BBR Congestion Control:

IETF 99 Update

Neal Cardwell, Yuchung Cheng,

C. Stephen Gunn, Soheil Hassas Yeganeh

Ian Swett, Jana Iyengar, Victor Vasiliev

Van Jacobson

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

IETF 99: Prague, July 17, 2017

Outline

- Review of BBR [also see: IETF 97 | IETF 98]
- New Internet Drafts specifying BBR (2)
  - Delivery rate estimation: draft-cheng-iccrg-delivery-rate-estimation
  - BBR congestion control algorithm: draft-cardwell-iccrg-bbr-congestion-control
- Active and upcoming work
- BBR deployment update: BBR now also used for QUIC traffic on google.com/YouTube

The problem: loss-based congestion control

- BBR motivated by problems with loss-based congestion control (Reno, CUBIC)
- Packet loss alone is **not** a good proxy to detect congestion
- If loss comes **before** congestion, loss-based CC gets low throughput
    - 10Gbps over 100ms RTT needs <0.000003% packet loss (infeasible)
    - 1% loss (feasible) over 100ms RTT gets 3Mbps
- If loss comes **after** congestion, loss-based CC bloats buffers, suffers high delays

BBR (Bottleneck BW and RTT)

- **Model** network path: track windowed max BW and min RTT on each ACK
- Control sending rate based on the model
- **Sequentially** probe max BW and min RTT, to feed the model samples
- Seek high throughput with a small queue
  - Approaches maximum available throughput for random losses up to 15%
  - Maintains small, bounded queue independent of buffer depth

	BBR	CUBIC / Reno	Vegas	DCTCP
Congestion signal	(Bottleneck) BW & RTT	Loss	RTT & Loss	ECN & Loss
(Primary) controller	Pacing rate	cwnd	cwnd	cwnd

Delivery rate estimation: Internet Draft

- draft-cheng-iccrg-delivery-rate-estimation
- On each ACK, provides a sample with:
  - 1: estimated rate at which network delivered the last flight of data packets
  - 2: whether this rate was application-limited (app ran out of data to send)
- Why a separate draft for delivery rate estimation?
  - Decomposes BBR into simpler pieces (sampling / modeling / control)
  - Can be implemented separately from BBR (e.g., in Linux TCP)
  - Is useful outside BBR (e.g., picking rate for adaptive bitrate streaming)

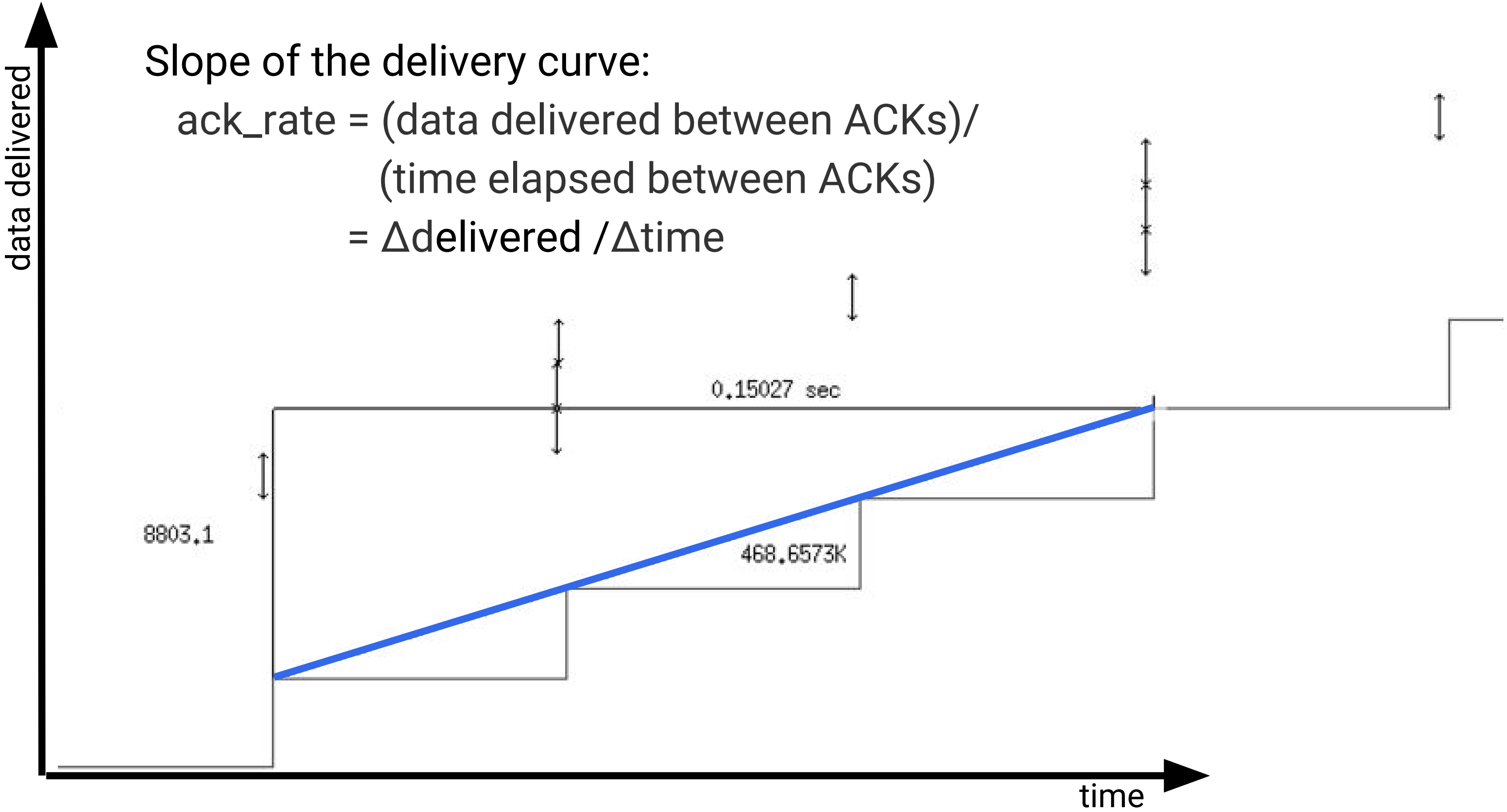
Delivery rate estimation: Design Principles

- Design principles
  - Purely passive
  - Generic: independent of congestion control or transport-specific mechanisms
    - So far: Linux TCP (GPLv2 | BSD style license), QUIC (.cc | .h BSD style license)
  - Track application-limited rate samples
  - Constant time computation
  - Err on the side of underestimating (rather than overestimating)
  - Continuous feedback on any ACK (e.g., SACK, non-SACK dupacks, etc.)
  - Use at least a full round of packets, rather than 1 packet
- Main alternative: packet dispersion metrics (inter-ACK spacing)
  - Various approaches: packet pair, packet train, chirping
  - Challenges:
    - ACK compression, ACK aggregation/decimation, stretch ACKs
    - Jitter/noise

Delivery rate estimation: tracking the ACK rate

Slope of the delivery curve:

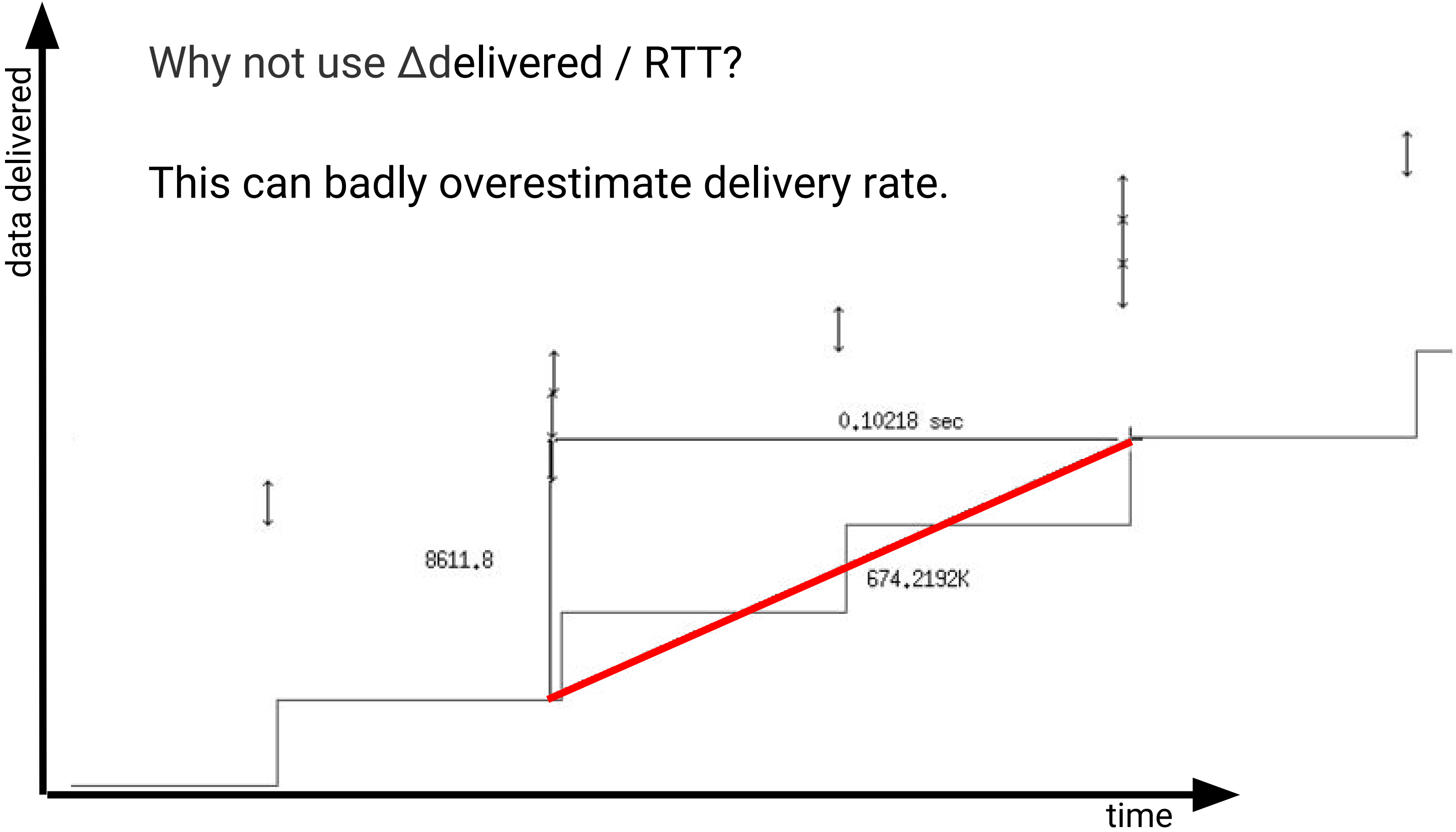
ack_rate = (data delivered between ACKs)/
(time elapsed between ACKs)
= Δdelivered /Δtime



Caveat: why not just Δdelivered/RTT?

Why not use Δdelivered / RTT?

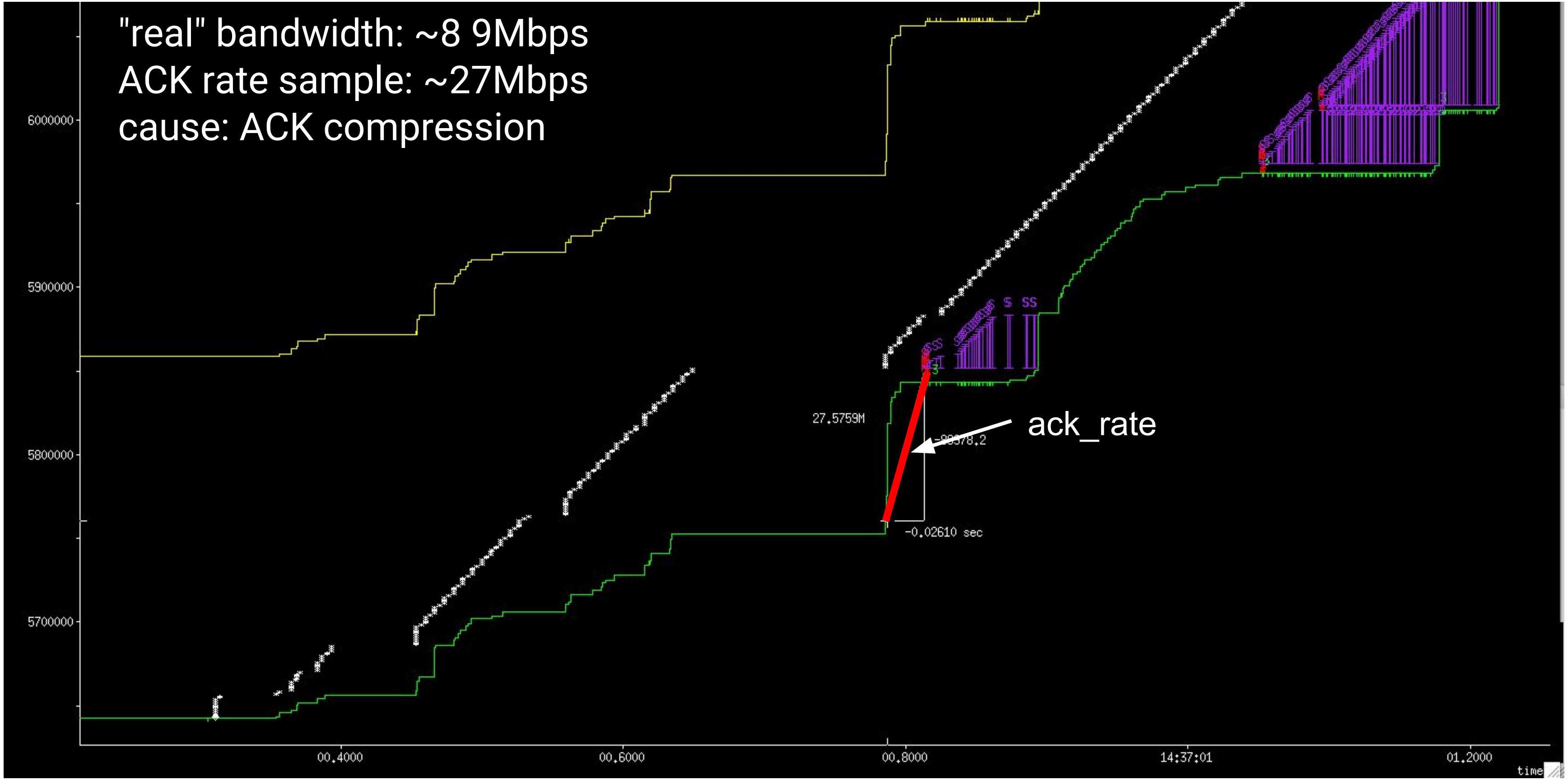
This can badly overestimate delivery rate.



