BBR Congestion Control Work at Google

IETF 102 Update

Neal Cardwell, Yuchung Cheng,

C. Stephen Gunn, Soheil Hassas Yeganeh

Ian Swett, Jana Iyengar*, Victor Vasiliev

Priyaranjan Jha, Yousuk Seung, Kevin Yang, Matt Mathis

Van Jacobson

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

Outline

- Overview and status of BBR congestion control
- BBR v2 research update
  - Research goals and focus areas
  - Design rationale
  - Illustrative lab results from BBR v2 research pre-release code
- Conclusion and ongoing work on BBR

BBR v1: 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 | IETF 101
  - Overview in Feb 2017 CACM

BBR v2: current research focus

- Improving coexistence/fairness with loss-based CC
  - Adapting BBR bandwidth-probing time scale to coexist with Reno/CUBIC
- Reducing queue pressure (packet loss, queueing delay)
  - Using loss and ECN signals for:
    - New model for safe range for in-flight data
    - New estimators for exiting STARTUP faster
- Speeding up min_rtt convergence
  - Making PROBE_RTT more frequent
- Reducing throughput variance
  - Making PROBE_RTT less drastic

BBR v2 design principles in a nutshell

- **Leave headroom: leave space for entering flows to grab**
- **React quickly: using loss/ECN, adapt to delivery process *now* to maintain flow balance**
- Don't overreact: don't do a multiplicative decrease on every round trip with loss/ECN
- **Probe deferentially: probe on a time scale to allow coexistence with Reno/CUBIC**
- Probe robustly: try to probe beyond estimated max bw, max volume before we cut est.
- **Avoid overshooting: start probing at an inflight measured to be tolerable**
- Grow scalably: **start probing at 1 extra packet**; grow exponentially to use free capacity

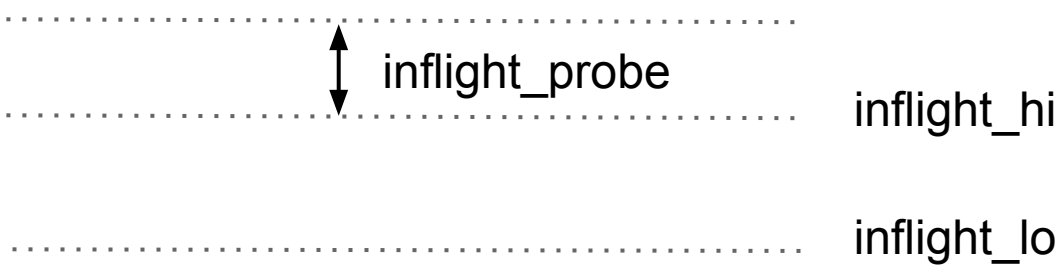
(**bold** = new in v2)

Key in v2: new model for safe range for in-flight

All of these new design principles need an explicit, independent, tight bound on in-flight data

BBR v2 model:

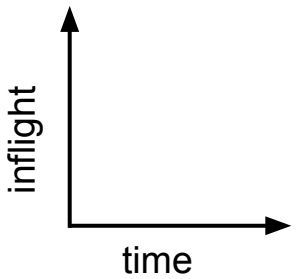
- Mostly cruise at an operating point that maintains flow balance and leaves headroom
  - inflight_lo: conservative in-flight bound based on recent loss/ECN signals
- Periodically probe beyond flow balance to probe robustly for higher volume, bandwidth
  - inflight_hi: max volume flow had in flight before signals of congestion (loss, ECN)
- If probing higher inflight doesn't trigger loss/ECN signals , grow probing rapidly
  - inflight_probe: incremental probe data beyond inflight_hi (during probing)



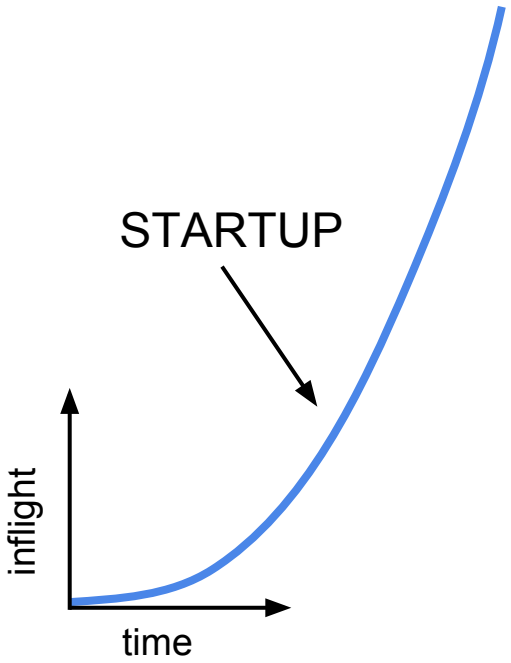
BBR v2 flow life cycle

- At a high level, BBR v2 has the same state machine states as v1
- But many of the mechanism details are new

(**bold** = new in v2)

