Modeling BBR's Interactions with Loss-Based Congestion Control

Ranysha Ware
rware@cs.cmu.edu
Carnegie Mellon University

Matthew K. Mukerjee
mukerjee@nefeli.io
Nefeli Networks

Srinivasan Seshan
srini@cs.cmu.edu
Carnegie Mellon University

Justine Sherry
sherry@cs.cmu.edu
Carnegie Mellon University

ABSTRACT

BBR is a new congestion control algorithm (CCA) deployed for Chromium QUIC and the Linux kernel. As the default CCA for YouTube (which commands 11+% of Internet traffic), BBR has rapidly become a major player in Internet congestion control. BBR's fairness or friendliness to other connections has recently come under scrutiny as measurements from multiple research groups have shown undesirable outcomes when BBR competes with traditional CCAs. One such outcome is a fixed, 40% proportion of link capacity consumed by a single BBR flow when competing with as many as 16 loss-based algorithms like Cubic or Reno. In this short paper, we provide the first model capturing BBR's behavior in competition with loss-based CCAs. Our model is coupled with practical experiments to validate its implications. The key lesson is this: under competition, BBR becomes window-limited by its 'in-flight cap' which then determines BBR's bandwidth consumption. By modeling the value of BBR's in-flight cap under varying network conditions, we can predict BBR's throughput when competing against Cubic flows with a median error of 5%, and against Reno with a median of 8%.

1 INTRODUCTION

In 2016, Google published a new algorithm for congestion control called BBR [4, 5]. Now deployed as the default congestion control algorithm (CCA) for Google services including YouTube, which commands 11% [13] of US Internet traffic, BBR consequently impacts a large fraction of Internet connections today. In this short paper, we focus on BBR's behavior – 'fairness' or 'friendliness' – when competing with legacy, loss-based CCAs such as Reno or Cubic.

We are not the first to investigate BBR's properties when competing with traditional loss-based CCAs. Experimental studies have noticed two key phenomena. First, in shallow-buffered networks, BBR's bandwidth probing phase causes buffer overflows and bursty loss for competing flows; these bursts can lead to Cubic and Reno nearly starving for bandwidth. This phenomena was first explored in [11] and BBRv2 is expected to patch the problem [7].[1]

In residential capacity links (*e.g.* 10-100Mbps) with deep buffers, studies [4, 9, 14, 16, 17] have generated conflicting reports on how BBR shares bandwidth with competing Cubic and Reno flows. We [17] and others [9, 14] observed a single BBR flow consuming a fixed 35-40% of link capacity when competing with as many as 16 Cubic flows. These findings contradict the implication of early presentations on BBR [4] which illustrated scenarios where BBR was *generous* to competing Cubic flows. In short, the state of affairs is confusing, with no clear indication as to *why* any of the empirically observed behaviors might emerge.

The contribution of this paper is to model BBR's behavior when it competes with traditional, loss-based congestion control algorithms in residential, deep-buffered networks (studies [12] suggest that residential routers typically have buffer depths 10-30× a bandwidth-delay product for a 100ms RTT). The key insight behind our model is that, while BBR is a rate-based algorithm when running alone, BBR degrades to window-based transmission when it competes with other flows. BBR's window is set to a maximum 'in-flight cap' which BBR computes as $2 \times RTT_{est} \times Btlbw_{est}$, for RTT_{est} and $Btlbw_{est}$, BBR's estimates of the baseline RTT and its share of bandwidth.

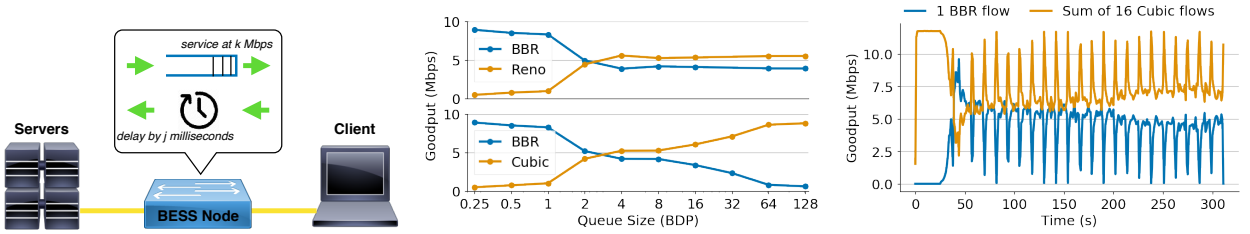
While the original BBR publication presented the in-flight cap as merely a safety mechanism – included to allow BBR to handle delayed ACKs [5] – this mechanism, unexpectedly, is the key factor controlling BBR's share of link capacity under competition. Our model focuses on how BBR estimates its in-flight cap under different network conditions; by computing what we expect BBR's in-flight cap to be, we can predict BBR's share of link capacity for long-lived flows. The size of the in-flight cap is influenced by several parameters: the link capacity and latency, the size of the bottleneck queue, and the number of concurrent BBR flows. But, notably absent, the number of competing loss-based (Cubic or Reno) flows *does not play a factor* in computing this in-flight cap. Hence, BBR's sending rate is not influenced by the number of competing traditional flows; this is the reason behind reports that BBR is 'unfair' to Cubic and Reno in multi-flow settings [9, 17].

