BBR TCP

Matt Mathis, based on the work of:

Neal Cardwell, Yuchung Cheng,
C. Stephen Gunn, Soheil Hassas Yeganeh,
Van Jacobson

https://groups.google.com/d/forum/bbr-dev

Google

Outline

- Problems with loss based Congestion Control
- Intro to BBR
  - Algorithm & Model
  - Basic behavior
- Results
  - Common environments
  - Deployment at Google
- Future work

Problems with loss-based congestion control

2011: many reported excessive buffering and delays on the Internet (a.k.a. bufferbloat)

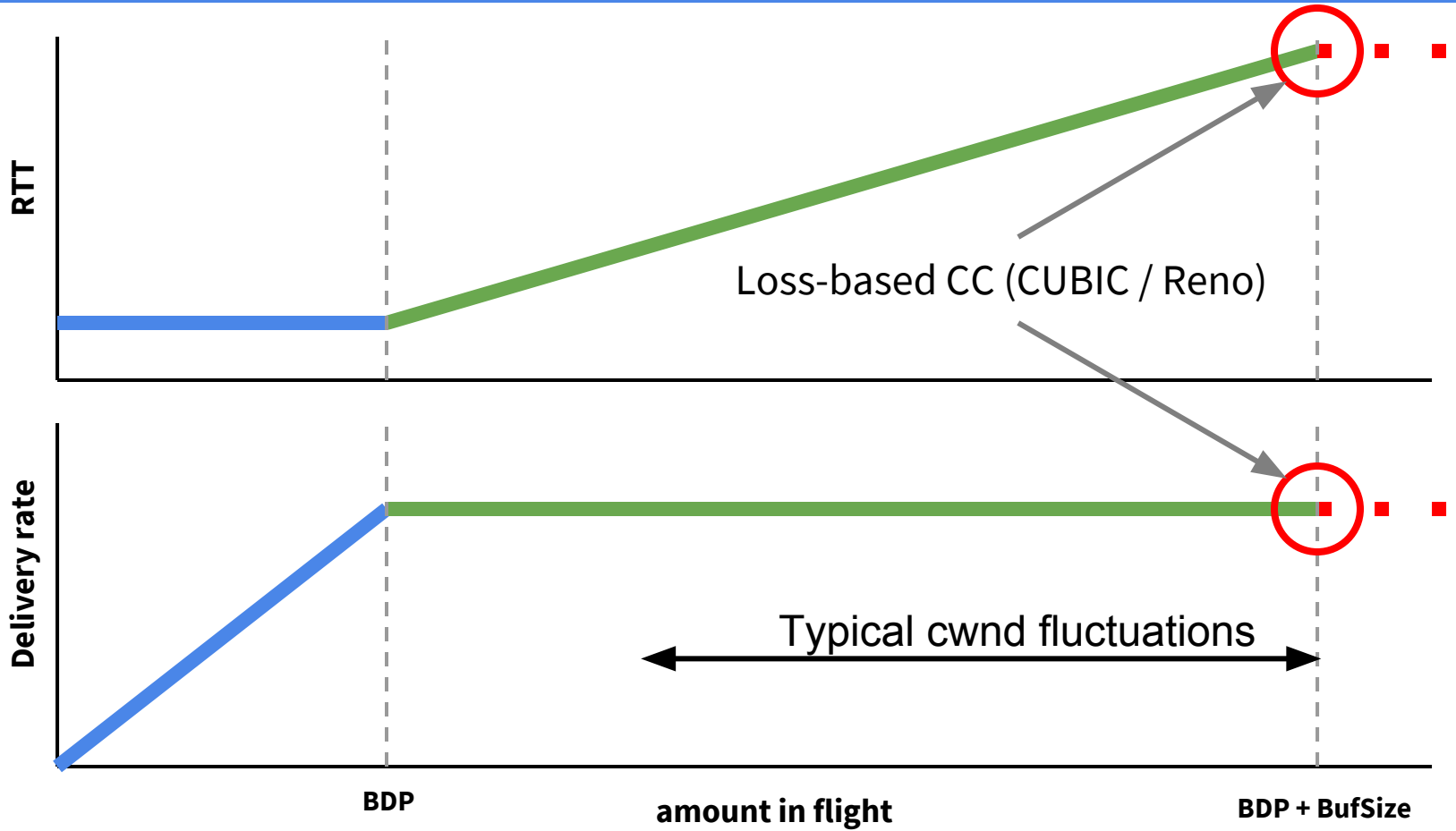
2012: single-connection HTTP/2 was much slower than multi-conn HTTP/1 on lossy links

2013: poor TCP throughput on WANs w/ commodity shallow-buffer switches

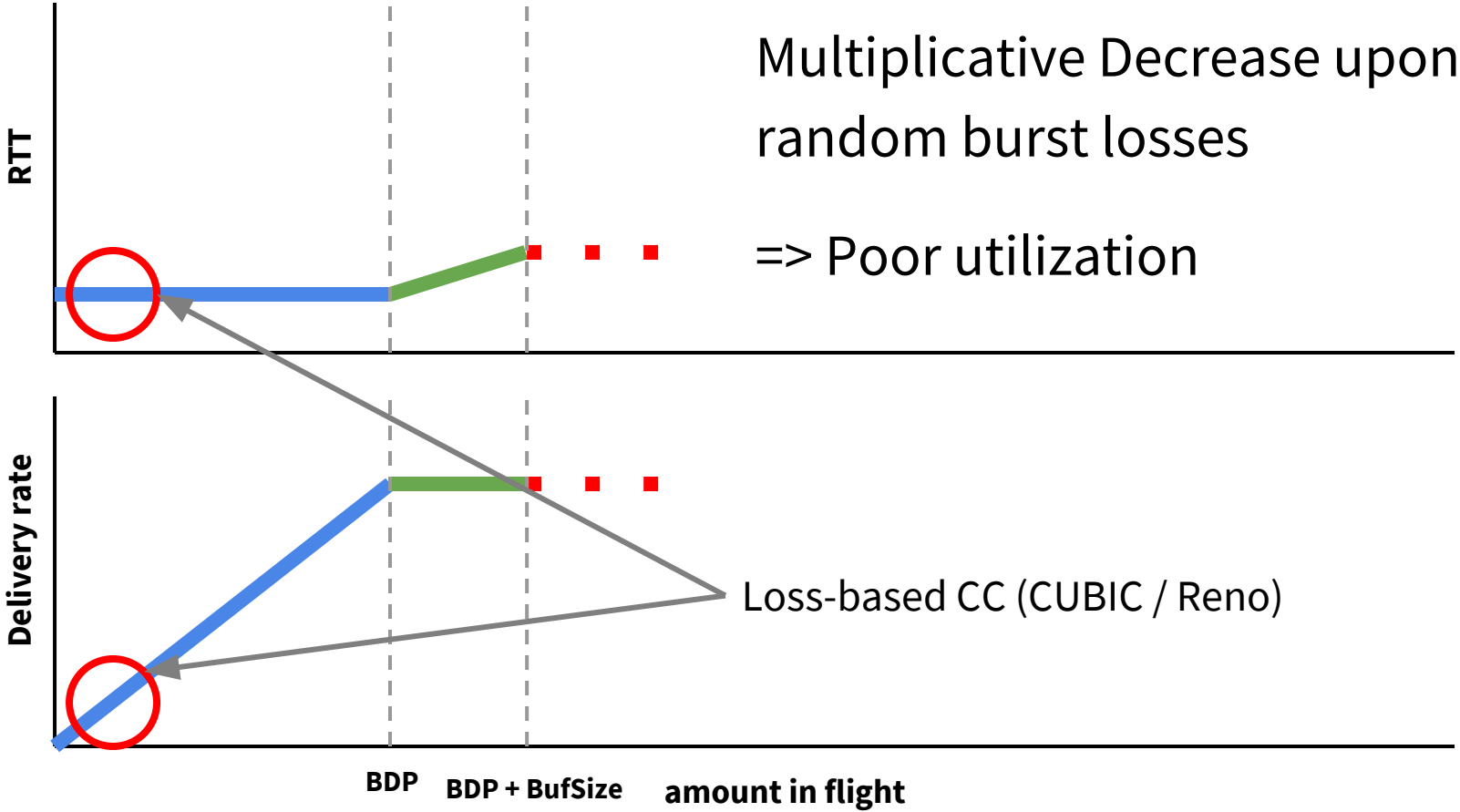
Culprit: loss-based congestion control (Reno, then CUBIC)

- Packet loss alone is **not** a good proxy to detect congestion
- Loss-based CC is overly sensitive to losses that come **before** congestion
  - 10Gbps over 100ms RTT needs <0.000003% packet loss
  - 1% loss over 100ms RTT gets 3Mbps
- Loss-based CC bloats buffers if loss comes **after** congestion

