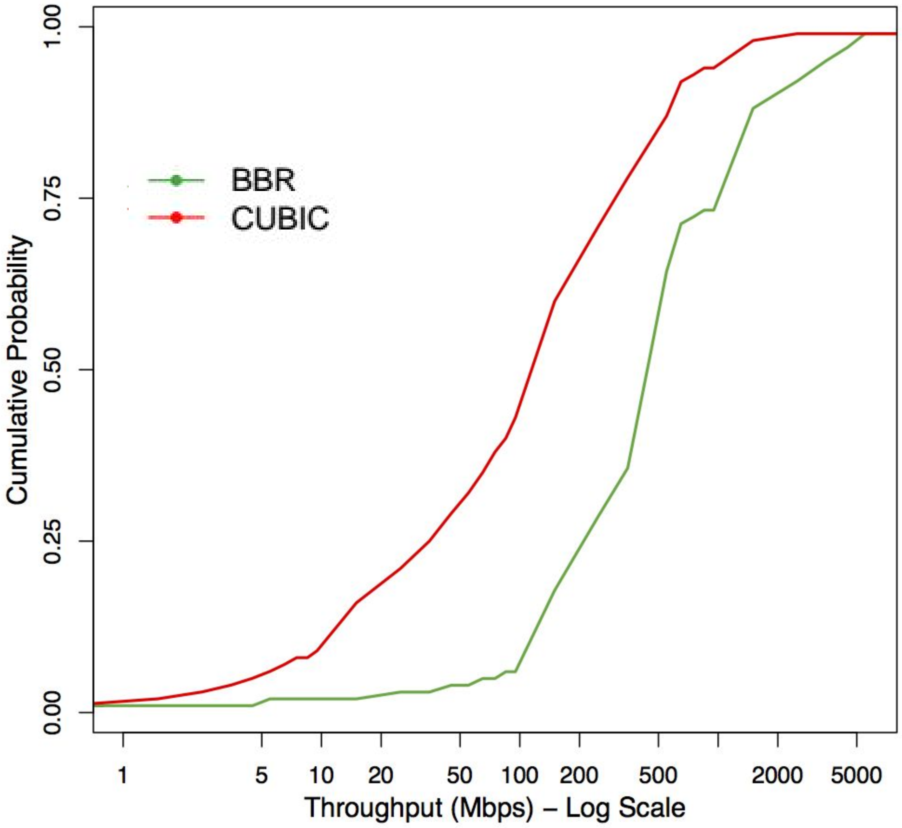
BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms

Low queue delay, despite bloated buffers



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

Active benchmarking tests on Google WAN



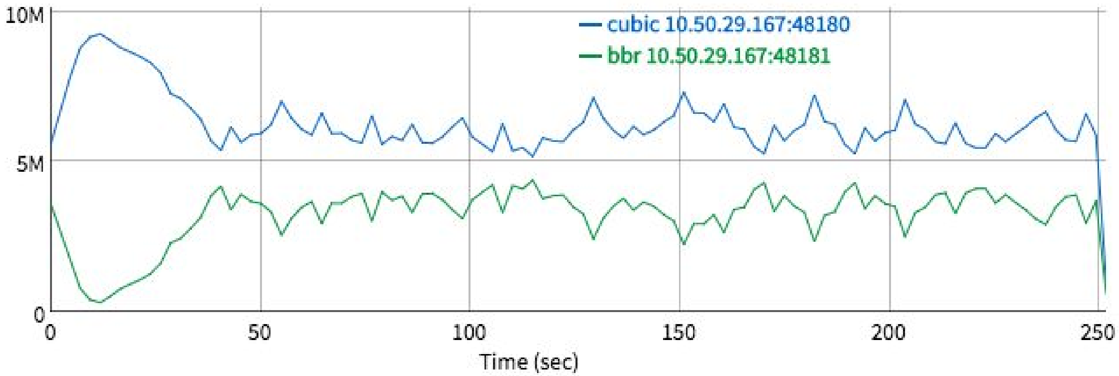
- BBR used for vast majority of TCP on Google B4
- Active probes across metros
  - 8MB PRC every 30s over warmed connections
  - On the lowest QoS (BE1) BBR is 2-20x faster than Cubic
  - BBR tput is often limited by default maximum RWIN (8MB)
- WIP: benchmarking RPC latency impact of all apps using B4 with higher max. RWIN