ACK compression

- ACK compression ("aggregation", "decimation", "stretching" ...):
    - What it is: ACK are delayed and then arrive in a burst
    - Cause: receiver or middlebox
    - Frequency: prevalent; very common in wifi, cellular, cable modem paths
    - Result: can result in excessive ACK rate samples

ACK compression: an example



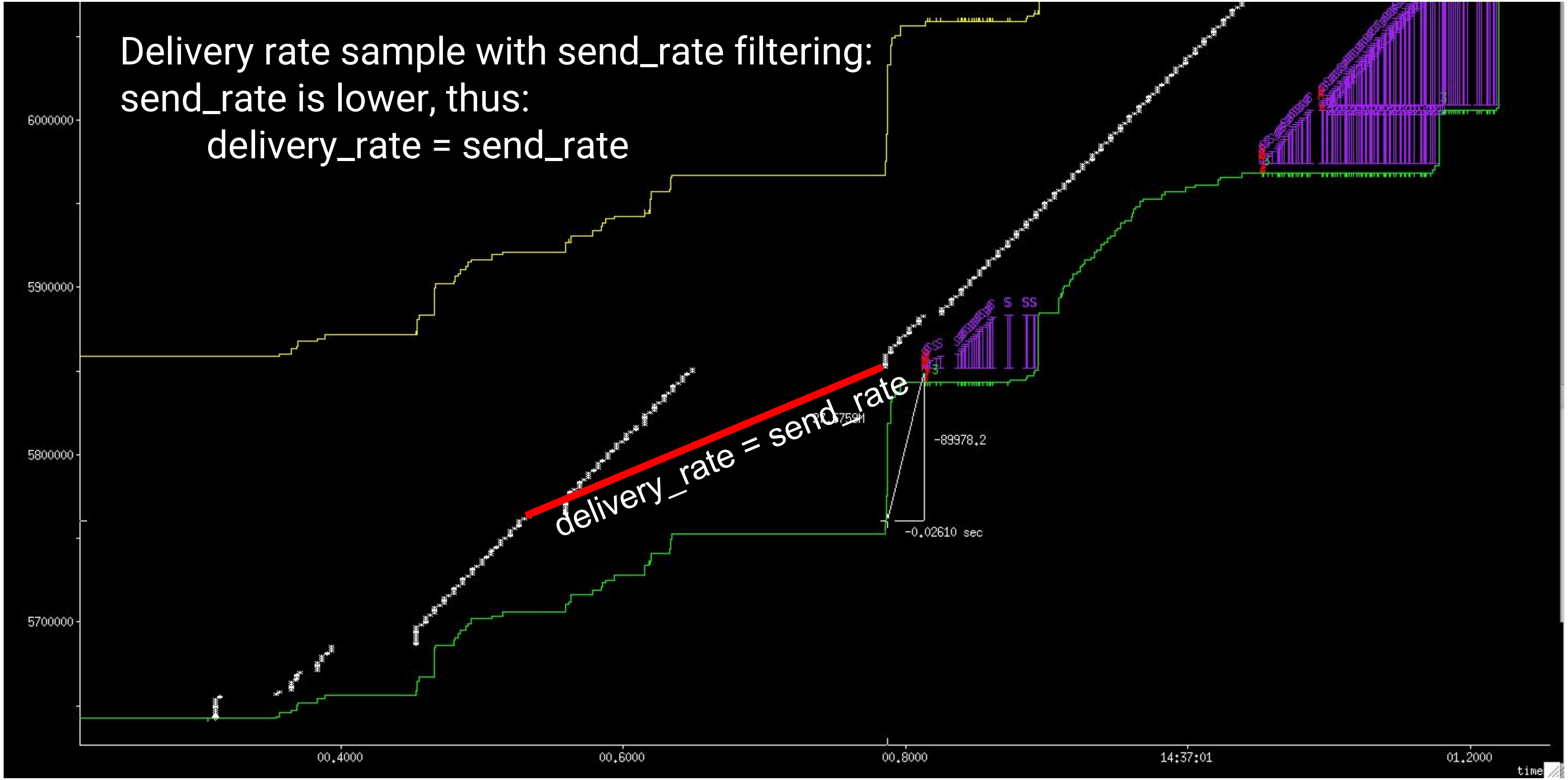
"real" bandwidth: ~8 9Mbps
ACK rate sample: ~27Mbps
cause: ACK compression

Filtering out ACK compression

- Our current approach is to simply filter out "implausibly high" ACK rates:
  - ACK rate cannot physically exceed send rate on a sustained basis
  - For each flight of data delivered between a send and ACK...
    - send_rate: rate at which flight is sent
    - ack_rate: rate at which flight is ACKed
    - delivery_rate = min(send_rate, ack_rate)

- This can be improved, to more thoroughly filter out implausible ACK rates
  - An active area of work for our team

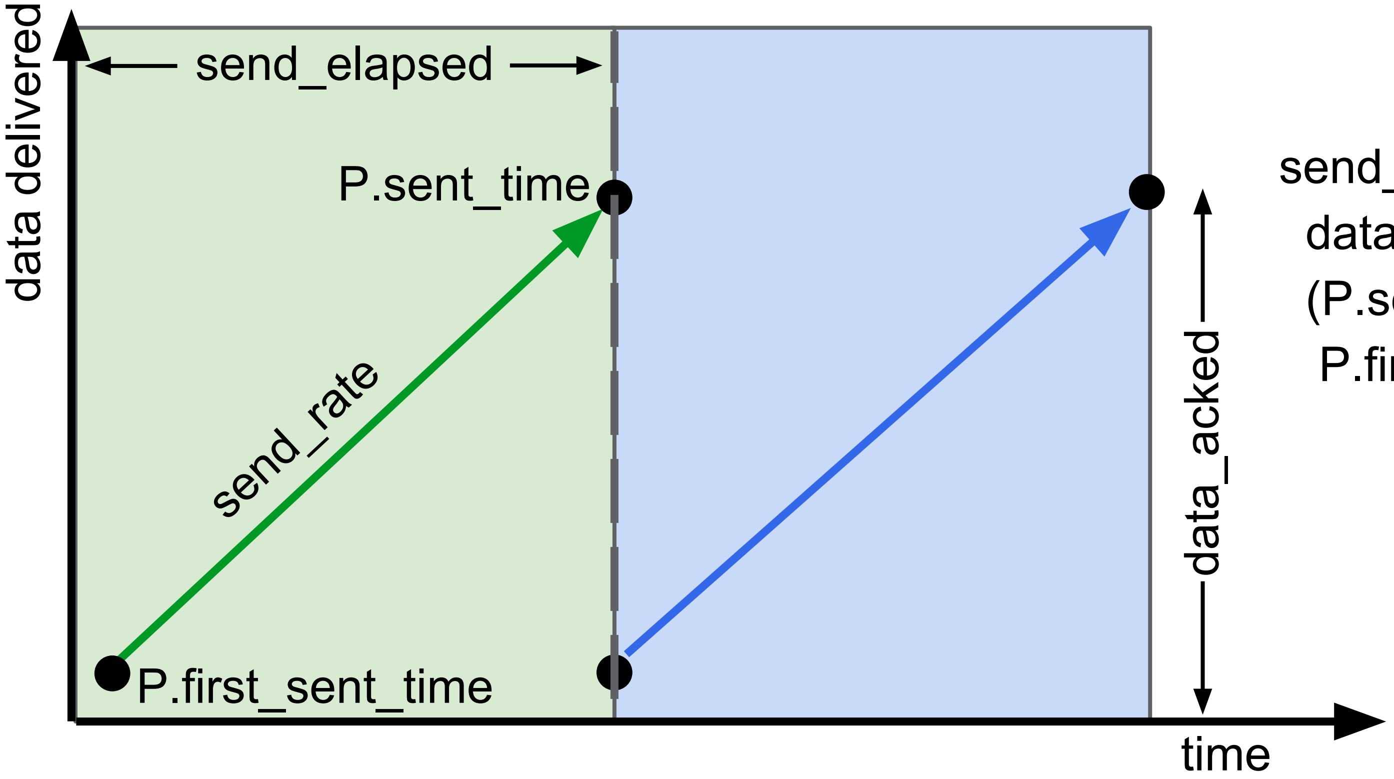
Filtering out ACK compression: an example



Delivery rate sample with send_rate filtering:
send_rate is lower, thus:
delivery_rate = send_rate

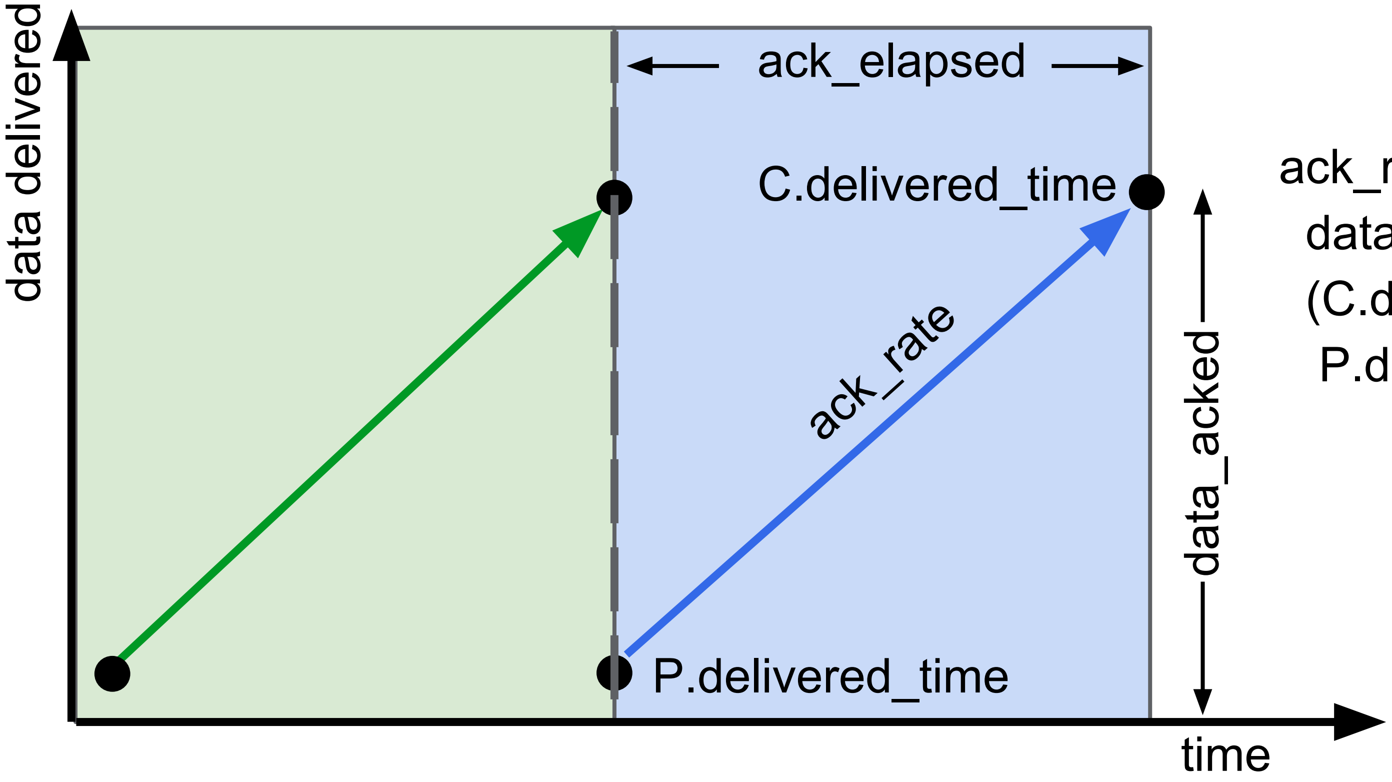
delivery_rate = send_rate

Delivery rate sampling: send_rate



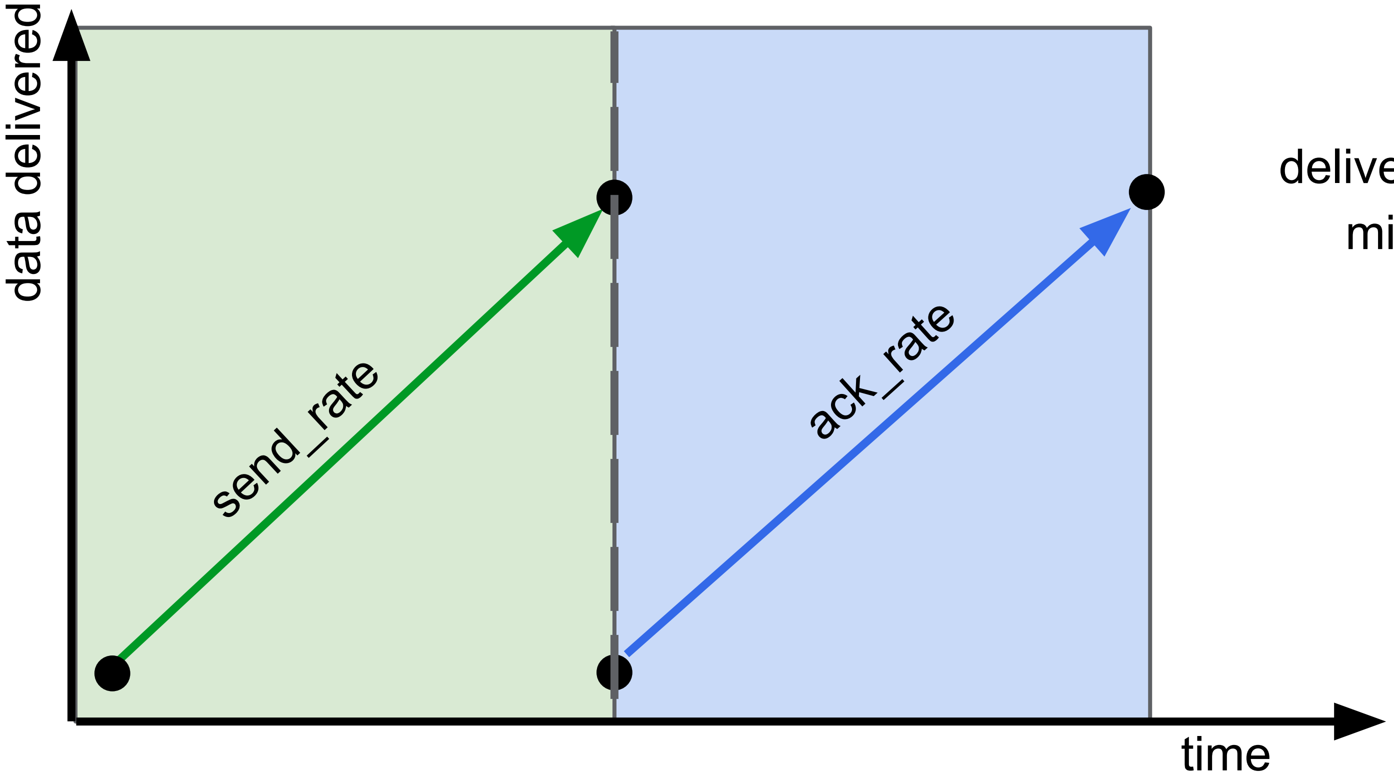
send_rate =
data_acked /
(P.sent_time
P.first_sent_time)