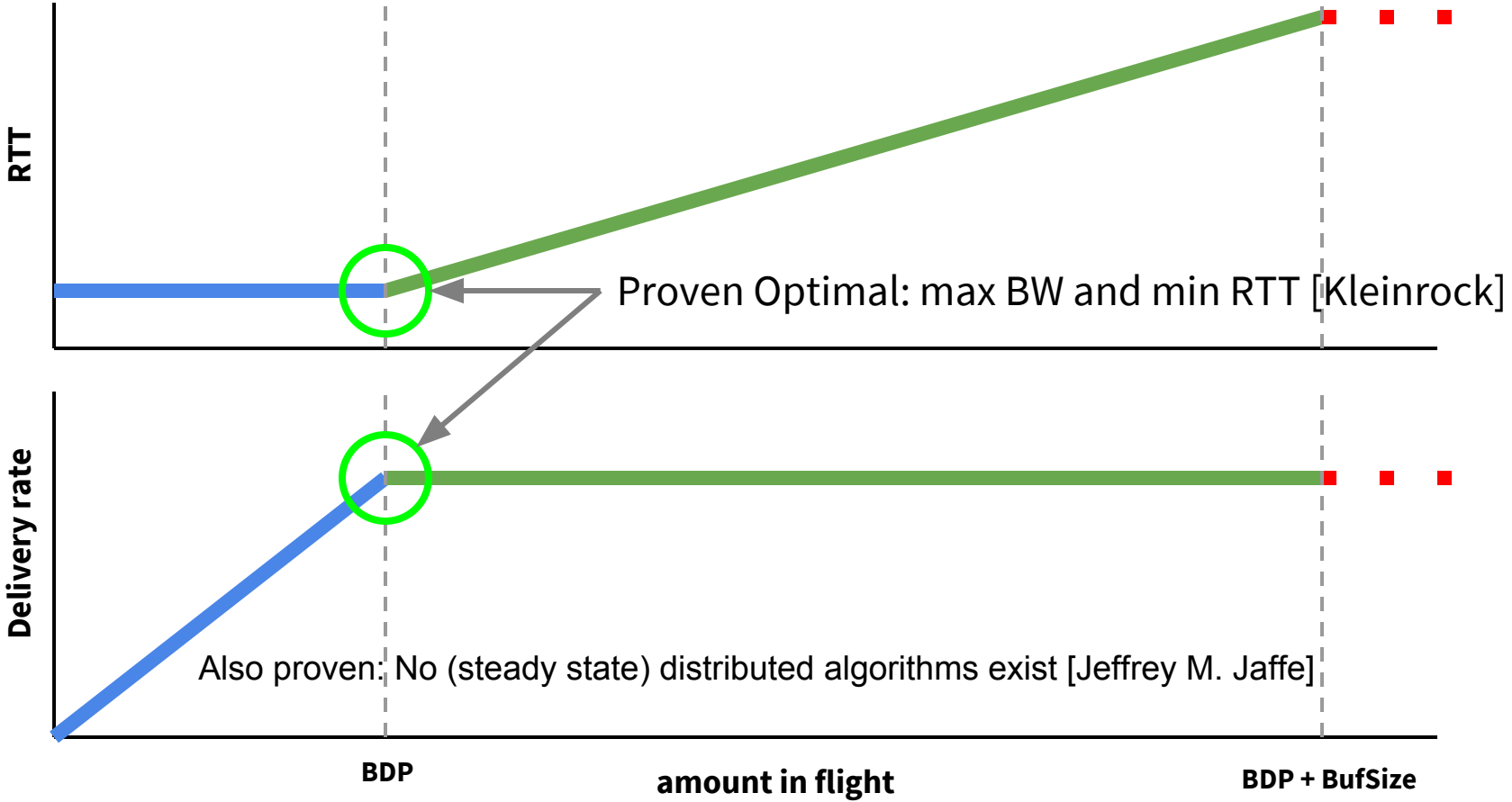
Loss-based congestion control in deep buffers



Loss-based congestion control in shallow buffers



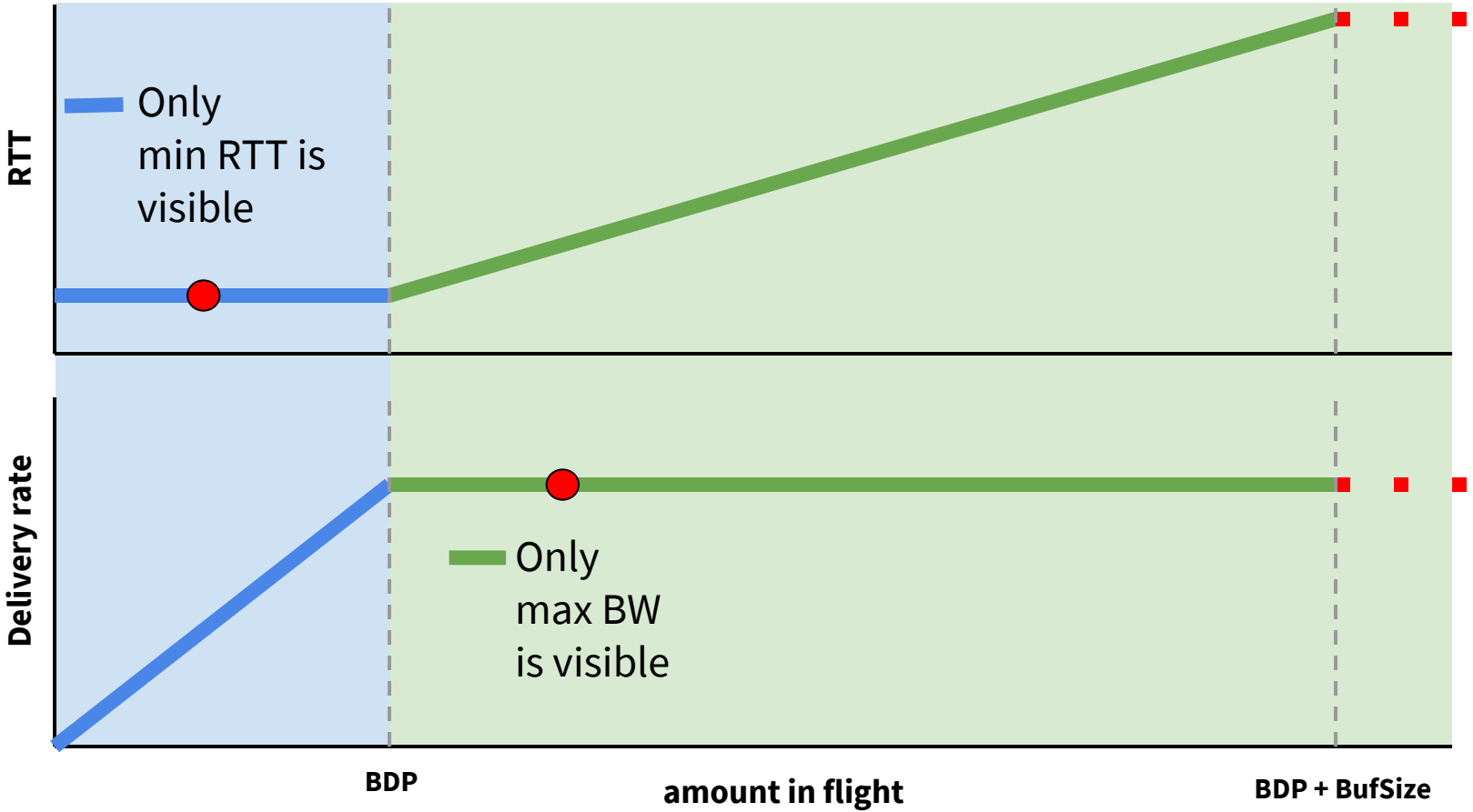
Optimal operating point



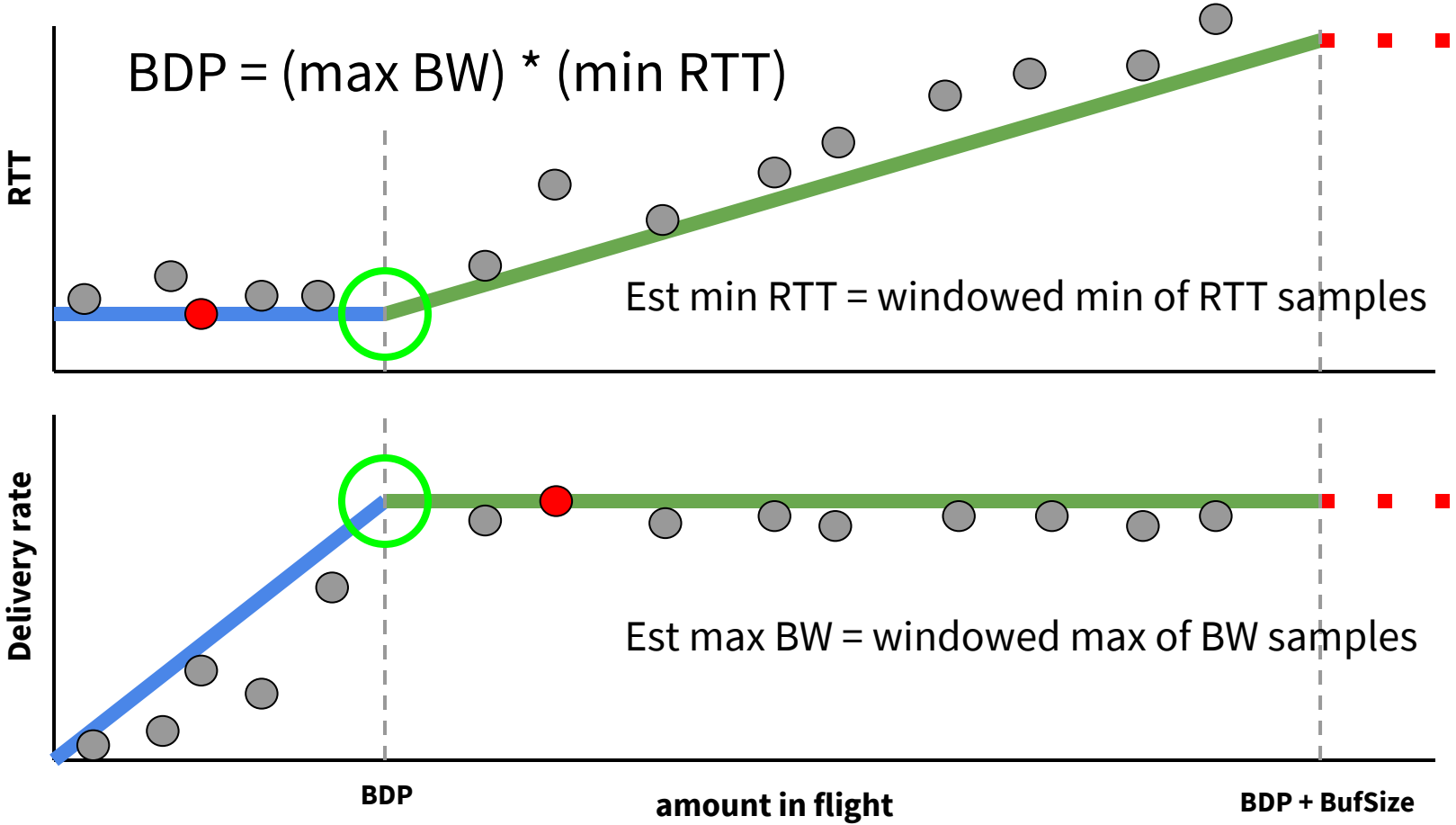
BBR Congestion Control: Overview

- **BBR** = **B**ottleneck **B**andwidth and **R**ound-trip propagation time
- **BBR seeks high throughput with a small queue by probing BW and RTT**
- Ground-up redesign of congestion control
  - *Not* loss-based, *Not* delay-based, *Not* ECN-based, *Not* AIMD-based
- **Models** the network path: probes and estimates max BW and min RTT
- Key insight: measure max BW and min RTT at different times
  - Assume path properties are mostly stationary