Deep dives & implementation

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

Sharing deep buffers with loss-based CC



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

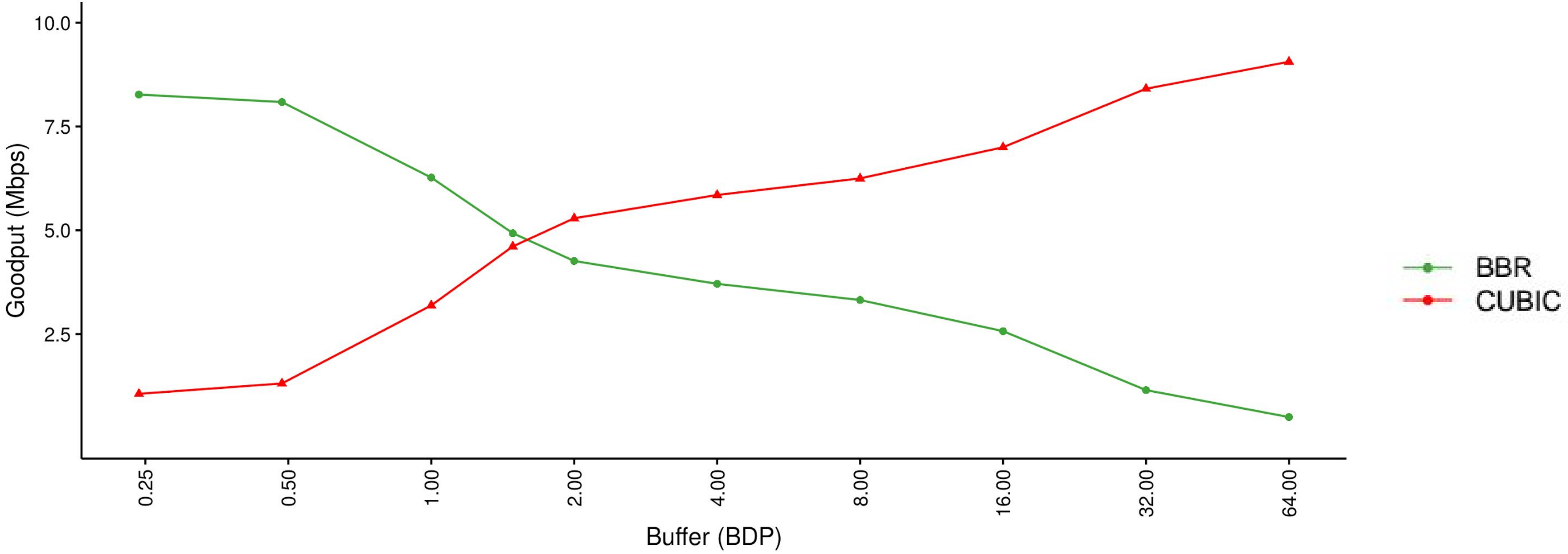
But BBR does not collapse; it adapts: BBR's bw and RTT probing tends to drive system toward fairness