Delivery rate sampling: ack_rate



ack_rate =
data_acked /
(C.delivered_time
P.delivered_time)

Delivery rate sampling: delivery_rate



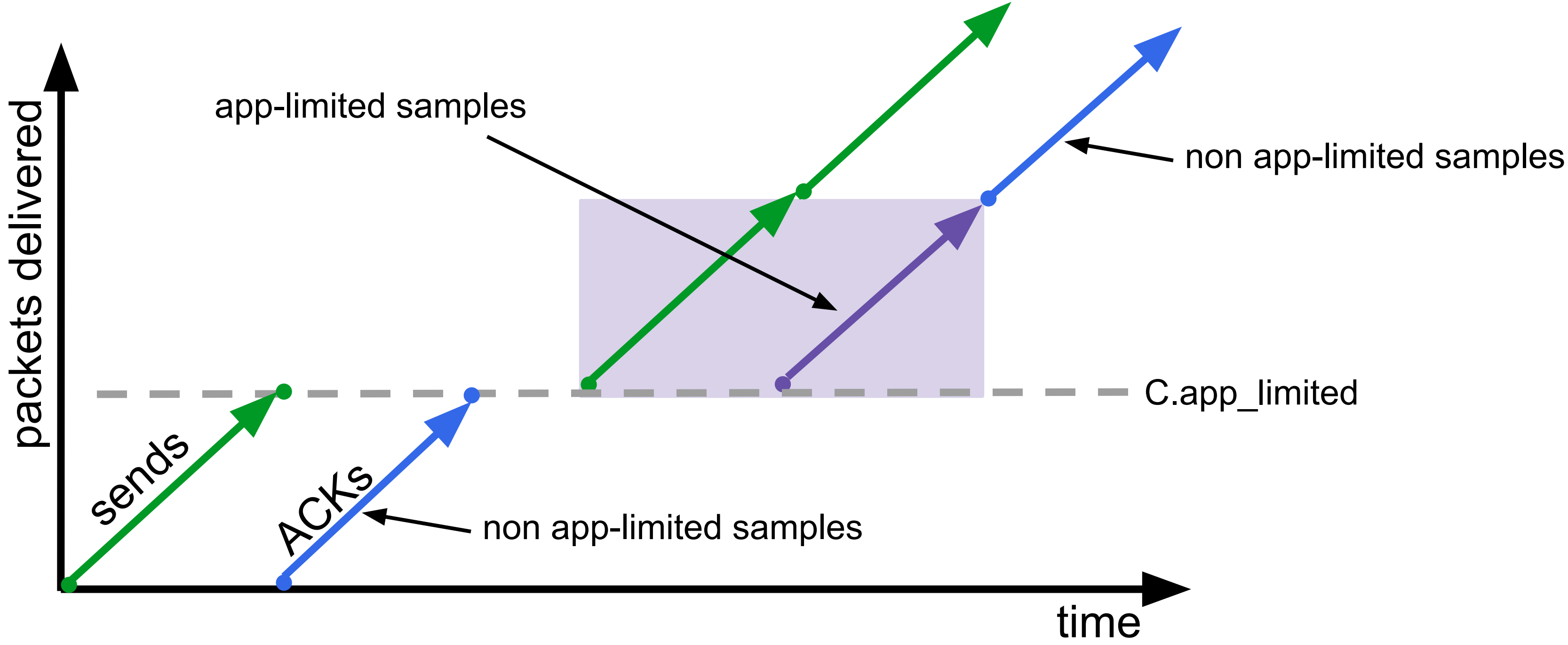
delivery_rate =
    min(send_rate,
        ack_rate)

Detecting application-limited delivery rates

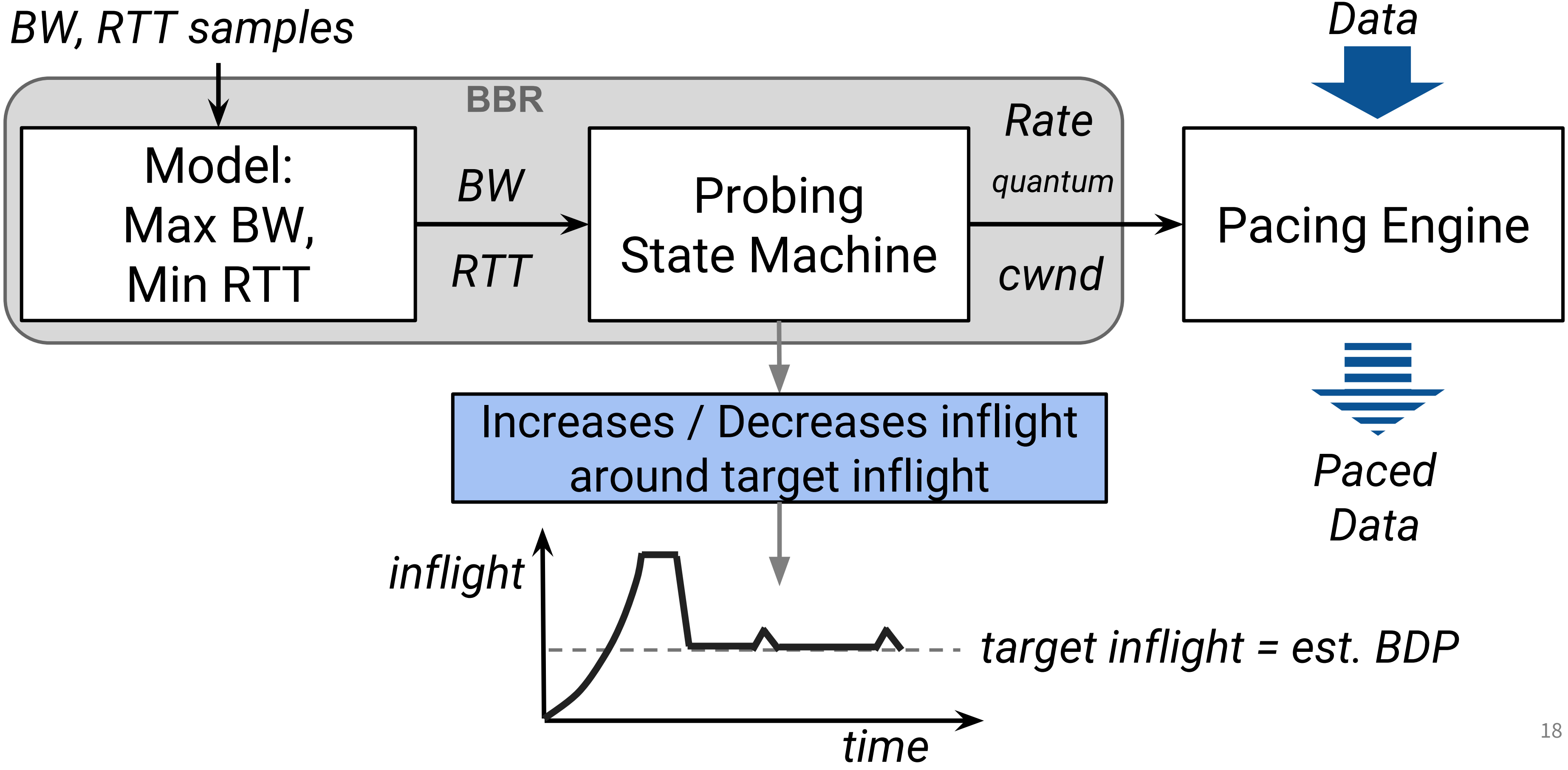
- Goal: track whether rate measures sender behavior (app-limited) or other bottleneck
  - Knowing if a rate sample is app-limited is critical
  - Congestion control wants to adapt to network rate, not application rate
- Rate sample is marked app-limited if app ran out of data to send
  - App-limited moments create a "bubble" of idle time in data pipeline
- Algorithm:
  - Upon app write(), transport marks flow app-limited if all conditions hold:
    - Transport send buffer has less than 1*SMSS of unsent data
    - Flow is not currently in process of transmitting a packet
    - Data estimated to be in flight is less than cwnd
    - All the packets marked lost have been retransmitted
  - Upon ACK, clear app-limited mark if all app-limited packets have been ACKed

Tracking application-limited behavior

When sender becomes app-limited, mark "bubble" with: C.app_limited = C.delivered + C.pipe
Sent packets are marked app-limited for the next round trip (while C.app_limited !=0).
When C.delivered passes C.app_limited, "bubble" is cleared by zeroing C.app_limited.



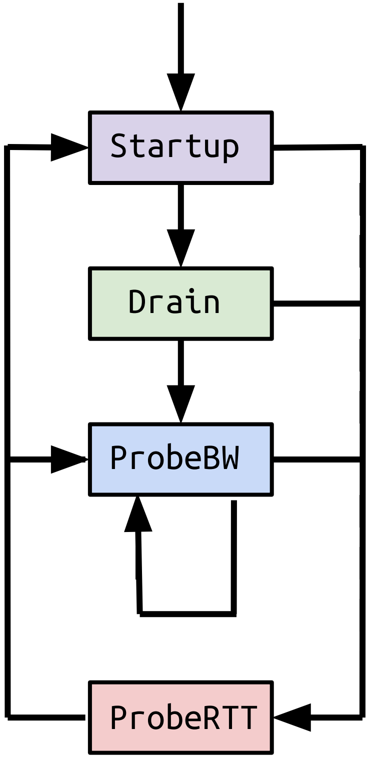
BBR congestion control: the big picture



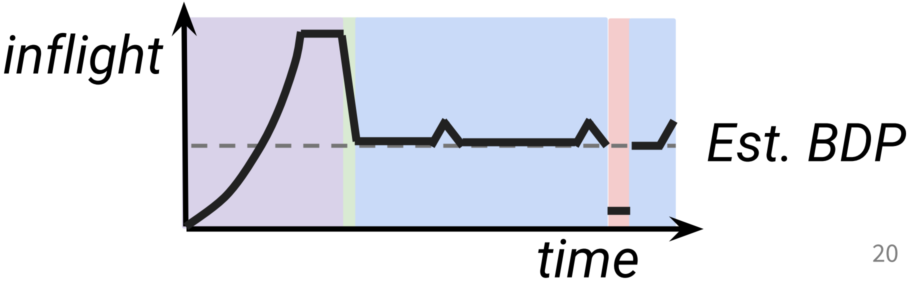
BBR congestion control algorithm: Internet Draft

- draft-cardwell-iccrg-bbr-congestion-control
- Network path model
  - BtlBw: estimated bottleneck bw available to the flow, from windowed max bw
  - RTprop: estimated two-way propagation delay of path, from windowed min RTT
- Target operating point
  - Rate balance: to match available bottleneck bw, pace at or near estimated bw
  - Full pipe: to keep inflight near BDP, vary pacing rate
- Control parameters
  - Pacing rate: max rate at which BBR sends data (primary control)
  - Send quantum: max size of a data aggregate scheduled for send (e.g. TSO chunk)
  - Cwnd: max volume of data allowed in-flight in the network
- Probing state machine
  - Using the model, dial the control parameters to try to reach target operating point

BBR: probing state machine



- State machine for 2-phase sequential probing:
    - 1: raise inflight to probe BtlBw, get high throughput
    - 2: lower inflight to probe RTprop, get low delay
    - At two different time scales: warm-up, steady state...
- Warm-up:
    - Startup: ramp up quickly until we estimate pipe is full
    - Drain: drain the estimated queue from the bottleneck
- Steady-state:
    - ProbeBW: cycle pacing rate to vary inflight, probe BW
    - ProbeRTT: if needed, a coordinated dip to probe RTT



BBR: current areas of research focus

- ACK aggregation (wifi, cellular, DOCSIS)
  - Improving bandwidth estimation
  - Provisioning enough data in flight
- Behavior in shallow buffers
- Datacenter behavior with large numbers of flows

Conclusion

- BBR Internet Drafts are out and ready for review/comments:
  - Delivery rate estimation: draft-cheng-iccrg-delivery-rate-estimation
  - BBR congestion control algorithm: draft-cardwell-iccrg-bbr-congestion-control
- Status of BBR:
  - New: BBR is now deployed for QUIC on Google.com, YouTube
    - With results improvements similar in character to those for TCP
  - All Google/YouTube servers and datacenter WAN backbone connections use BBR
    - Better performance than CUBIC for web, video, RPC traffic
  - Code is available as open source in Linux TCP (dual GPLv2/BSD), QUIC (BSD)
  - Work under way for BBR in FreeBSD TCP @ NetFlix
- Actively working on improving the BBR algorithm
  - Always happy to hear test results or look at packet traces...

