BBR Congestion Control Work at Google

IETF 101 Update

Neal Cardwell, Yuchung Cheng,

C. Stephen Gunn, Soheil Hassas Yeganeh

Ian Swett, Jana Iyengar, Victor Vasiliev

Priyaranjan Jha, Yousuk Seung

Van Jacobson

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

Google

IETF 101: London, March 2018

Outline

- Overview and status of BBR congestion control
- BBR and network paths with aggregation
- Meditations on packet loss
- Conclusion and ongoing work on BBR

BBR v1.0: overview and status

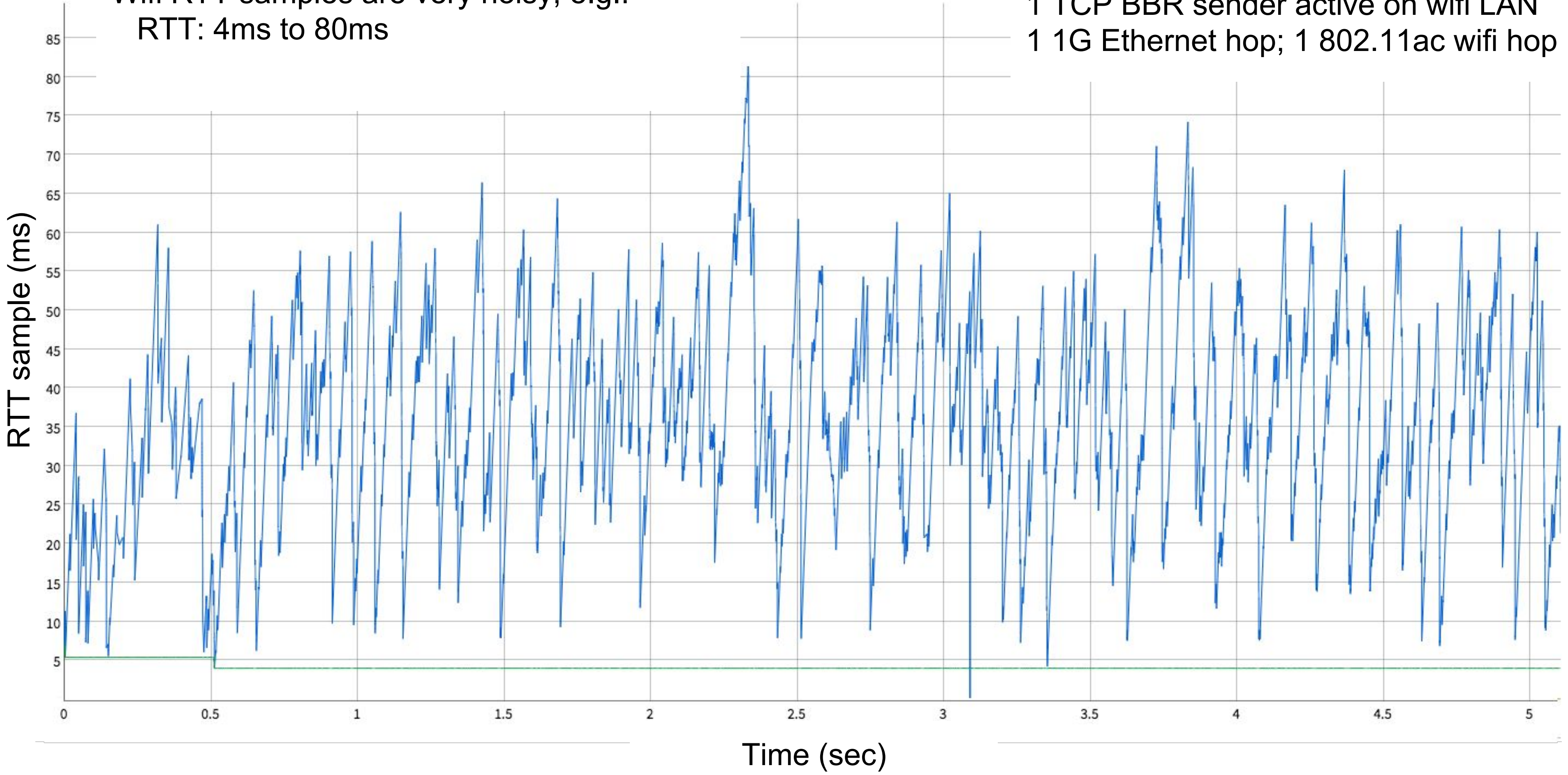
- BBR milestones already mentioned at the IETF:
    - BBR is used for TCP and QUIC on Google.com, YouTube
    - All Google/YouTube servers and datacenter WAN backbone connections use BBR
        - Better performance than CUBIC for web, video, RPC traffic
    - Code is available as open source in Linux TCP (dual GPLv2/BSD), QUIC (BSD)
    - Active work under way for BBR in FreeBSD TCP @ NetFlix
    - BBR Internet Drafts are out and ready for review/comments:
        - Delivery rate estimation: draft-cheng-iccrg-delivery-rate-estimation
        - BBR congestion control: draft-cardwell-iccrg-bbr-congestion-control
    - IETF presentations: IETF 97 | IETF 98 | IETF 99 | IETF 100
    - Overview in Feb 2017 CACM

Aggregation and BBR

- Aggregation: data and/or ACKs sent in bursts
  - Generally used for amortizing overheads to increase efficiency
    - In shared media (wifi, cellular, cable modems)
    - In interrupt processing (TSO, GSO, LRO, GRO)
  - Very widespread
- To handle aggregation, BBR decomposes this into two main problems...
- 1: bounding the data sending rate: bandwidth estimation
  - draft-cheng-iccrg-delivery-rate-estimation
  - Presented at IETF 99 (slides | video)
  - Research is ongoing: sampling bandwidth over longer intervals
- 2: bounding the volume of in-flight data: cwnd provisioning
  - To a first approximation, BBR v1 tries to bound in-flight data near estimated BDP:
    - cwnd ~= 2 * estimated_bdp ~= 2 estimated_bandwidth * min_rtt
  - But how well does this cope with high degrees of aggregation and RTT variance?

Wifi min_rtt is **far** from the "typical" RTT
Wifi RTT samples are very noisy; e.g.:
RTT: 4ms to 80ms

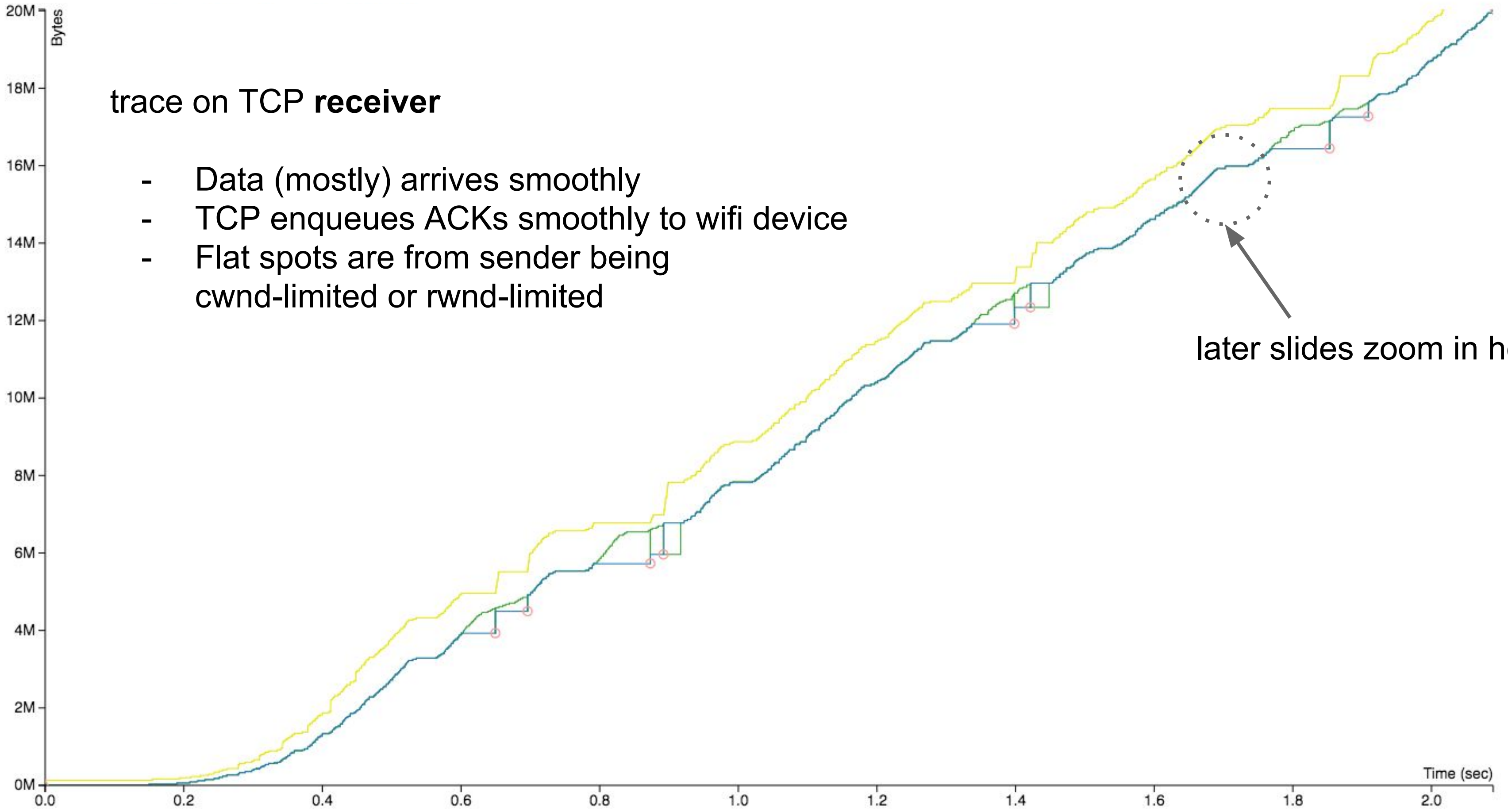
Per-ACK RTT samples (ms)
1 TCP BBR sender active on wifi LAN
1 1G Ethernet hop; 1 802.11ac wifi hop



Aggregation deep-dive: a trace of TCP BBR over wifi

- An example: a single bulk TCP transfer of 20 MBytes over wifi
- Path:
    - Linux TCP BBR v1.x sender
    - Public Internet
    - Cable modem
    - Home wifi network
    - Laptop receiver
- Packet traces captured on sender and receiver with tcpdump