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

  -> CUBIC: 6.31 Mbps vs BBR: 3.26 Mbps

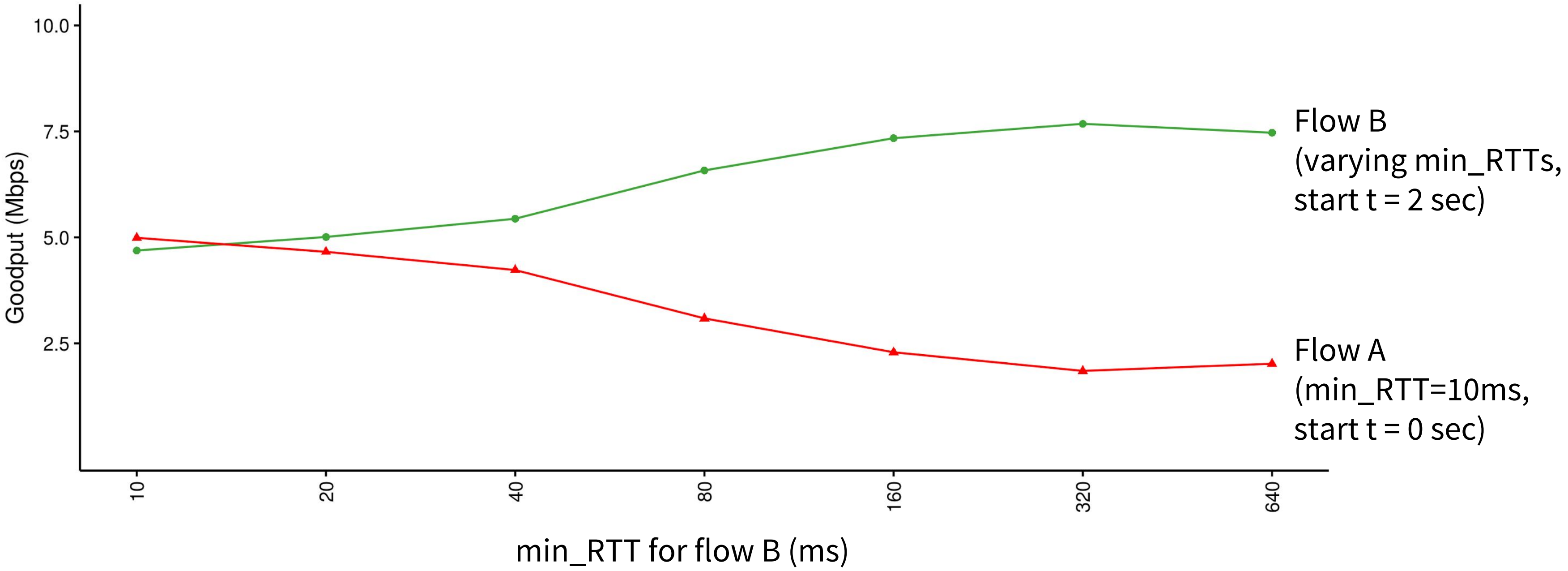
Current dynamics w/ with loss-based CC

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



BBR multi-flow behavior: RTT fairness

Compare the goodput of two competing BBR flows with short (A) and long (B) min_RTT

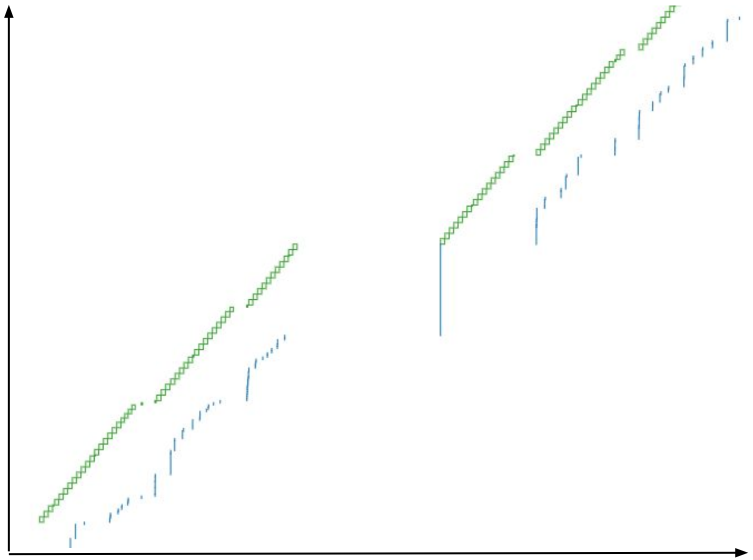


BBR flows w/ higher RTT have an advantage; but BBR flow with 64x higher min_RTT only has <4x higher bw