Q & A

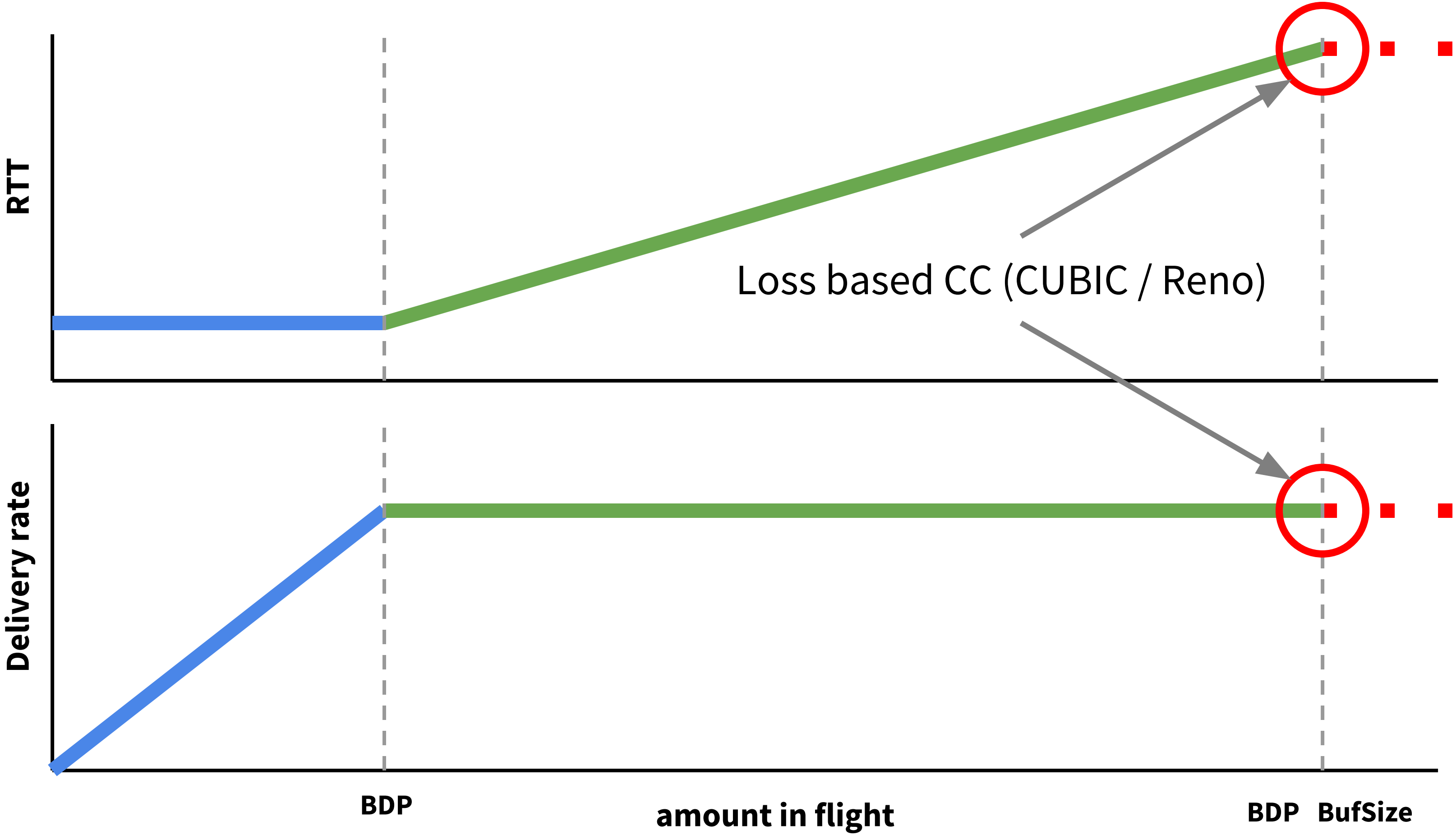
https://groups.google.com/d/forum/bbr dev

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

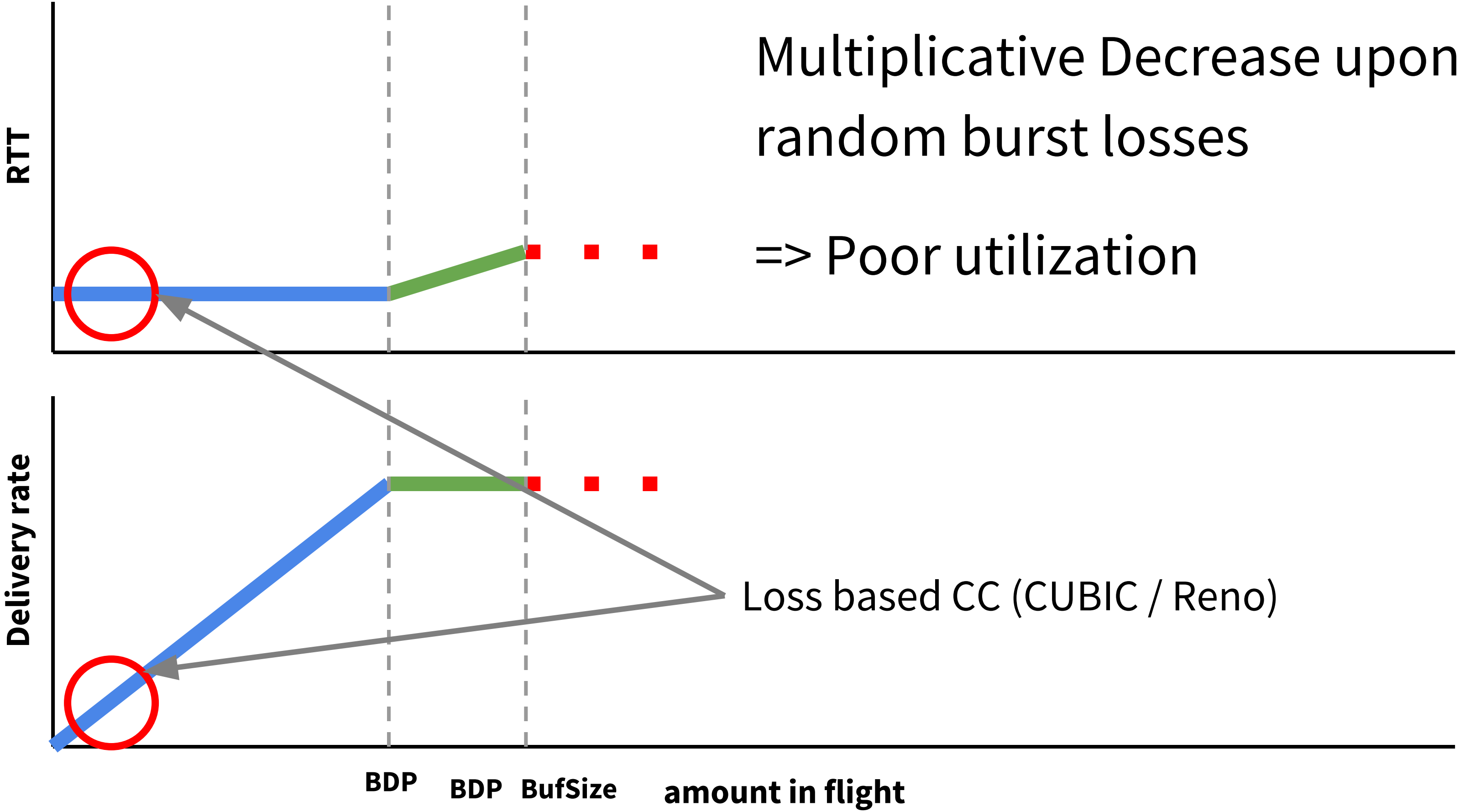
Special thanks to Eric Dumazet, Nandita Dukkipati, Pawel Jurczyk, Biren Roy, David Wetherall, Amin Vahdat, Leonidas Kontothanassis, and {YouTube, google.com, SRE, BWE} teams.

Backup slides from previous BBR talks...

Loss based congestion control in deep buffers



Loss based congestion control in shallow buffers

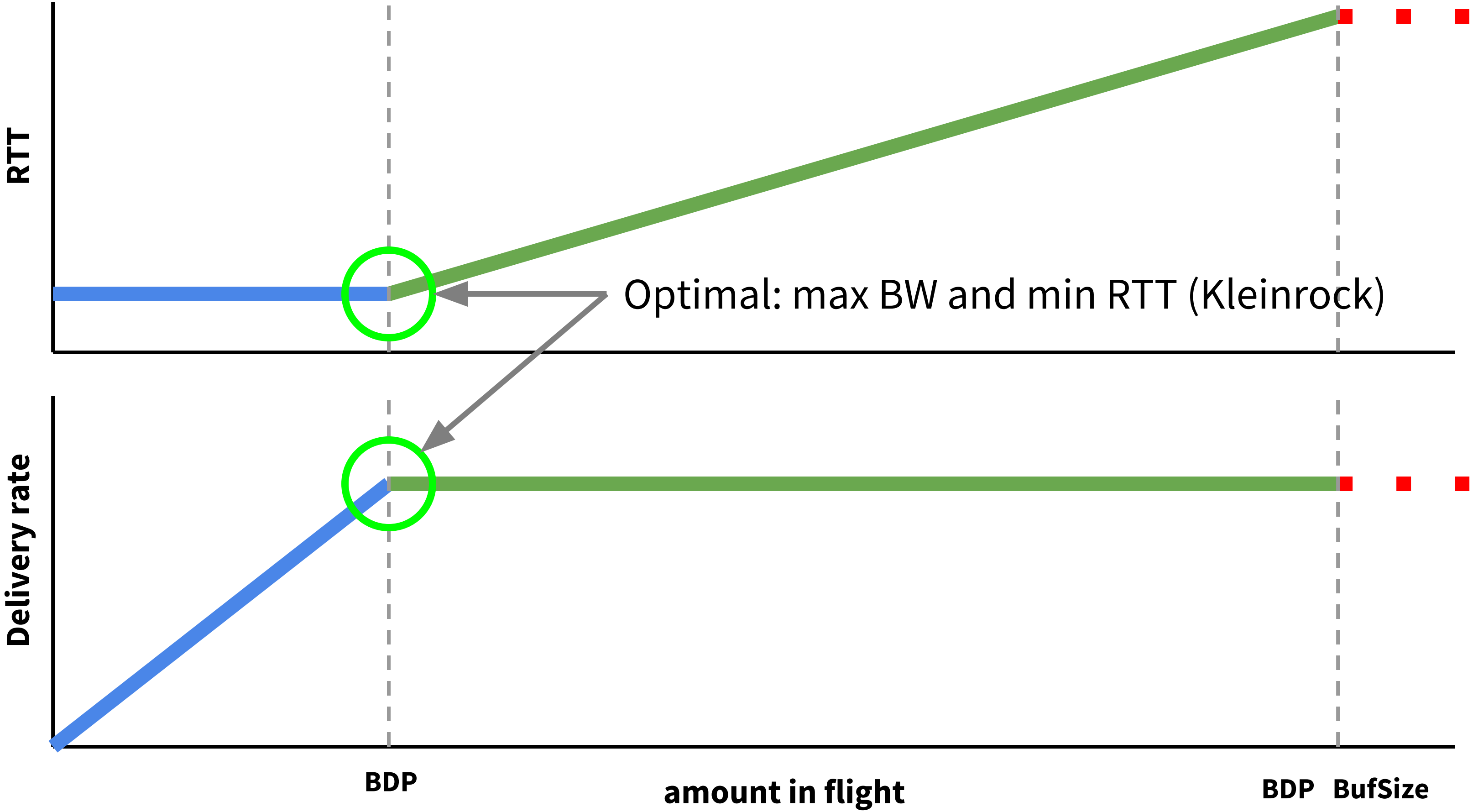


Multiplicative Decrease upon random burst losses

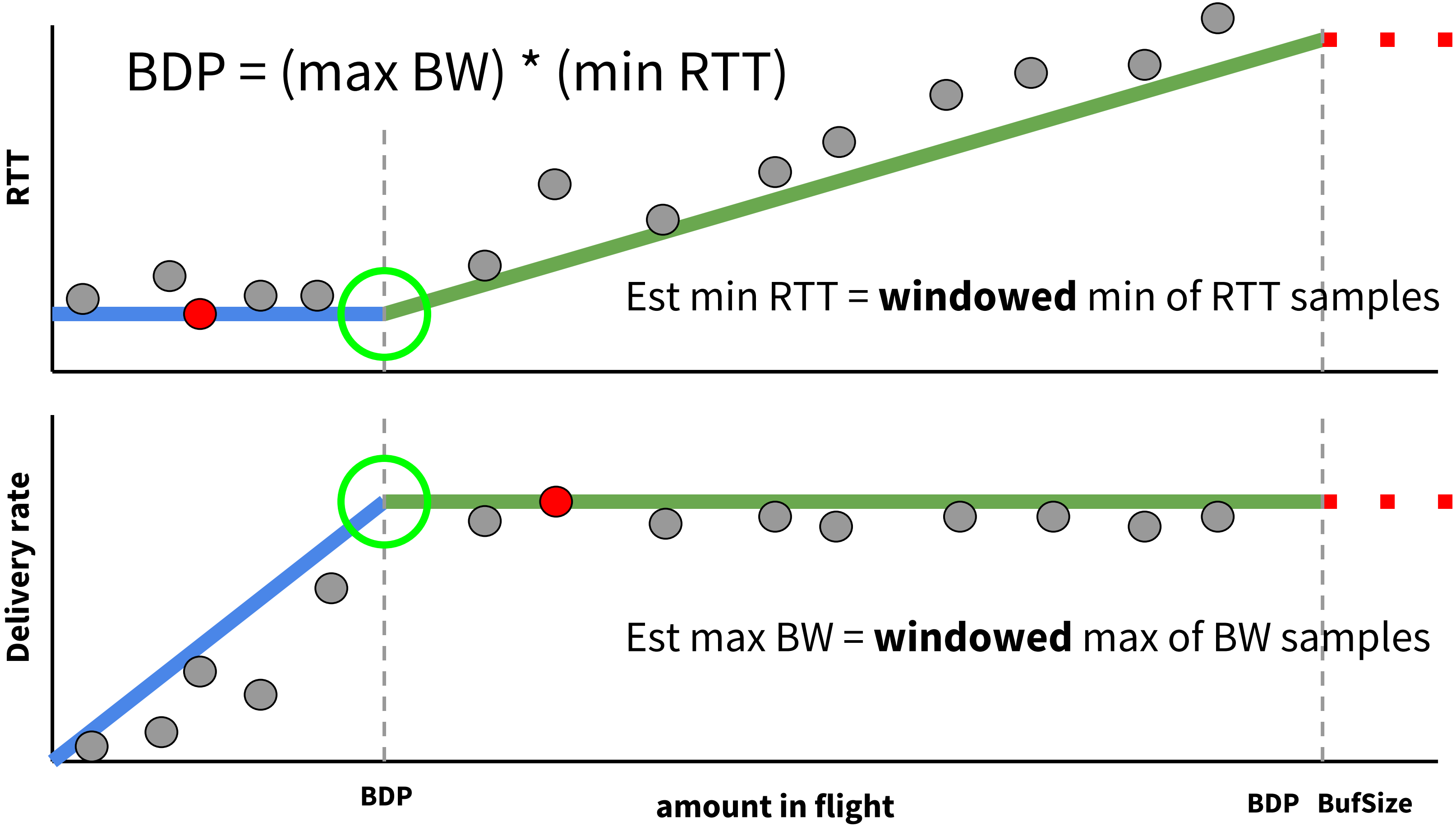
=> Poor utilization

Loss based CC (CUBIC / Reno)

