BBR v2: A Model-based Congestion Control Performance Optimizations

Neal Cardwell, Yuchung Cheng,

Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang,

Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, Matt Mathis

Van Jacobson

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

IETF 106: Singapore, Nov 2019

Outline

- BBR v2 performance optimizations
- Status of the BBR v2 code and Google deployment
- Conclusion

BBR v2 performance optimizations

BBR v2 performance optimizations: goals

(1) Ensure BBRv2 is suitable as a general-purpose CC for LAN, WAN, datacenter, VM guest
  - And as a drop-in replacement for Reno, CUBIC, DCTCP...
    - Yields performance improvements across these environments
    - Has suitable coexistence behavior with Reno, CUBIC

(2) As a stepping stone to (1), deploy BBRv2 for all TCP/QUIC traffic at Google

(3) To prepare for (2), test BBRv2 host and network performance
  - Ensure it exceeds or approximately matches the preceding TCP/QUIC CC algorithms used at Google and widely deployed elsewhere:
    - e.g. Reno, CUBIC, DCTCP

BBR v2 optimizations: summary

- All Google production kernel changes, including BBRv2 internal deployments, must pass a suite of rigorous application benchmarks
  - Wide-ranging applications: search, databases, storage, etc.
  - Metrics: CPU usage, throughput, CPU/operation, median and tail RPC latency
  - Not just small/simple bulk transfer dumbbell tests, throughput, loss rate, fairness
- In the production deployments of congestion control algorithms, many details matter:
  - Host-side: CPU usage, interrupt rate, data packet rate, ACK rate, offload burst size
  - Workloads: Thousands of flows per sender, and/or sharing a tiny BDP, app-limited transfers
- To handle these, implemented performance improvements for BBRv2:

1: Fast path	to match CPU usage for Reno/CUBIC/DCTCP
2: Improved TSO autosizing	to match CPU usage for Reno/CUBIC/DCTCP
3: Faster ACKs	to fix an issue uncovered with Linux TCP
4: Reducing queues with many flows	to enhance BBRv2 beyond Reno/CUBIC/DCTCP

1: TCP BBR v2: fast path

- Much real-world traffic is app-limited: web, RPC, adaptive bit-rate video
- CUBIC/Reno/DCTCP have a de-facto fast path when processing each ACK:
  - if (not cwnd_limited) then return;
- Problem: BBR CPU usage, throughput regressions on some app-limited workloads
  - Why? BBR prioritized simplicity, so ran the entire algorithm on every ACK:
    - Update entire model of the network path (max_bw, min_rtt, etc.)
    - Update probing state machine
    - Adjust all control parameters: pacing rate, offload chunk size, cwnd
  - Caused 2-5% CPU and throughput regression vs CUBIC/DCTCP in some tests
- Solution: a BBRv2 fast path:
  - Only run the portions of the algorithm that are strictly needed, based on each ACK
  - Resolved those CPU regressions, without sacrificing throughput or latency

2: TCP BBR v2: TSO autosizing improvements

- Core Linux TCP stack uses a TSO autosizing algorithm to adapt offload chunk size:

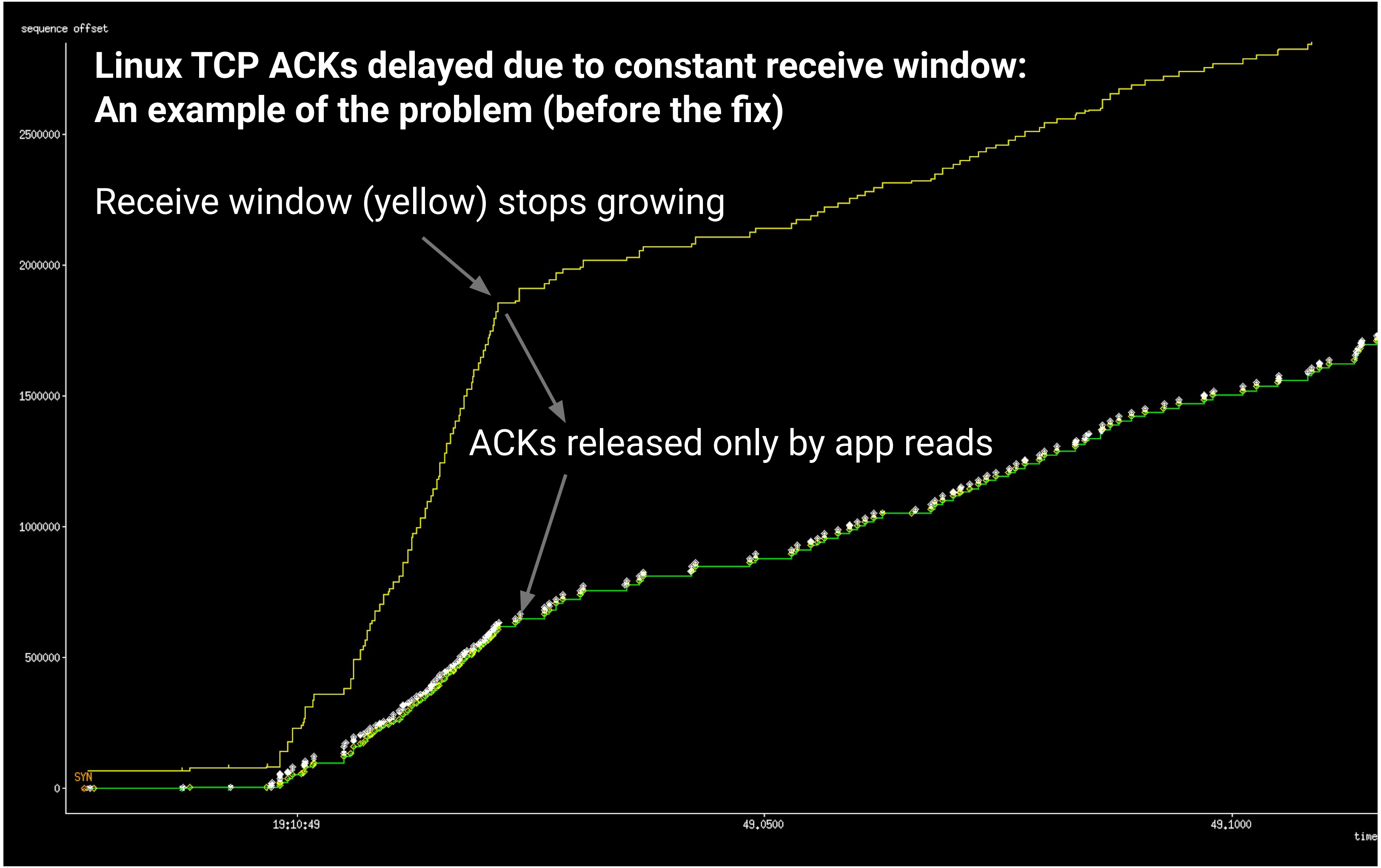
```
pacing_rate = K_scale * cwnd / srtt // K_scale = 2 in slow start; else 1.2
size = pacing_rate * 1ms       // chunk size is proportional to pacing rate
size = max(2 * SND.MSS, size)
size = min(0xFFFF, size)
```

