BBR Congestion Control

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

Google

IETF 97: Seoul, Nov 2016
Congestion and bottlenecks
Congestion and bottlenecks

![Diagram showing sender and receiver with delivery rate, BDP, and BDP + BufSize on a graph.](Image)
Optimal: max BW and min RTT (Gail & Kleinrock. 1981)
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
But to see both max BW and min RTT, must probe on both sides of BDP...

- Only min RTT is visible
- Only max BW is visible
One way to stay near \((\text{max BW, min RTT})\) point:

**Model** network, update max BW and min RTT estimates on each ACK

**Control** sending based on the model, to...

- Probe both max BW and min RTT, to feed the model samples
- **Pace** near estimated BW, to reduce queues and loss [move queue to sender]
- Vary pacing rate to keep inflight near BDP (for full pipe but small queue)

That's **BBR** congestion control:

- **BBR** = Bottleneck Bandwidth and Round-trip propagation time

**BBR** seeks high tput with small queue by probing BW and RTT *sequentially*
BBR: model-based walk toward max BW, min RTT

optimal operating point
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.
Performance results
BBR and CUBIC: Start-up behavior

CUBIC (red)
BBR (green)
ACKs (blue)
BBR multi-flow convergence dynamics

Converge by sync'd PROBE_RTT + randomized cycling phases in PROBE_BW

- Queue (RTT) reduction is observed by every (active) flow
- Elephants yield more (multiplicative decrease) to let mice grow: each flow learns its fair share

bw = 100 Mbit/sec
path RTT = 10ms
Fully use bandwidth, despite high loss

BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms
Low queue delay, despite bloated buffers

BBR vs CUBIC: synthetic bulk TCP test with 8 flows, bottleneck_bw=128kbps, RTT=40ms
BBR is 2-20x faster on Google WAN

- BBR used for all TCP on Google B4
- Graph: bw for B4 active probers
- Most BBR flows so far rwin-limited
  - max RWIN here was 8MB
  - 10 Gbps x 100ms = 125MB BDP
- after lifting rwin limit:
  - BBR 133x faster than CUBIC
Deep dives & implementation
Top priority: reducing queue usage

- Current active work for BBR
- Motivation:
  - Further reduce delay and packet loss
  - Better fairness w/ loss-based CC in shallow buffers
  - Better fairness w/ higher-RTT BBR flows
  - Lower tail latency for cross-traffic
- Mechanisms:
  - Draining queue more often
    - Drain inflight down to BDP each gain cycle
  - Estimate available buffer; modulate probing magnitude/frequency
    - In shallow buffers, BBR bw probing makes loss-based CC back off
Sharing deep buffers with loss-based CC

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
Current dynamics w/ with loss-based CC

CUBIC vs BBR goodput: bw = 10Mbps, RTT = 40ms, 4 min. bulk xfer, varying buffer sizes
BBR multi-flow behavior: RTT fairness

Compare the goodput of two competing BBR flows with short (A) and long (B) min_RTT

BBR flows w/ higher RTT have an advantage; but BBR flow with 64x higher min_RTT only has <4x higher bw
bw = 10 Mbit/sec, buffer = 1000 packets
Common real-world issues

- **ACK compression**
  - One TCP ACK for up to +200 packets
  - Particularly wireless & cable networks
  - BBR strategy: cap inflight <= 2*BDP

- **Application idles**
  - Paces at BW restarting from idle

- **Inappropriate receive window**
  - Linux default 3MB => 240Mbps on 100ms RTT

- **Token-bucket traffic policers**
  - Explicitly model policers
  - Details presented in maprg
Implementation and deployment status

- **Linux v4.9 TCP**
  - A congestion control module with dual GPL/BSD licence
  - Requires fq/pacing qdisc (BBR needs pacing support)
  - Fully deployed for WAN between Google datacenters
  - Being deployed on Google.com and YouTube

- **QUIC implementation under way**
  - Production experiments have started
  - {vasilvv,ianswett,jri}@google.com

- **FreeBSD implementation under way**
  - rrs@netflix.com
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is BBR fair to Cubic/Reno?</td>
<td>Buffer &gt;= 1.5*BDP: Yes; Else: WIP</td>
</tr>
<tr>
<td>Is BBR 1/sqrt(p)?</td>
<td>No</td>
</tr>
<tr>
<td>Is BBR {delay</td>
<td>loss</td>
</tr>
<tr>
<td>Is BBR ack-clocked?</td>
<td>No</td>
</tr>
<tr>
<td>Does BBR require pacing?</td>
<td>Yes</td>
</tr>
<tr>
<td>Does BBR require an FQ scheduler?</td>
<td>No, but it helps</td>
</tr>
<tr>
<td>Does BBR require receiver or network changes</td>
<td>No</td>
</tr>
<tr>
<td>Does BBR improve latency on short flows?</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Conclusion

BBR: model-based congestion control

- Goal is to maximize bandwidth then minimize queue
- Orders of magnitude higher bandwidth and lower latency

BBR will continue to evolve as we gain more experience

- Help us make it better! https://groups.google.com/forum/#!forum/bbr-dev
  Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson
Backup slides...
BBR: control logic details

Controls sending based on the model, to move toward network's best operating point:

- send rate near available bandwidth (primary):
  - Pacing rate = pacing_gain * BtlBw

- volume of data in flight near BDP (secondary):
  - Max inflight = cwnd_gain * BDP = cwnd_gain * BtlBw * RTprop
BBR state transition diagram
BBR: state machine details

**STARTUP**: exponential growth to quickly fill pipe (like slow-start)
- stop growth when bw estimate plateaus, not on loss or delay (Hystart)
- pacing\_gain = 2.89, cwnd\_gain = 2.89

**DRAIN**: drain the queue created in **STARTUP**
- pacing\_gain = 0.35 = 1/2.89, cwnd\_gain = 2.89

**PROBE\_BW**: cycle pacing\_gain to explore and fairly share bandwidth (cwnd\_gain = 2 in all phases):
- [1.25, 0.75, 1, 1, 1, 1, 1, 1] (1 phase per min RTT)
- Pacing\_gain = 1.25 => probe for more bw
- Pacing\_gain = 0.75 => drain queue and yield bw to other flows
- Pacing\_gain = 1.0 => cruise with full utilization and low, bounded queue

**PROBE\_RTT**: if needed, occasionally send slower to probe min RTT
- Maintain inflight of 4 for at least max(1 round trip, 0.2 sec); pacing\_gain = 1.0
BBR: life of a typical bulk flow

bw = 100 Mbit/sec
rtt = 100ms
buffer = 10 BDP
Comparing RTT fairness for BBR and CUBIC

Compare the goodput of two competing BBR or CUBIC flows with short (A) and long (B) min_RTT

bw = 10 Mbit/sec, buffer = 1000 packets

Flow A (min_RTT=10ms, start t = 0 sec)
Flow B (varying min_RTTs, start t = 2 sec)

BBR flows w/ higher RTT have an advantage; flow with 64x higher min_RTT has <4x higher bw

CUBIC flows w/ lower RTT have an advantage; flow with 64x higher min_RTT has 4.6x higher bw
How BBR Fits into Transport Stacks

ACK processing, loss detection - What to send
Congestion control - How fast to send
Smart packet scheduler - When to send

Smart packet scheduler

TSO
(How large a burst)

TSQ
(How much to enqueue)

Pacing
(When to send)

Fair queuing
(How to share and mix)

NIC
TCP

TCP Small Queues (TSQ)

TSO autosizing

Pacing

Fair queuing

NIC

link
Fully use bandwidth, despite high loss

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