Full wifi trace: receiver

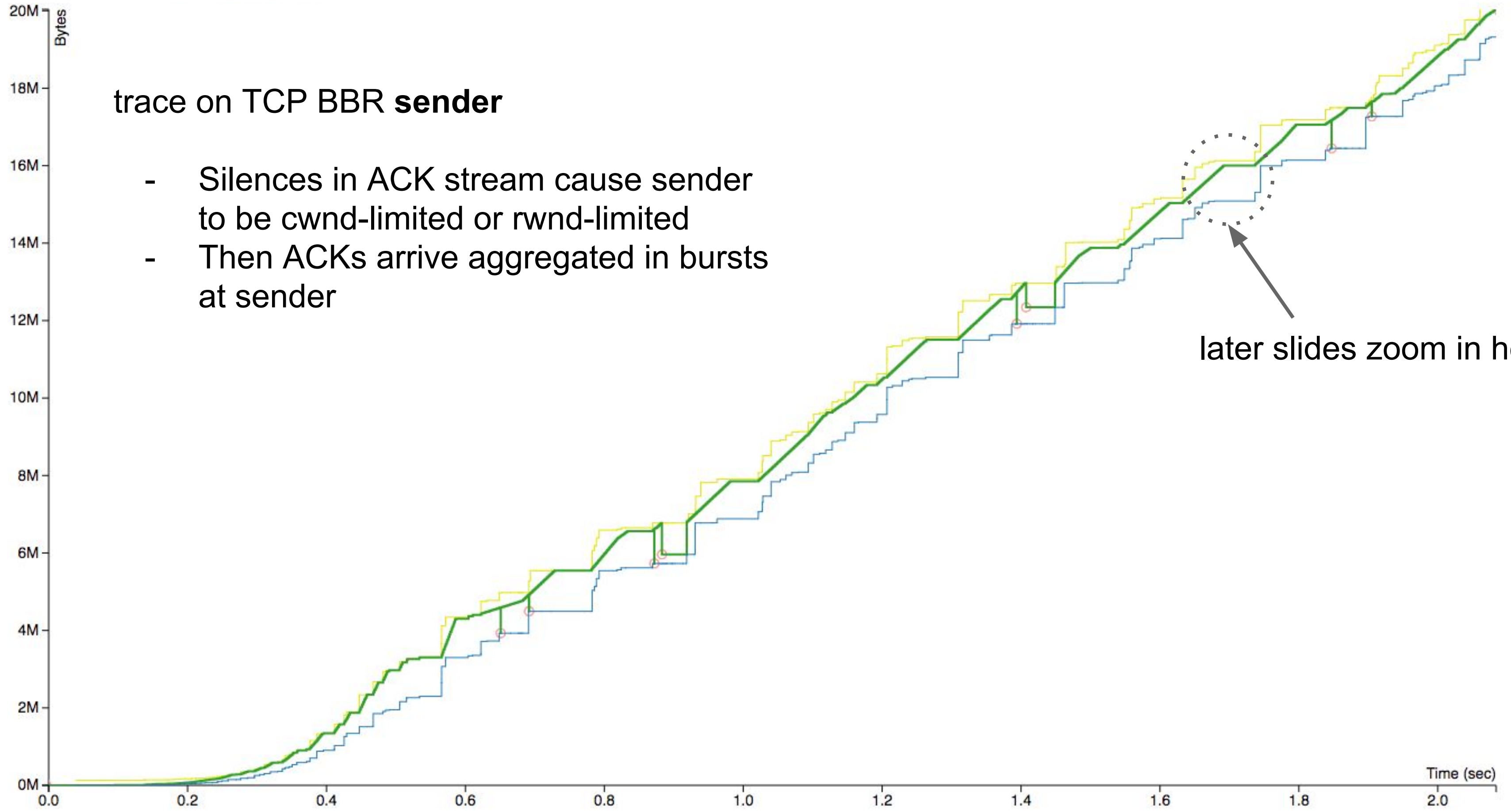


trace on TCP **receiver**

- Data (mostly) arrives smoothly
- TCP enqueues ACKs smoothly to wifi device
- Flat spots are from sender being cwnd-limited or rwnd-limited

later slides zoom in here

Full wifi trace: sender

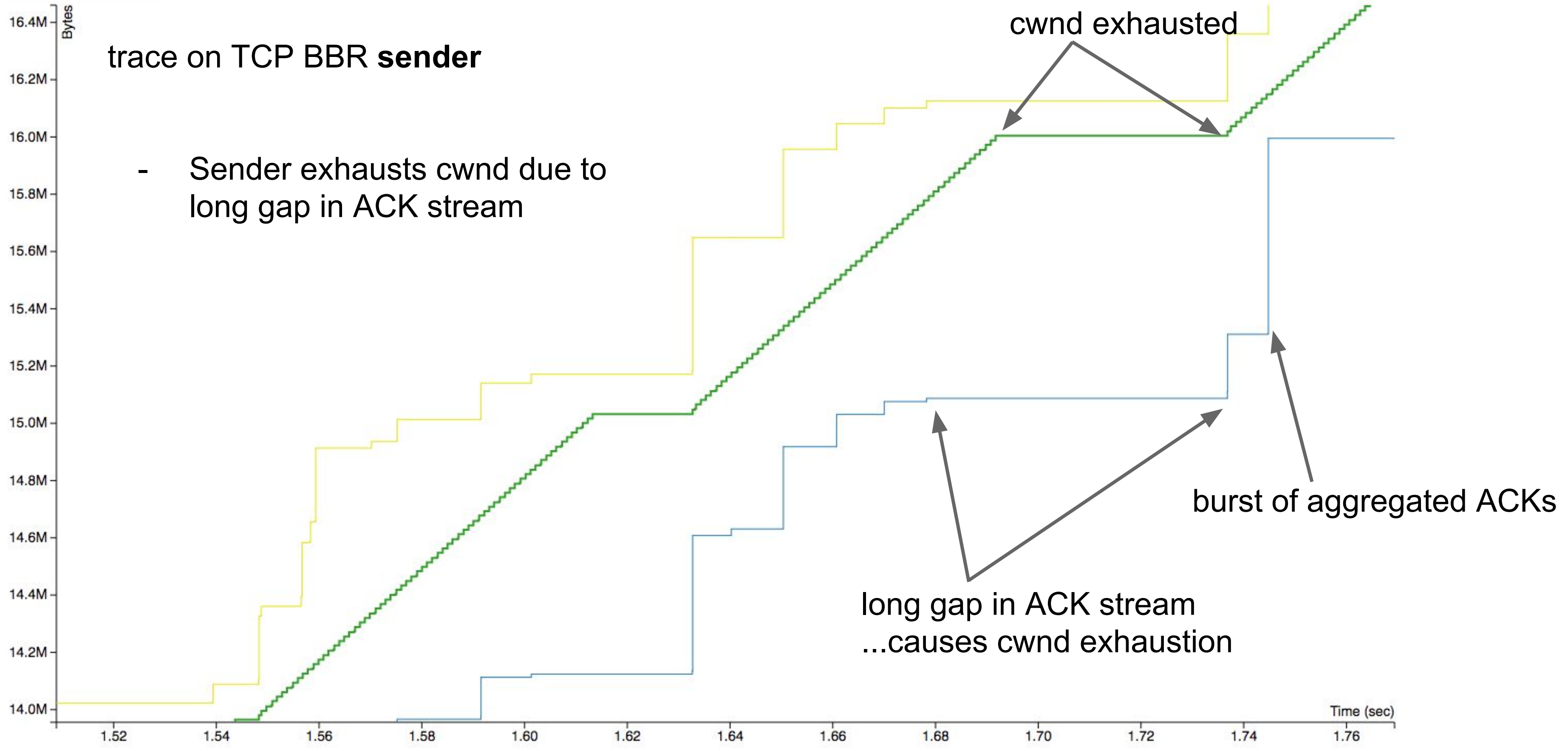


trace on TCP BBR **sender**

- Silences in ACK stream cause sender to be cwnd-limited or rwnd-limited
- Then ACKs arrive aggregated in bursts at sender

later slides zoom in here

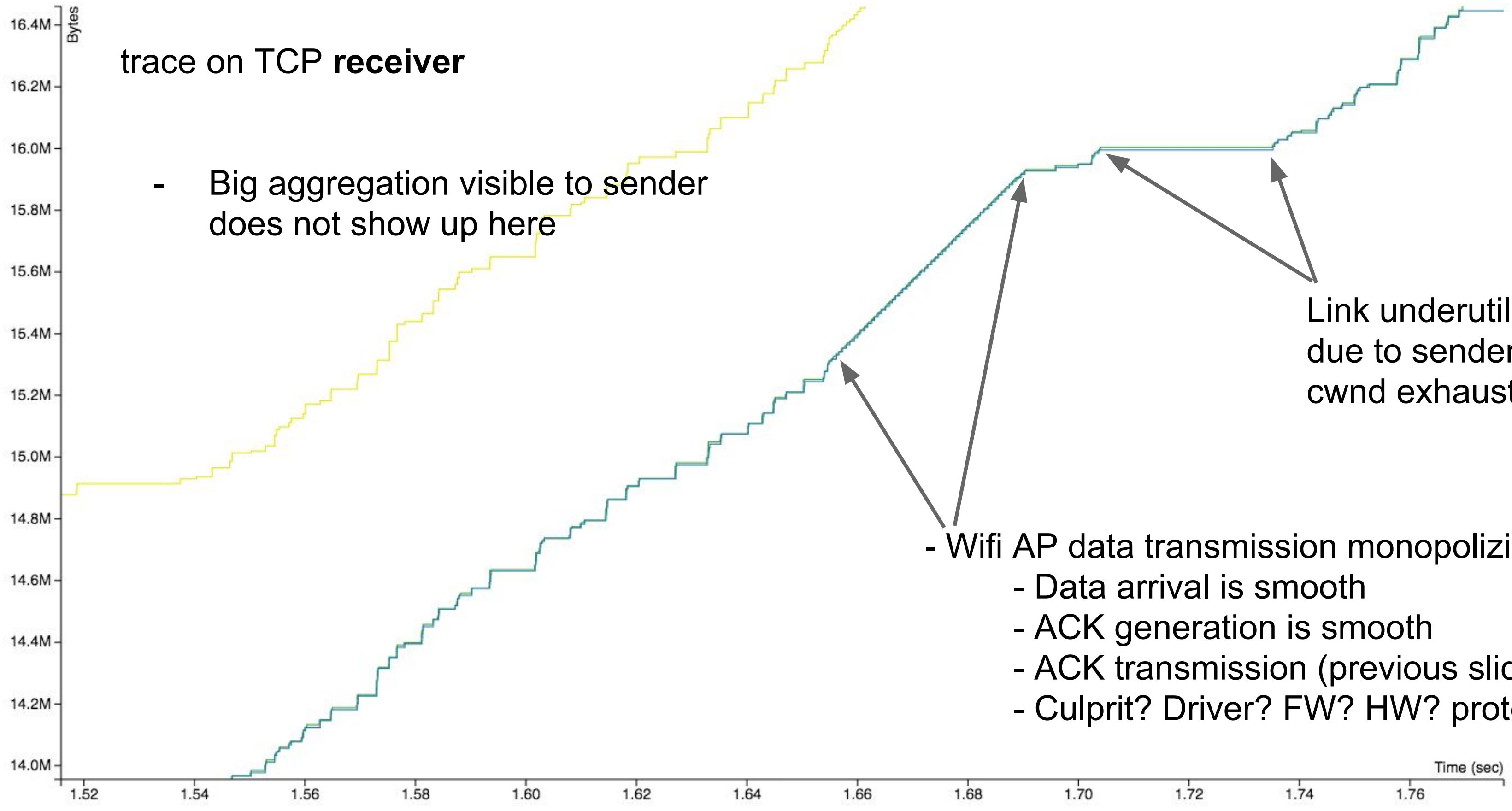
Zoomed-in wifi trace: sender



trace on TCP BBR **sender**

- Sender exhausts cwnd due to long gap in ACK stream

Zoomed-in wifi trace: receiver



trace on TCP **receiver**

- Big aggregation visible to sender does not show up here

Link underutilized due to sender cwnd exhaustion

- Wifi AP data transmission monopolizing link?
    - Data arrival is smooth
    - ACK generation is smooth
    - ACK transmission (previous slide) paused
    - Culprit? Driver? FW? HW? protocol?

prietary

How to bound in-flight data with aggregation?