Optimal operating point



Estimating optimal point (max BW, min RTT)

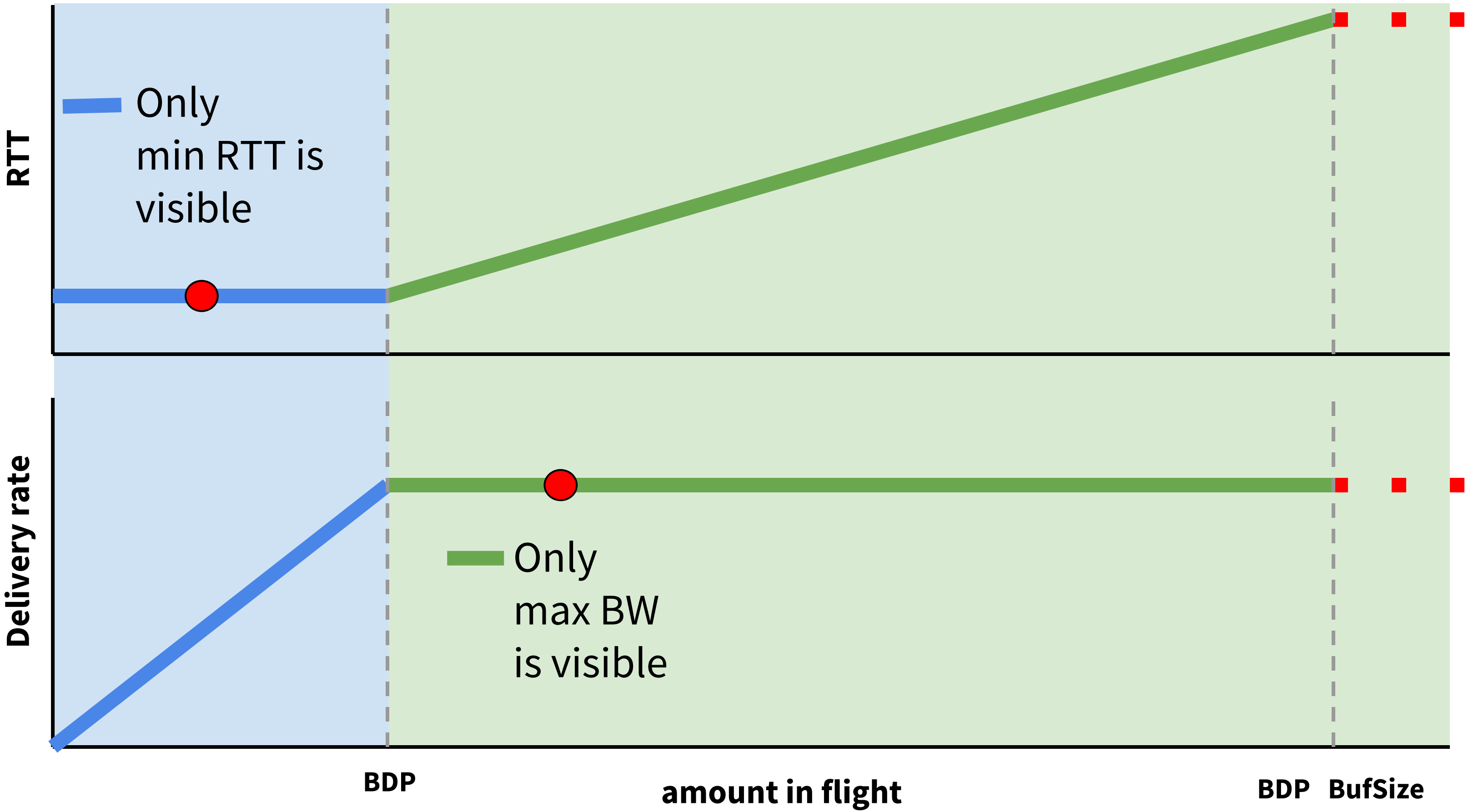


BDP = (max BW) * (min RTT)

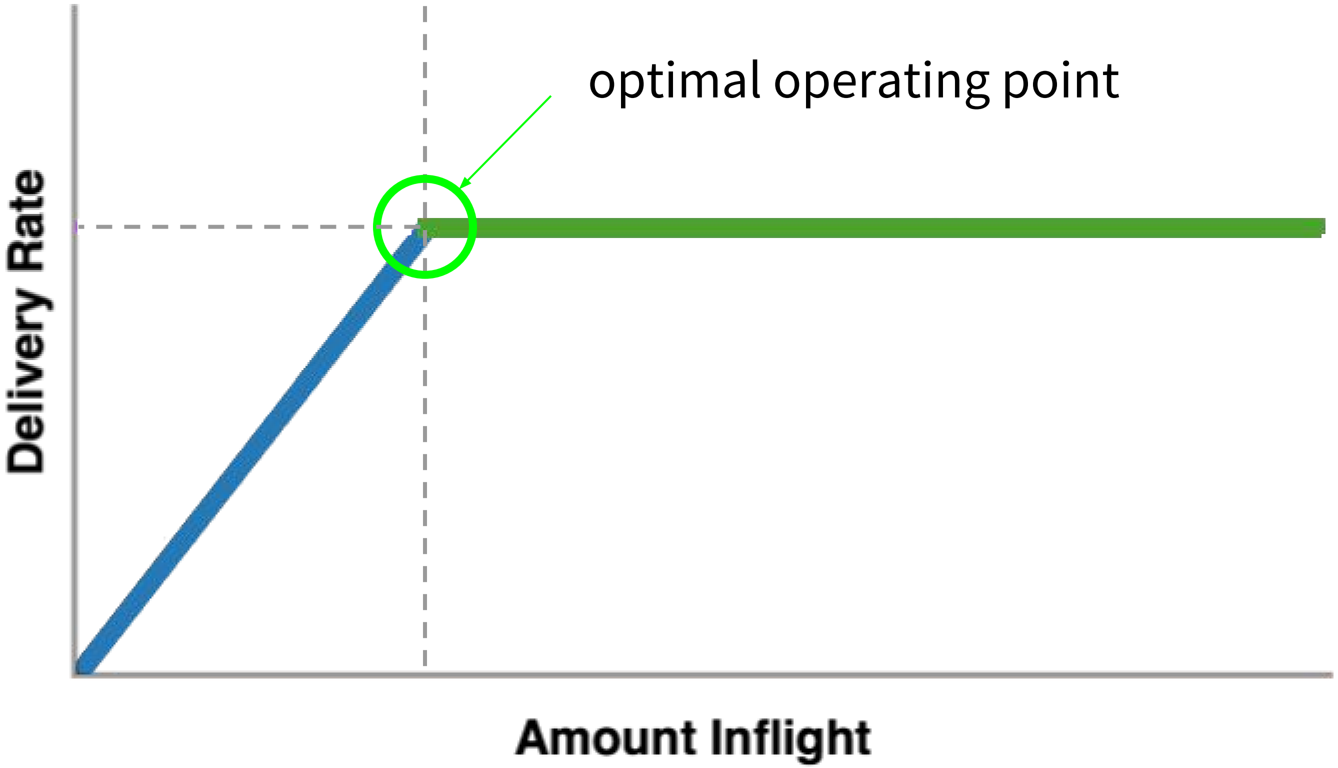
Est min RTT = **windowed** min of RTT samples

Est max BW = **windowed** max of BW samples

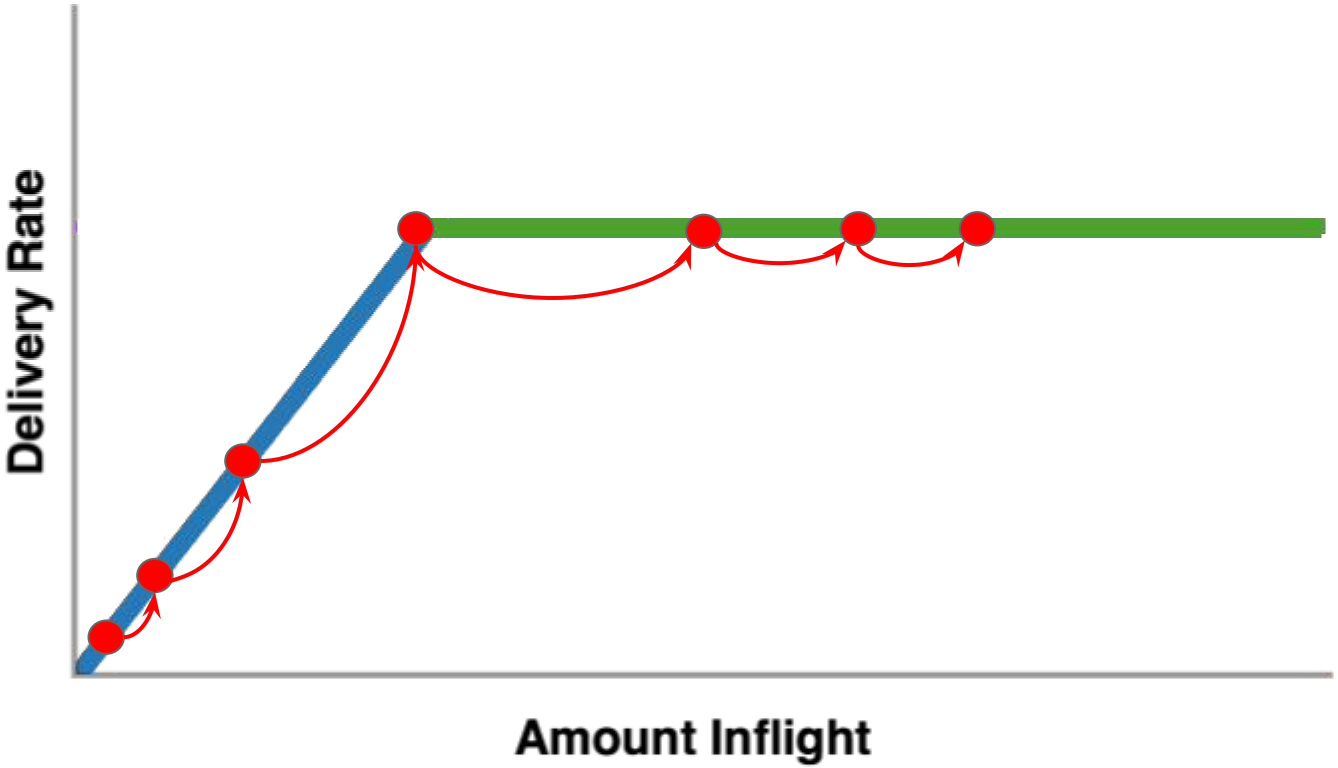
To see max BW, min RTT: probe both sides of BDP



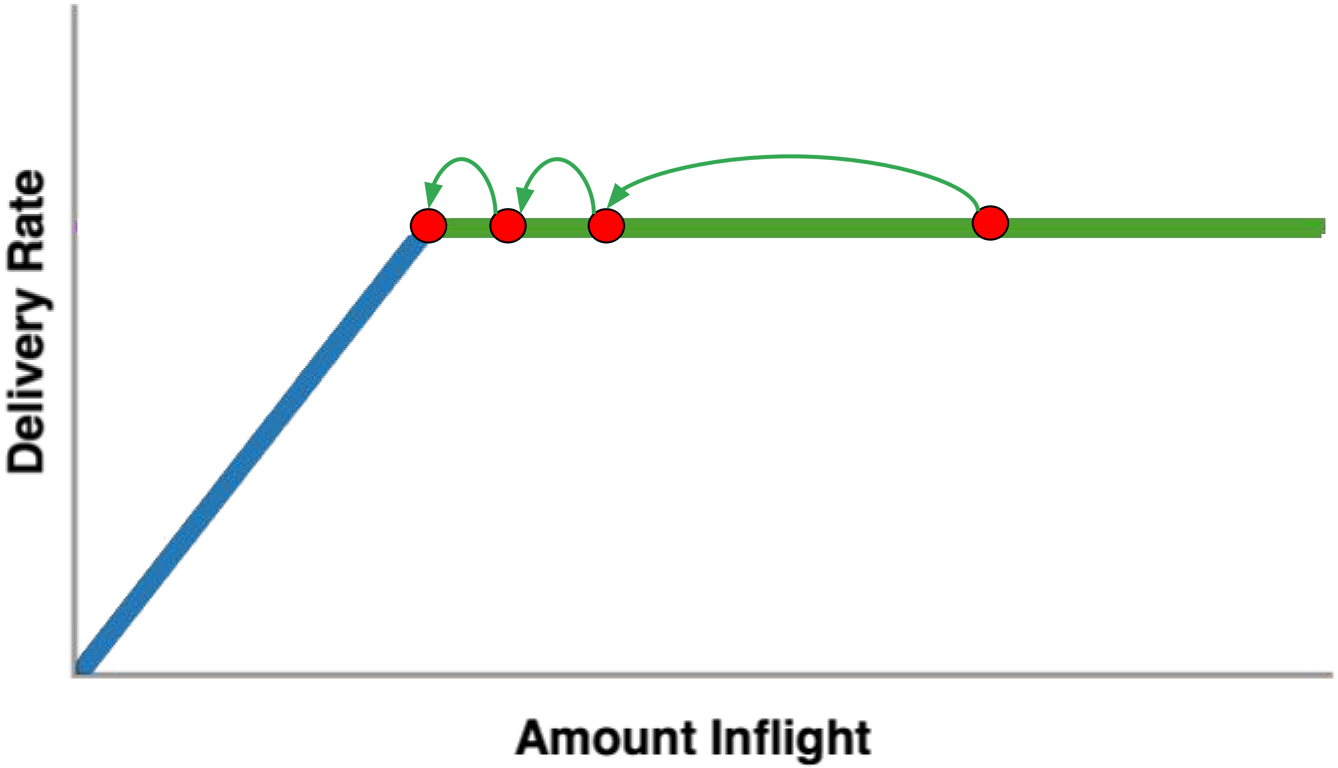
BBR: model based walk toward max BW, min RTT