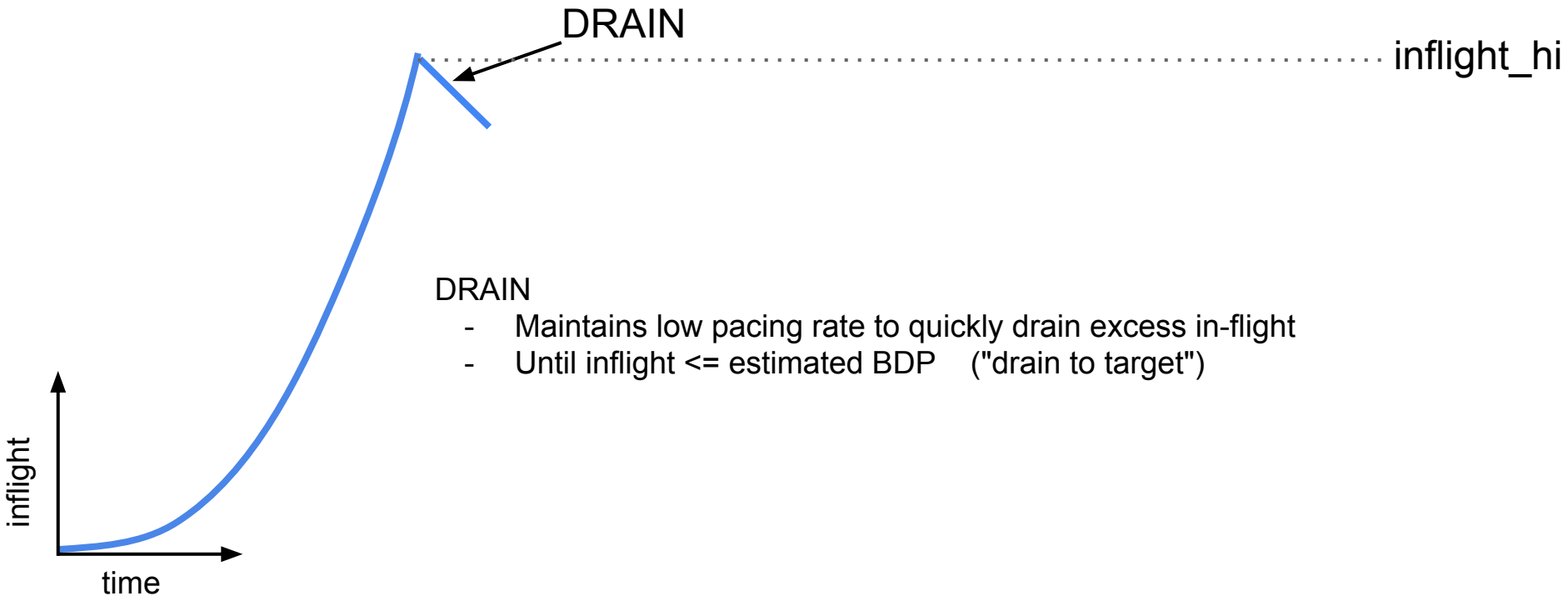


BBR v2 flow life cycle



STARTUP
- Doubles sending rate and inflight
- **Sets inflight_hi to estimated max safe in-flight volume if:**
  - **Filtered loss rate is too high**
  - **Filtered ECN rate signal is too high**
- Exits when either:
  - Bandwidth samples plateau
  - **inflight_hi is set**

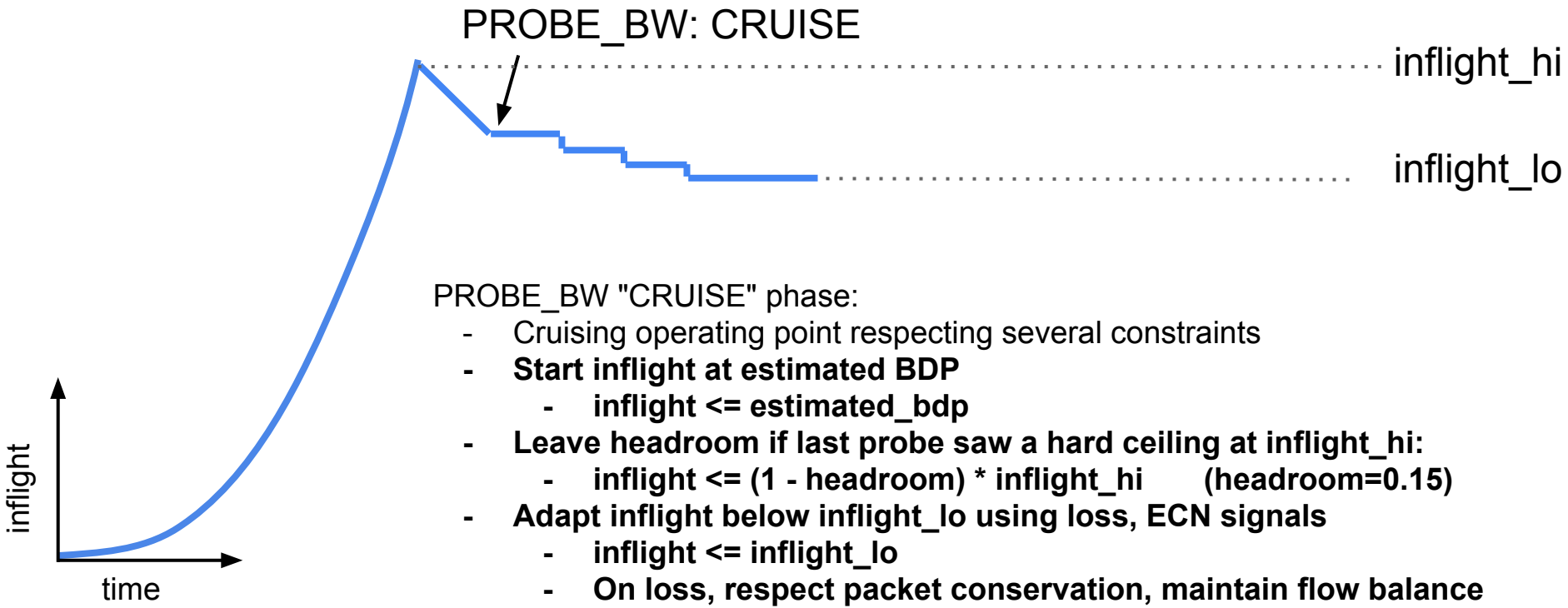
BBR v2 flow life cycle



DRAIN

- Maintains low pacing rate to quickly drain excess in-flight
- Until inflight <= estimated BDP ("drain to target")

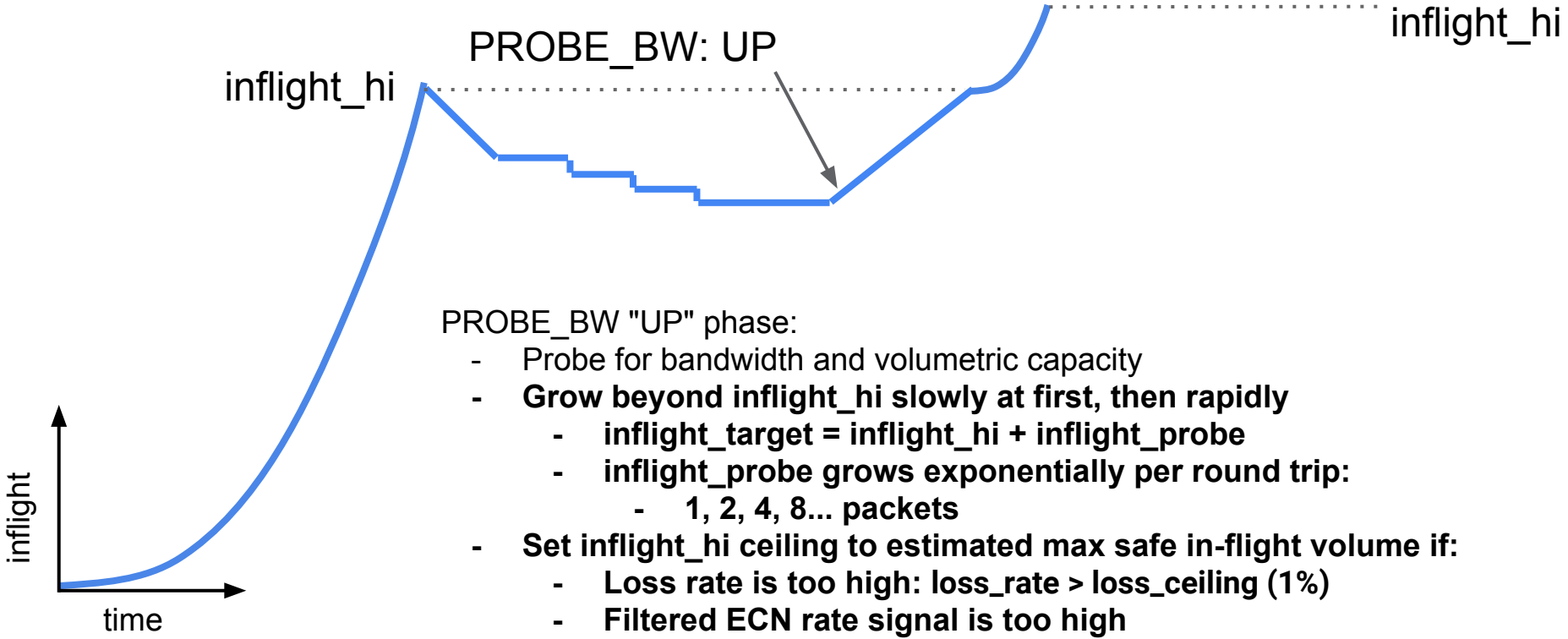
BBR v2 flow life cycle



PROBE_BW "CRUISE" phase:

- Cruising operating point respecting several constraints
- **Start inflight at estimated BDP**
  - **inflight <= estimated_bdp**
- **Leave headroom if last probe saw a hard ceiling at inflight_hi:**
  - **inflight <= (1 - headroom) * inflight_hi (headroom=0.15)**
- **Adapt inflight below inflight_lo using loss, ECN signals**
  - **inflight <= inflight_lo**
  - **On loss, respect packet conservation, maintain flow balance**
    - **inflight_lo -= packets_lost**
  - **On filtered ECN signals, cut inflight using EWMA mark rate α**
    - **inflight_lo -= k * (α - hysteresis) * packets_delivered**

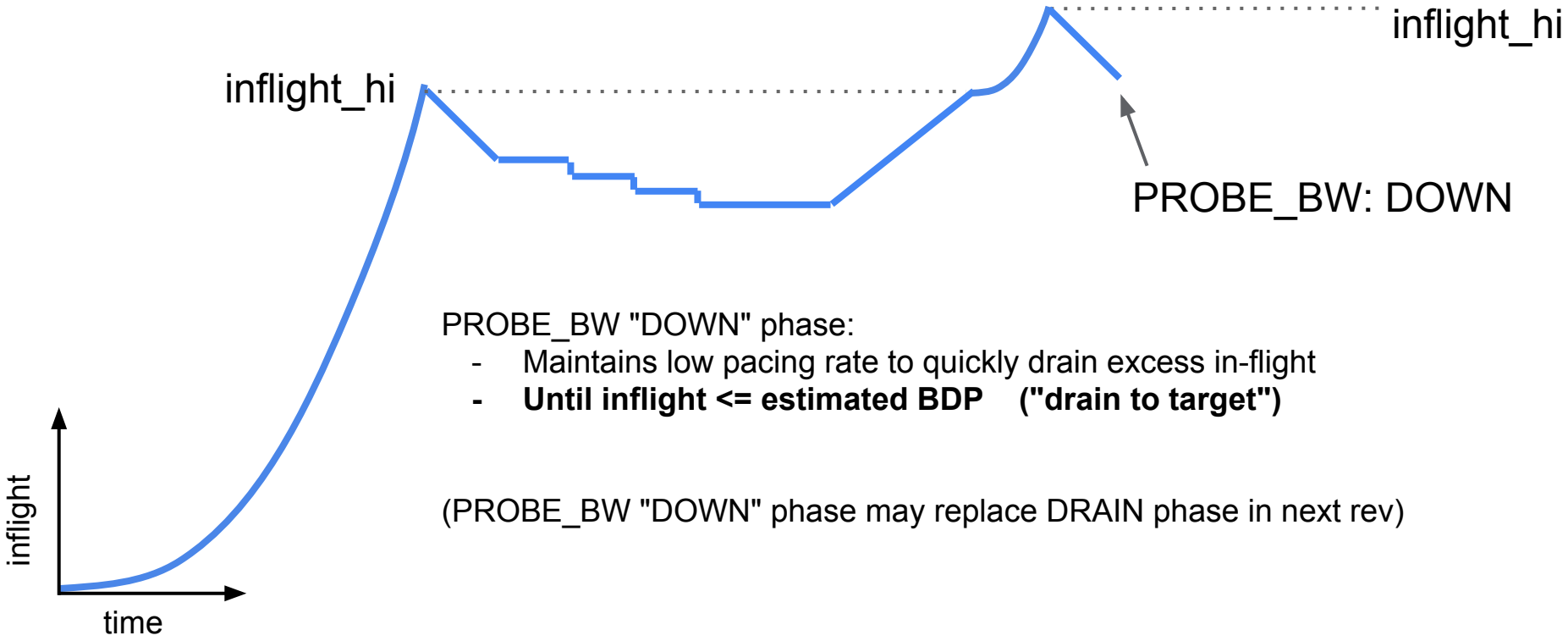
BBR v2 flow life cycle



PROBE_BW "UP" phase:

- Probe for bandwidth and volumetric capacity
- **Grow beyond inflight_hi slowly at first, then rapidly**
  - **inflight_target = inflight_hi + inflight_probe**
  - **inflight_probe grows exponentially per round trip:**
    - **1, 2, 4, 8... packets**
- **Set inflight_hi ceiling to estimated max safe in-flight volume if:**
  - **Loss rate is too high: loss_rate > loss_ceiling (1%)**
  - **Filtered ECN rate signal is too high**
- **Terminate probing upon any of:**
  - **Estimated queue is too high (inflight > 1.25 * estimated_bdp)**
  - **Hit inflight_hi loss or ECN ceiling**

BBR v2 flow life cycle



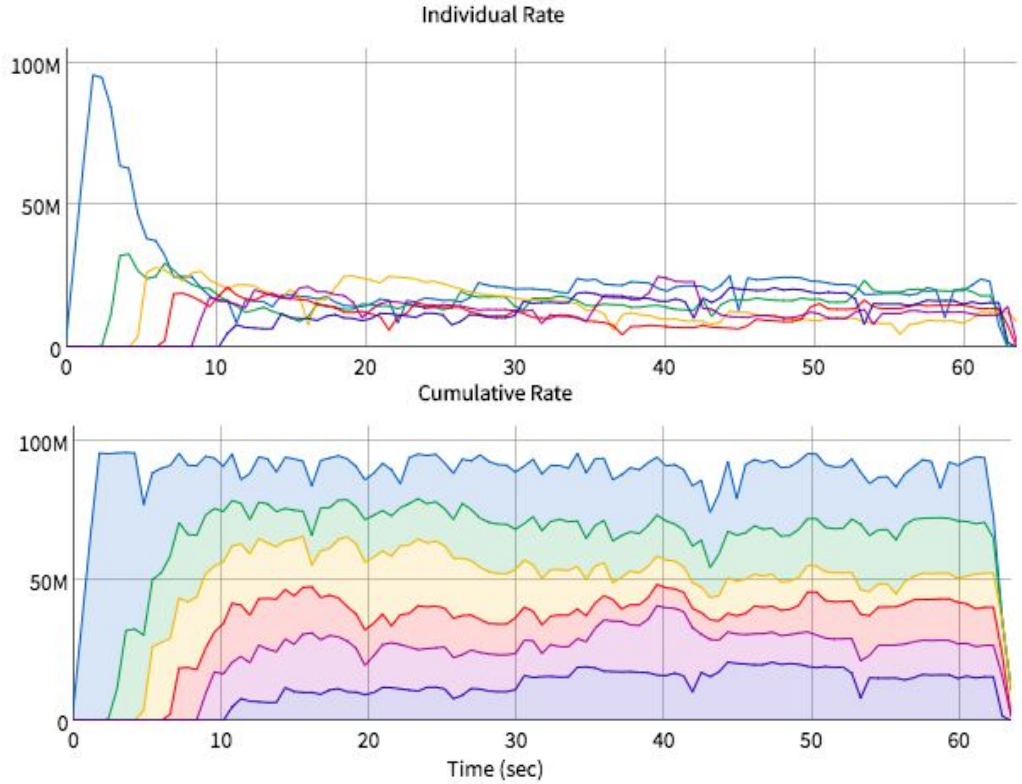
PROBE_BW "DOWN" phase:

- Maintains low pacing rate to quickly drain excess in-flight
- **Until inflight <= estimated BDP ("drain to target")**

(PROBE_BW "DOWN" phase may replace DRAIN phase in next rev)

BBR v2 life cycle example: shallow buffer

- Example: 6 BBR, 100M, 100ms, buffer = 5% of BDP (41 packets); t={0, 2, 4, 6, 8, 10} sec



	bw	retrans
BBR 1	23.6 M	2.0%
BBR 2	16.9 M	0.9%
BBR 3	15.3 M	0.8%
BBR 4	12.3 M	0.9%
BBR 5	14.2 M	0.8%
BBR 6	14.3 M	0.7%

Coexistence with loss-based congestion control

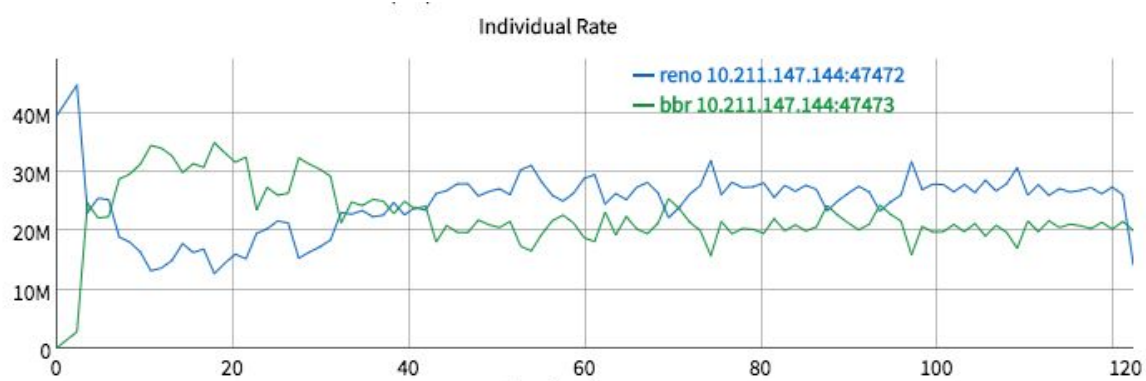
- **Goal**: ensure Reno, CUBIC can continue to work well in contexts where they do today:
  - Intra-datacenter/LAN traffic: support 100M through 40G Ethernet
  - Internet last mile traffic: support up to 25Mbps (4K Video) at RTT of 40ms
- **Challenge**: Reno/CUBIC need *long* periods of no loss to utilize high BDPs

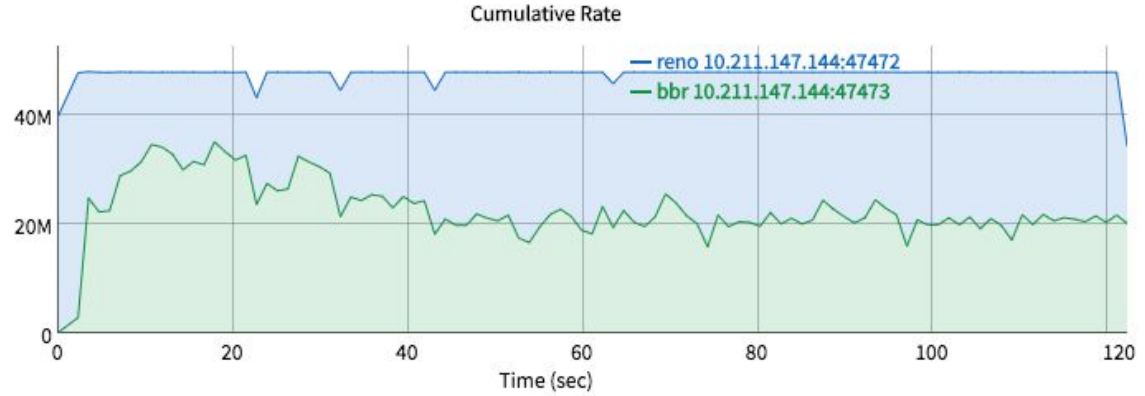
Coexistence: design

- **Strategy**:
  - 1: Estimate the bandwidth available to our flow
  - 2: Adapt (within bounds) the frequency at which a BBR flow probes and knowingly risks packet loss to allow Reno (and thus CUBIC) to reach our bandwidth
- Dual time scales, including bounded Reno pseudo-emulation (like CUBIC)
  - T_bbr: BBR-native time-scale: 2-5 secs, as an increasing log-like function of bw
  - T_reno: Reno-coexistence time scale: min(BDP in packets, 50) * RTT
  - Time between bandwidth probe phases (PROBE_BW "UP" phases):
    - T_probe = min(T_bbr, T_reno)
- Bandwidth estimator filter window now simply covers last 2 PROBE_BW cycles