- BBR aims for minimum data required in-flight to achieve available bandwidth
- BBR v1 cwnd ~= (pipe_budget) + (delay_variation_budget)

  = (bw*min_rtt) + (bw*min_rtt)

- Sufficient for full utilization iff: delay_variation <= min_rtt
- But with a high degrees of aggregation...
  - Exhausts cwnd for low-RTT connections, causing link underutilization
  - The lower the min_rtt, the more likely sender will exhaust cwnd
- Well-provisioned CDN edge servers are often close enough for these shortfalls over wifi:
  - e.g.
    - min_rtt: 1-20ms
    - wifi delay_variation=1-80ms

BBR aggregation estimator

- How much data can be ACKed as an aggregate over the time scale of cwnd?
- Estimate the windowed max degree of aggregation
- Use this to provision extra in-flight data to keep sending during inter-ack silences
- Degree of aggregation is estimated as extra data ACKed beyond expected:
    - extra_acked = excess data ACKed beyond expected amount over this interval
      = actual_acked - estimated_bw*interval
    - max_extra_acked = maximum *recent* extra_acked
    - target_cwnd = f(estimated_bdp, offload_quantum) + max_extra_acked
- How *recent*?
    - Approximate sliding window of 10 (packet-timed) round trips
- What's the upper bound on extra_acked?
    - Bounded by cwnd at time of measuring extra_acked

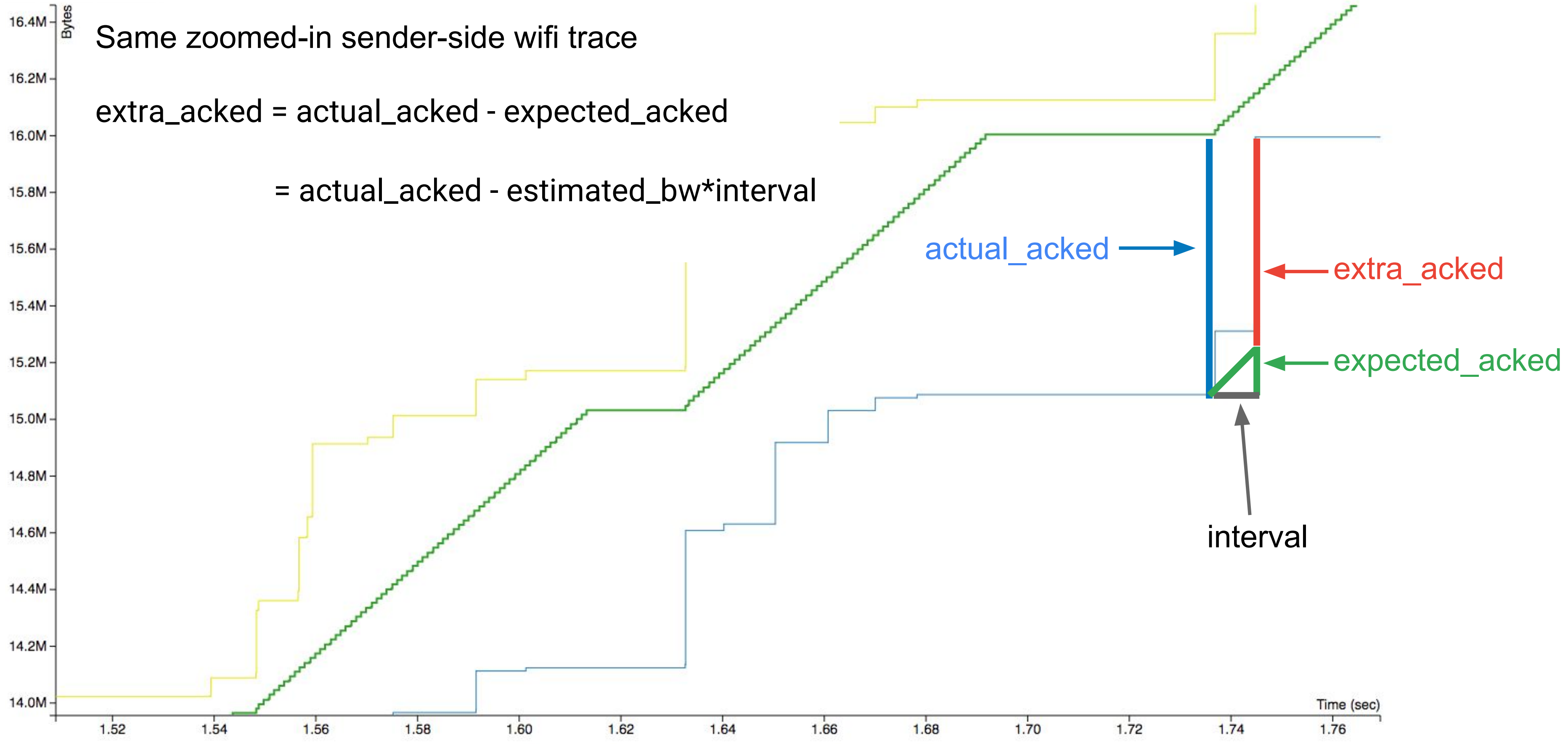
Google

BBR aggregation estimator: visualizing the dynamics

Same zoomed-in sender-side wifi trace

extra_acked = actual_acked - expected_acked

= actual_acked - estimated_bw*interval



BBR aggregation estimator: status and results

- Code is deployed for QUIC BBR
  - See max_ack_height_ in bbr_sender.cc
- TCP implementation is being rolled out for Google.com and YouTube experiments
- Example performance of TCP implementation in controlled lab setting:
  - host1 (ethernet) -> AP -> (wifi) host2
  - 4x increase in 2.4 GHz: ~25 Mbps to ~100 Mbps
  - 10x increase in 5.0 GHz: ~25 Mbps to ~260 Mbps

Adaptive draining for shorter queues

- In steady state BBR uses "gain cycling": pace at pacing_gain * estimated_bandwidth
    - Probe for bandwidth: pacing_gain > 1
    - Drain queue: pacing_gain < 1
    - Hold steady: pacing_gain = 1
- In BBR v1.0 "drain" phase was held for 1.0 * min_rtt
    - With multiple flows, inflight drifted up to 2 * estimated_bdp
- "Drain to Target": adaptively holds the "drain" phase until inflight ~ estimated_bdp
- Ingredients to make this work:
    - Bounded time in drain phase: need to refresh bandwidth estimate
    - Randomized phases: avoid elephants and mice probing and draining in sync
    - Aggregation estimator: to adaptively allow higher cwnd with aggregation effects
- Seeing lower loss rates with small-scale YouTube experiments
    - Planning global YouTube, Google.com tests next

Recent contributions from the community

- BBR implementation available for ns-3:
  - Code on github
  - M. Claypool, J.W. Chung, and F. Li. "BBR' - An Implementation of Bottleneck Bandwidth and Round-trip Time Congestion Control for ns-3", Technical Report WPI-CS-TR-18-01, Computer Science, Worcester Polytechnic Institute, January 2018
- BBR being tested in Stanford's Pantheon of Congestion Control:
  - http://pantheon.stanford.edu/

Meditations on packet loss

- The question is not **whether** to use loss as an input signal
    - Reno, CUBIC, and BBR (v1 and v2) all use packet loss as an explicit signal
- The key question: **how** can packet loss be used effectively as a signal to adapt transport behavior?
- Where packet loss results from congestion, there is still the question of **time scale**:
    - If buffer is 1% of BDP, a link can be 99% idle even over the course of a round-trip with losses
    - If a buffer is 100% of BDP, a link can be 90% idle over the 10 round trips following a loss
    - Given this, on what time scale should a CC re-probe for bandwidth?
- There's a tension between minimizing loss and maximizing application performance
- Reducing queuing and packet loss generally reduces application latency, up to a point
- Minimizing queuing and packet loss does not *always* mean maximizing application performance
    - Sending slowly and probing for bandwidth slowly cuts loss but can increase app latency
    - We've seen this repeatedly in datacenter workloads, with different CC algorithms

Conclusion

- Status of BBR v1.0
  - Deployed widely at Google
  - Open source for Linux TCP and QUIC
  - Documented in IETF Internet Drafts
- Actively working on BBR v2.0
  - Linux TCP and QUIC at Google; current focus areas:
    - Aggregation (e.g. wifi)
    - Packet loss signals
    - Dynamics of sharing with loss-based congestion control
  - Work under way for BBR in FreeBSD TCP @ NetFlix
  - Always happy to see patches, hear test results, or look at packet traces...

Q & A

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

Internet Drafts, 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...

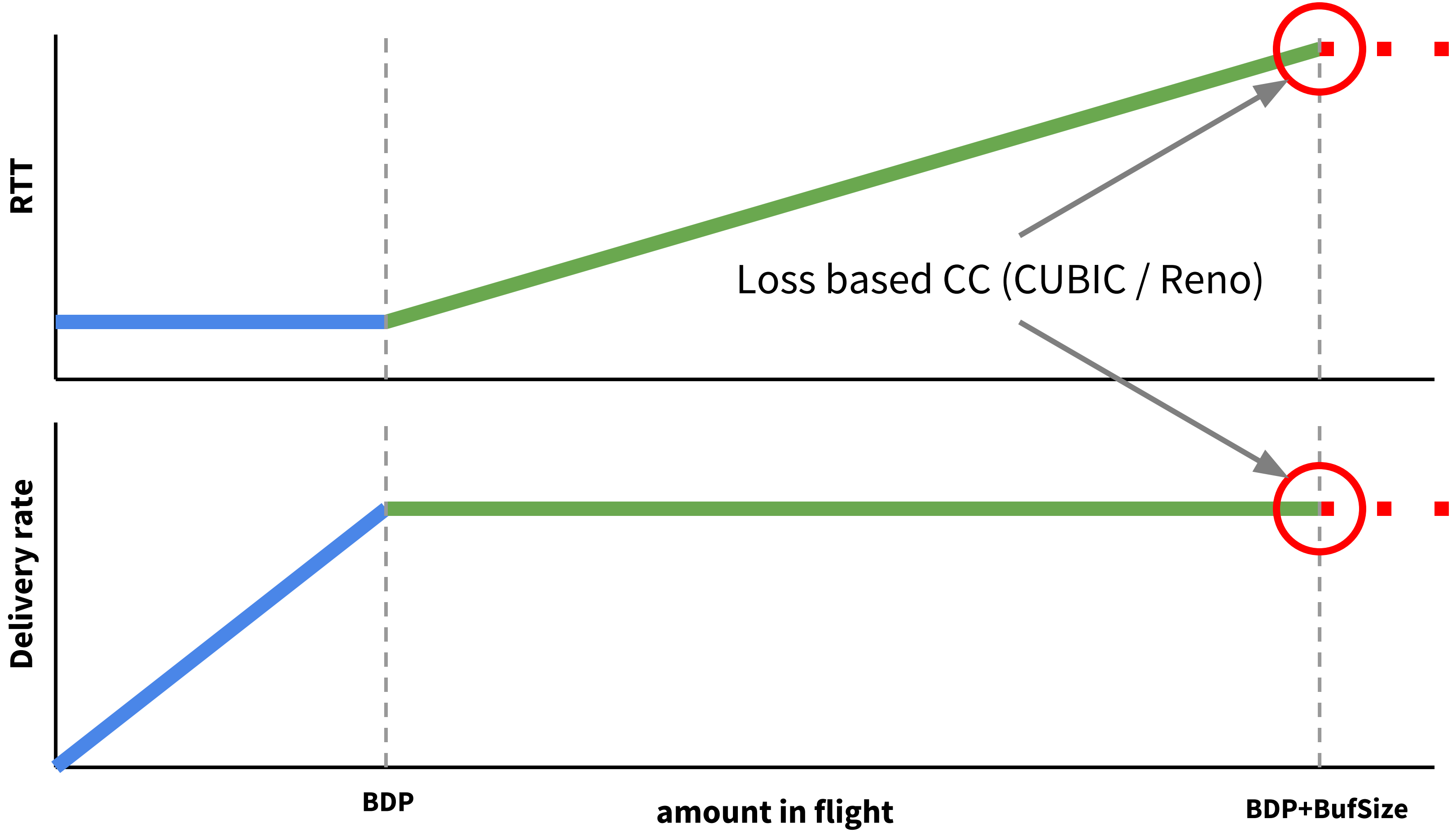
Issues with loss-based congestion control

- Loss-based congestion control (Reno, CUBIC) has fundamental scaling issues
- 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