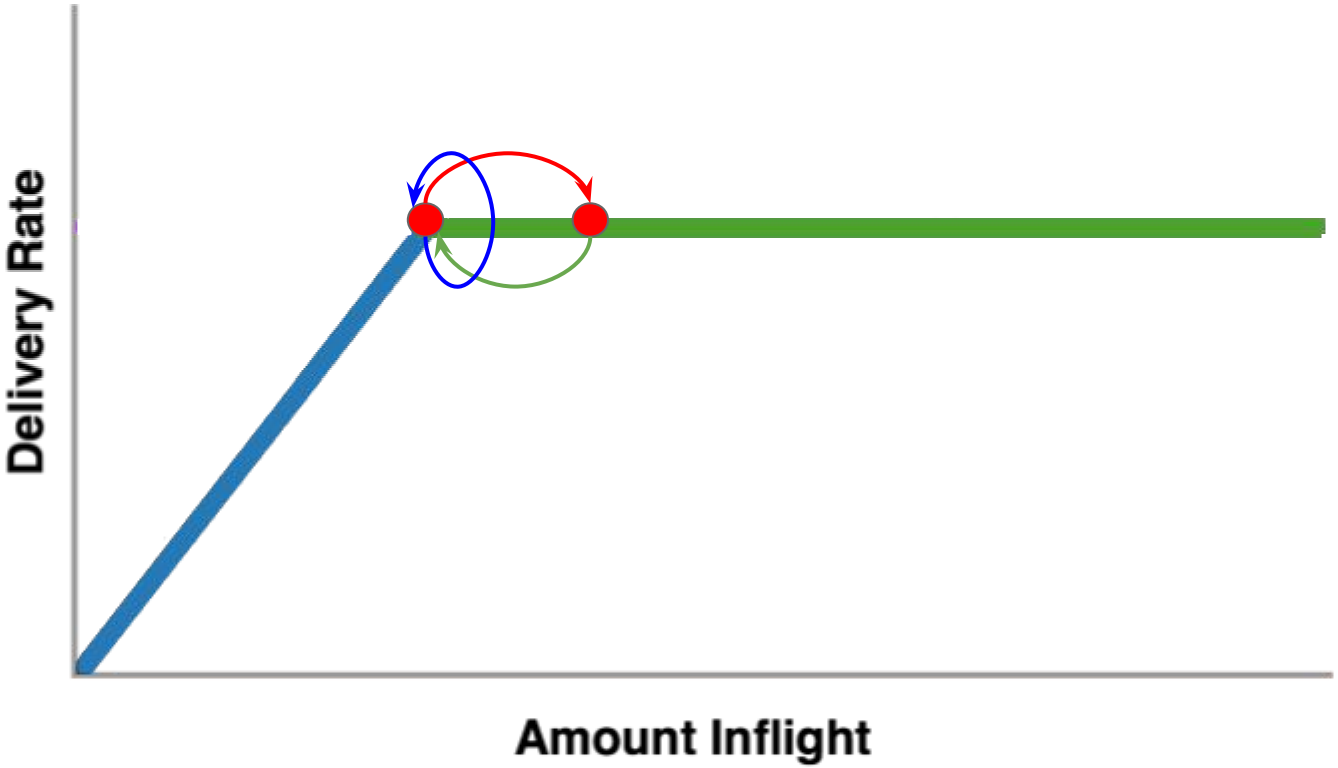
STARTUP: exponential BW search



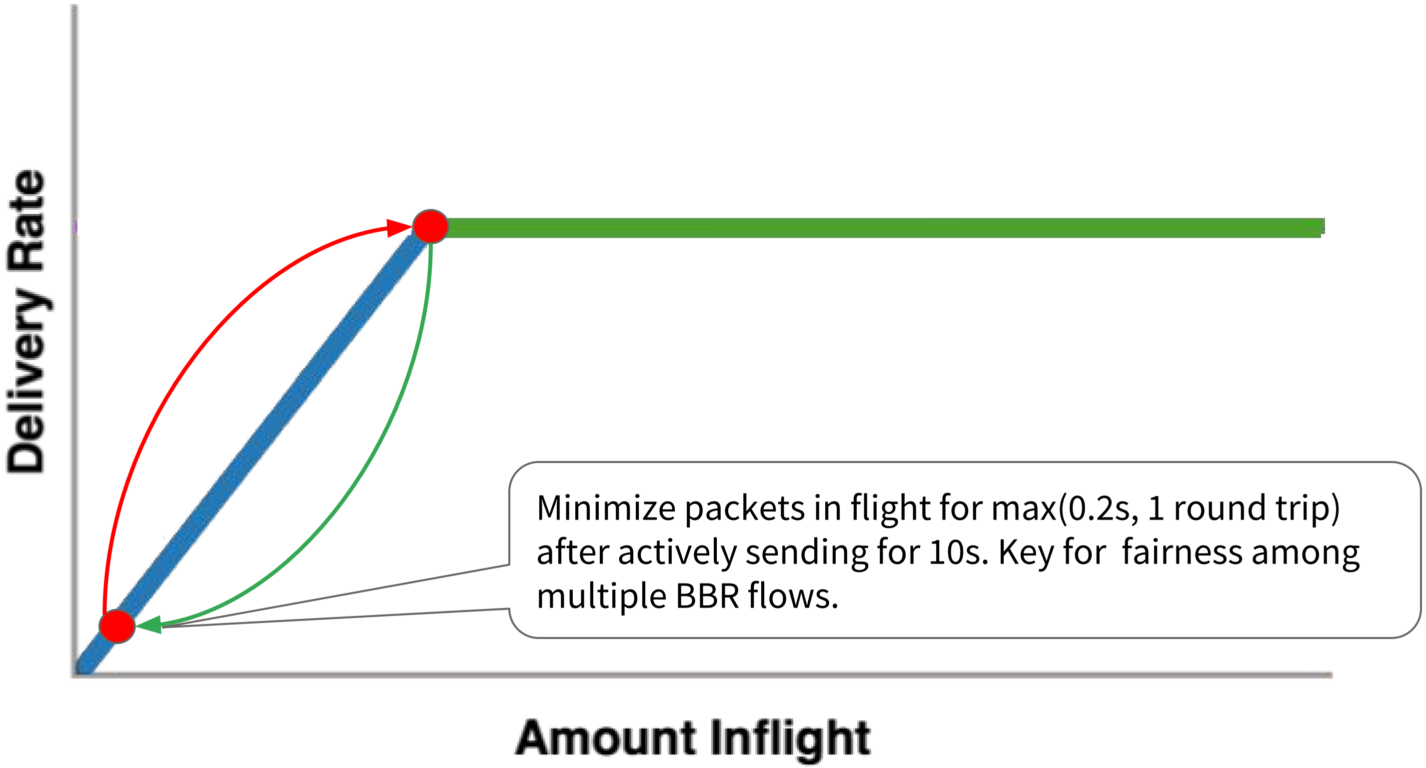
DRAIN: drain the queue created during STARTUP



PROBE_BW: explore max BW, drain queue, cruise



PROBE_RTT: drains queue to refresh min RTT



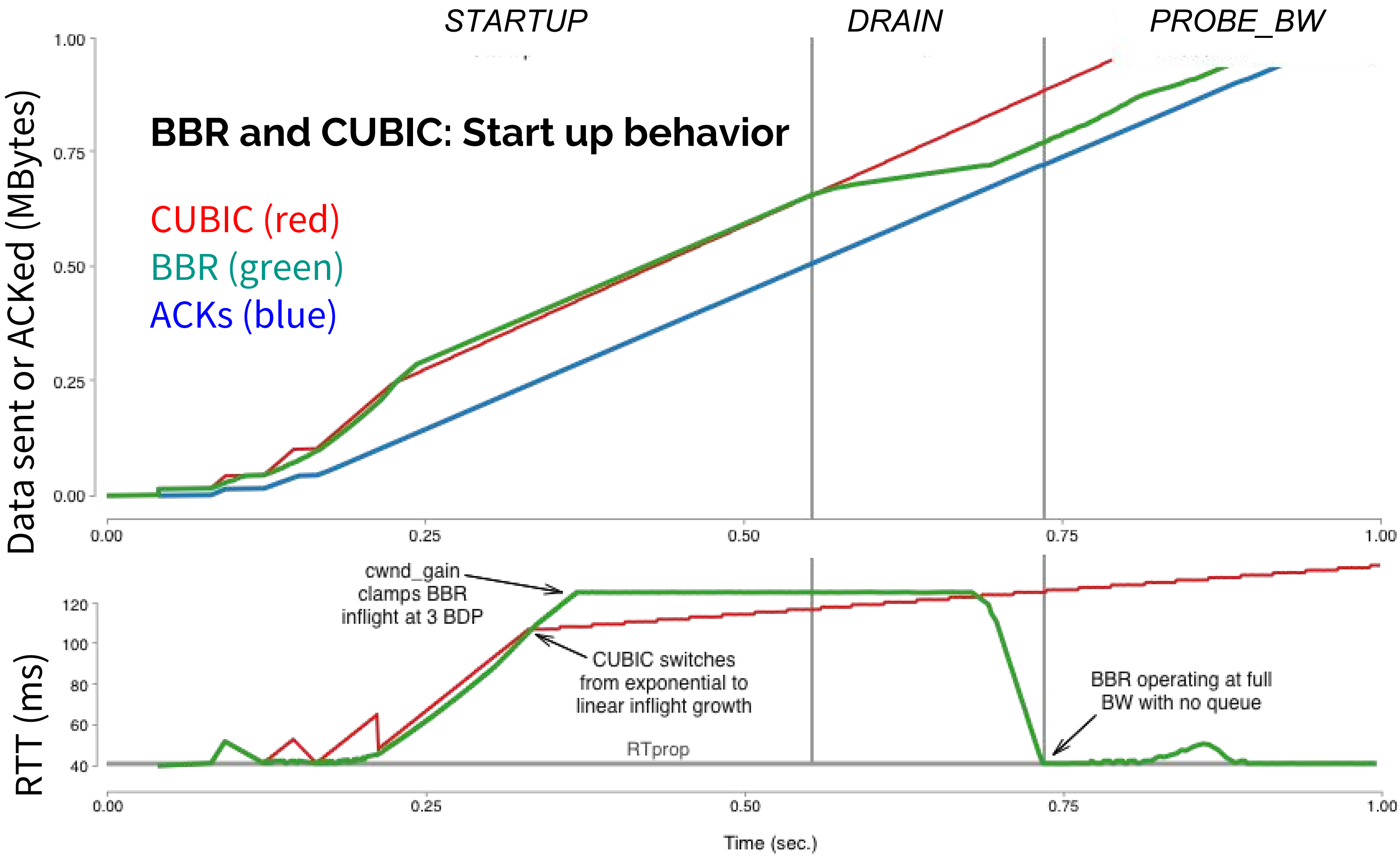
Minimize packets in flight for max(0.2s, 1 round trip) after actively sending for 10s. Key for fairness among multiple BBR flows.