bw = 10 Mbit/sec, buffer = 1000 packets

Common real-world issues

- ACK compression
  - One TCP ACK for up to +200 packets
  - Particularly wireless & cable networks
  - BBR strategy: cap inflight <= 2*BDP
- Application idles
  - Paces at BW restarting from idle
- Inappropriate receive window
  - Linux default 3MB => 240Mbps on 100ms RTT
- Token-bucket traffic policers
  - Explicitly model policers
  - Details presented in maprg



Implementation and deployment status

- Linux v4.9 TCP
  - A congestion control module with dual GPL/BSD licence
  - Requires fq/pacing qdisc (BBR needs pacing support)
  - Employed for vast majority of traffic for Google's WAN.
  - Being deployed on Google.com and YouTube
- QUIC implementation under way
  - Production experiments have started
  - {vasilvv,ianswett,jri}@google.com
- FreeBSD implementation under way
  - rrs@netflix.com

BBR FAQ

Is BBR fair to Cubic/Reno?	Buffer >= 1.5*BDP: Yes; Else: WIP
Is BBR 1/sqrt(p)?	No
Is BBR {delay|loss|ECN|AIMD}-based?	No. It is congestion-based
Is BBR ack-clocked?	No
Does BBR require pacing?	Yes
Does BBR require an FQ scheduler?	No, but it helps
Does BBR require receiver or network changes	No
Does BBR improve latency on short flows?	Yes

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

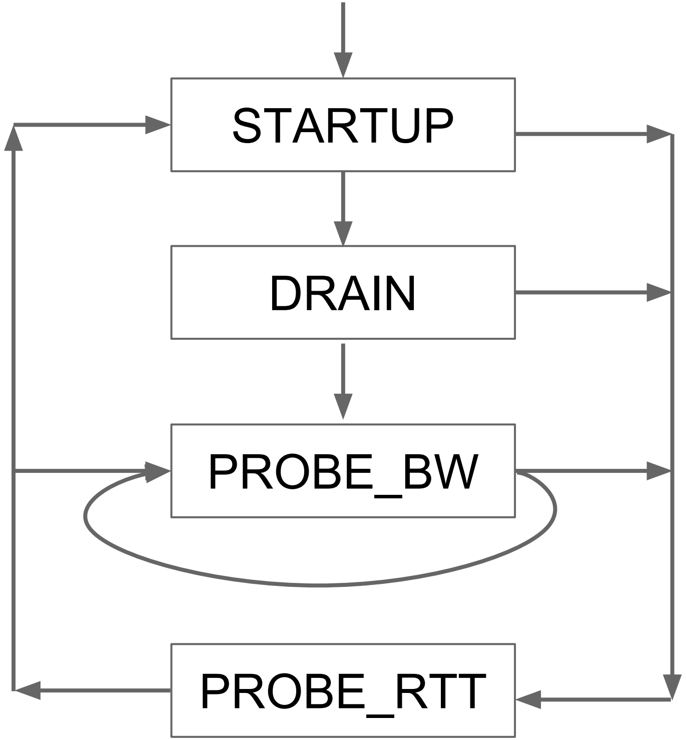
Backup slides…

BBR: core estimators and control logic

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

- On each ACK, update our model of the network path:
  - bottleneck_bandwidth = windowed_max(delivered / elapsed, 10 round trips)
  - min_rtt = windowed_min(rtt, 10 seconds)
- send rate near available bandwidth (primary):
  - Pacing rate = pacing_gain * BtlBw
- volume of data in flight near BDP (secondary):
  - Max inflight = cwnd_gain * BDP = cwnd_gain * BtlBw * RTprop