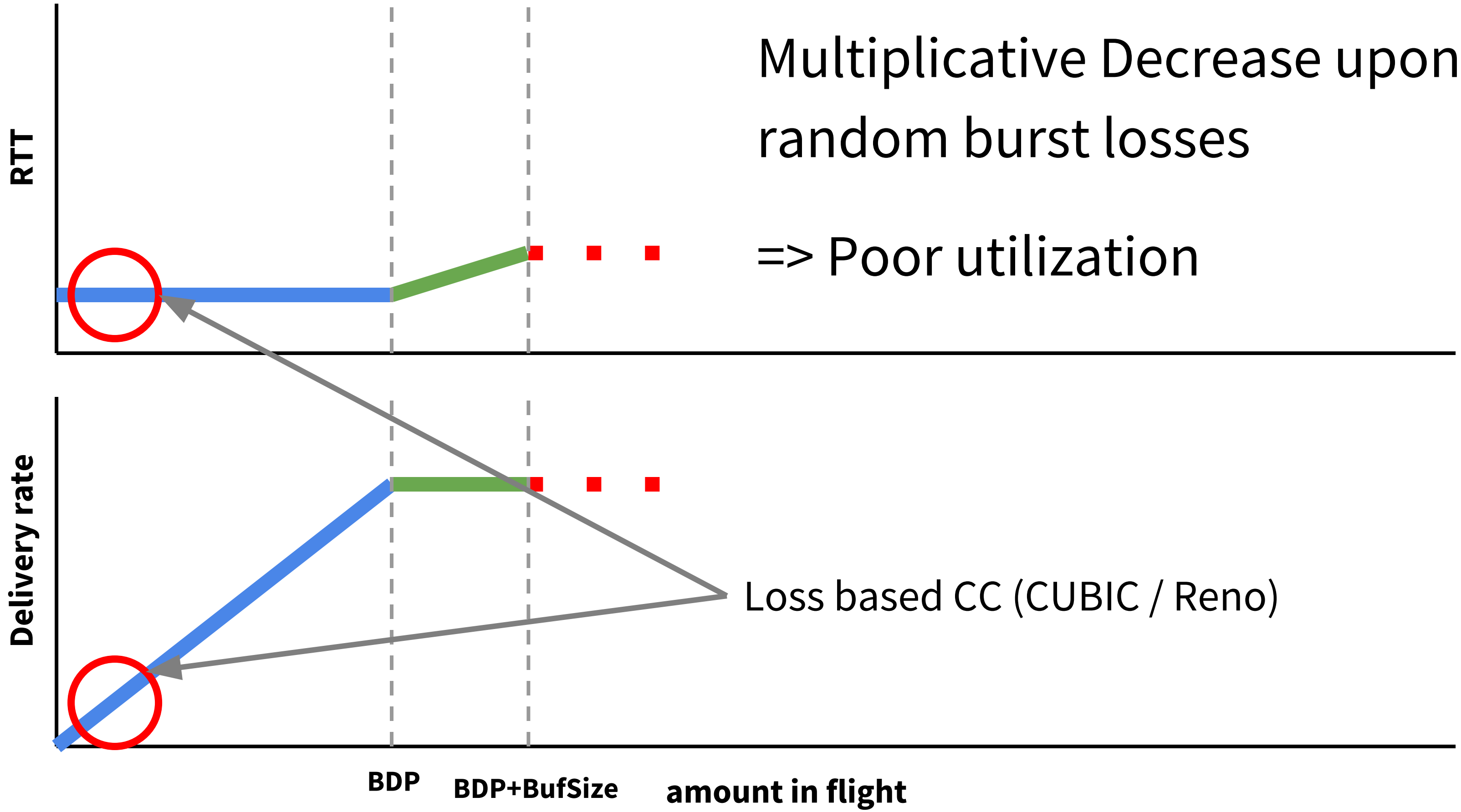
BBR (Bottleneck BW and RTT)

- **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

Loss based congestion control in deep buffers



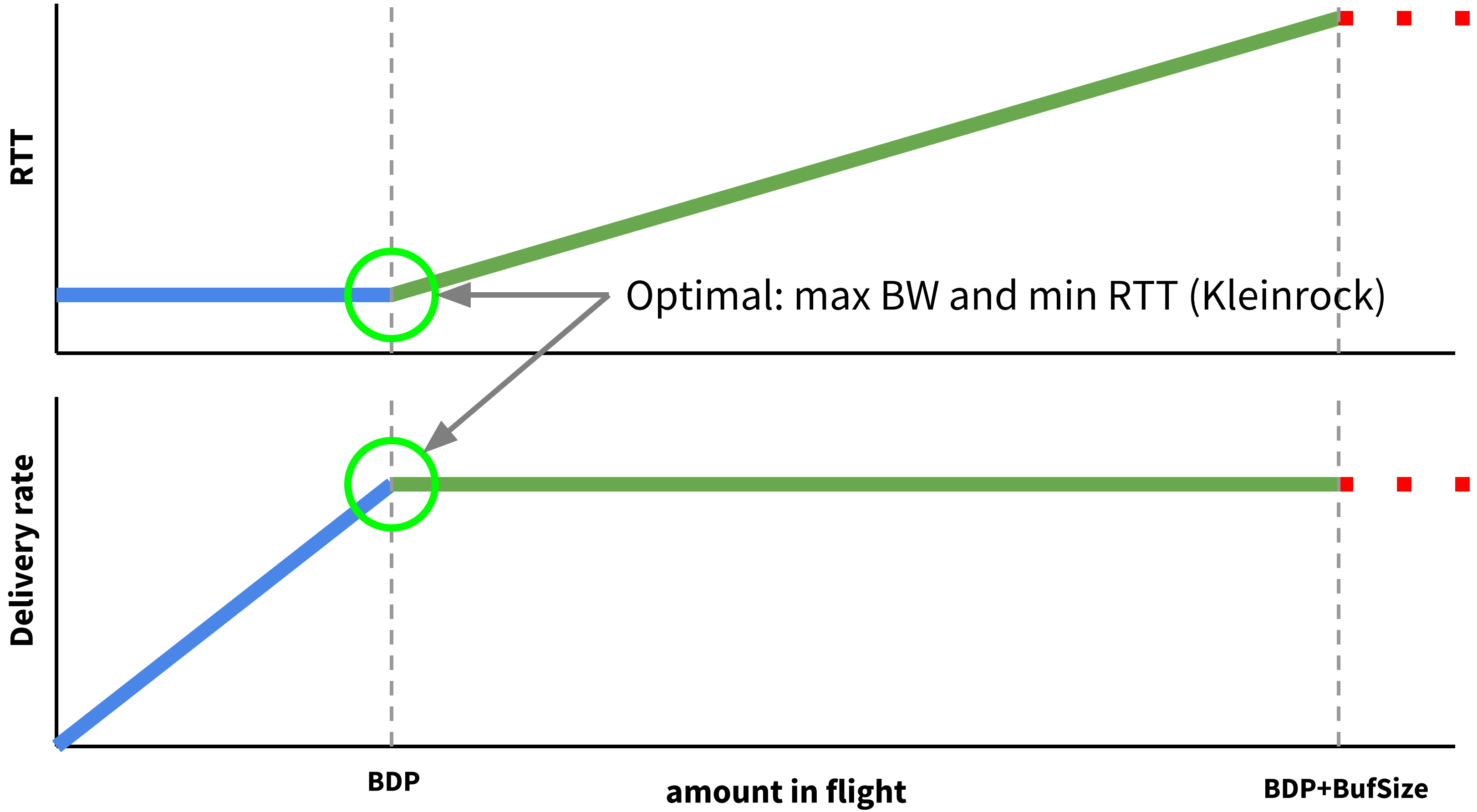
Loss based congestion control in shallow buffers



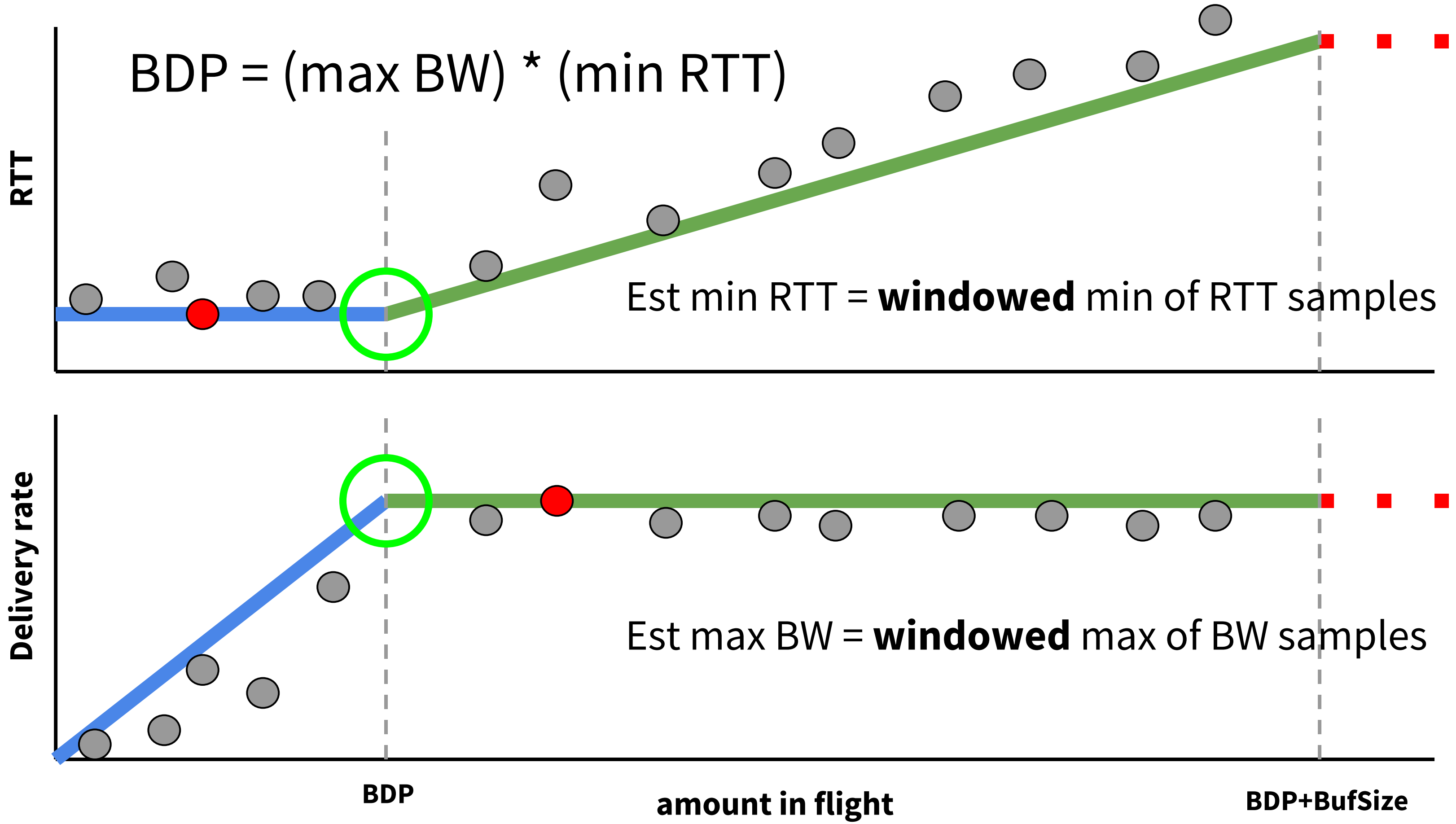
Multiplicative Decrease upon random burst losses