BBR and CUBIC: Start up behavior

CUBIC (red)

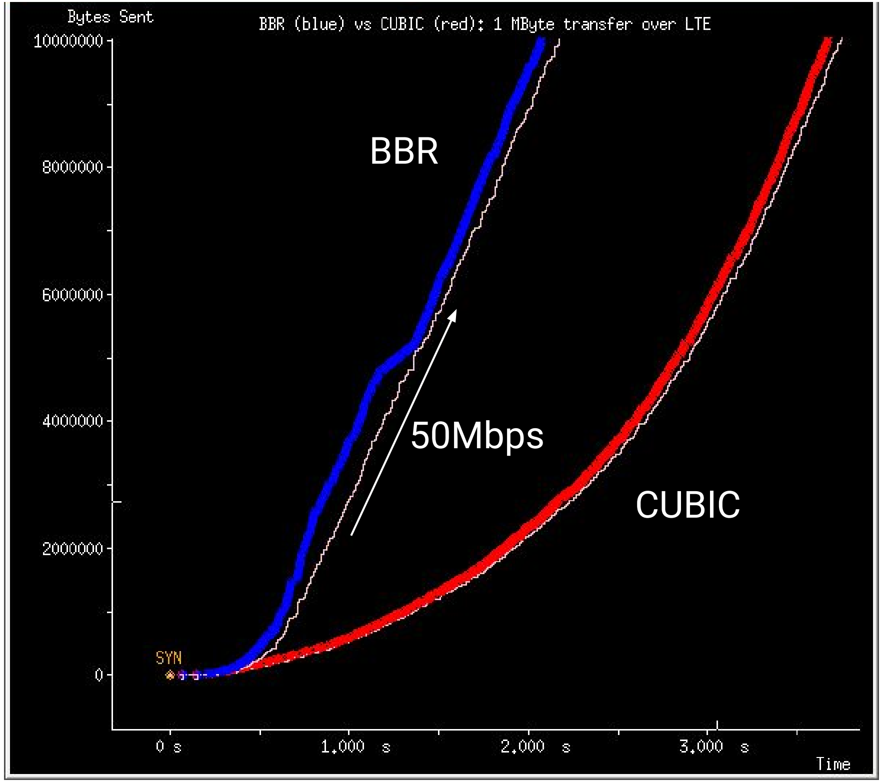
BBR (green)

ACKs (blue)



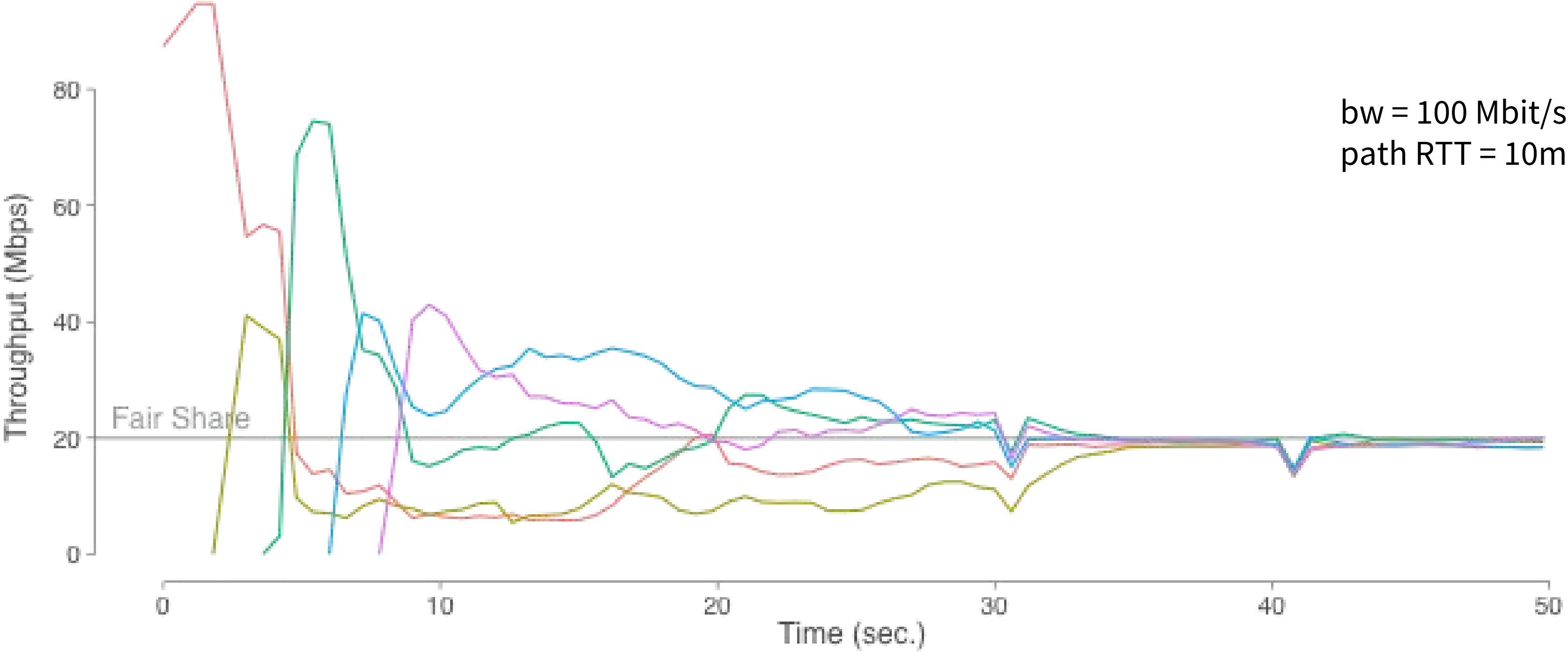
BBR: faster for short flows, too

	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

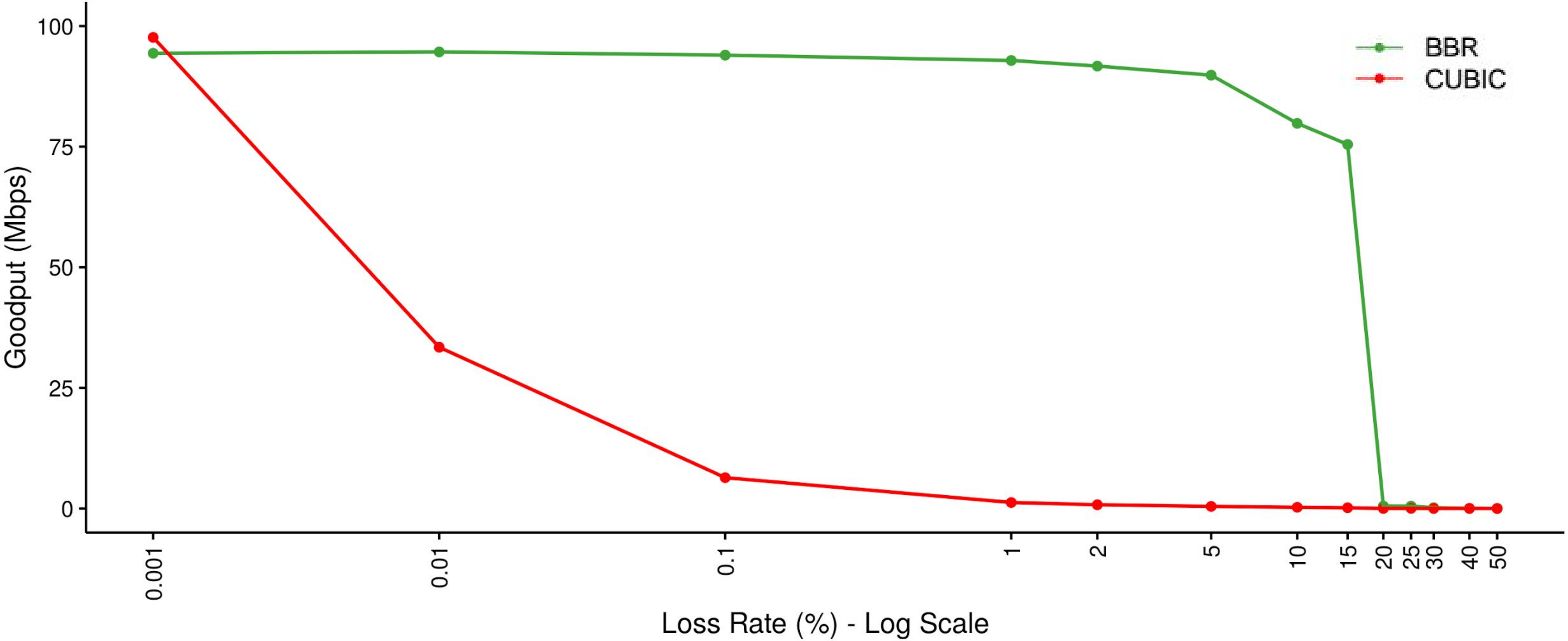
BBR multi flow convergence dynamics



bw = 100 Mbit/sec
path RTT = 10ms

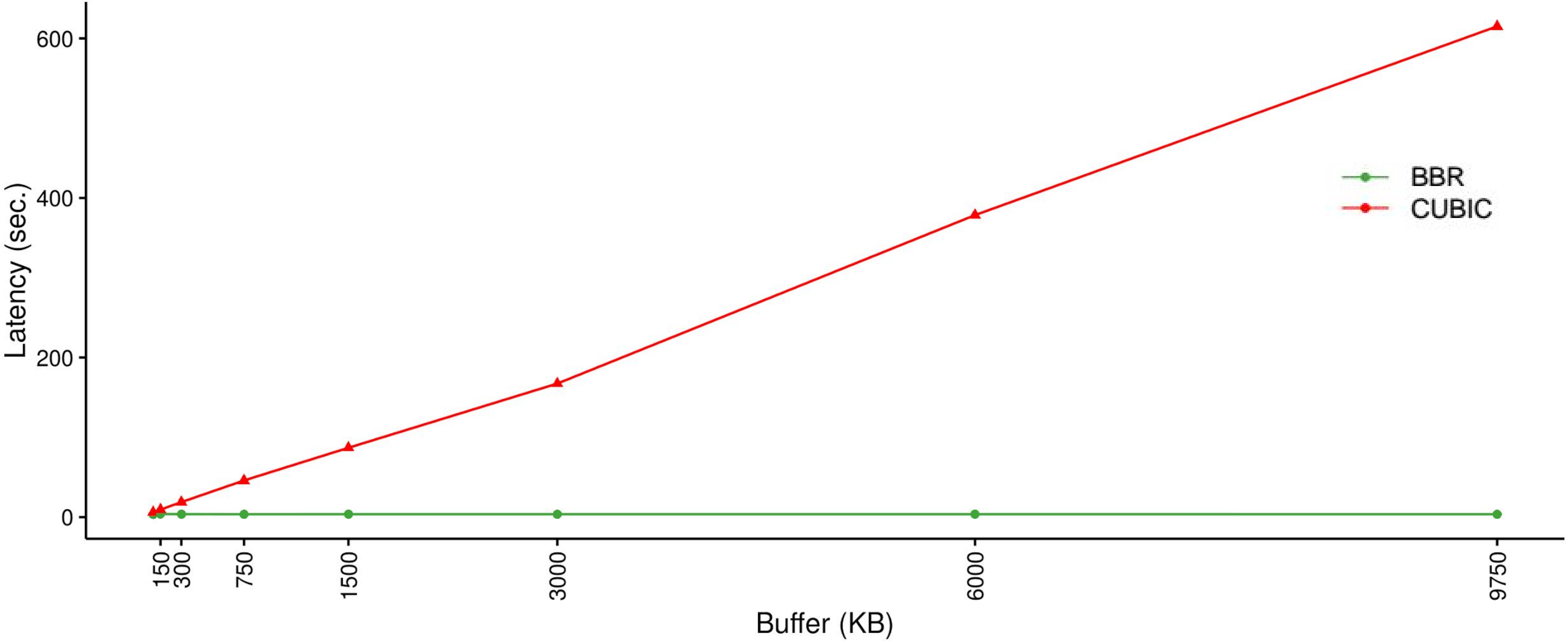
1. Flow 1 briefly slows down to reduce its queue every 10s (PROBE_RTT mode)
2. Flow 2 notices the queue reduction via its RTT measurements
3. Flow 2 schedules to enterslow down 10 secs later (PROBE_RTT mode)
4. Flow 1 and Flow 2 gradually converge to share BW fairly

BBR: fully use bandwidth, despite high packet loss



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

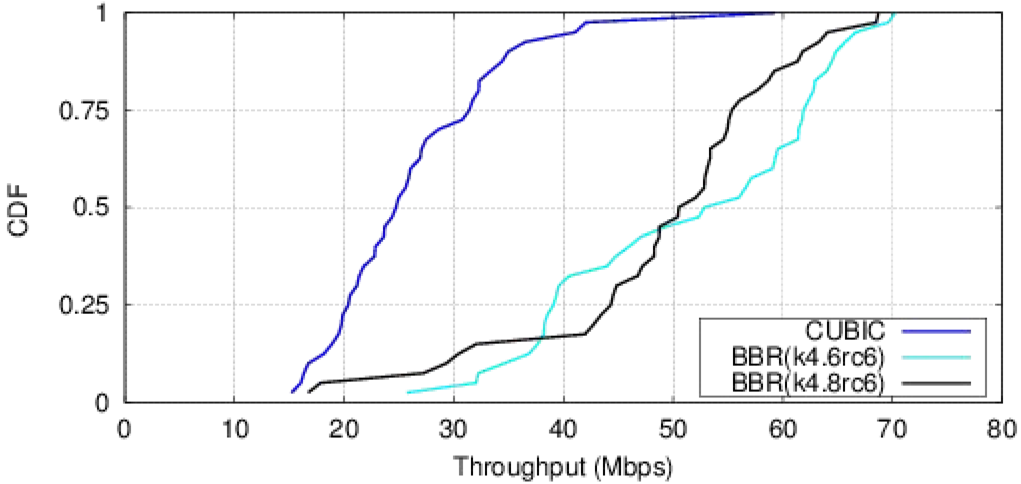
BBR: low queue delay, despite bloated buffers



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

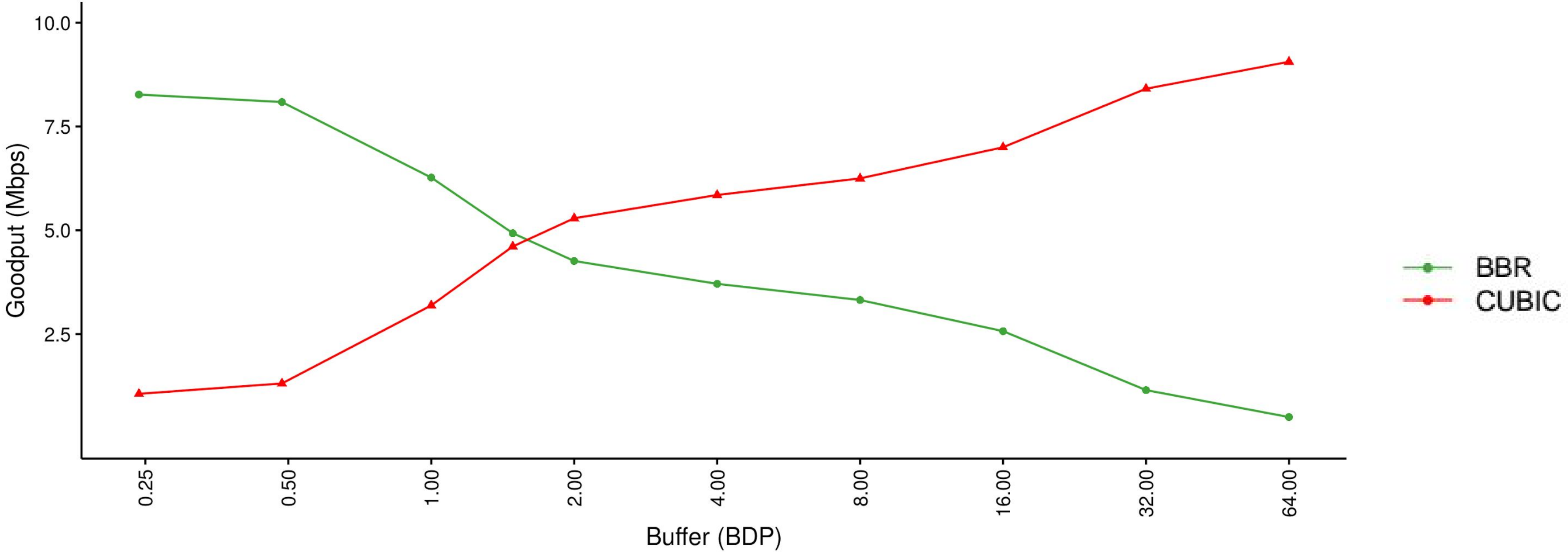
BBR: robust detection of full pipes > faster start up