- If the sender host is bottleneck for many flows (e.g. thousands) there's no ECN or loss
  - Thus DCTCP/CUBIC/Reno cwnd, pacing rate, and TSO chunk size can be very large
  - Thus DCTCP/CUBIC/Reno CPU usage is low for such workloads
- Problem: BBR CPU use from setting pacing rate purely based on flow's bandwidth share
  - With many flows, flow's fair share is tiny, so BBR offload chunks were tiny
- Solution: for BBR, adapt offload chunk size based on min_rtt as well as pacing rate:

```
r = min_rtt >> K_r
size += 0xFFFF >> r
```

3: TCP BBR v2: faster ACKs

- Linux TCP delayed ACKs
  - DCTCP, BBRv2 emit an ACK when incoming CE bits change
    - But that mechanism does not trigger under continuous CE marking
  - Linux TCP "ACK per 2*RCV.MSS" logic only ACKed data immediately if both…
    - (1) > RCV.MSS bytes received since sending last ACK, ***and***
    - (2) Next offered receive window >= previous offered receive window
- Problem: If receive window stops growing, this causes sender to become cwnd-limited, waiting for app read to free receive buffer space and trigger ACK
  - Caused 2x higher p99 RPC latency with sustained congestion in some tests
- Solution: remove check (2)
  - Resolved these RPC latency issues
    - Fix applies to any congestion control interacting with a Linux TCP receiver



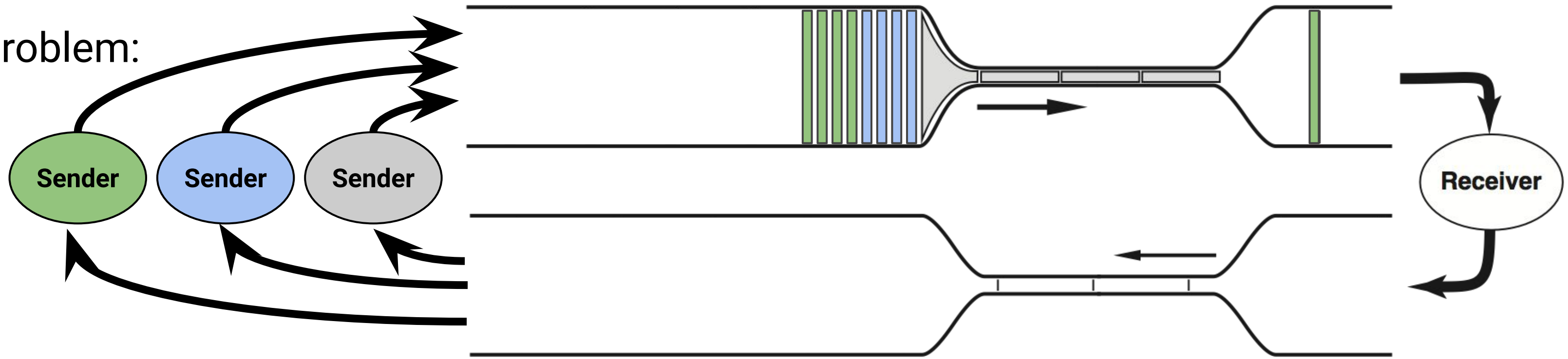
Linux TCP ACKs delayed due to constant receive window:
An example of the problem (before the fix)
Receive window (yellow) stops growing
ACKs released only by app reads
sequence offset
2500000
2000000
1500000
1000000
500000
0
SYN
19:10:49
49.0500
49.1000
time

4: Reducing queues with many flows

Standing queues with many flows sharing small BDPs

- When there are many flows and a low BDP, thus far BBRv2 (like BBRv1, DCTCP, CUBIC, Reno...) runs window-limited, and has a standing queue at the bottleneck when: number_of_flows * min_cwnd > BDP [Morris, "TCP Behavior with Many Flows", '97]
- Problem: in these cases there is a standing queue that is roughly: queue = number_of_flows * min_cwnd - BDP
  - BBRv2 min_cwnd (aside from RTO timeout recovery) is 4 MSS
    - To avoid stop-and-wait behavior with TCP's "ACK every other packet" policy

The problem:



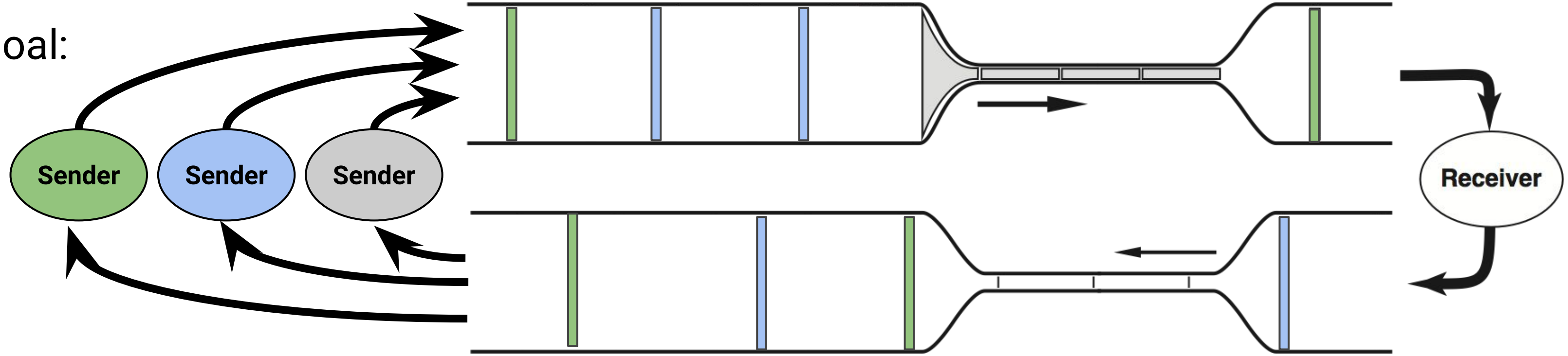
TCP BBR v2: design principles for scalable pacing

Solution: improve use of ECN/loss signal to adapt **pacing** rate to match available bandwidth:

- Multiplicatively decrease pacing rate and in-flight data in response to ECN
  - Quickly match available bandwidth, drain queue, allow convergence toward fair
- Additively increase when ACK suggests queue is low enough (CE=0)
  - Fairness is helped by an additive component to the flow dynamics (Chiu/Jain, '89)

[Also see: SIGCOMM 2015 "TIMELY" paper]

The goal:



TCP BBR v2: pacing to scale to num_flows > BDP

- At end of each round trip with ECN marks:

```
// alpha: EWMA of ECN mark rate
inflight_lo = (1 - alpha*ecn_factor) * inflight_lo
bw_lo       = (1 - alpha*ecn_factor) * bw_lo // <- NEW
```

- Experimental changes (this is work in progress):
  - Reduce bw_lo multiplicatively, in proportion to EWMA ECN mark rate
    - Use ECN to adjust both rate and volume, as with loss
  - Increase ecn_factor from 0.3 to 0.5 (similar to DCTCP)
  - Decouple bw_hi from max bw sample
    - Refine bw_hi to be the maximum bw that showed tolerable ECN mark rate
    - Multiplicative cut to bw_hi if no bw sample has ECN mark rate < ecn_thresh
  - Additive increase to rate and volume of inflight data when starting bw probing:

```
inflight_hi     += 1pkt
bw_hi           += 1pkt/min_rtt
```

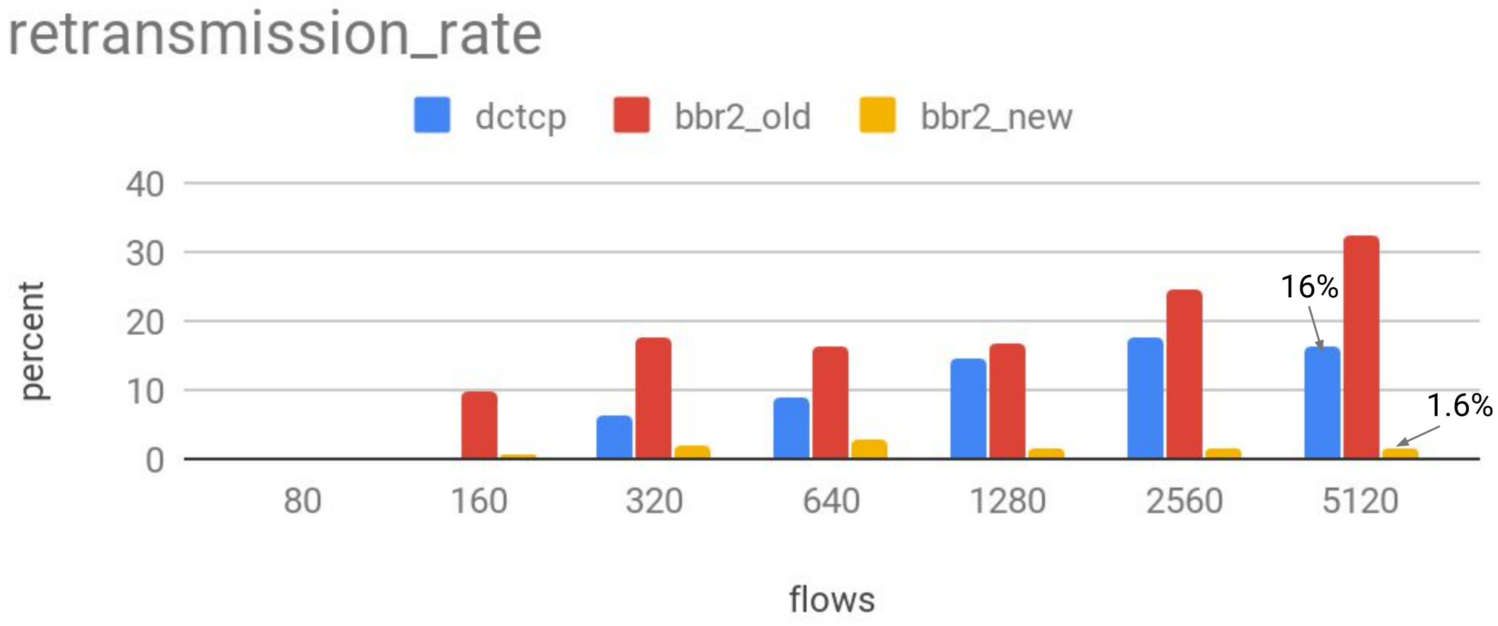
TCP BBR v2: experiments with scalability