BBR state transition diagram



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

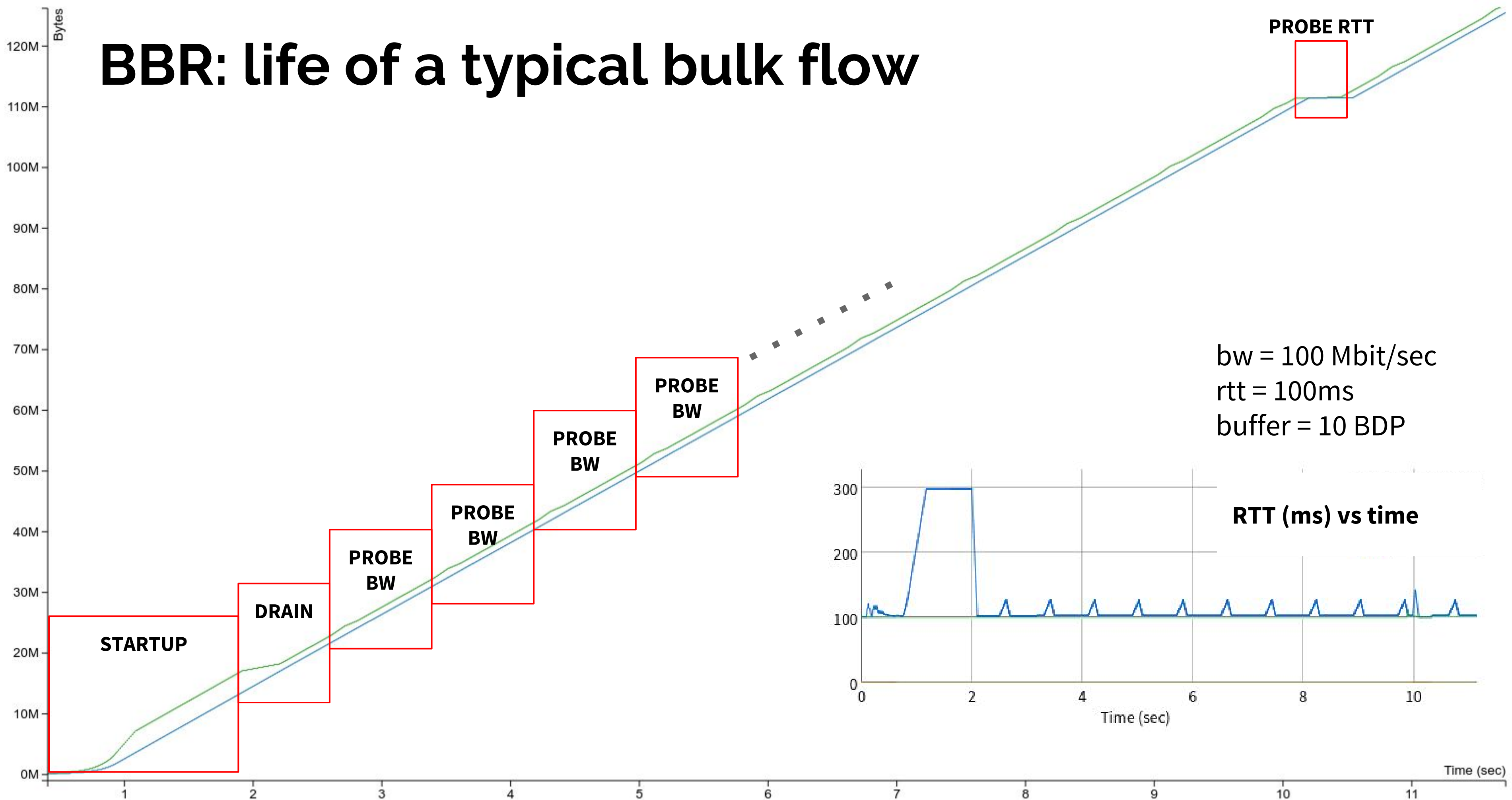
PROBE_BW: cycle pacing_gain to explore and fairly share bandwidth (cwnd_gain = 2 in all phases):