To see max BW, min RTT: probe both sides of BDP



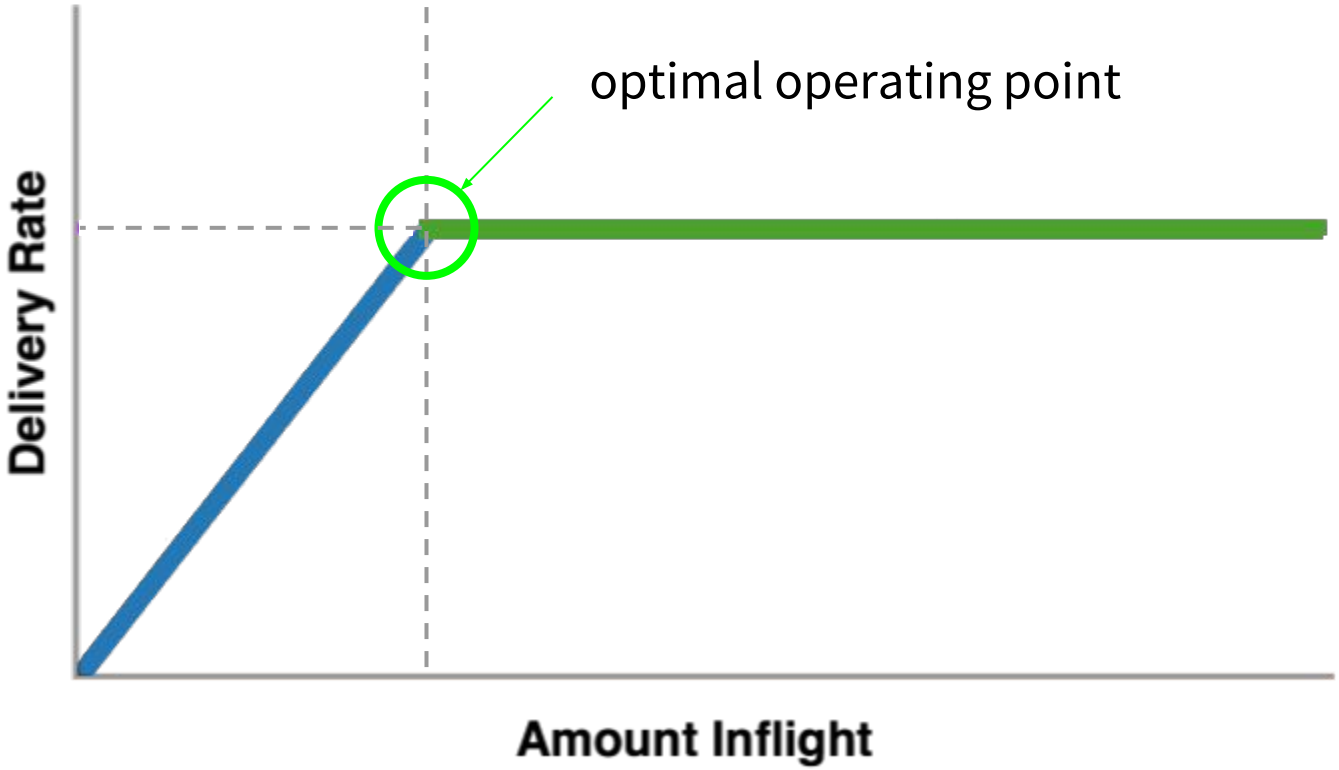
Estimating optimal point (max BW, min RTT)



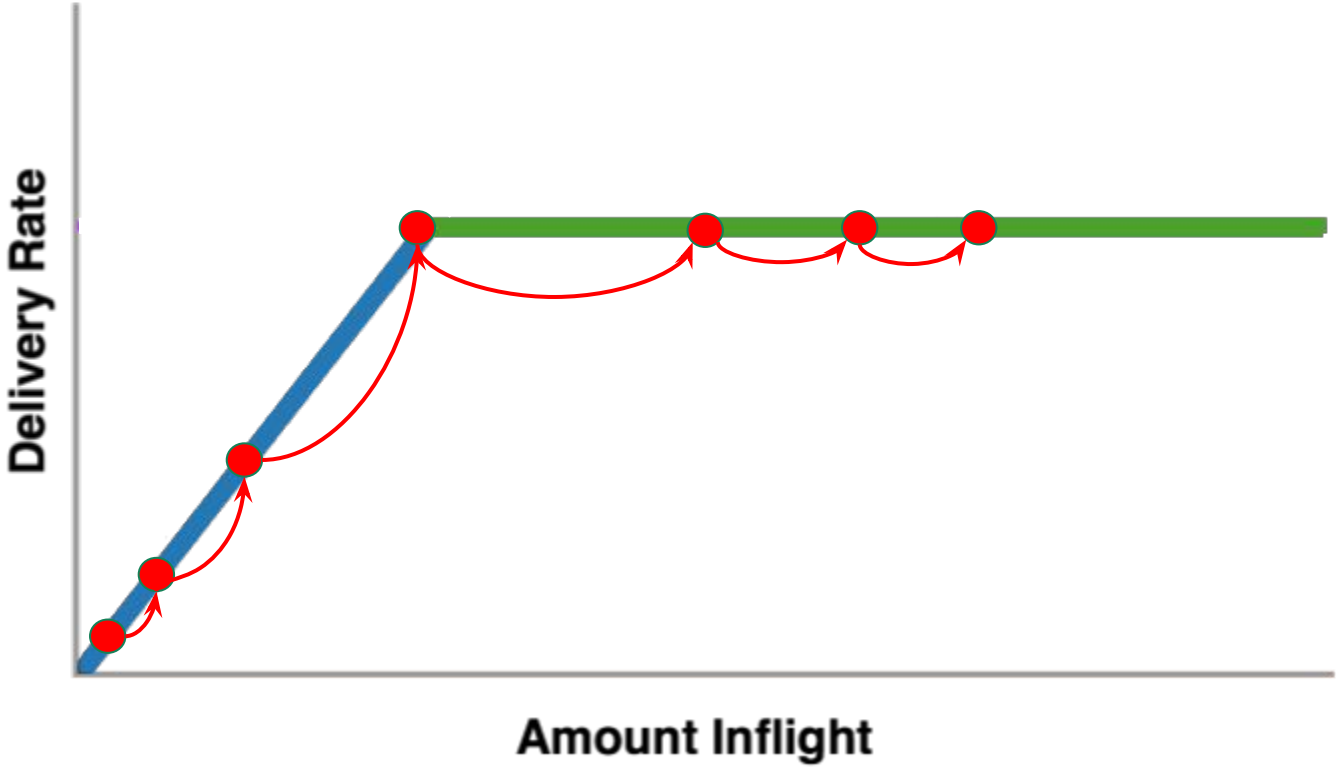
BBR: Core Design

- **Measure** the network path on every ACK
  - Update max BW (8 round-trip sliding window)
  - Update min RTT (10 second sliding window)
- **Model** network path
  - Path BDP = max BW * min RTT
- **Control** sending based on the model, to...
  - Sequentially probe max BW and min RTT, to drive the measurements
  - Mostly pace at estimated BW, to minimize queues and loss
  - Probe up, 1 in 8 RTTs to refresh max BW estimate
  - Probe down 1 RTT every 10 seconds to refresh min RTT estimate
- **Secondary control** limit inflight to be less than 2 * path BDP