- 2 sender machines, 1 receiver machine, on the same switch
- All 3 machines have 50 Gbit/s NIC
- Switch configured for DCTCP-style marking: CE iff instantaneous queue > 80 KBytes
- N 60-second bulk netper flows split across both sender machines, to single receiver
- Metrics collected using ss:
    - throughput
    - smoothed RTT, sampled every 100ms
    - retransmission rate
    - CE rate
    - Jain fairness index
- Comparing 3 congestion control variants:
    - DCTCP: latest Linux DCTCP as of 2019-11-10, up to and including [2874c5fd2842]
    - bbr2_old: baseline bbr2
    - bbr2_new: bbr2 with the experimental changes

Many flows sharing small BDPs: Experiment results

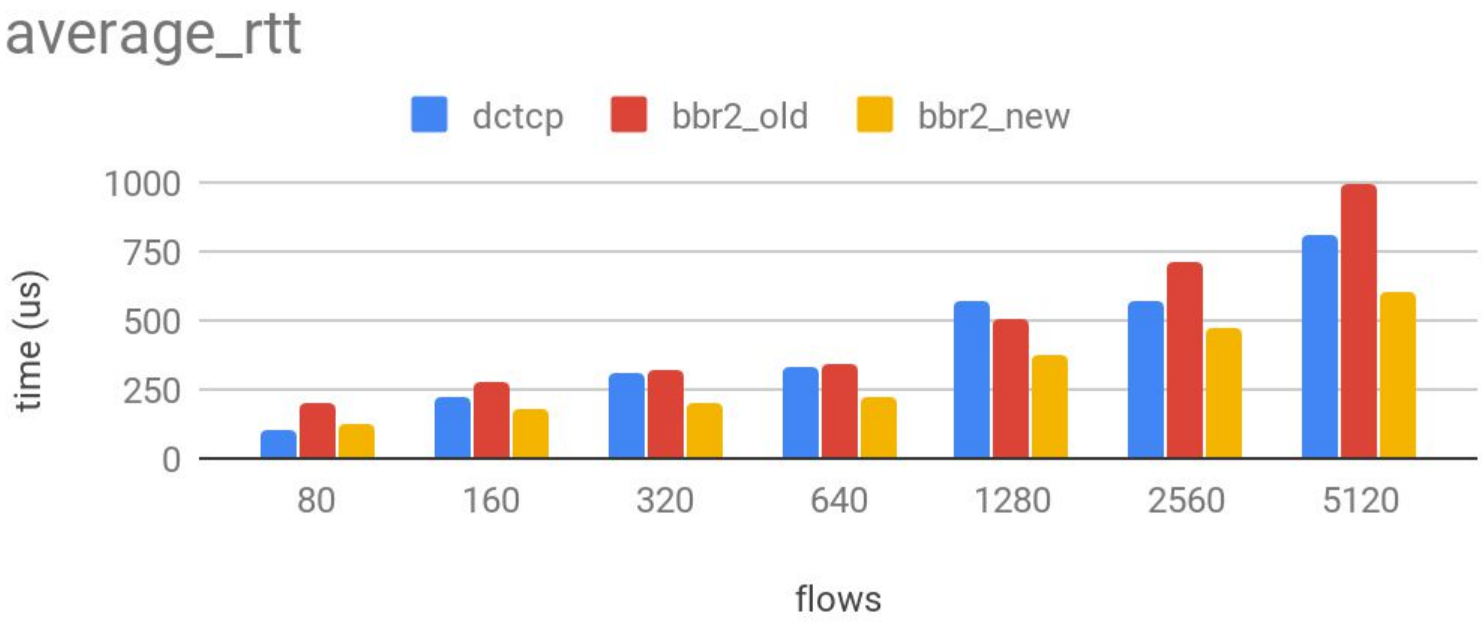
retransmission rate

- bbr2_new has a much lower retransmission rate than DCTCP, for num_flows >= 320



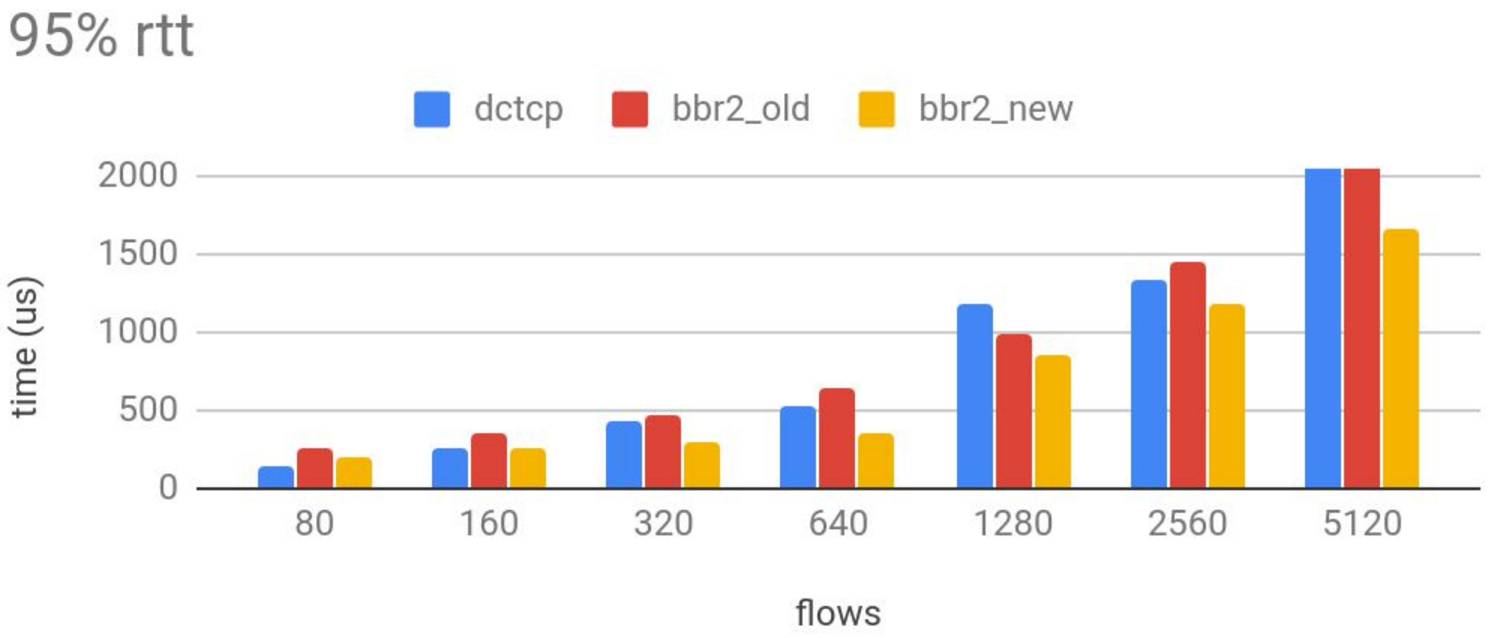
average RTT

- bbr2_new has a lower average RTT, for num_flows >= 160