Coexistence with loss-based congestion control

- Example: 1 Reno, 1 BBR, 50M, 40ms, buffer = 1xBDP; start time {0, 2} secs





	bw	retrans
Reno	24.9 M	0.19%
BBR	23.1 M	0.09%

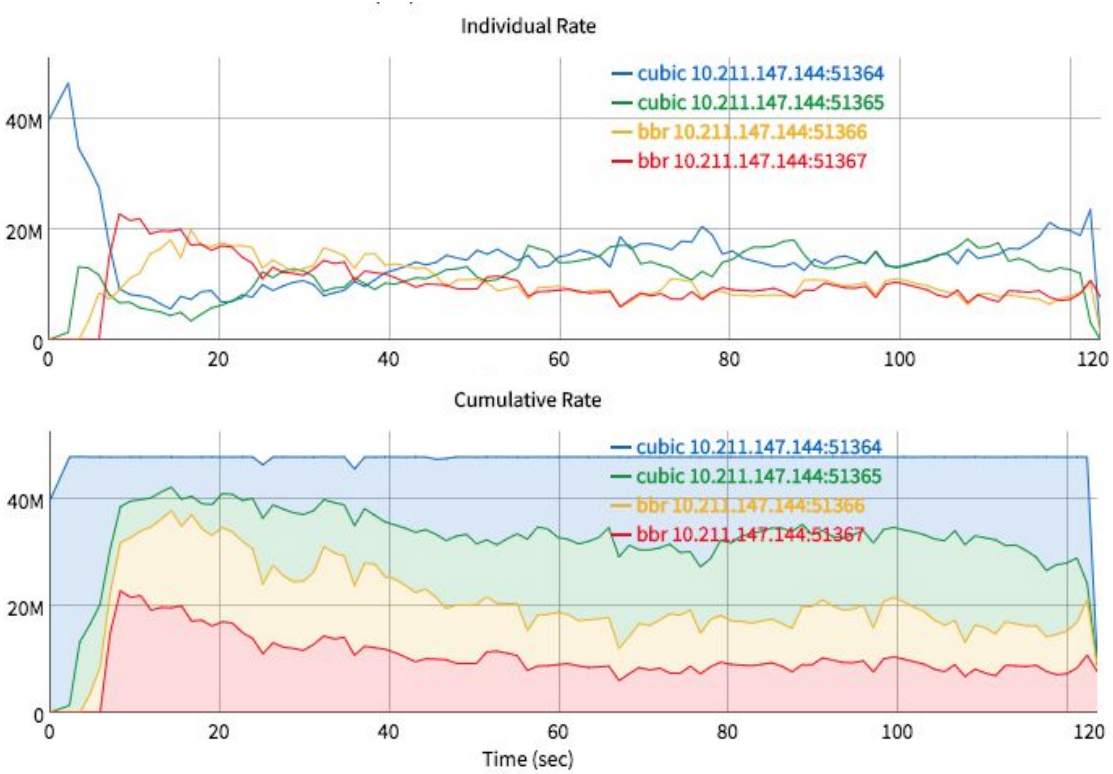
Improved fairness:

BBR v1: 92% of bw

BBR v2: 48% of bw

Coexistence with loss-based congestion control

- Example: 2 CUBIC, 2 BBR, 50M, 40ms, buffer = 1xBDP; start time {0, 2, 4, 6} secs



	bw	retrans
CUBIC 1	14.8 M	0.40%
CUBIC 2	12.4 M	0.12%
BBR 1	10.8 M	0.16%
BBR 2	10.9 M	0.21%

Improved fairness:

BBR v1: 94% of bw

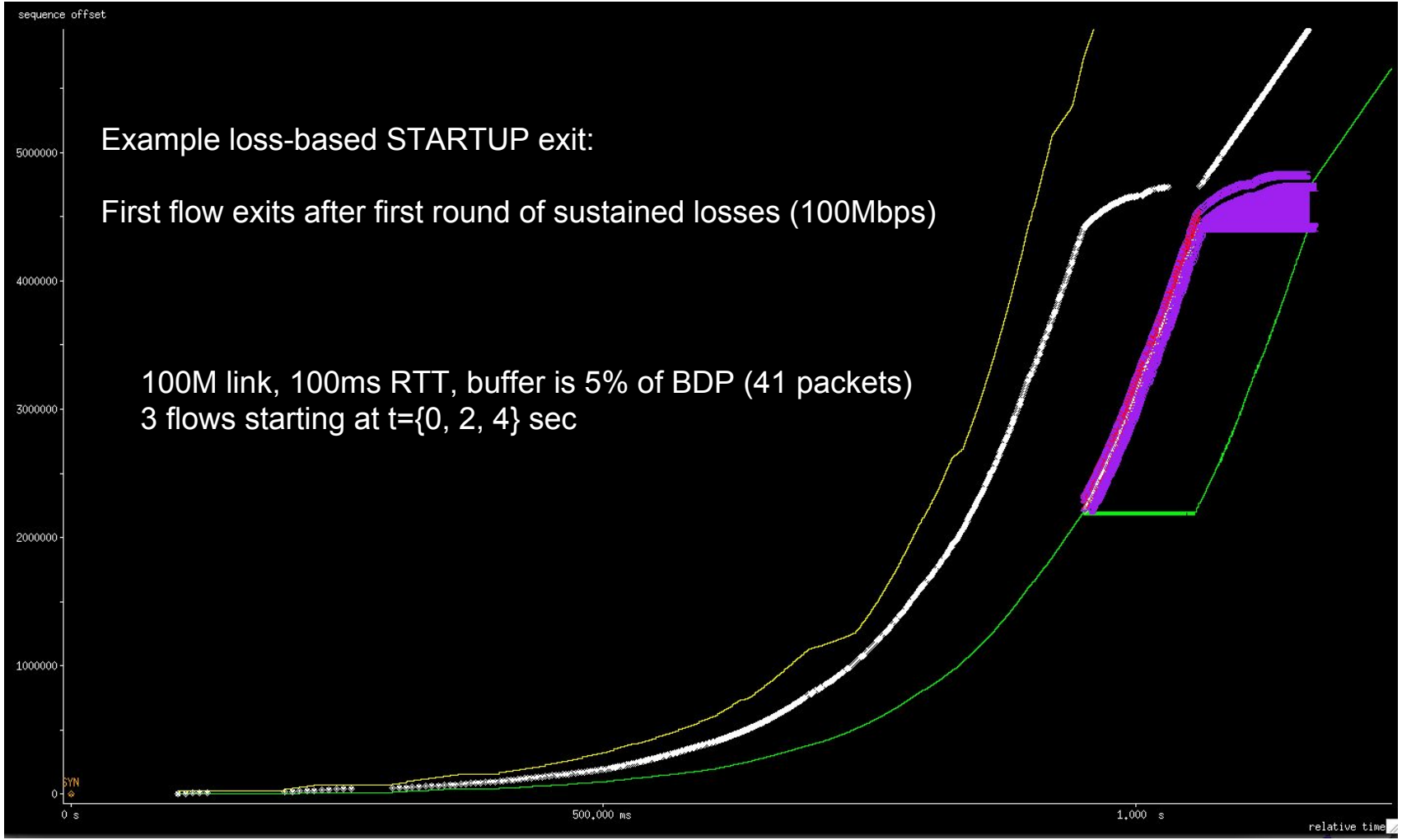
BBR v2: 44% of bw

Exiting STARTUP faster: motivation

- BBR STARTUP
  - Doubles sending rate each round trip (analogous to slow-start)
  - Sending rate is up to 2x the delivery rate
  - Inflight is up to 2x to 2.89x the BDP
- Staying in startup after the pipe and buffer are both full can cause sustained high loss
  - And thereby drive loss-based CC down to low rates
- Thus exiting STARTUP quickly is important
  - For reducing loss
  - For fairness