=> Poor utilization

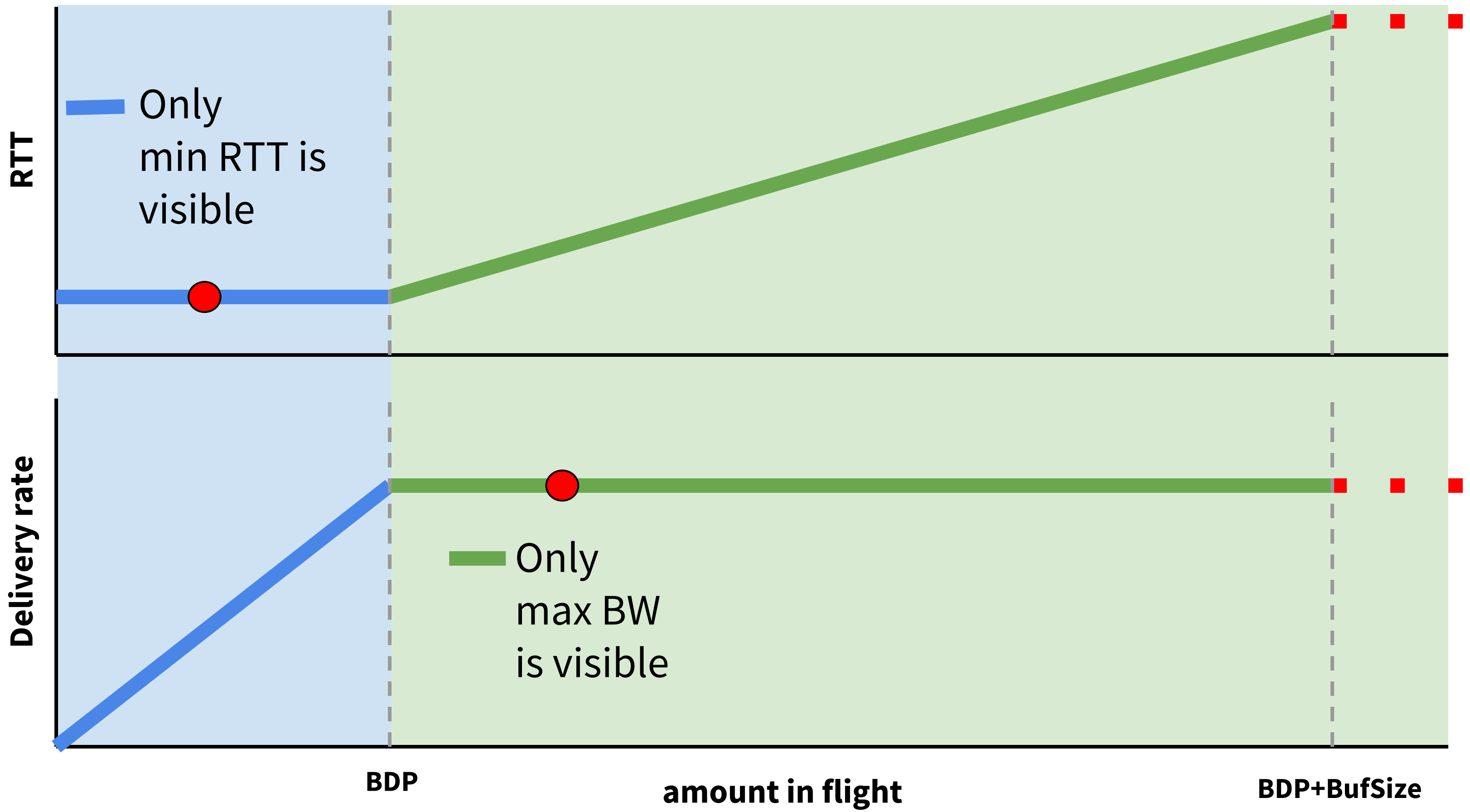
Optimal operating point



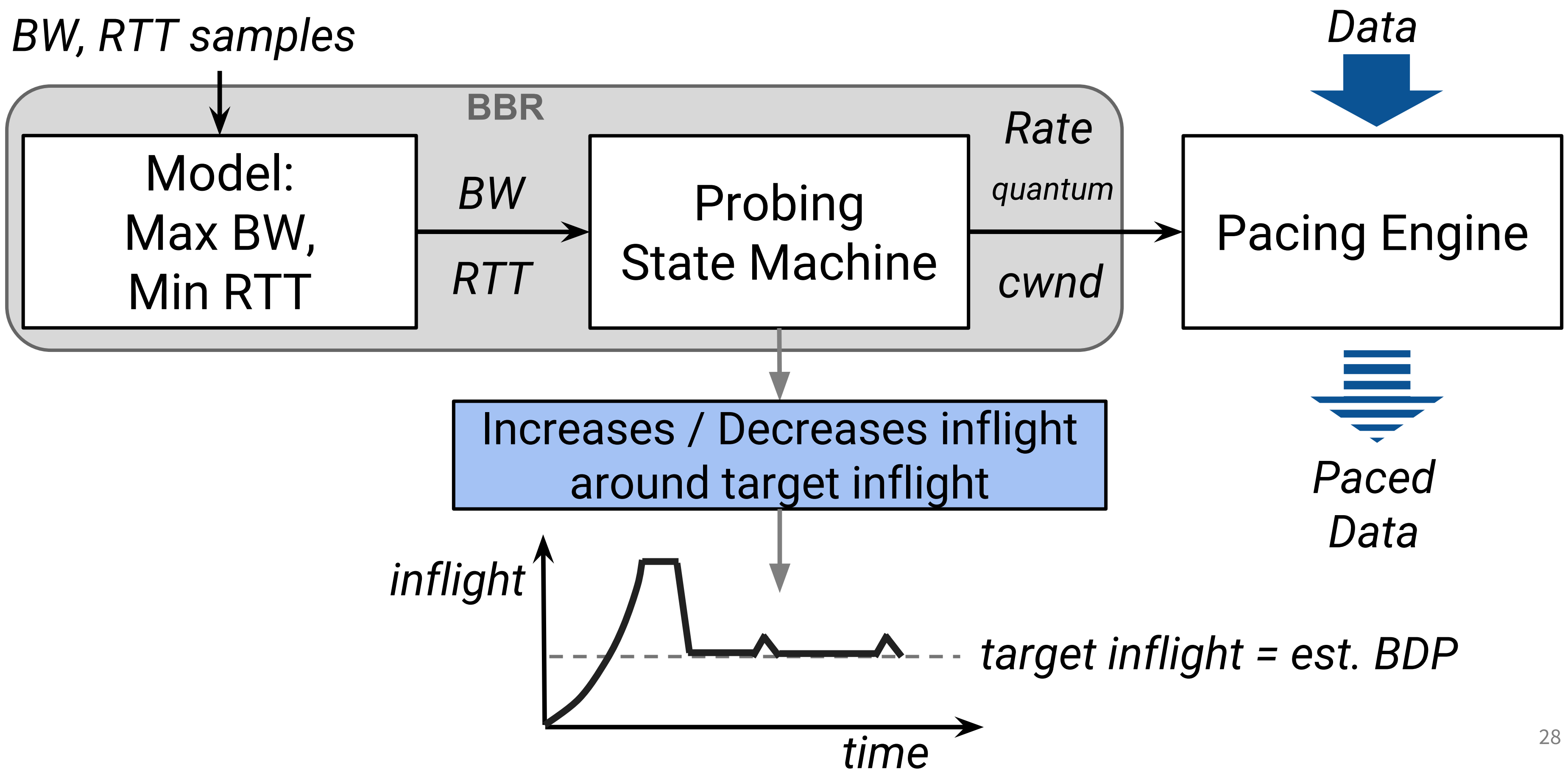
Estimating optimal point (max BW, min RTT)



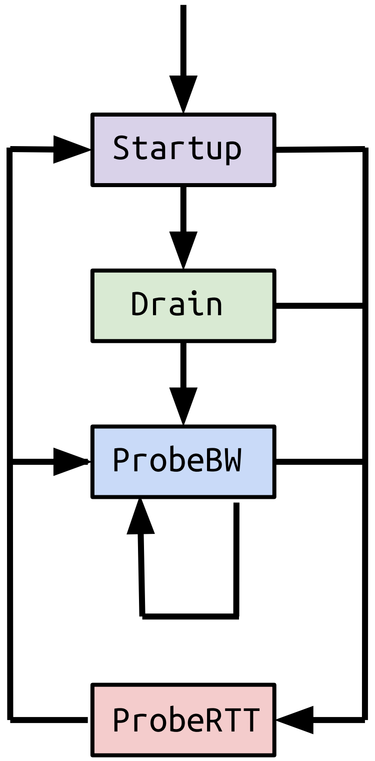
To see max BW, min RTT: probe both sides of BDP



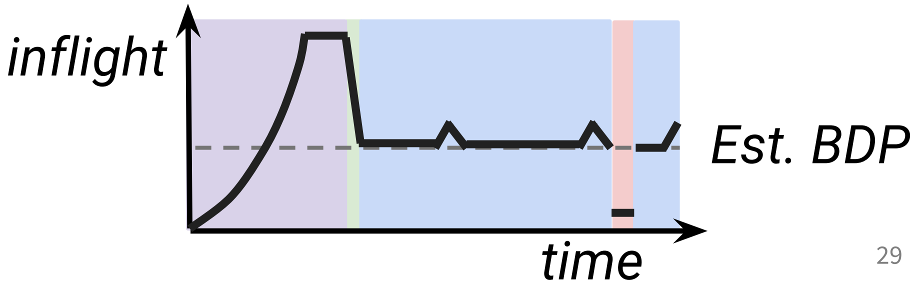
BBR congestion control: the big picture



BBR: probing state machine



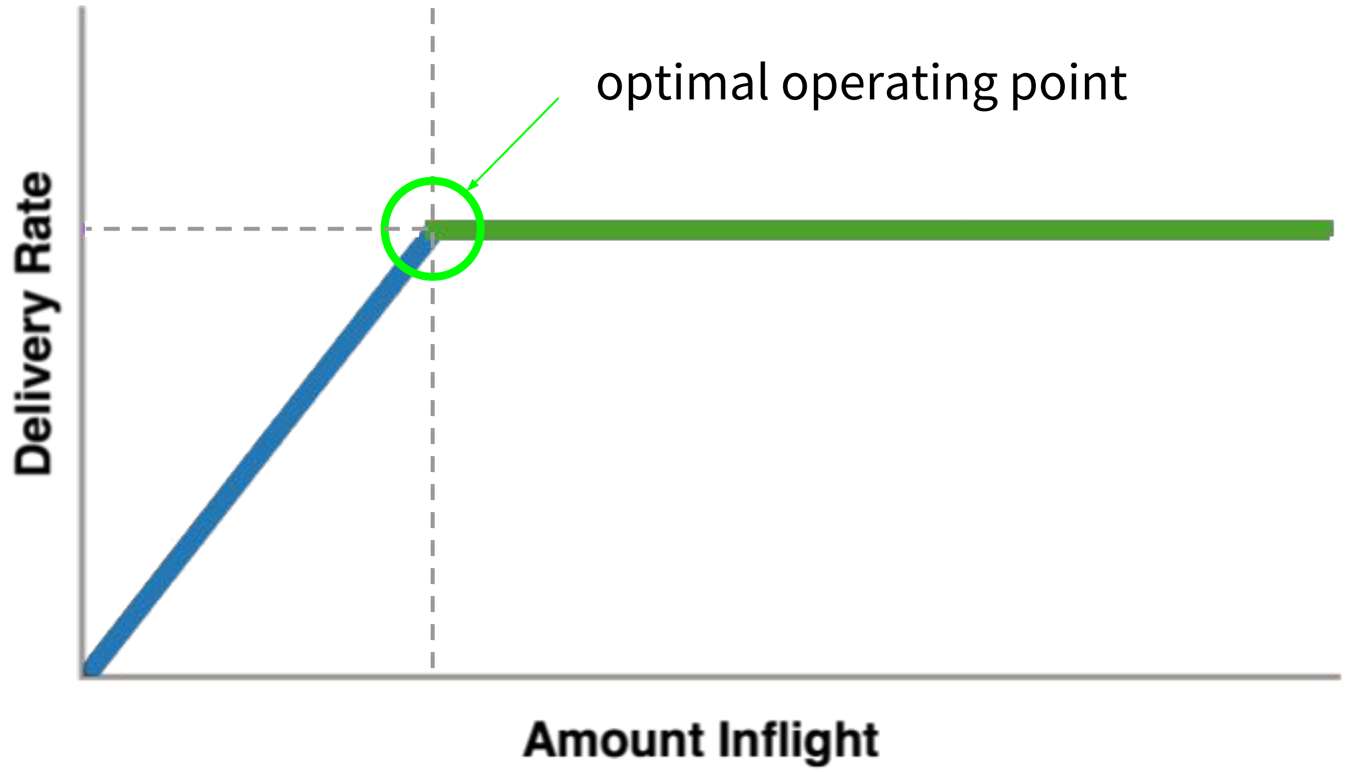
- State machine for 2-phase sequential probing:
  - 1: raise inflight to probe BtlBw, get high throughput
  - 2: lower inflight to probe RTprop, get low delay
  - At two different time scales: warm-up, steady state...
- Warm-up:
  - Startup: ramp up quickly until we estimate pipe is full
  - Drain: drain 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