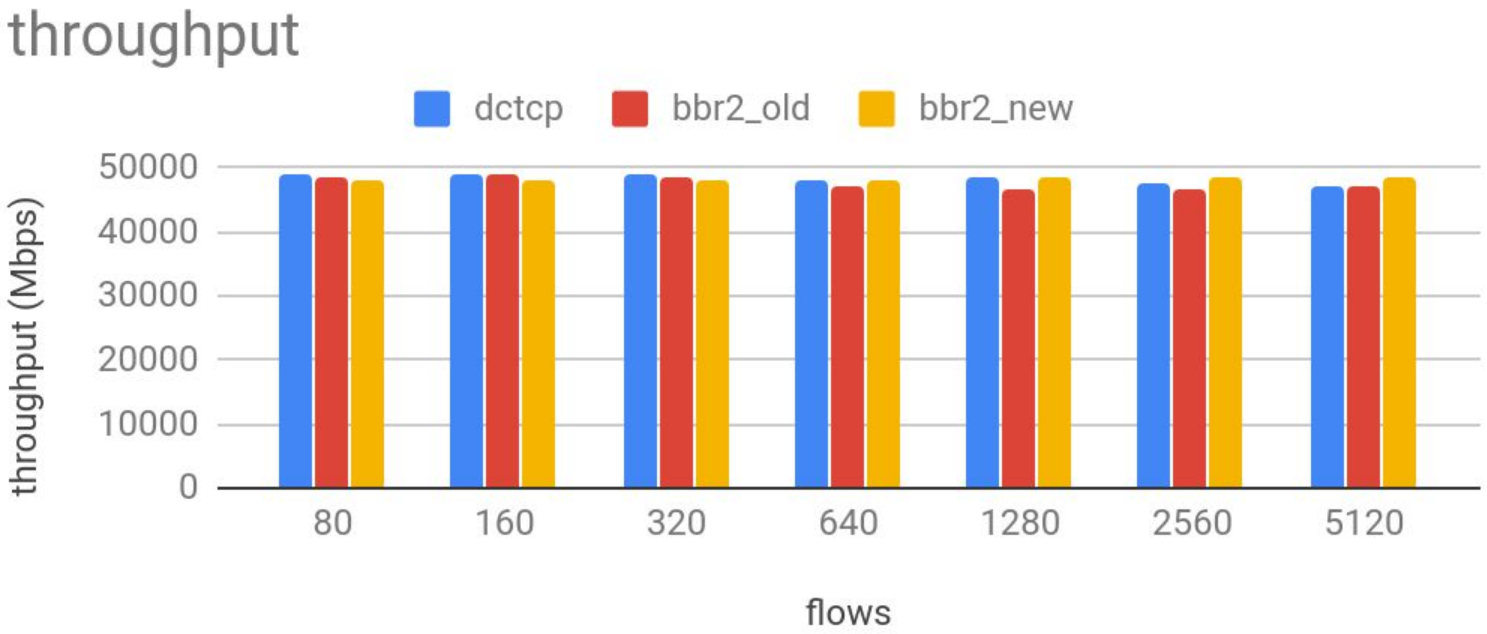
p95 RTT

- bbr2_new has a lower 95% RTT, for num_flows >= 160



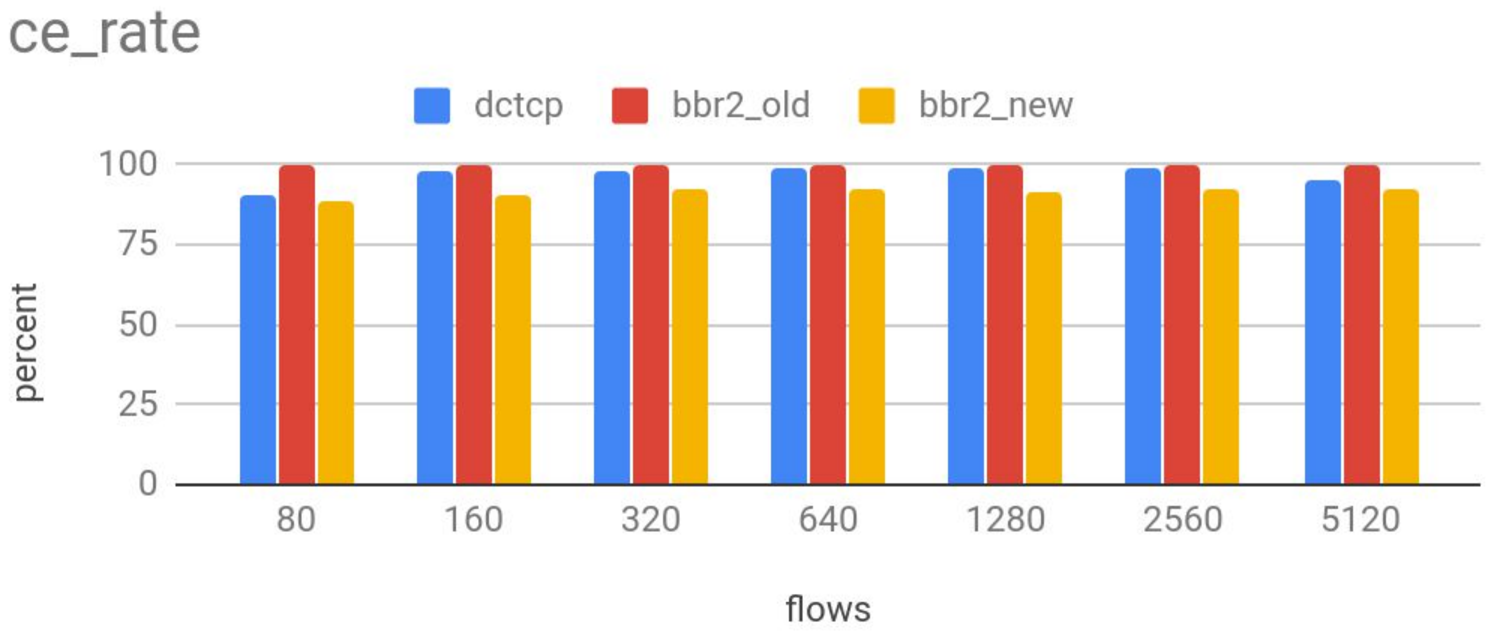
throughput

- All variants have similar throughputs



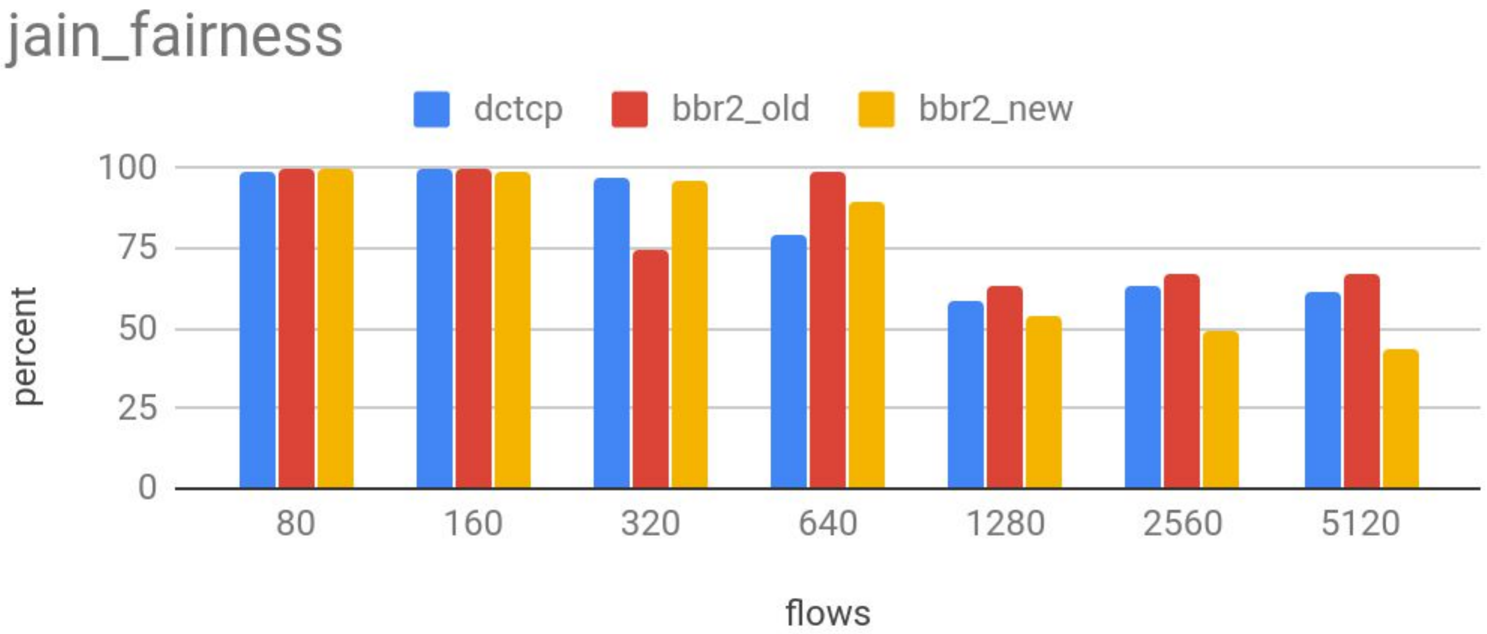
ECN mark rate

- DCTCP and bbr2_new avoid saturation of the congestion experienced (CE) signal



fairness

- All three variants have reasonable, though not ideal, fairness



Wrapping up...