Exiting STARTUP faster: using Loss, ECN signals

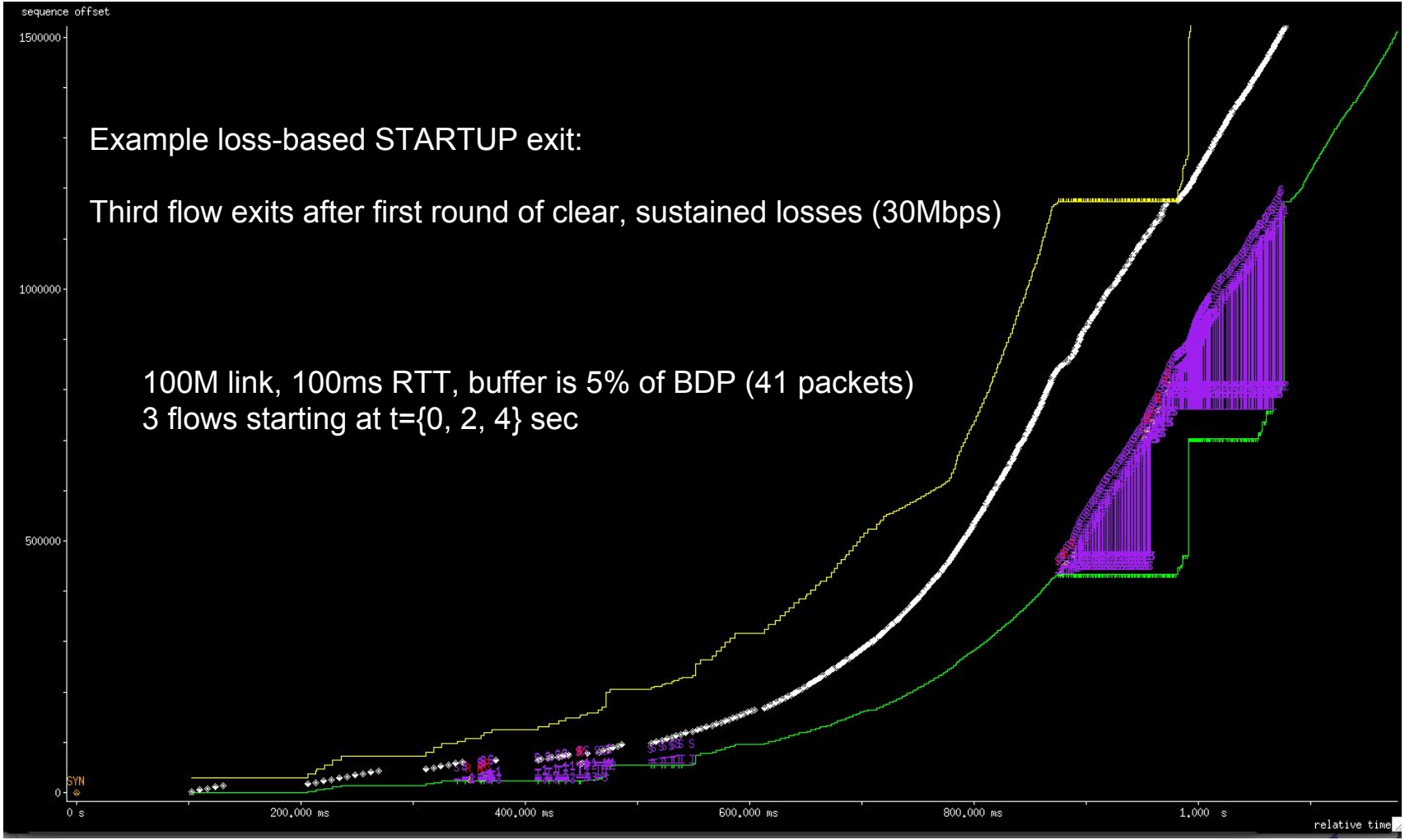
- BBR v1 STARTUP exit
  - After 3 round trips where bw didn't increase by 25%, exit STARTUP, enter DRAIN
- V2 prototype: Loss-based STARTUP exit
  - After entering fast recovery in STARTUP
    - Enter packet conservation (match current delivery bw, avoid further losses)
  - At end of each round trip in recovery:
    - If loss_rate > loss_ceiling (1%) and num_loss_gaps > K (8) then
      - Enter DRAIN and drain in-flight to estimated BDP
    - Else
      - Exit packet conservation and continue
- V2 prototype: ECN-based STARTUP exit
  - At end of each round trip that saw ECE-marked packets:
    - If ecn_mark_rate > target_ecn_mark_rate (50%) then
      - Enter DRAIN and drain in-flight to estimated BDP



Example loss-based STARTUP exit:

First flow exits after first round of sustained losses (100Mbps)

100M link, 100ms RTT, buffer is 5% of BDP (41 packets)
3 flows starting at t={0, 2, 4} sec


sequence offset
1500000
1000000
500000
0
SYN
0 s
200,000 ms
400,000 ms
600,000 ms
800,000 ms
1,000 s
relative time
Example loss-based STARTUP exit:
Third flow exits after first round of clear, sustained losses (30Mbps)
100M link, 100ms RTT, buffer is 5% of BDP (41 packets)
3 flows starting at t={0, 2, 4} sec

Improving PROBE_RTT

The problems:

- Slow min_rtt convergence because ProbeRTT is rare
  - Slow convergence (20-30 secs), because PROBE_RTT is rare (every 10 secs)
- Risk of high tail latency because ProbeRTT is drastic
  - Could cause high tail latency because PROBE_RTT cuts inflight low (4 packets)

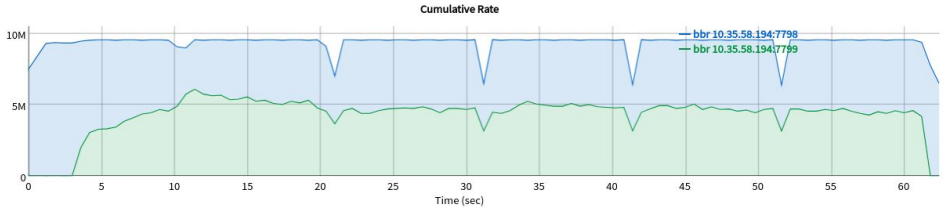
Proposed approach for improving PROBE_RTT:

- Make it less drastic: e.g. inflight ~= 0.75x estimated_bdp
  - Why 0.75x? 0.75 * 1.25 = 0.9375 (so our inflight should be less than real BDP)
- Make it more frequent: e.g. every 2.5 secs
  - Why 2.5s? (0.2*0.75 + 2.5*1.0) / (0.2+2.5) = 98.1% (so our utilization should still meet our target)