- **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):

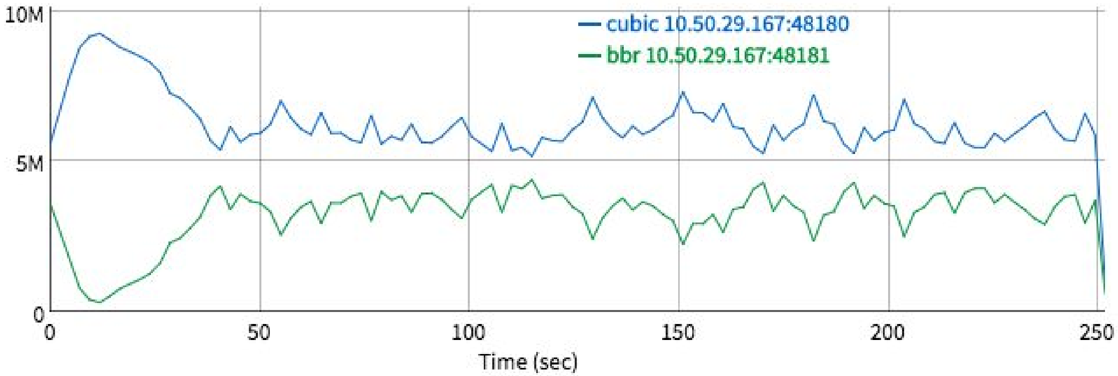


Improving dynamics w/ with loss based CC

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

Improving BBR

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

Experiment: modeling available buffer space

Goal: How to reduce buffer pressure and improve fairness in **shallow** buffers?

What if: we try to use no more than half of flow's estimated share of the bottleneck buffer?

full_rtt: average of RTT samples in first round of loss recovery phases in last N secs

```
if (full_rtt)
    my_buffer_target = (full_rtt   min_rtt) * bw / 2
    my_max_cwnd = bw * min_rtt   my_buffer_target
```

Next: how to probe gently but scalably when there are no recent losses?

e.g.: my_buffer_target *= 1.25 for each second of active sending?

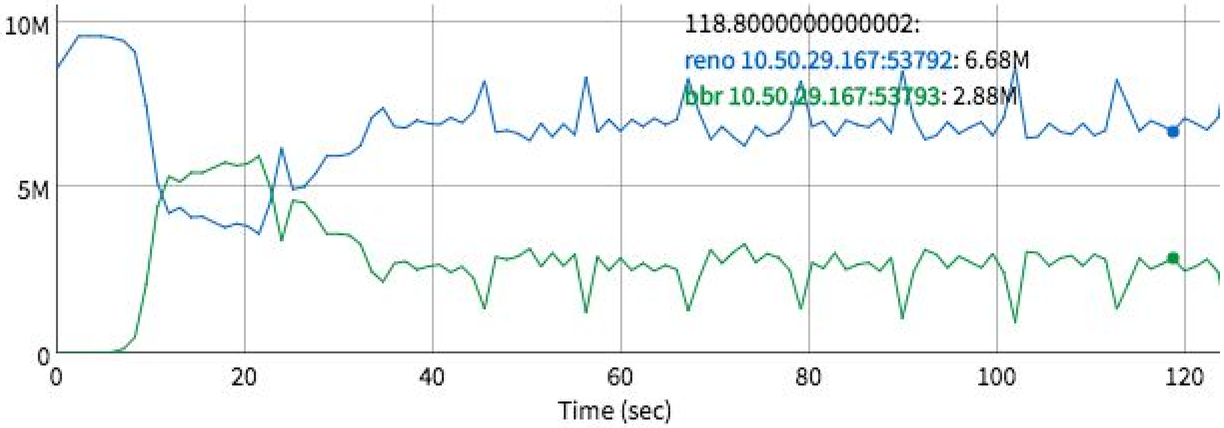
March 2017 experiments...

- 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
    - More frequent lower rate pacing to keep inflight closer to available BDP

.... resulting fairness?

In **deep** buffers, BBR's fairness to Reno matches or exceeds CUBIC's fairness to Reno...

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



1x Reno

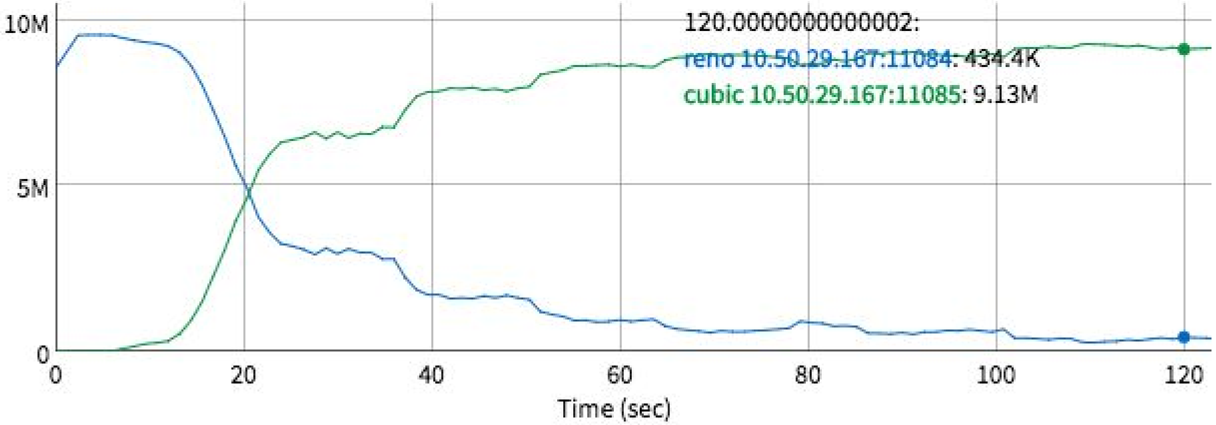
1x BBR

10 Mbps bw

40ms RTT

1 MByte buffer

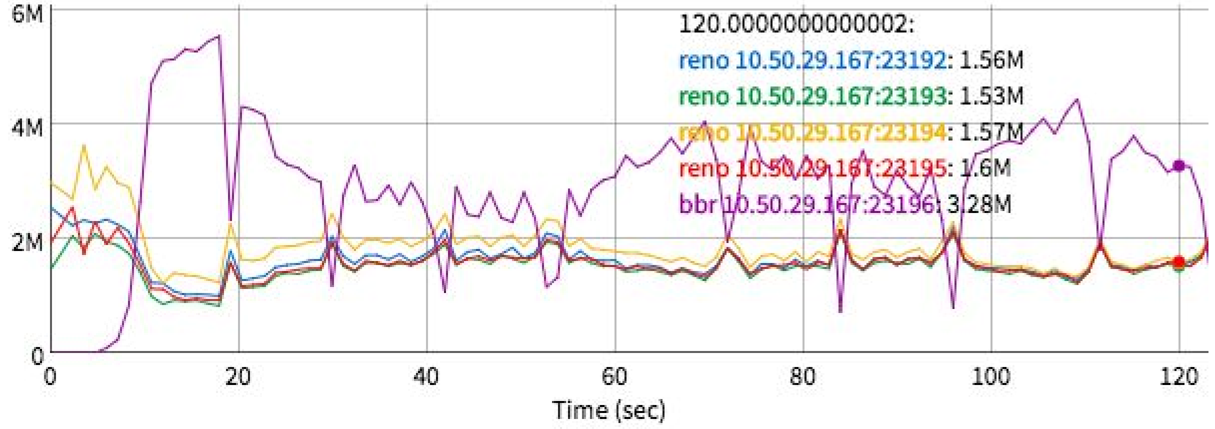
120 sec test



1x CUBIC

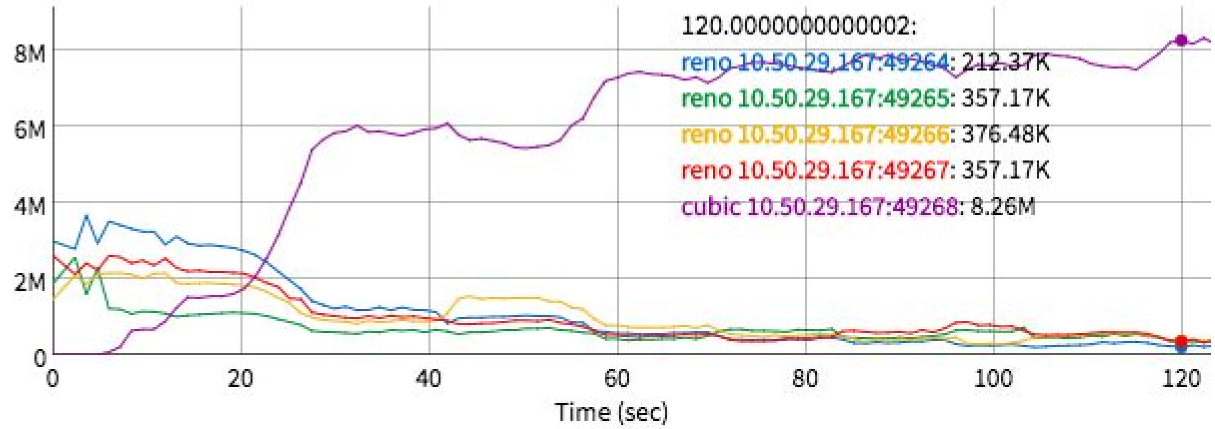
1x Reno

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



1x BBR

4x Reno



1x CUBIC

4x Reno

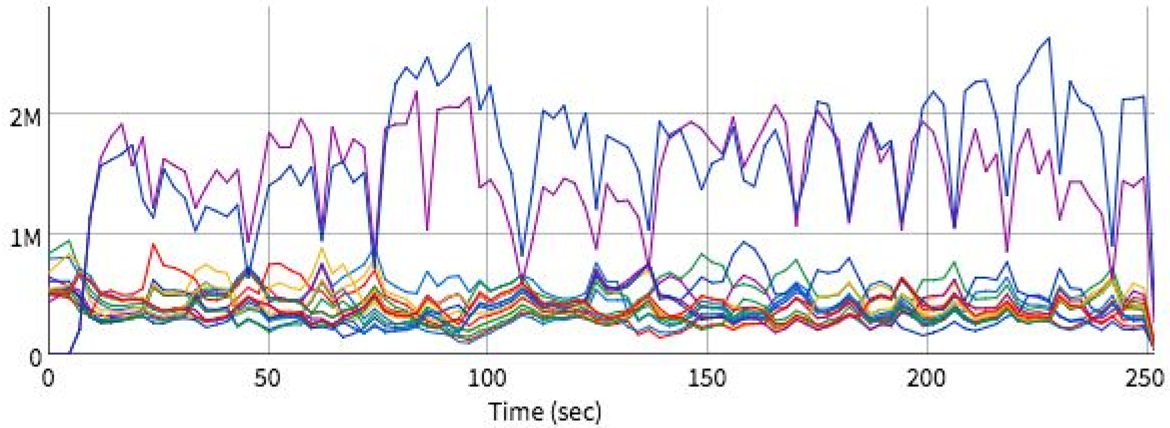
10 Mbps bw

40ms RTT

1 MByte buffer

120 sec test

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



2x BBR

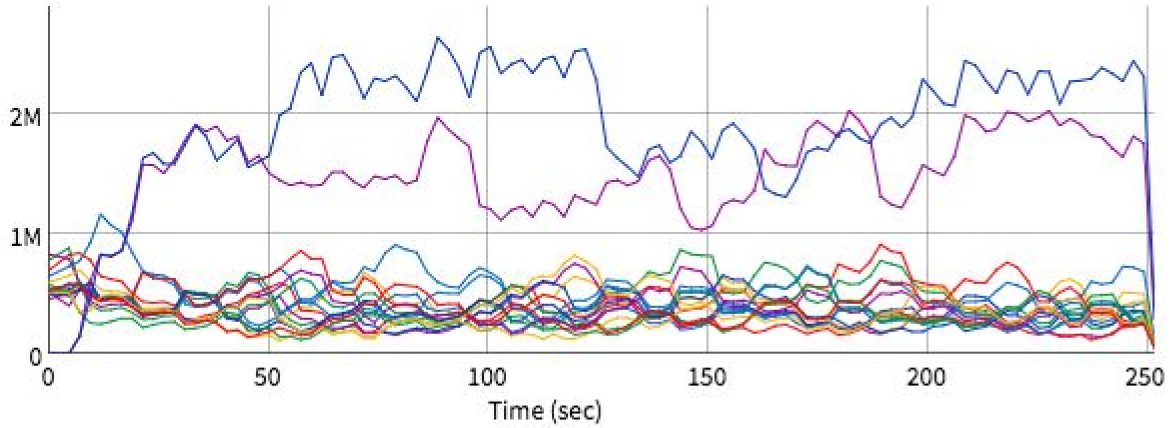
16x Reno

10 Mbps bw

40ms RTT

1 MByte buffer

240 sec test



2x CUBIC

16x Reno