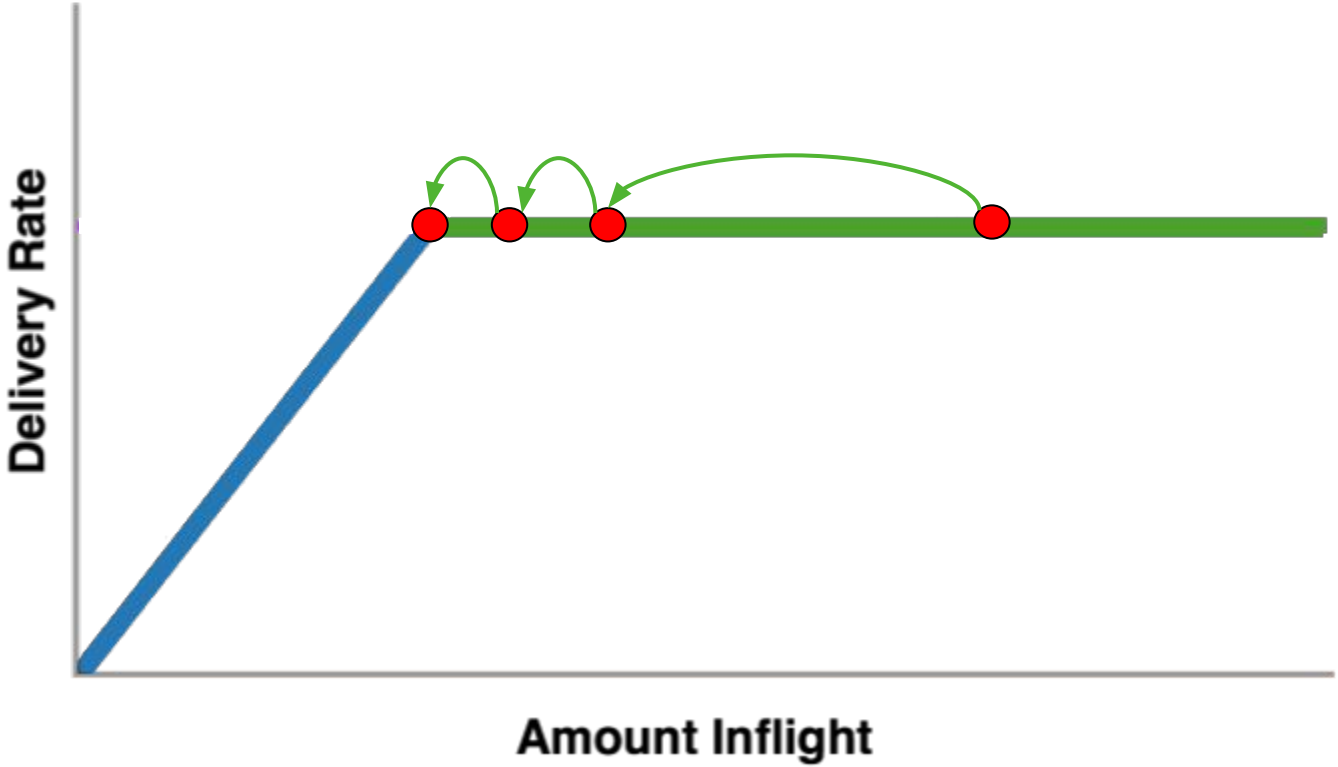
BBR: model-based walk toward max BW, min RTT



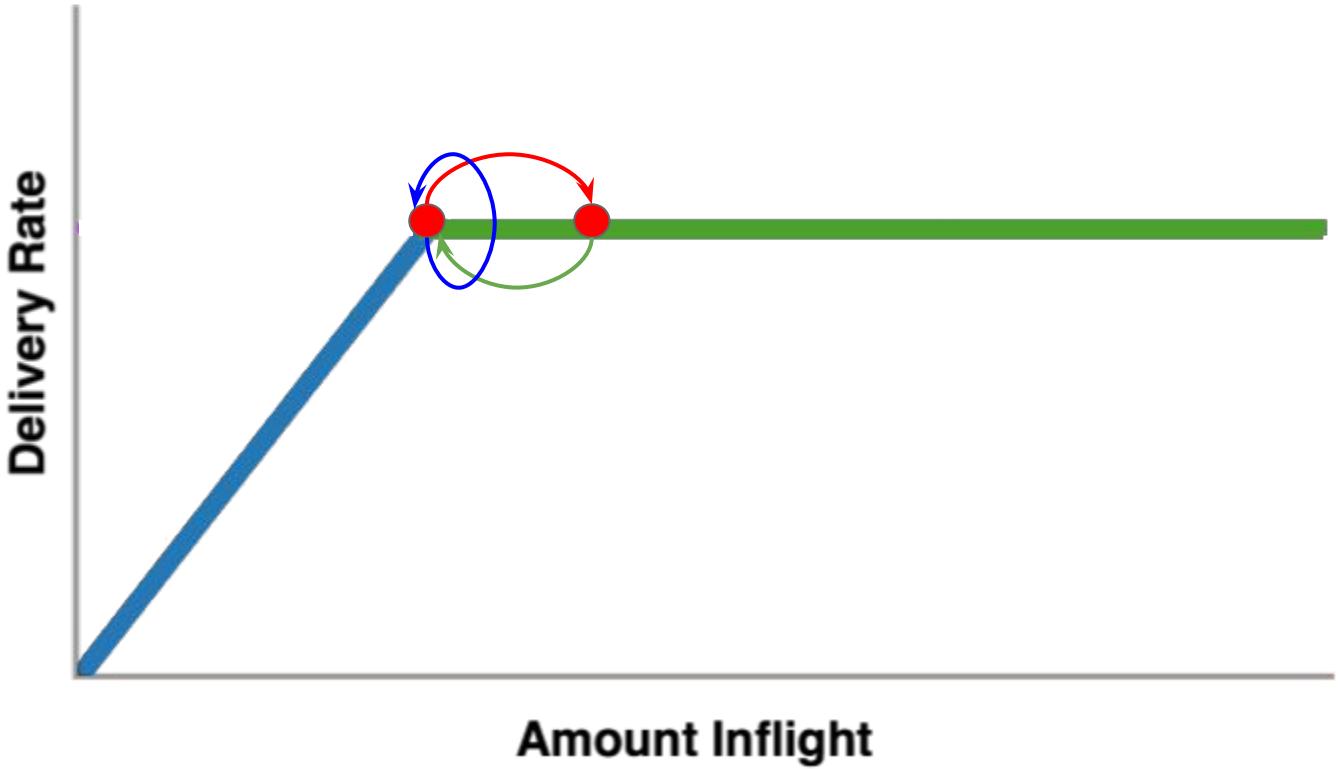
STARTUP: exponential BW search



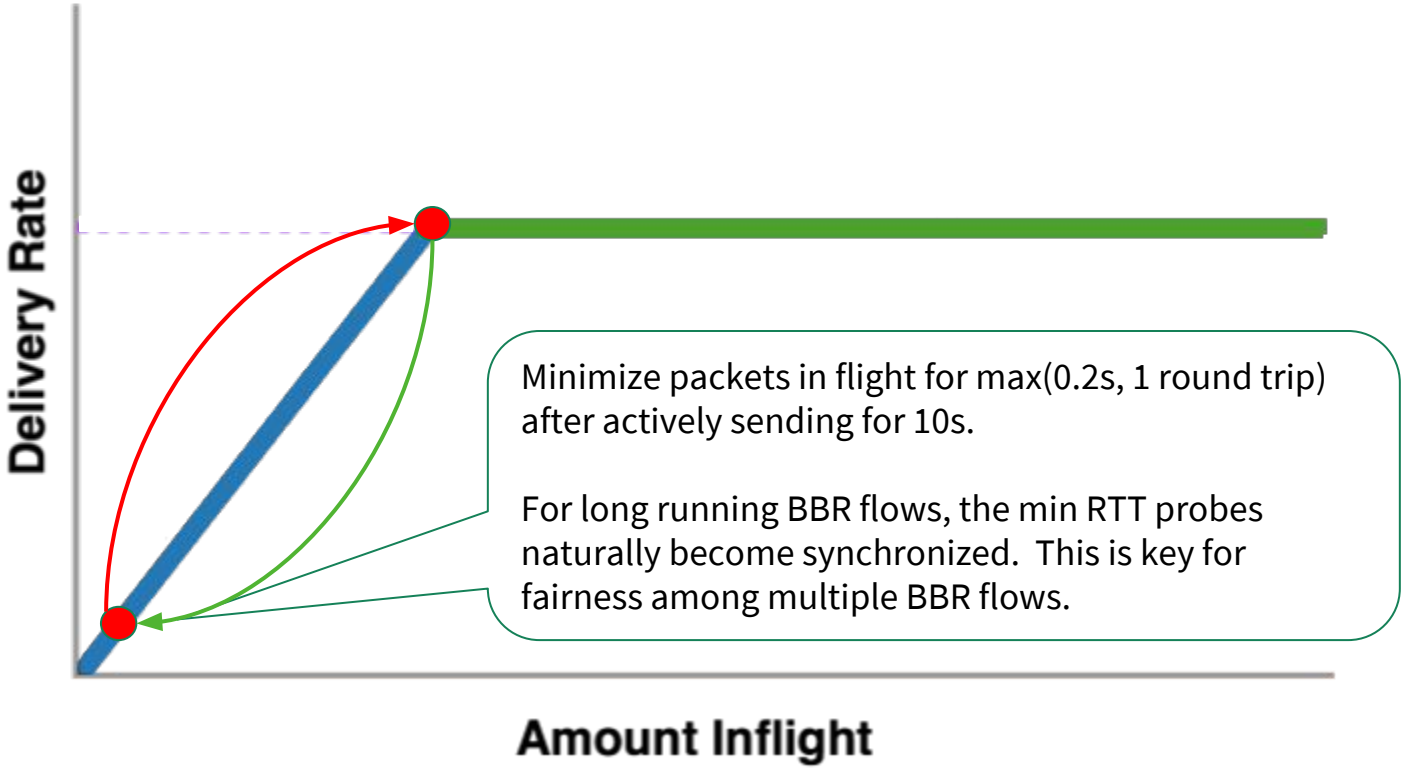
DRAIN: drain the queue created during STARTUP