Status of BBR v2 algorithm and code

- Performance improvements #1-4 (and some fixes) pushed to "alpha/preview" release:
  - Linux TCP (dual GPLv2/BSD): **github.com/google/bbr/blob/v2alpha/README.md**
- As noted at IETF 105, BBR v2 was released in July 2019 for QUIC:
  - Chromium QUIC (BSD):on chromium.org in bbr2_sender.{ cc, h }

- BBR v2 release is ready for research experiments
  - We invite researchers to share...
    - Ideas for test cases and metrics to evaluate
    - Test results
    - Algorithm/code ideas
  - Always happy to see patches or look at packet traces...
- BBR v2 algorithm was described at IETF 104 [slides | video]
- BBR v2 open source release was described at IETF 105 [slides | video]

BBR v2 deployment status at Google

- YouTube: deployed for a small percentage of users
    - Reduced queuing delays: RTTs lower than BBR v1 and CUBIC
    - Reduced packet loss: loss rates closer to CUBIC than BBR v1
- Internal: continuing test pilot program between and within some Google data-centers
    - BBRv2 being deployed as default TCP congestion control for internal Google traffic
- Continuing to iterate using production experiments and lab tests

Conclusion

- Actively working on BBR v2 at Google
  - Tuning performance to enable full-scale roll-out at Google
  - Improving the algorithm to scale to larger numbers of flows
  - We invite the community share test results, issues, patches, or ideas
- Work under way for BBR in FreeBSD TCP @ Netflix as well
- Food for thought:
  - What should KPIs (key performance indicators) for congestion control look like?
    - How to tell if CC is "doing well" in a production environment...
      - ...given dynamic traffic, routing, and topologies?

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, C. Stephen Gunn, Kevin Yang, Jana Iyengar, Pawel Jurczyk, Biren Roy, David Wetherall, Amin Vahdat, Leonidas Kontothanassis, and {YouTube, google.com, SRE, BWE} teams.

Backup slides...

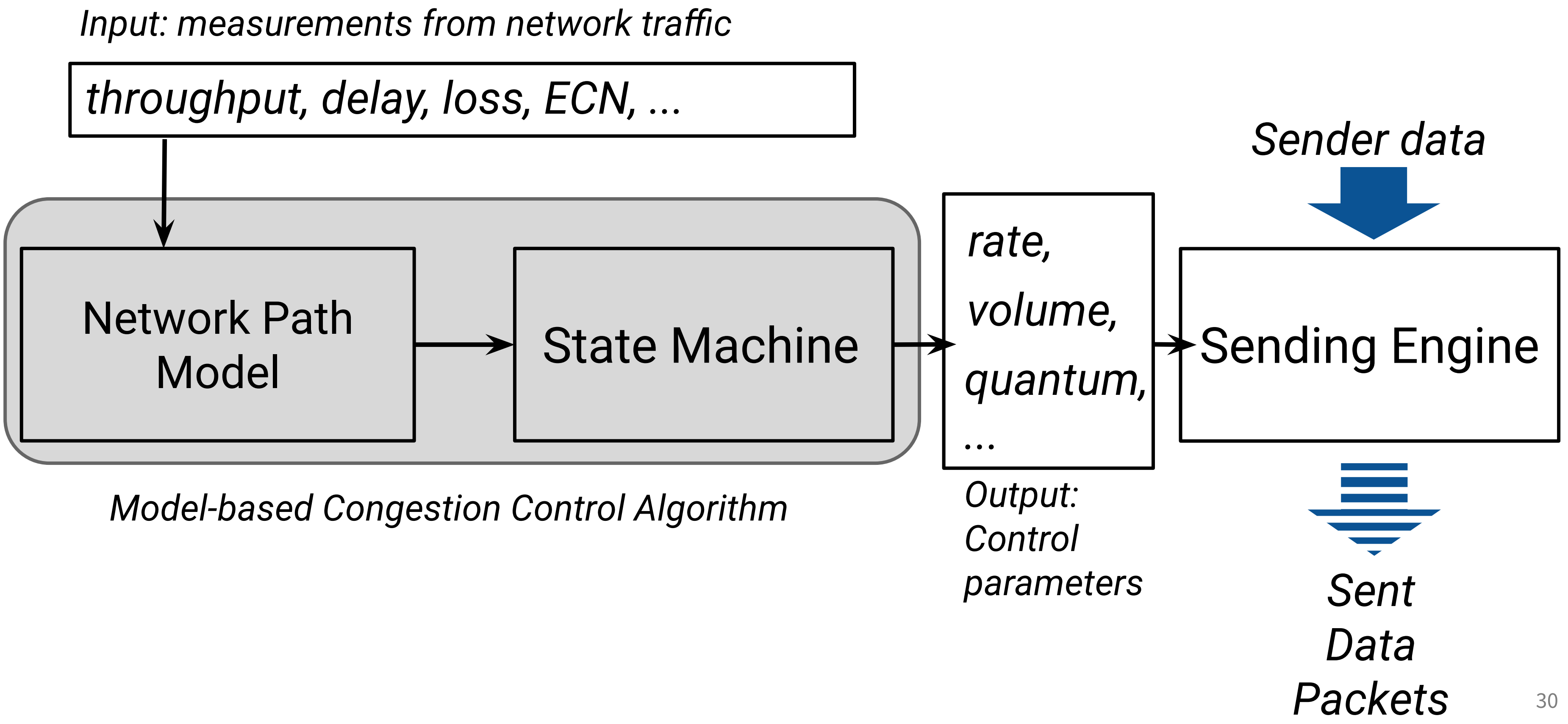
BBR v1 status: deployment, release, documentation

- BBR v1 used for TCP/QUIC on Google.com/YouTube, Google WAN backbone
  - Better performance than CUBIC for web, video, RPC traffic
- BBR v1 code open source in Linux TCP (dual GPLv2/BSD), Chromium QUIC (BSD)
- BBR v2 **preview** code available: Linux TCP (dual GPLv2/BSD), Chromium QUIC (BSD)
- Active BBR work under way for BBR in FreeBSD TCP @ Netflix
- BBR v1 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: 97 | 98 | 99 | 100 | 101 | 102 | 104 (v2 design overview) | 105
- BBR v1 Overview in Feb 2017 CACM