In what follows, we discuss our testbed in §2 and early measurements of BBR's 'fairness' or 'friendliness' in §3. We then provide a primer on the BBR algorithm in §4. We then develop our analysis of BBR in §5 along with an explanation of BBR's convergence to 40% of link capacity. We connect our results to related work in §6 and and conclude in §7.

[1] BBRv2 has very recently been released; in this paper we focus on BBRv1 which was the only version available at the time of this study.

IMC '19, October 21–23, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6948-0/19/10…$15.00
https://doi.org/10.1145/3355369.3355604

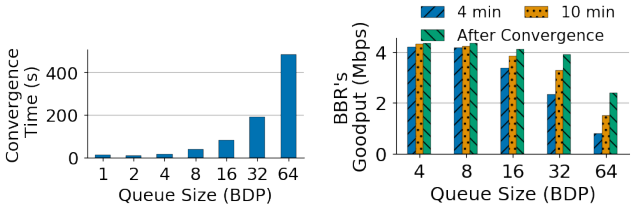


(a) Testbed for congestion experiments which introduces queueing at a controlled bottleneck.

(b) Average goodput for two competing flows over 4 min in a 40ms×10Mbps network with varying queue sizes.

(c) BBR's goodput over time competing with 16 Cubic flows in a 40ms×10Mbps network with a 32 BDP queue.

Figure 1: Testbed and initial measurements of BBR's empirical behaviors.



(a) Convergence time for 1 BBR flow and 1 Cubic flow over varying queue sizes

(b) Goodput for 1 BBR flow and 1 Cubic flow over varying measurement intervals.

Figure 2: BBR vs Cubic in a 40ms × 10Mbps network

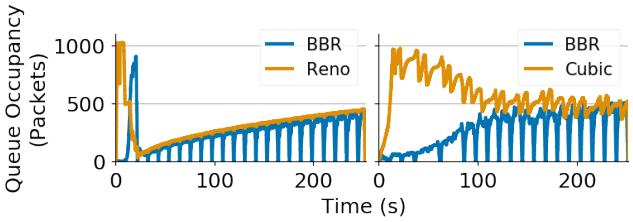


Figure 3: BBR and Cubic or Reno's queue when competing for 4 minutes over a network with a 64 BDP (1024 packet) queue.

2 TESTBED

Throughout this paper, we show experiments generated in the testbed illustrated in Fig. 1a. Each experiment involves three servers: a server/sender, a BESS [10] software switch, and a client/receiver. All servers are running Linux 4.13 (using internal TCP pacing), have Intel E5-2660V3 processors, and have dual-port Intel X520 10Gb NICs. Senders and receivers use iPerf 3 [1] to generate/receive traffic. Within BESS, traffic is serviced at a configurable rate below the link capacity to introduce queueing. The queue size is set to ratios relative to BDP; since the BESS queue module only supports powers-of-two sizes we rounded to the nearest power-of-two. To configure delay, we hold all ACKs for a configurable amount of time. Unless otherwise noted, we set bandwidth to 10 Mbps and RTT to 40ms, following Google's parameters in IETF presentations [4, 6].

3 BBR IN COMPETITION

A natural concern when deploying a new CCA on the Internet is how the new CCA will interact with other deployed algorithms. Will the new CCA be 'fair' to existing CCAs, or starve them?