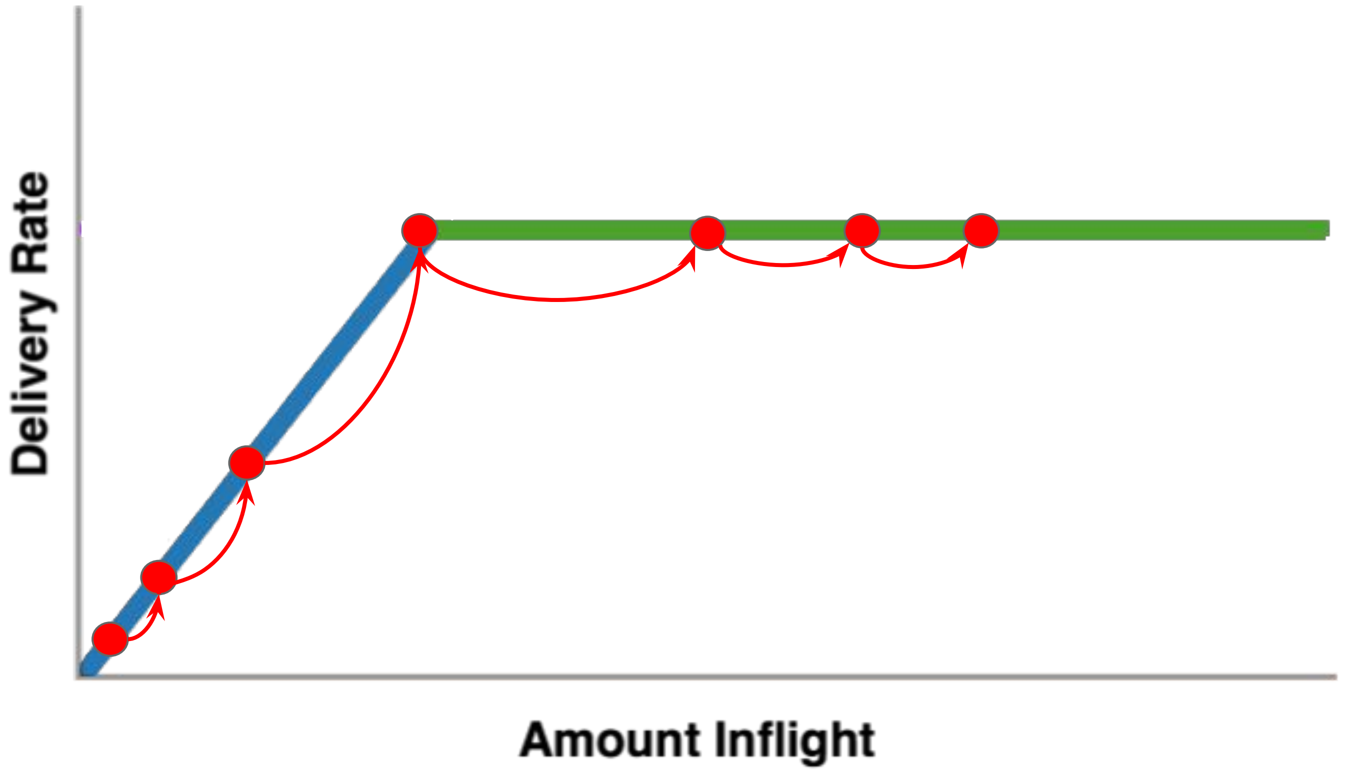
BBR congestion control algorithm: Internet Draft

- 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

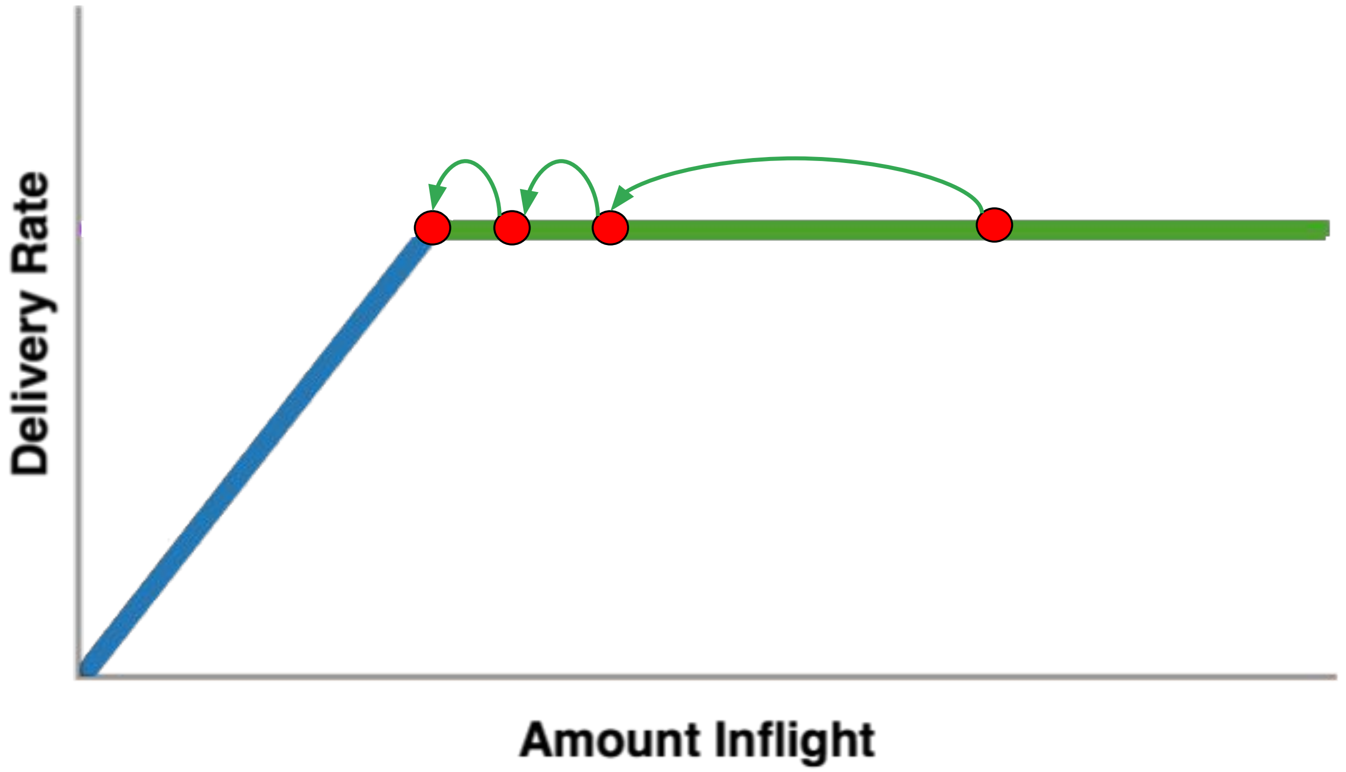
BBR: model based walk toward max BW, min RTT



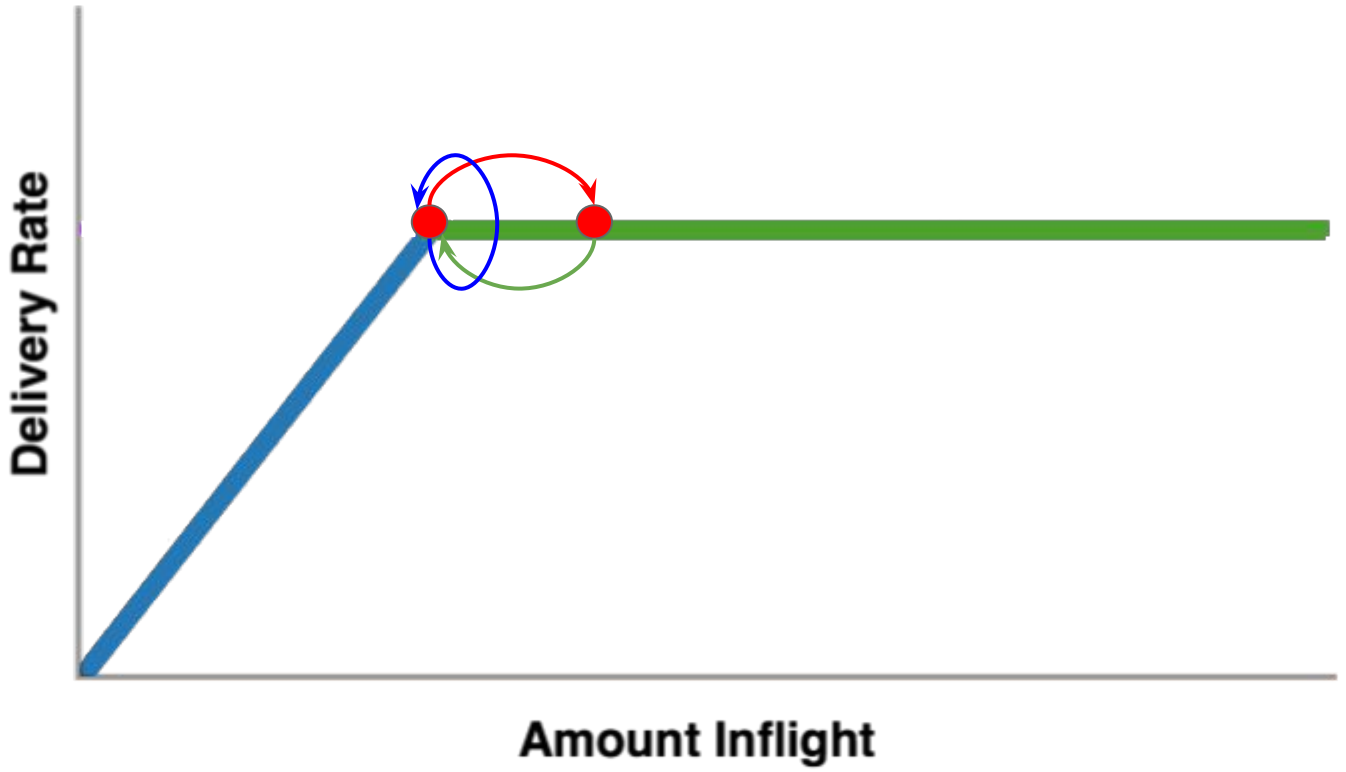
STARTUP: exponential BW search



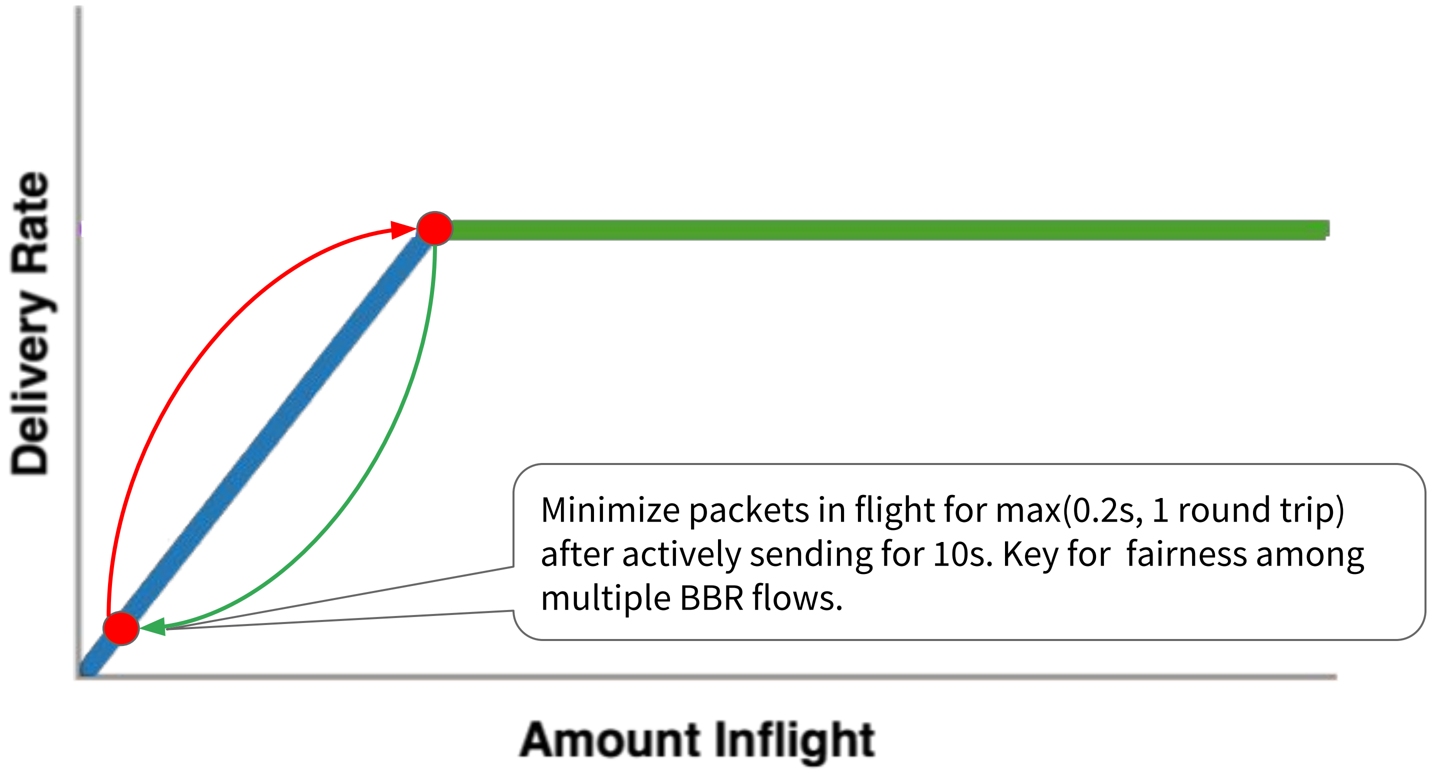
DRAIN: drain the queue created during STARTUP