PROBE_BW: explore max BW, drain queue, cruise

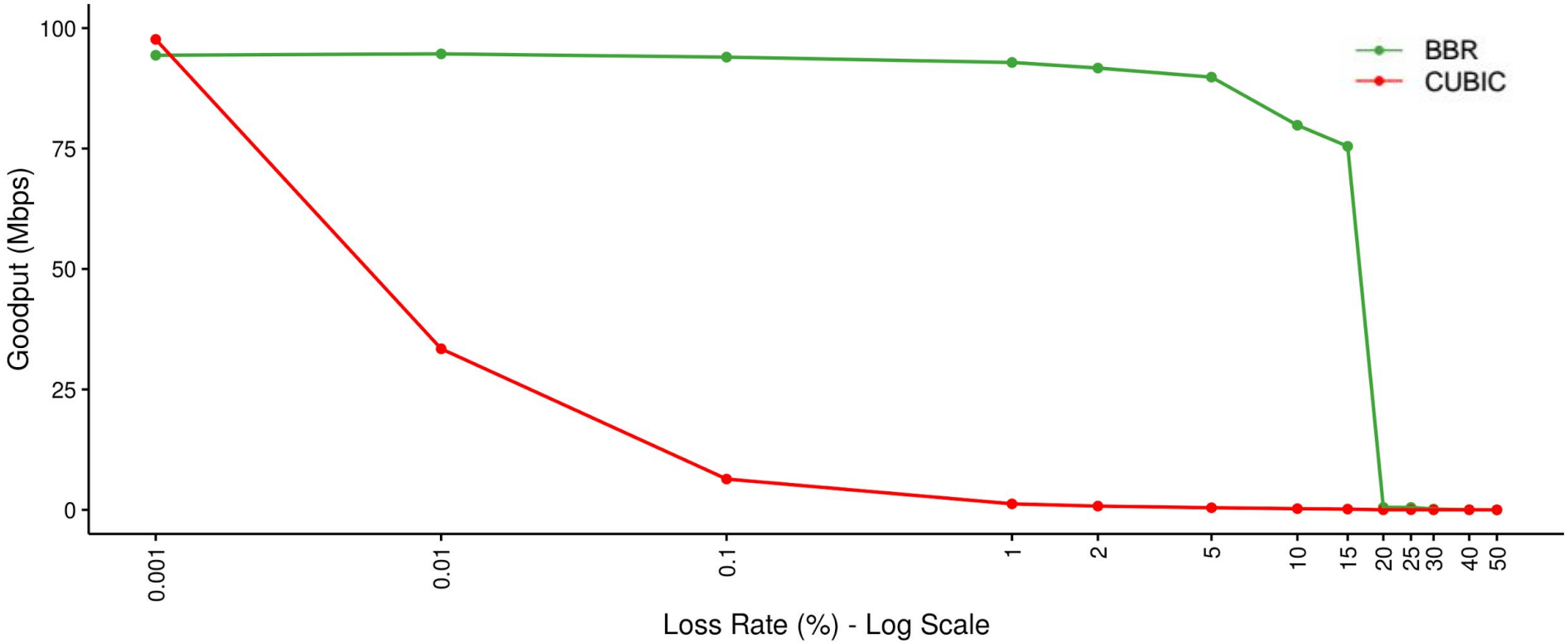


PROBE_RTT: drains queue to refresh min RTT



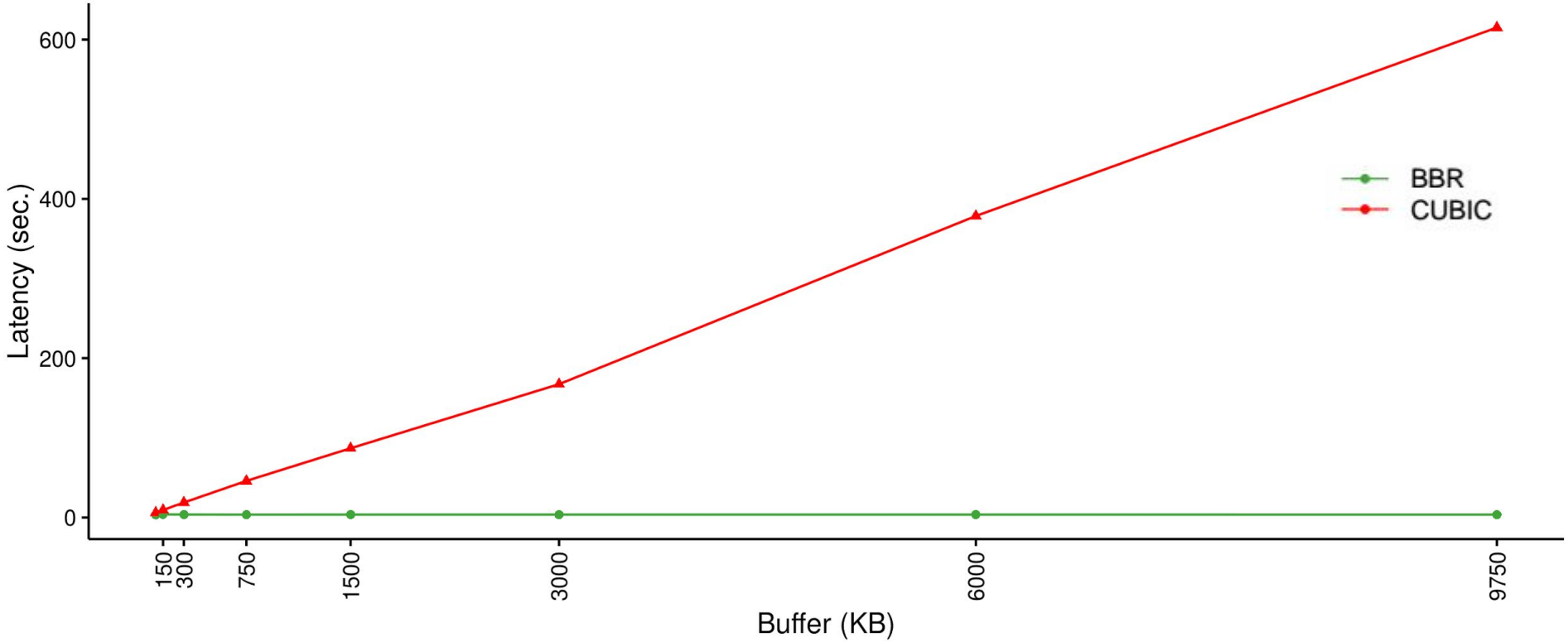
Results

BBR: fully use bandwidth, despite high packet loss



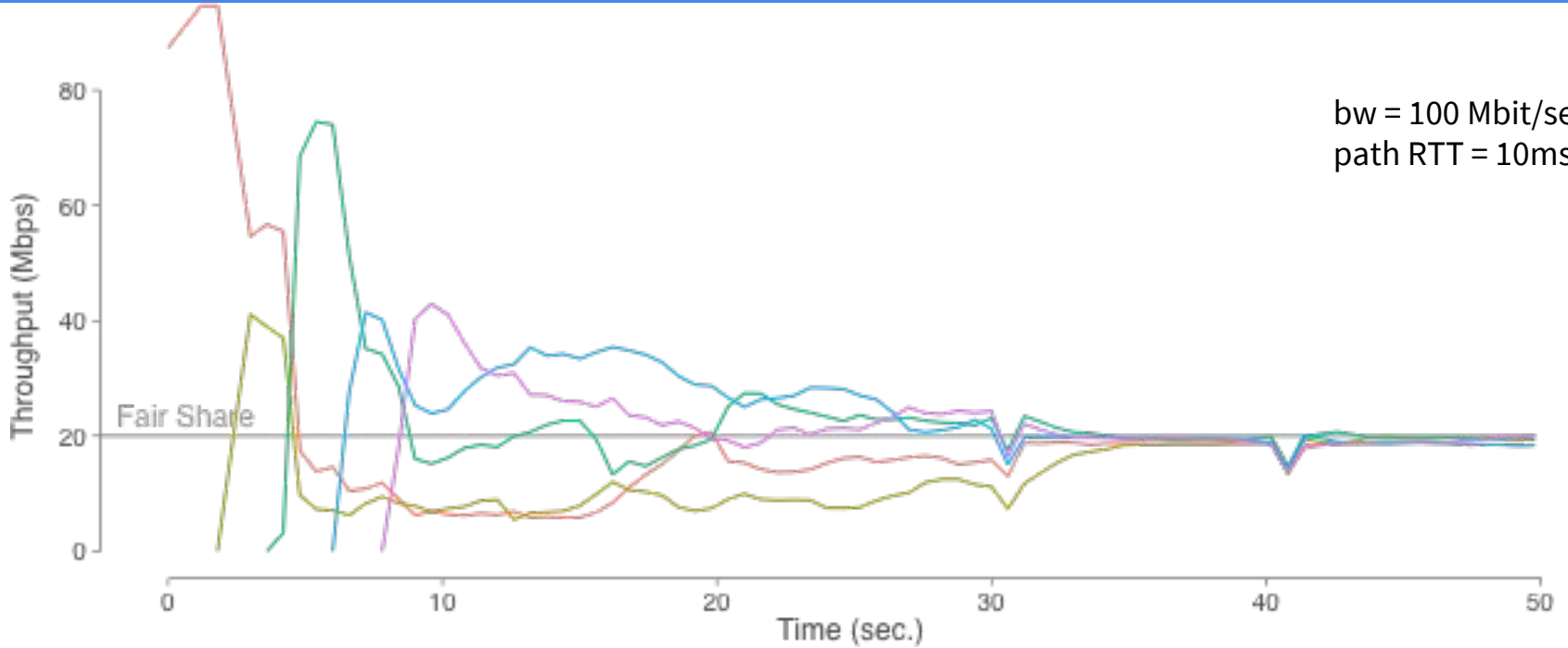
BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms

BBR: low queue delay, despite excess buffer space

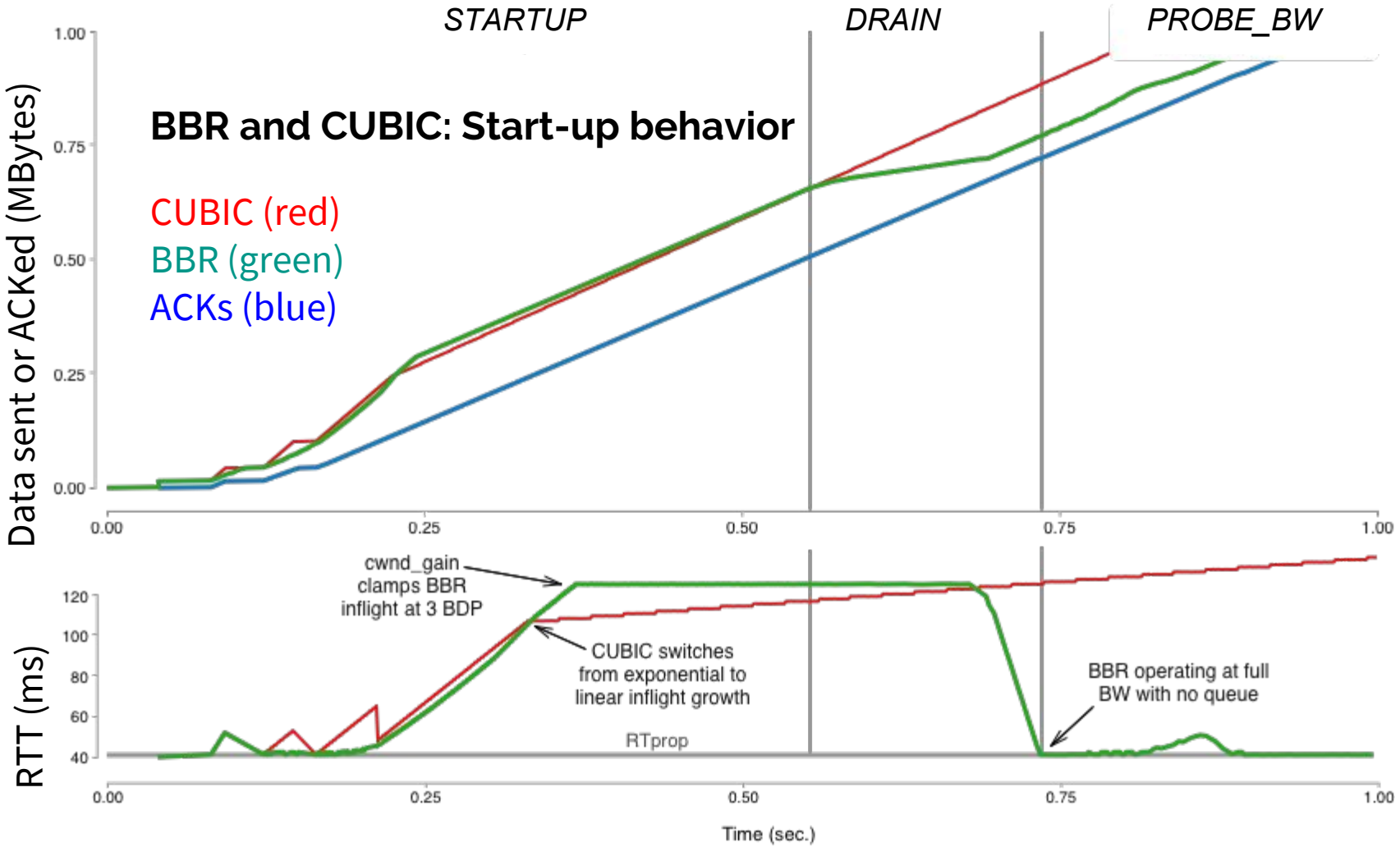


BBR vs CUBIC: synthetic bulk TCP test with 8 flows, bottleneck_bw=128kbps, RTT=40ms

BBR multi-flow convergence dynamics

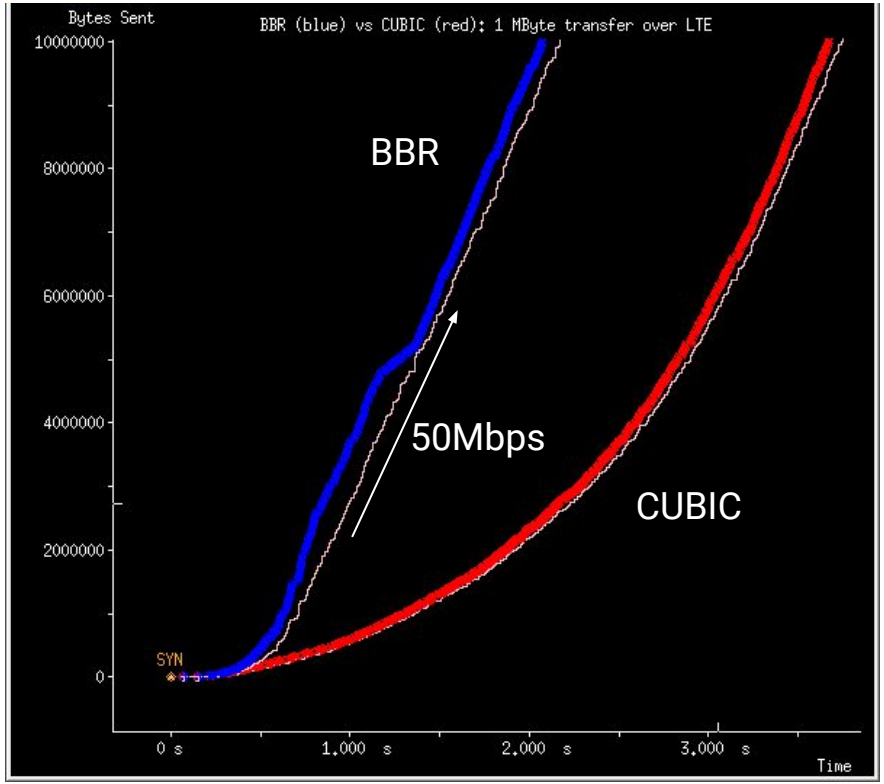


1. Flow 1 briefly slows down to reduce its queue every 10s (PROBE_RTT mode)
2. Flow 2 notices the queue reduction via its RTT measurements
3. Flow 2 schedules to enter slow down 10 secs later (PROBE_RTT mode)
4. Flow 1 and Flow 2 gradually converge to share BW fairly


STARTUP
DRAIN
PROBE_BW
BBR and CUBIC: Start-up behavior
CUBIC (red)
BBR (green)
ACKs (blue)
Data sent or ACKed (MBytes)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
cwnd_gain
clamps BBR
inflight at 3 BDP
CUBIC switches
from exponential to
linear inflight growth
BBR operating at full
BW with no queue
RTprop
RTT (ms)
120
100
80
60
40
Time (sec.)

BBR: faster for short flows, too

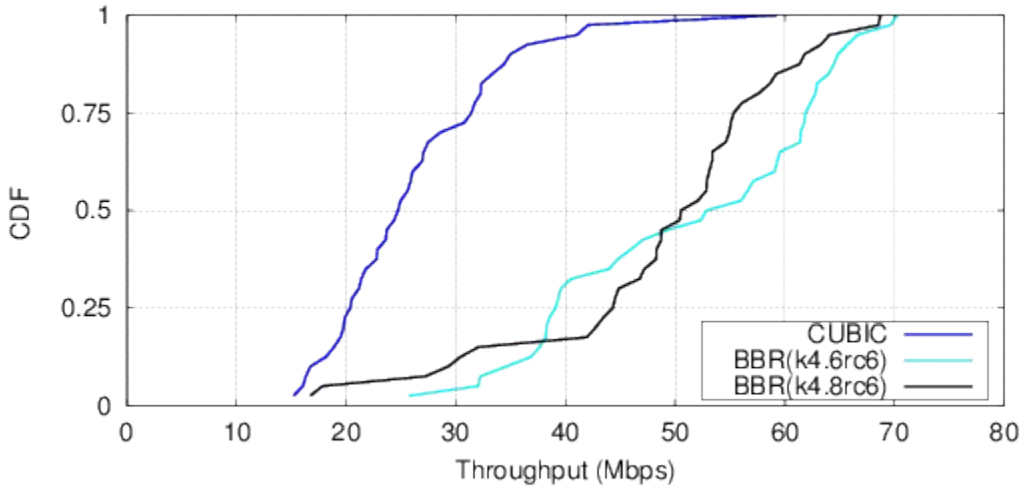
	Cubic (Hystart)	BBR
Initial rate	10 packets / RTT	
Acceleration	2x per round trip	
Exit acceleration	A packet loss or significant RTT increase	Delivery rate plateaus



BBR and Cubic time series overlaid. BBR downloads 1MB 44% faster than Cubic. Trials produced over LTE on Neal's phone in New York

BBR: robust detection of full pipes -> faster start-up

- **BBR** STARTUP: estimate reached full BW if **BW** stops increasing significantly
- **CUBIC** Hystart: estimate reached full BW if **RTT** increases significantly
- But delay (RTT) can increase significantly well before full BW is reached!
  - Shared media links (cellular, wifi, cable modem) use slotting, aggregation
- e.g.: 20 MByte transfers over LTE (source: post by Fung Lee on bbr-dev list, 2016/9/22):



BBR is deployed for WAN TCP traffic at Google

vs CUBIC, BBR yields:

- 2% lower search latency on google.com
- 13% larger Mean Time Between Rebuffers on YouTube
- 32% lower RTT on YouTube
- Loss rate increased from 1% to 2%

Lessons from deploying a new C.C. in the real Internet

- Cellular or Wi-Fi gateways adjust link rate based on the backlog
  - BBR needs to deliberately keep some queue
- Delay alone is an extremely noisy signal
  - e.g. waiting for a slot on a shared medium (radio or cable)
- NICs or middleboxes hold up or decimate ACK packets
  - e.g. one TCP ACK for up to +200 packets
- Token-bucket traffic policers allow bursts, then drop
  - Common in developing regions and mobile carriers
- Middle-boxes modify TCP receive window to throttle sender
- ...