- [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

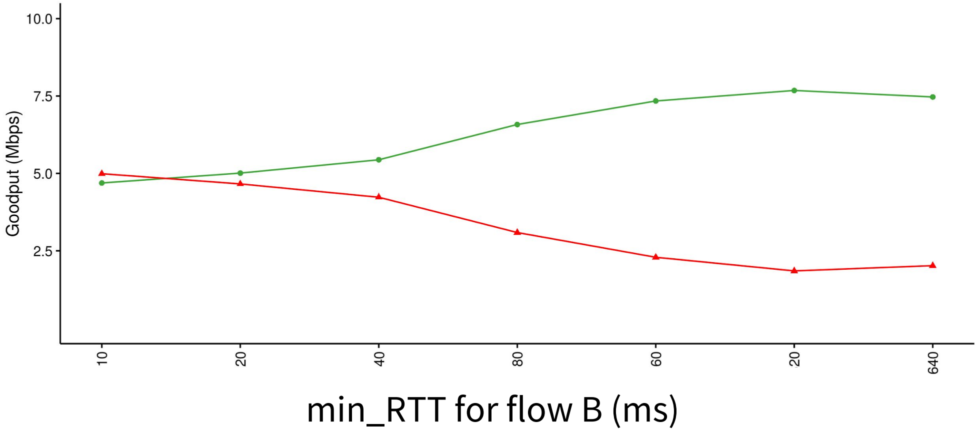
PROBE_RTT: if needed, occasionally send slower to probe min RTT

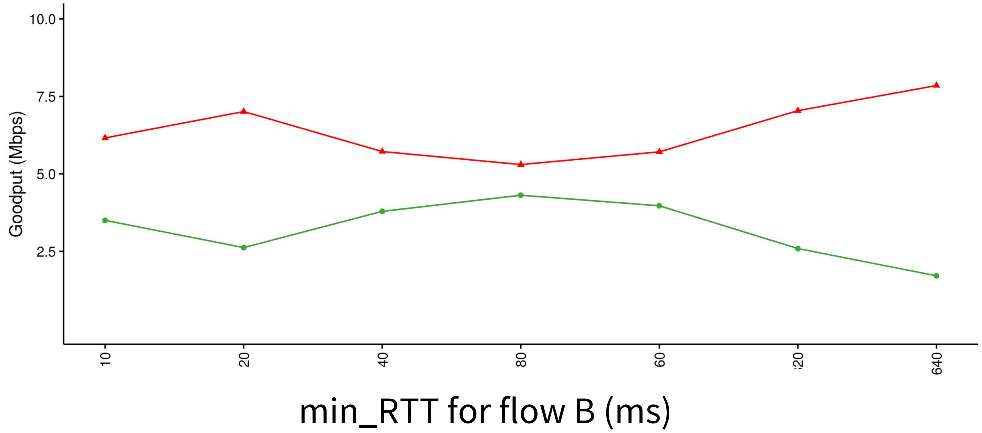
- Maintain inflight of 4 for at least max(1 round trip, 0.2 sec); pacing_gain = 1.0

BBR: life of a typical bulk flow



Comparing RTT fairness for BBR and CUBIC





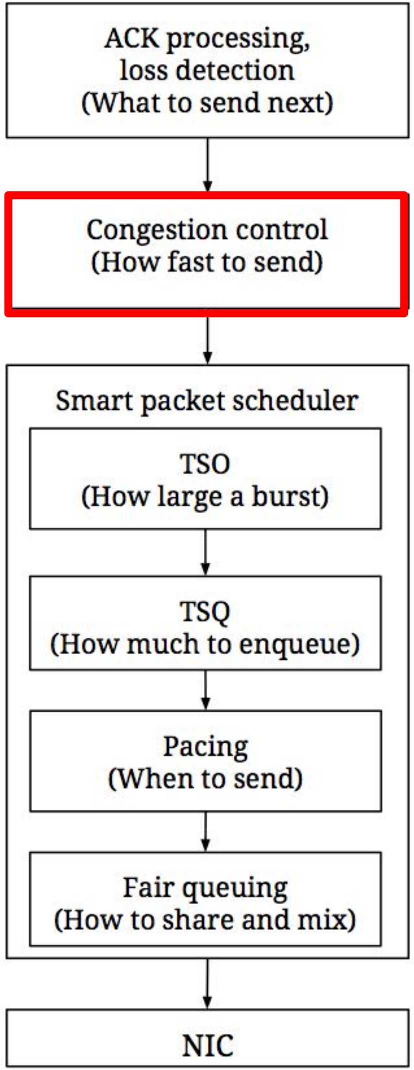
Compare the goodput of two competing BBR or CUBIC flows with short (A) and long (B) min_RTT