PROBE_BW: explore max BW, drain queue, cruise



PROBE_RTT: drains queue to refresh min RTT



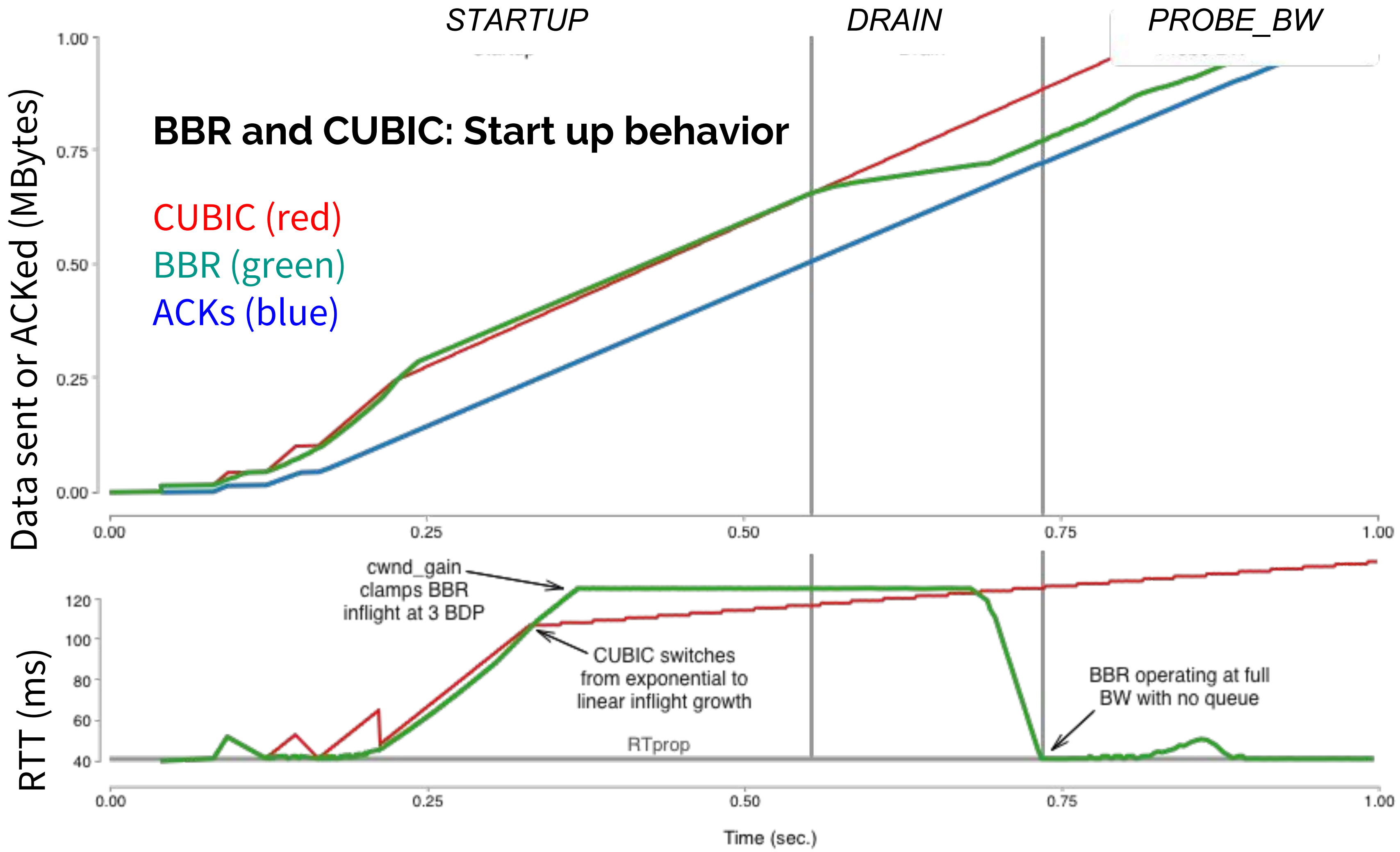
Minimize packets in flight for max(0.2s, 1 round trip) after actively sending for 10s. Key for fairness among multiple BBR flows.

BBR and CUBIC: Start up behavior

CUBIC (red)

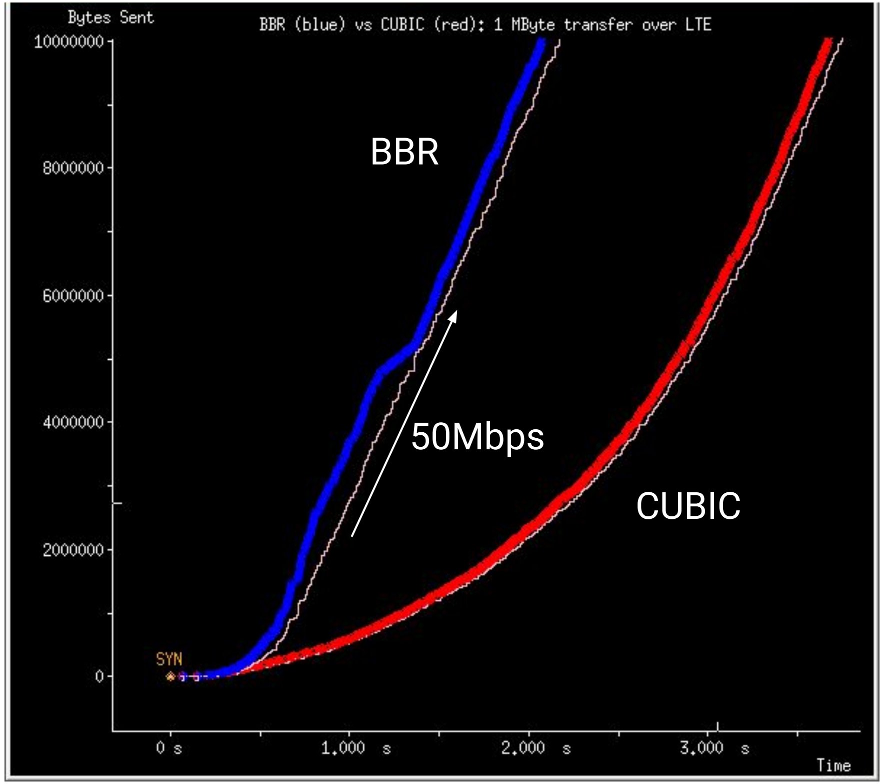
BBR (green)

ACKs (blue)