Lessons from implementing a new C.C.

- RFCs are not a sufficient roadmap for implementing a CC protocol
- Need a framework providing much more detailed accounting of what's going on
  - Per-connection totals: SACKed, lost, retransmitted, in-flight
  - Attaching detailed per-TSO-chunk accounting info
    - (Re)transmit/SACK state => enables SACK scoreboard => RFC6675 recovery
    - Transmission times => enables time-based loss reasoning => RACK
    - Delivery rate state => enables delivery-based CC => BBR

Future work

Improving BBR

In principle BBR can be even better:

- Smaller queues: lower delays, less loss, more fair with Reno/CUBIC
  - Potential: cut RTT and loss rate in half for bulk flows
- Higher throughput with wifi/cellular/DOCSIS
  - Potential: 10-20% higher throughput for some paths
- Lower tail latency by adapting magnitude of PROBE_RTT
  - Potential: usually PROBE_RTT with cwnd = 0.75*BDP instead of cwnd=4

End goal: improve BBR to enable it to be the default congestion control for the Internet

We have some ideas for tackling these challenges

We also encourage the research community to dive in and improve BBR!

Following are some open research areas, places where BBR can be improved...

Open research challenges and opportunities with BBR

Some of the areas with work (experiments) planned or in progress:

- Reducing queuing/losses on shallow-buffered networks and/or with cross-traffic:
  - Quicker detection of full pipes at startup
  - Gentler PRR-inspired packet scheduling during loss recovery
  - Refining the bandwidth estimator for competition, app-limited traffic
  - Refining cwnd provisioning for TSO quantization
  - More frequent pacing at sub-unity gain to keep inflight closer to available BDP
  - Explicit modeling of buffer space available for bandwidth probing
- Improving fairness vs. other congestion controls
- Reducing the latency impact of PROBE_RTT by adaptively scaling probing
- Explicitly modeling ACK timing, to better handle wifi/cellular/cable ACK aggregation

Looking for ways to contribute?

- More data is always useful
- And Google uses many kinds of networking traffic, but not all
- So... testing, testing, testing!
  - HPC WANs & LANs
  - wifi LANs
  - AQM: experiences with Codel, PIE, DOCSIS AQM, ...
  - ...
- We love pcaps (or recipes to reproduce behaviors)
- If you're trying to use BBR in a realistic scenario, and it's not working, let us know!

HPC is not the same as Google

(My background is HPC)

- Key differences
  - Many paths where host BW == WAN BW
  - Small integer aggregation (2-10 flows) is common
- We test two cases at global scale:
  - Access bottlenecks near the user (generally over buffered)
  - High aggregation (>1000 flows) fabric and WAN paths

Conclusion

- Loss-based C.C., a 30-year-old approach, is failing on modern fast networks
  - Packet loss signal alone is too late (big queues) or too noisy (underutilization)
- BBR uses BW & RTT (instead of a window) to model the network
  - Goal: maximize bandwidth, then minimize queue
- Deployed on Google.com, YouTube, B4/B2 (for TCP)
  - Better performance for web, video, RPC traffic
  - Open sourced in Linux TCP, QUIC
- Applying BBR to QUIC, FreeBSD TCP @ NetFlix
- Actively working on improving the algorithm

Q & A

https://groups.google.com/d/forum/bbr-dev

research paper, code, mailing list, talks, etc.

Based on work by

Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson

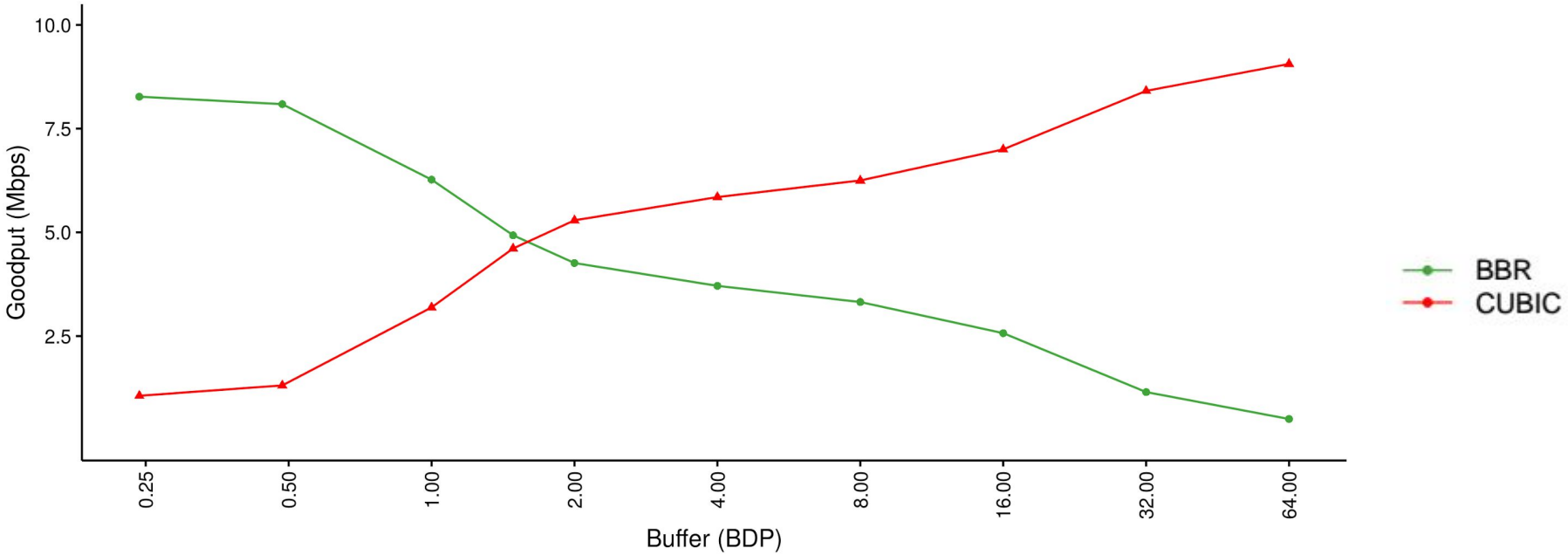
with special thanks to

Eric Dumazet, Ian Swett, Jana Iyengar, Victor Vasiliev, Nandita Dukkipati, Pawel Jurczyk, Biren Roy, David Wetherall, Amin Vahdat, Leonidas Kontothanassis, and {YouTube, google.com, SRE, BWE} teams.

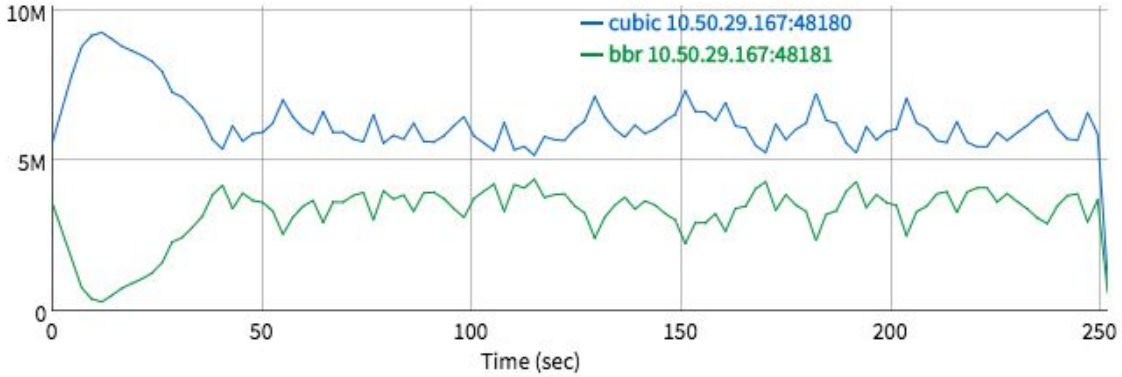
Backup slides...

Improving dynamics w/ with loss-based CC

1xCUBIC v 1xBBR goodput: bw=10Mbps, RTT=40ms, 4min transfer, varying buffer sizes



BBR and loss-based CC in deep buffers: an example



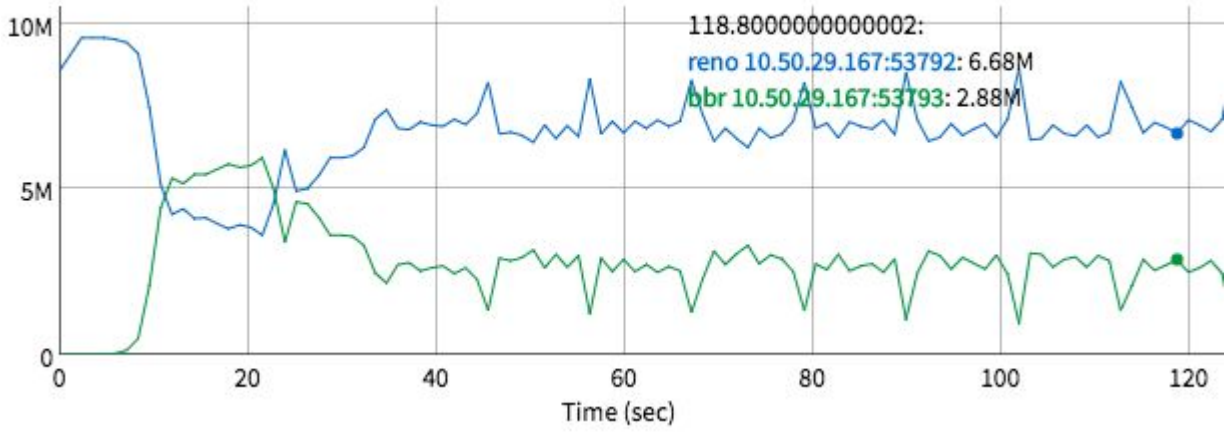
At first CUBIC/Reno gains an advantage by filling deep buffers