What's new in BBR v2: a summary

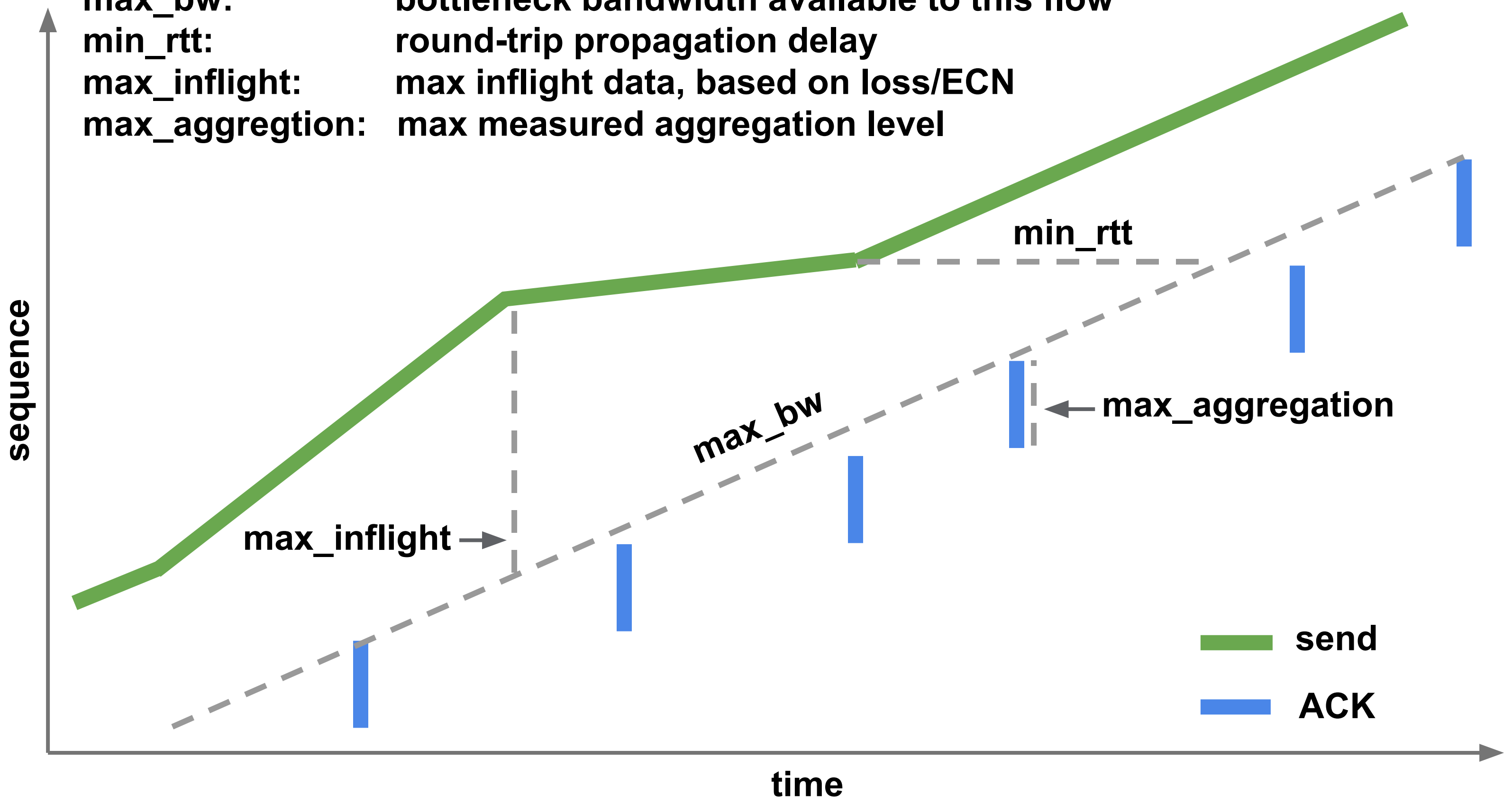
	CUBIC	BBR v1	BBR v2
Model parameters to the state machine	N/A	Throughput, RTT	Throughput, RTT, max aggregation, max inflight
Loss	Reduce cwnd by 30% on window with any loss	N/A	Explicit loss rate target
ECN	RFC3168 (Classic ECN)	N/A	DCTCP-inspired ECN
Startup	Slow-start until RTT rises (Hystart) or any loss	Slow-start until tput plateaus	Slow-start until tput plateaus or ECN/loss rate > target

BBR congestion control: the big picture



BBR v2: the network path model

max_bw: bottleneck bandwidth available to this flow
min_rtt: round-trip propagation delay
max_inflight: max inflight data, based on loss/ECN
max_aggregtion: max measured aggregation level



BBR v2: what's new?

- Properties maintained between BBR v1 and BBR v2:
  - High throughput with a targeted level of random packet loss
  - Bounded queuing delay, despite bloated buffers
- Improvements from BBR v1 to BBR v2 (as discussed at IETF 104 [slides | video]):
  - Improved coexistence when sharing bottleneck with Reno/CUBIC
  - Much lower loss rates for cases where bottleneck queue < 1.5*BDP
  - High throughput for paths with high degrees of aggregation (e.g. wifi)
  - Using DCTCP/L4S-style ECN signals
  - Vastly reduced the throughput reduction in PROBE_RTT
- Following are a few tests, to illustrate the core properties maintained and improved...
  - Metrics we're evaluating in these:
    - throughput, queuing latency, retransmit rate, fairness