An early BBR presentation [4] provided a glimpse into these questions. A graph in the presentation measures 1 BBR flow vs. 1 Cubic flow over 4 minutes, and illustrates a correlation between the size of the bottleneck queue and BBR's bandwidth consumption. We set out to replicate Google's experiments and easily did so – shown in Fig. 1b – as did other studies [14]. The implication of these graphs is that BBR is generous to existing CCAs in typical buffer bloated networks, especially to Cubic.

Subsequent studies in our group and others questioned both the results – what fraction of the link BBR consumed – as well as the implication of generosity [9, 14, 17]. Some data [17] showed that BBR converged to different rates – around 40% of the link capacity for queue sizes up to 32×BDP, matching the Reno graph, but not matching the Cubic graph. We show in Figs. 3 and 2 that this incongruity is merely the result of differing experimental conditions and the amount of time it takes for BBR to converge to its steady-state share of link capacity. Where BBR quickly matches Reno's queue occupancy – and therefore consumption of the link capacity – BBR takes longer to scale up when competing with Cubic (Fig. 3). As a consequence, the 'average goodput' one computes is dependent on how long one measures the competition between BBR and Cubic (Fig. 2b). Furthermore, to reach convergence can take on the order of minutes in very deep buffered networks (Fig. 2b).

Another set of experiments [9, 17] suggest that BBR may consume far more than its 'fair' share of link capacity. Fig. 1c shows goodput over time of BBR vs 16 Cubic flows in the same 40ms × 10Mbps scenario. BBR consumes an outsized share of bandwidth, leaving just over half to be shared by the sixteen other connections.

Unfortunately, relying only on these empirical studies leave us like the blind men and the elephant, each relying on only pieces of the overall picture to understand BBR's characteristics. To get to the bottom of *why* BBR behaves in the way it does, and to *predict* how BBR might behave in unobserved scenarios, we turn to modeling in the rest of this paper.

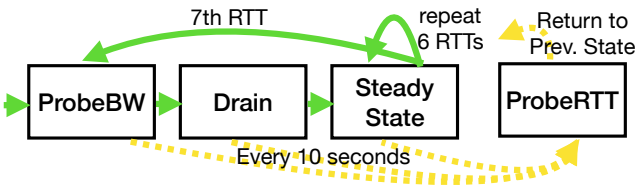


Figure 4: BBR's steady-state operation.

4 BBR PRIMER

BBR is designed to be a rate-based algorithm. BBR maintains two key variables: $Btlbw_{est}$ BBR's estimate of the available throughput for it to transmit over the network, and RTT_{est} BBR's estimate of the baseline round-trip time. BBR paces packets at $Btlbw_{est}$ rate. Assuming that BBR is transmitting over a single link with no queueing (and a sender which ACKs instantaneously), BBR should expect to never have more than $Btlbw_{est} \times RTT_{est}$ unacked packets outstanding.

As a failsafe and to keep the pipe full in networks that delay or aggregate ACKs, BBR implementations impose a 'in-flight cap' – it will never allow itself to have *more* than $2 \times Btlbw_{est} \times RTT_{est}$ unacknowledged packets outstanding [5, 6]. As we will show, this cap turns out to be the central parameter controlling BBR's link utilization in competition with Cubic and Reno.

To estimate $Btlbw_{est}$ and RTT_{est}, BBR cycles (post-startup) through a simple state machine illustrated in Fig. 4.[2]

Estimating the rate. BBR sends at a fixed rate BW_{est}. BBR sets its initial rate using its own version of 'slow start'; henceforth BBR 'probes for bandwidth' (ProbeBW in Fig. 4) one out of every 8 RTTs. During this stage, BBR inflates the rate to $1.25 * Btlbw_{est}$ and observes the achieved throughput during that interval. BBR then lowers its rate to $(0.75 * Btlbw_{est})$ to drain any excess packets out of queues. BBR's $Btlbw_{est}$ is then the max observed packet delivery rate over the last 8 RTTs. It then sends at the newly-recalculated $Btlbw_{est}$ for the next 6 RTTs before probing again.

Estimating the RTT. BBR also keeps track of the smallest observed RTT. If BBR goes 10 seconds without observing a smaller RTT, it enters ProbeRTT. During ProbeRTT, BBR caps the amount of data it has in-flight to only 4 packets and measures the RTT for those packets for at least 200ms and one packet-timed round-trip. [3] BBR drops its sending rate to try to ensure none of its own packets are occupying queues in the network: in Fig. 1c one can observe BBR dropping its rate to almost zero on ten-second intervals. After ProbeRTT, BBR returns to the state it was in previously.