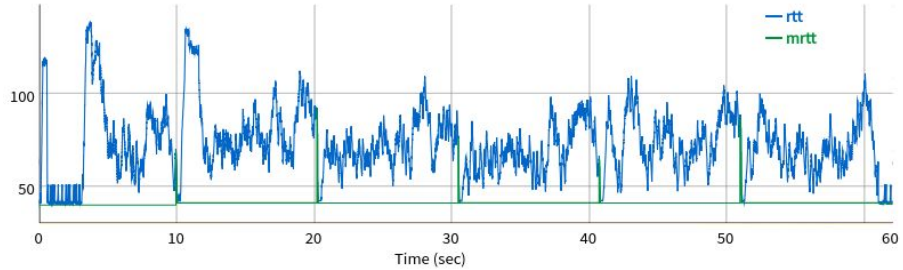
PROBE_RTT example: BBR vs BBR

Before (BBR v1):

Rates for 2 BBR flows



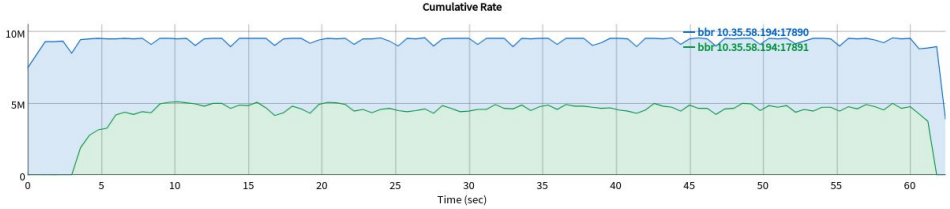
RTT



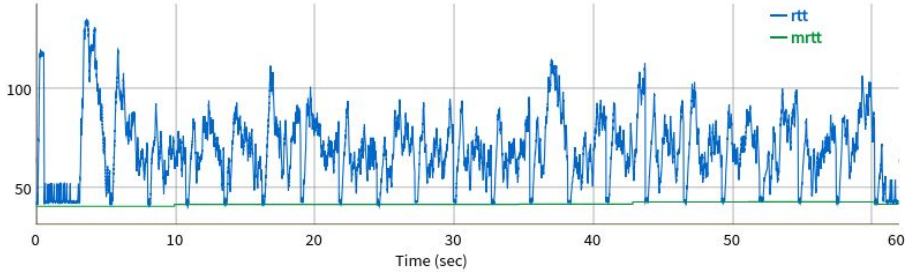
10M, 40ms, buffer = 10xBDP

After (BBR v2):

Rates for 2 BBR flows



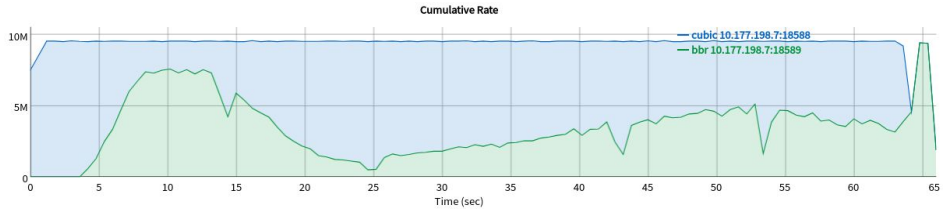
RTT



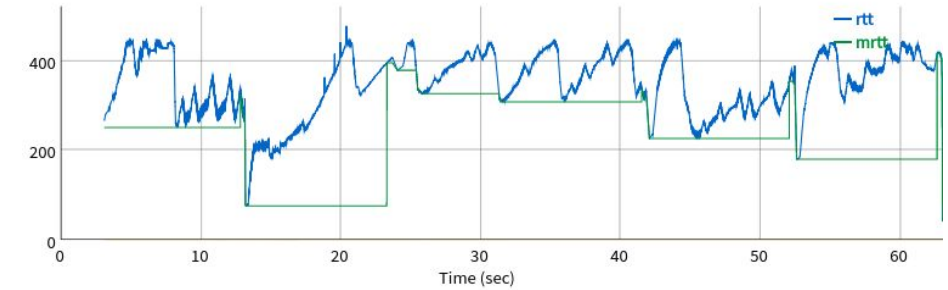
PROBE_RTT example: BBR vs CUBIC

Before (BBR v1):

Rates for 1 Cubic and 1 BBR



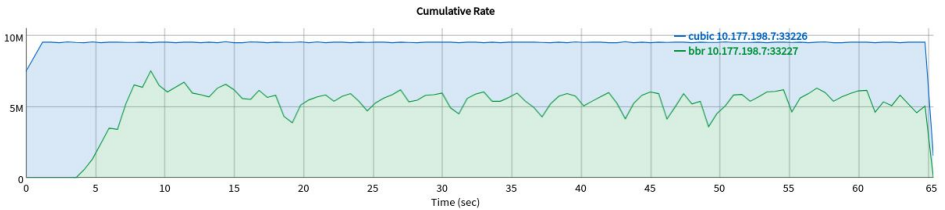
RTT



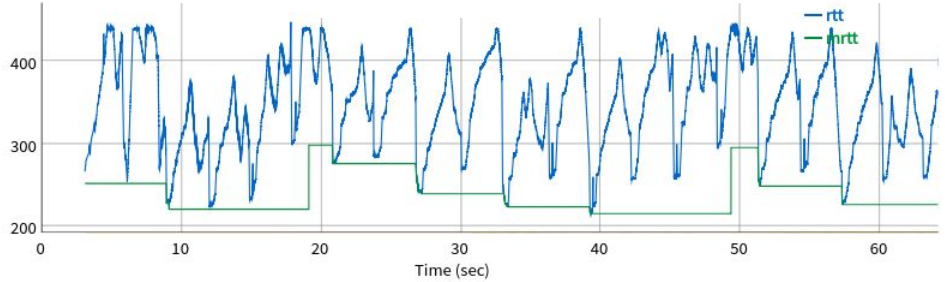
10M, 40ms, buffer = 10xBDP

After (BBR v2):

Rates for 1 Cubic and 1 BBR



RTT



Conclusion

- Status of BBR v1
  - Deployed widely at Google
  - Open source for Linux TCP and QUIC
  - Documented in IETF Internet Drafts
- Actively working on BBR v2
  - Linux TCP and QUIC at Google; current focus areas:
    - Packet loss and ECN signals
    - Coexistence 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

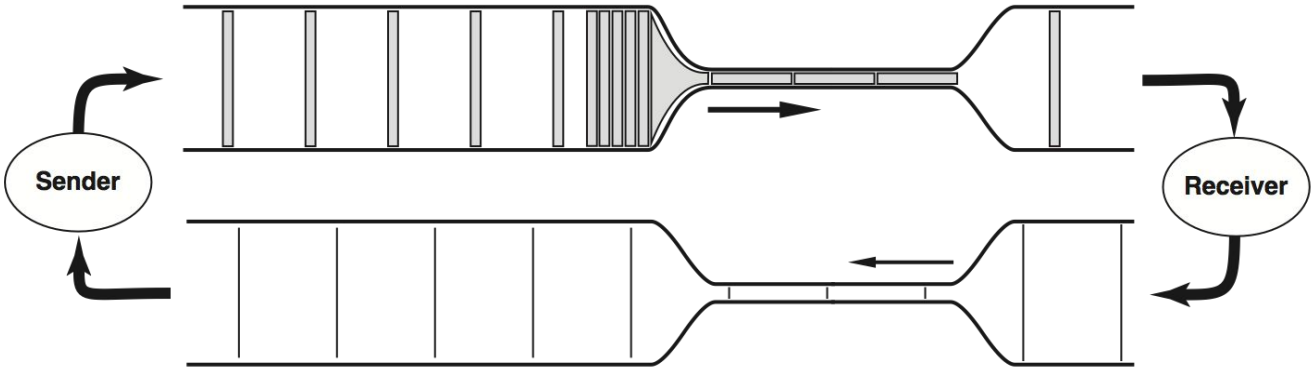
Internet Drafts, paper, code, mailing list, talks, etc.