bw = 10 Mbit/sec, buffer = 1000 packets

```
Flow A (min_RTT=10ms,      start t = 0 sec)
Flow B (varying min_RTTs, start t = 2 sec)
```

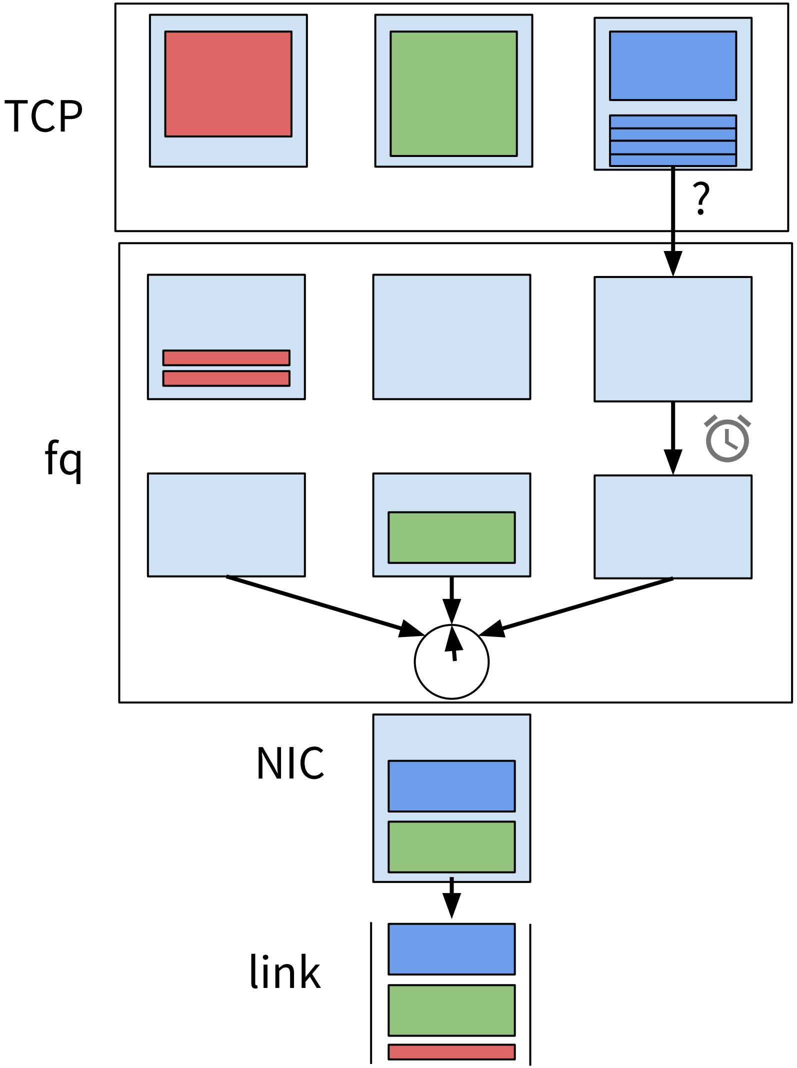
BBR flows w/ higher RTT have an advantage;
flow with 64x higher min_RTT has <4x higher bw

CUBIC flows w/ lower RTT have an advantage;
flow with 64x higher min_RTT has 4.6x higher bw



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



TSO autosizing

TCP Small Queues (TSQ)

Pacing

Fair queuing