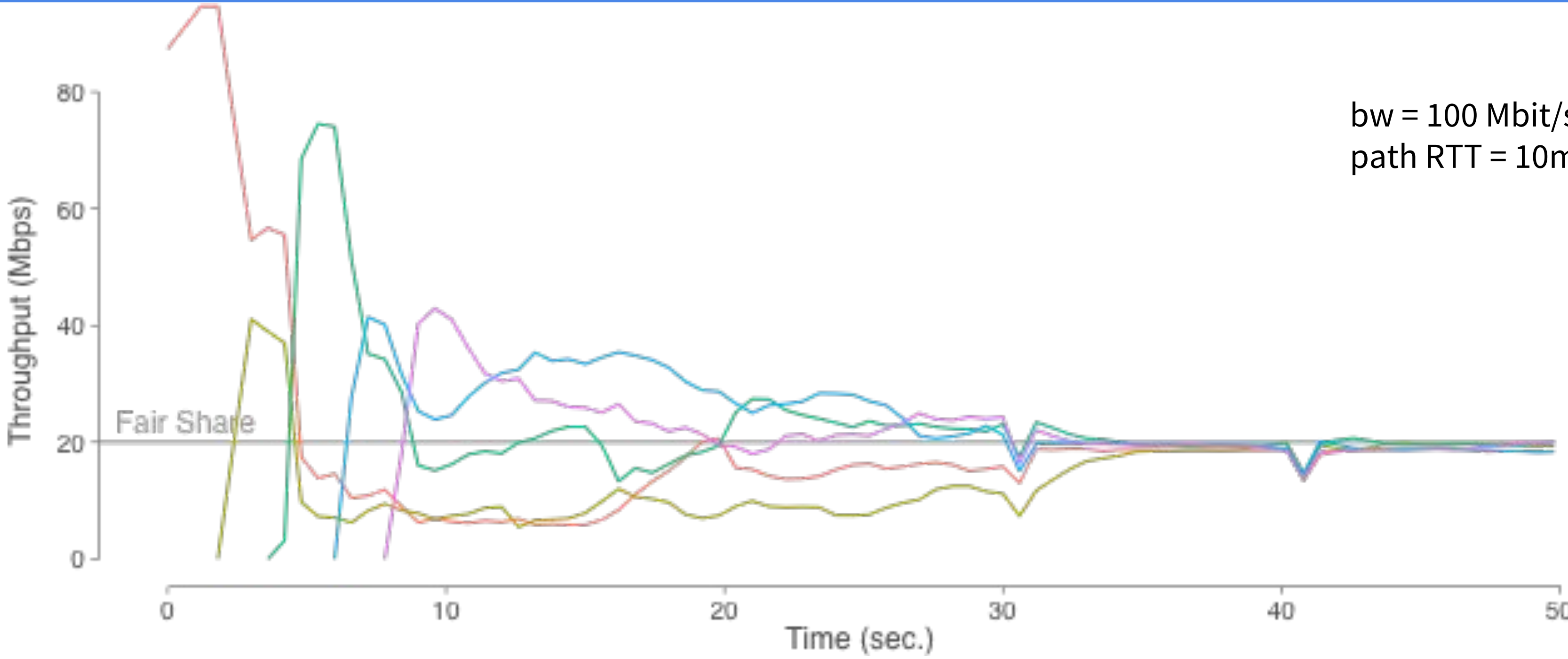
BBR: faster for short flows, too

	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

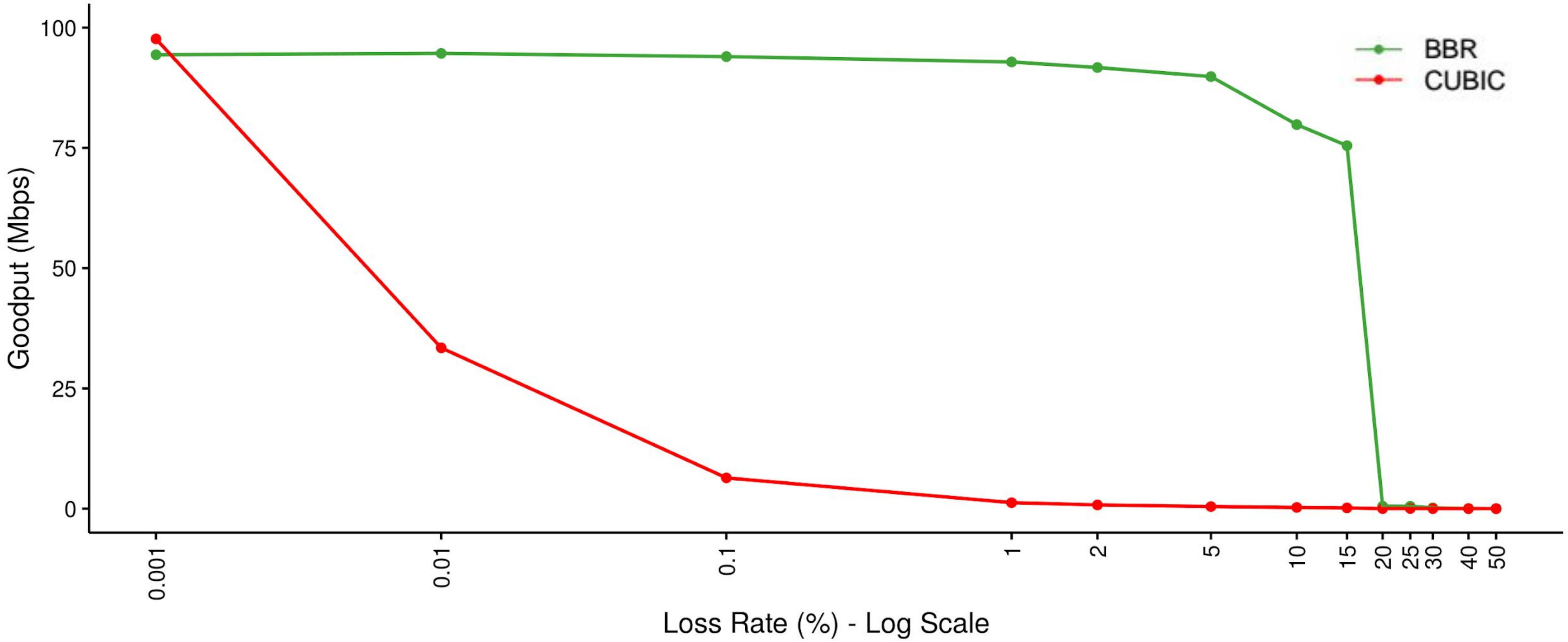
BBR multi flow convergence dynamics



bw = 100 Mbit/sec
path RTT = 10ms

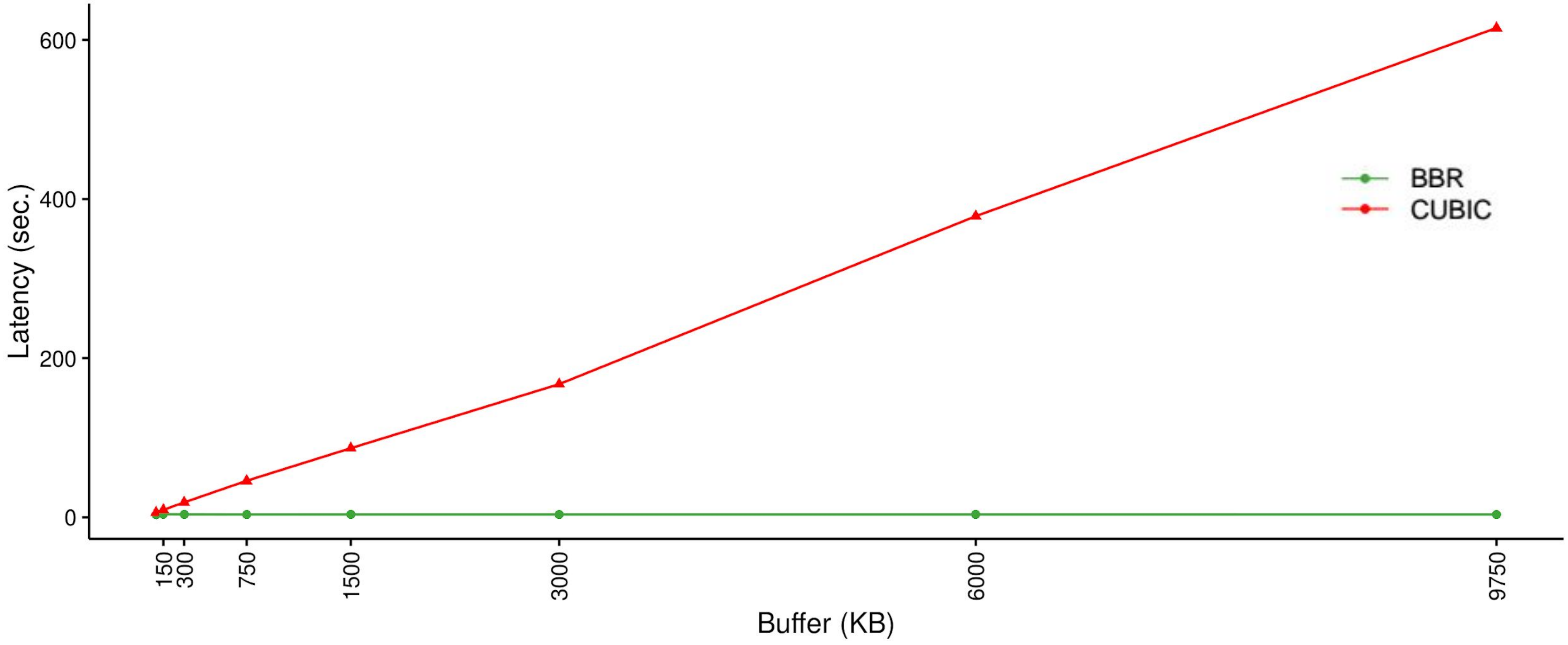
1. Flow 1 briefly slows down to reduce its queue every 10s (PROBE_RTT mode)
2. Flow 2 notices the queue reduction via its RTT measurements
3. Flow 2 schedules to enter slow-down 10 secs later (PROBE_RTT mode)
4. Flow 1 and Flow 2 gradually converge to share BW fairly

BBR: fully use bandwidth, despite high packet loss



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

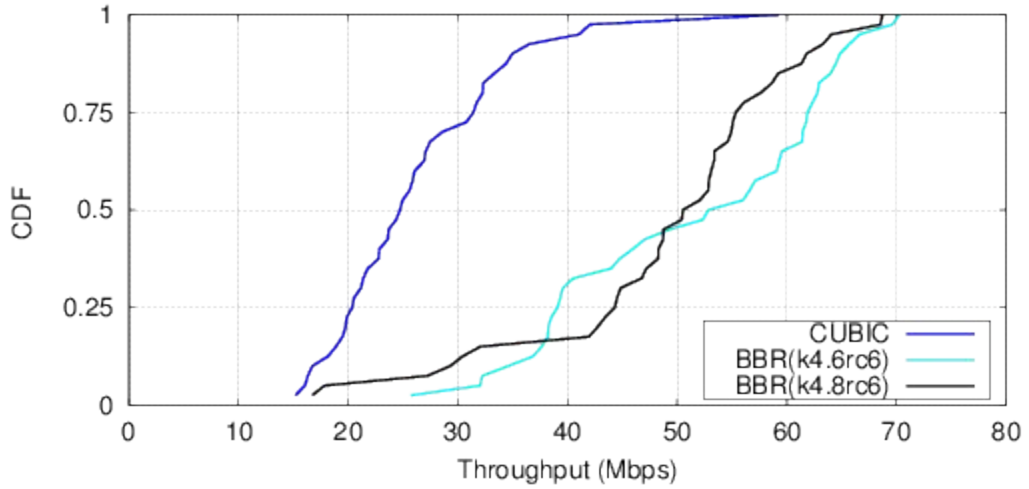
BBR: low queue delay, despite bloated buffers



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

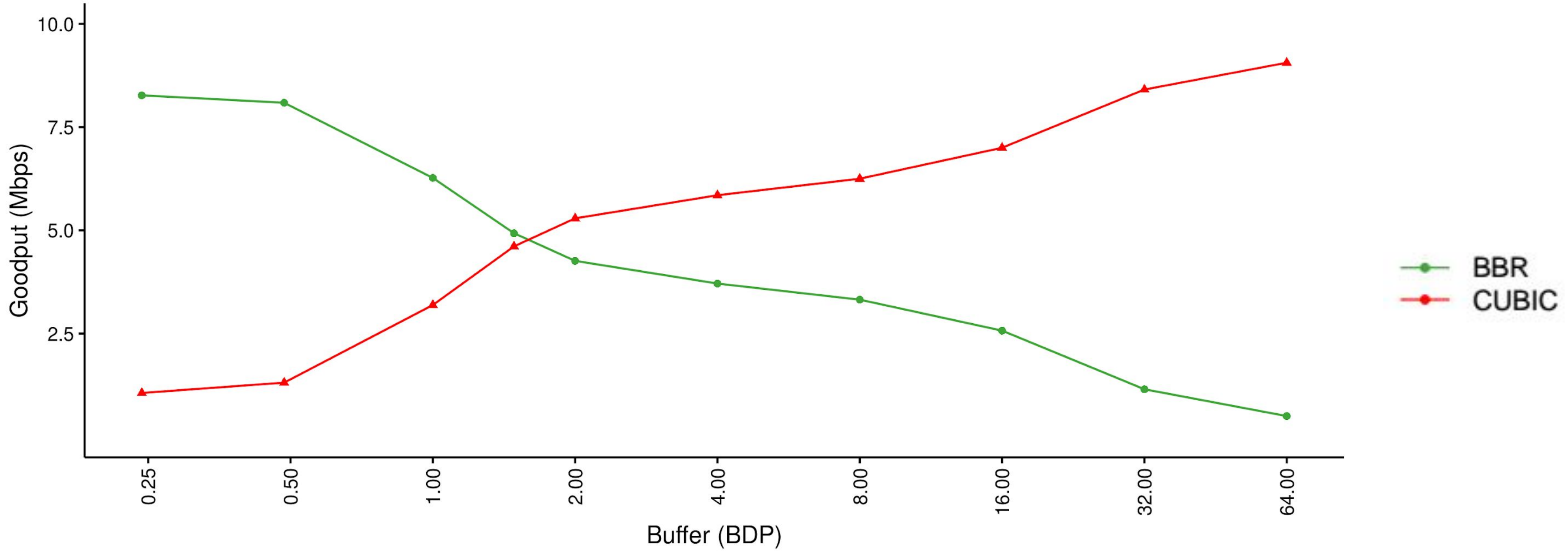
BBR: robust detection of full pipes > faster start up

- **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]):

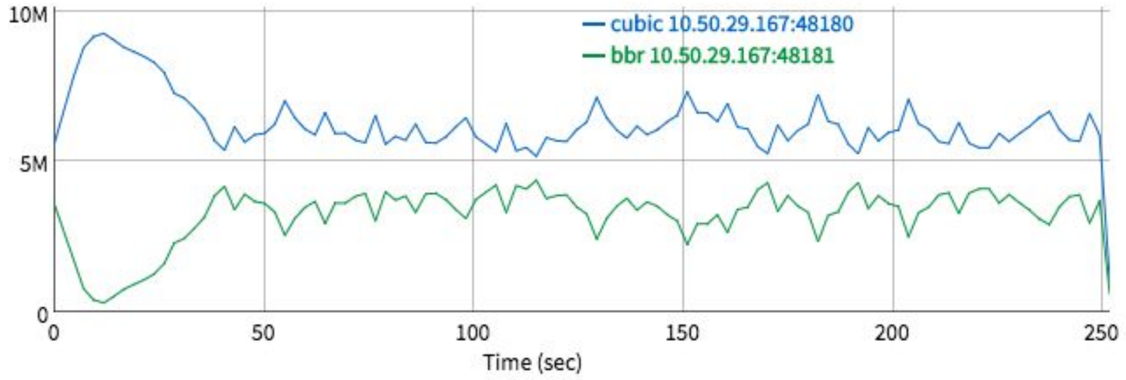


Improving dynamics w/ with loss based CC

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...