5 ANALYSIS AND MODELING

We model BBR's post-convergence share of link capacity when competing with loss-based CCAs in three phases.

[2]Our state machine figure differs from the 'standard' BBR figure [3] by focusing on only steady-state operation rather than startup, and separating apart the three sub-phases of ProbeBW.

[3]A "packet-timed round-trip" means that a data packet is sent and then the sender waits for that packet or some late packet to be acknowledged

(1) Simple Model of In-flight Cap: We first model a simple scenario to understand how BBR's in-flight cap controls BBR's sending rate. In this scenario, the queue is highly bloated, baseline RTTs are negligible, and there are only two flows (one BBR, one loss-based) competing.

(2) Extended Model of In-flight Cap: After demonstrating that BBR's in-flight cap controls its sending rate, we develop a more robust model, covering scenarios with multiple BBR flows, finite queue capacities, and non-negligible RTTs.

(3) Model of Probing Time: BBR's in-flight cap is only 4 packets during ProbeRTT, hence BBR spends time without transmitting data every ten seconds. To predict BBR's sending rate overall, we must reduce the rate predicted by the in-flight cap proportionally to the amount of time BBR spends in ProbeRTT.

5.1 Assumptions and Parameters

Table 1 lists the parameters in our model. We use these parameters to compute p, Cubic/Reno's share of the link capacity at convergence, and $1 - p$, BBR's share of link capacity at convergence. Our model is based on the following assumptions:

(1) Flows have infinite data to send; their sending rates are determined by their CCA, which is either BBR, Cubic, or Reno.

(2) All flows experience the same congestion-free RTT and the available link capacity is fixed.

(3) All packets are a fixed size.

(4) The network is congested and the queue is typically full; a flow's share of throughput hence equals its share of the queue.

(5) All loss-based CCA's are synchronized [15]. All BBR flows are synchronized [5]. All flows begin at the same time.

5.2 Simple Model: BBR's ProbeBW State

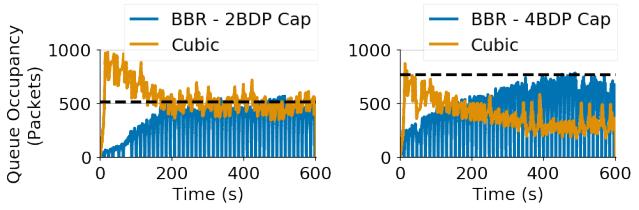
The first insight of our model is that BBR is controlled by its in-flight cap: in BBR's ProbeBW phase, BBR aggressively pushes out loss-based competitors until it reaches its in-flight cap.

Model: Why this happens follows from the BBR algorithm and loss-based CCAs' reaction to packet losses. Assume a link capacity c, where BBR and the loss-based CCAs, in aggregate, are consuming all of the available capacity. By probing for 125% of its current share of bandwidth, BBR pushes extra data into the network (offered load $> c$) leading to loss for all senders. Loss-based algorithms back off, dropping their window sizes and corresponding sending rate. BBR does not react to losses and instead *increases* its sending rate, since it successfully sent more data during bandwidth probing than it did in prior cycles. The loss-based CCA returns to ramping up its sending rate, and together the combined throughput of the two becomes slightly *higher* than the link capacity and the two flows begin to fill the bottleneck buffer. This process continues until BBR hits an in-flight cap; we expect that in the absence of a cap it would consume the entire link capacity.

Validation: We modified BBR in our testbed to run with a in-flight cap of $4 \times BDP$. In Fig. 5b we show one run with our elevated in-flight cap along with a run with the standard cap in a testbed in a 40

Parameter	Description
N	Number of BBR flows sharing bottleneck
q	Bottleneck queue capacity (packets)
c	Bottleneck link capacity (packets per second)
l	RTT when there is no congestion (seconds)
X	Queue capacity as multiple of BDP: $q = Xcl$
d	Flow completion times after convergence (seconds)

Table 1: Description of model parameters



(a) 2 BDP in-flight cap **(b) 4 BDP in-flight cap**

Figure 5: BBR vs Cubic in a 10Mbps×40ms testbed with a 32 BDP queue. Black dashed line is the model (5).

ms × 10 Mbps network with a 32 BDP packet queue. BBR increases its share of the link capacity; we show in the next subsection that this increased share matches our prediction of a window-limited sender with a window the size of the in-flight cap.