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

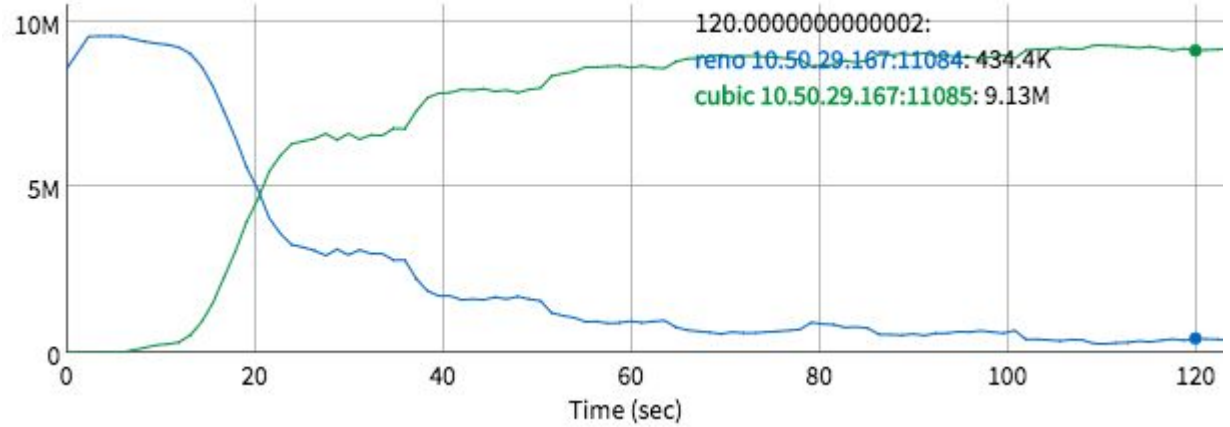
-> CUBIC: 6.31 Mbps vs BBR: 3.26 Mbps

In deep buffers: BBR, CUBIC friendliness to 1x Reno



1x Reno

1x BBR



1x CUBIC

1x Reno

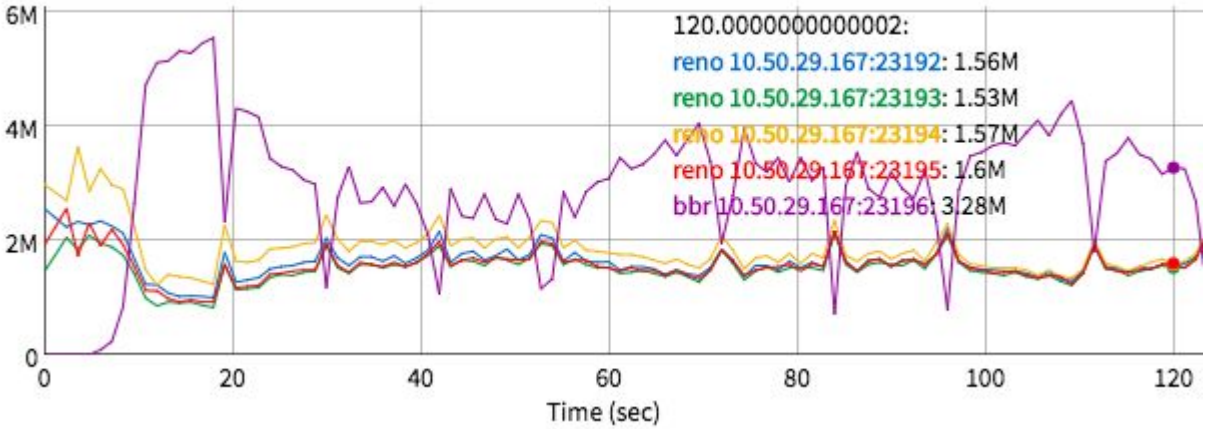
10 Mbps bw

40ms RTT

1 MByte buffer

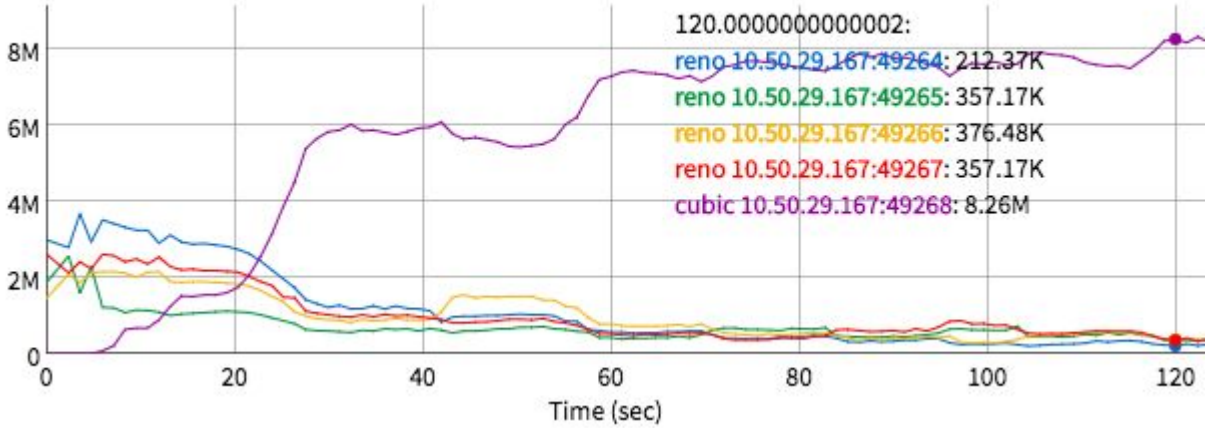
120 sec test

In deep buffers: BBR, CUBIC friendliness to 4x Reno



1x BBR

4x Reno



1x CUBIC

4x Reno

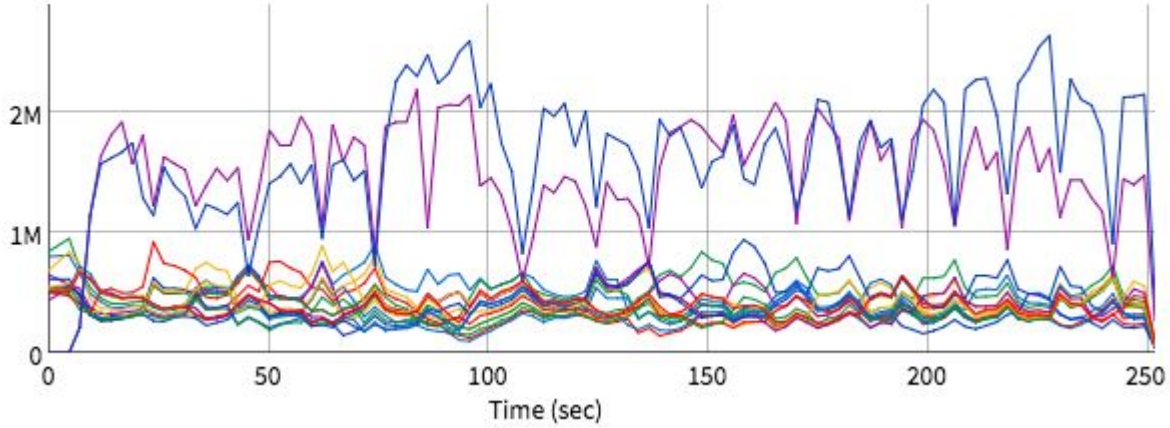
10 Mbps bw

40ms RTT

1 MByte buffer

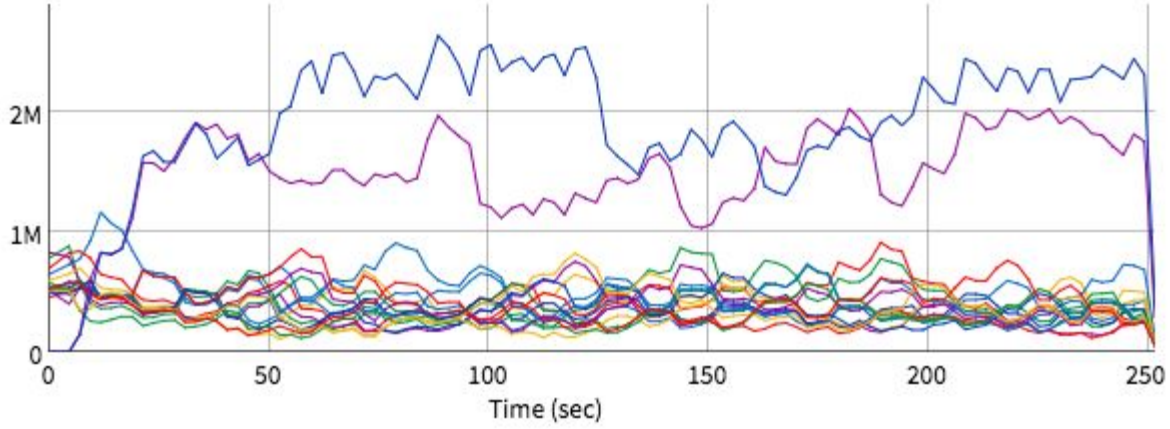
120 sec test

In deep buffers: BBR, CUBIC friendliness to 16x Reno



2x BBR

16x Reno



2x CUBIC

16x Reno

10 Mbps bw

40ms RTT

1 MByte buffer

240 sec test

An opportunity: leveraging BBR's path model

- Linux TCP rate sampling code is available to Linux apps using any C.C.
  - getsockopt(TCP_INFO)
- BBR exports its bandwidth (bw) and two-way propagation delay (min_rtt) estimates
  - getsockopt(TCP_CC_INFO)
  - Better than cwnd/rtt
- Possible applications:
  - Exporting BW to video apps to pick the best video format
  - ...