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...

Reducing queue pressure

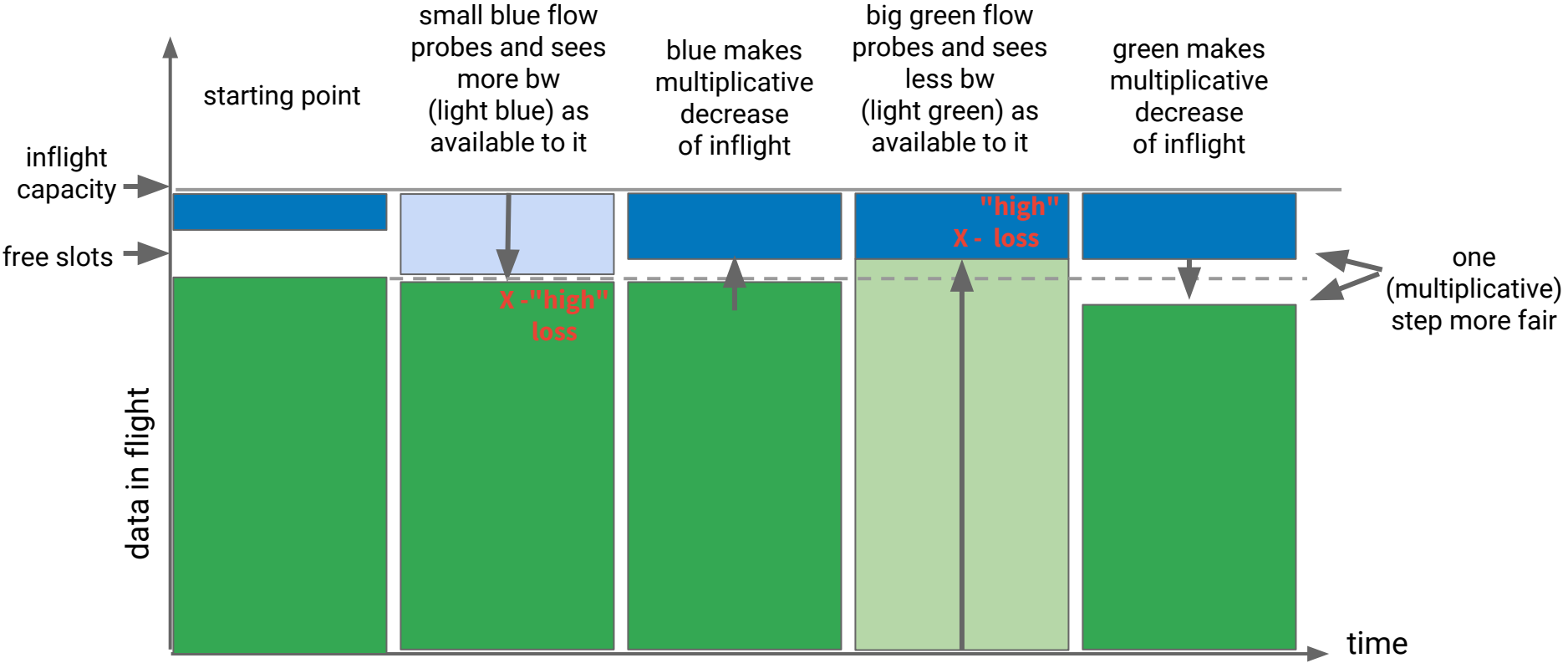
- To reduce queue pressure (and delay, loss) need an explicit, tight bound on in-flight data
- Building a robust model for a bound on in-flight data is challenging
- Challenges with signals:
  - ECN: Few paths outside datacenters provide ECN
  - Delay: Common to have no usable delay signal (policers, shallow WAN buffers)
  - Loss: Not all packet loss means a bottleneck is full on a sustained basis



Leaving headroom

- With shallow buffer (or zero buffer, like policers) CCs should aim to keep some fraction of slots free (in pipe and/or buffer)
- Operating point for in-flight data while cruising should leave some headroom
- Why leave headroom?
  - To reduce queue: lower loss, delay
  - To allow room for dynamic/intermittent cross traffic without excessive loss or delay
  - To not starve loss-based CC (Reno/CUBIC)
  - For faster, more scalable convergence toward a fair share
- Headroom accelerates fairness convergence dynamics
  - Small flows can slow down big flows by claiming those free slots at a higher rate than big flows

Headroom and convergence toward fair share



2 flows; same RTT; capacity = aggregate inflight beyond which all packets are dropped; used/free slots may be in queue or link

Google

NOTE: Whichever flow probes first can claim the free slots and increase its share (as with Reno/CUBIC convergence)

The bottleneck-probing dilemma

- The dilemma:
  - Sometimes a flow should respect flow balance:
    - If a path *is* full, traffic needs to adapt to restore flow balance
  - Sometimes a flow should exceed flow balance:
    - Any single loss or ECN mark is ambiguous
    - So to tell if a bottleneck is full, traffic needs to eventually send faster to probe
  - But if bottleneck *is* full and buffer is shallow, bandwidth probing causes loss
    - Causes low throughput for loss-based CC
    - Causes High tail latency due to loss recovery for any CC
- It's tricky to thread the needle...

BBR v2: probing growth rationale

- Are there more free slots (pipe and/or buffer) available? 2 main cases:
  - No: There are **not** more free slots: here we need to grow gently at first
    - We are filling buffer slots and causing higher delay and loss
    - Need to grow slowly at first to control blast radius
    - We don't want full multiplicative/exponential growth of inflight
  - Yes: There **are** more free slots: here we need to grow scalably (asymptotically very superlinear)
    - If more bw becomes available (e.g. other flows leave) need to grow 1000x rapidly
      - e.g. grow from 10Mbps to 10Gbps in a "usable" time scale
    - Growth should be fast enough to avoid forcing users to open more connections
    - Is a cubic curve as scalable as we'd like? Probably not:
      - Cubic: need wait_time *= 10 to get bw *= 1000
    - Ideally we want some exponential component:
      - Exponential w/ k=2: wait_time += 10 to get bw *= 1024
- Implication: inflight_probe starts small, grows exponentially