5.3 Simple Model: BBR's In-flight Cap

To understand the impact of the in-flight cap on BBR's performance, we build a model making two simplifying assumptions (we relax these assumptions later): (1) There is only 1 BBR flow competing with any number loss-based CCAs, and (2) The queue capacity is much greater than the BDP ($q \gg cl$).

Model: Recall that the in-flight cap is calculated as:

$$\text{inflight}_{cap} = 2 \times RTT_{est} \times Btlbw_{est} \quad (1)$$

With a queue capacity of q we can assume that, at any given point of competition p from loss-based flows, BBR will consume the remaining bandwidth:

$$Btlbw_{est} = (1-p)c. \quad (2)$$

About every 10 seconds, BBR enters ProbeRTT to measure the baseline RTT, draining any packets BBR has in the queue.

When there is no competing traffic, 1 BBR flow can successfully measure the baseline RTT l during ProbeRTT. When there is competing traffic from loss-based CCAs, there will be $p \times q$ data in the queue. Assuming a negligible baseline RTT ($q \gg cl$) — as bufferbloat increases, queuing delay becomes the dominant factor in latency — we have:

$$RTT_{est} = \frac{pq}{c}. \quad (3)$$

Plugging (2) and (3) into (1) and reducing gives:

$$\text{inflight}_{cap} = 2(p - p^2)q. \quad (4)$$

We know from the previous subsection that BBR will increase its rate until it is limited by the in-flight cap. To compute this, we set inflight_{cap} equal to the amount of data BBR has in-flight and solve for p:

$$2 \times (p - p^2)q = (1-p)q$$
$$p = \frac{1}{2} \quad (5)$$

We can now see that while 1 BBR flow increases its sending rate during ProbeBW, once it intersects the in-flight cap it will not be able to consume more than 50% of the available capacity.

Validation: This simple model for the in-flight cap in a deep-buffered network says if the BDP cap is 2, then BBR should occupy about half the queue after convergence. Similarly, if the BDP cap is 4, then BBR should occupy at most 75% of the queue after convergence. Fig. 5 shows BBR converging at each of these points in a deep-buffered network with a 32 BDP queue.

Note: This simple model demonstrates why BBR retains the same share of link capacity regardless of the number of competing Cubic or Reno flows. ProbeBW is aggressive enough to force one or many loss-based flows to back off; the bandwidth cap is set simply by the queue occupancy of the competing loss-based flows – but not *how many* loss-based flows there are. The calculations behind ProbeBW and the in-flight cap lack any signal to infer number of competing loss-based flows and adapt to achieve equal shares/fairness.

5.4 Extended Model: In-flight Cap

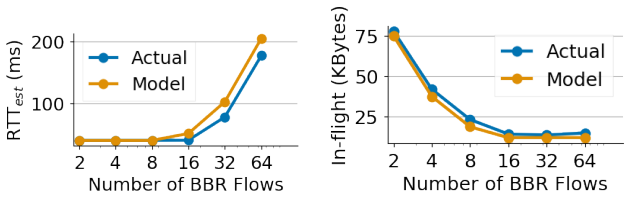
Our simple model assumes a buffer-bloated network and only one BBR flow. In this section, we show how BBR's in-flight cap changes given the size of the queue (bloated or not) and with an increasing number of BBR flows.

Multiple BBR Flows Alone: To understand multiple BBR flows competing with loss-based flows, we first need to understand multiple BBR flows competing in the *absence of other traffic*. After convergence, each BBR flow has a *slightly* overestimated $Btlbw_{est}$ near their fair share: $\frac{1}{N} \times c + \epsilon$. The additional ϵ is – similar to our discussion in §5.2 – due to the aggression of ProbeBW. Here, BBR flows compete against each other; BBR uses a max() operation to compute $BtlBw_{\text{est}}$ over multiple samples of sending rates resulting in, usually, a slight *overestimate* of its fair share. While we ignore this ϵ in our modeling, its existence forces the aggregate of BBR flows to send at a rate slightly higher than c, filling queues until each flow reaches its bandwidth cap and becomes window-limited and subsequently ACK-clocked.

However, the cap may also be elevated due to the presence of multiple competing flows. During ProbeRTT, each flow will limit inflight to 4 packets, so that they can drain all of their packets from the queue and measure the baseline RTT. For N BBR flows, this means in aggregate they will have $4N$ packets inflight. However, if $4N$ packets is greater than the BDP, the queue will not drain during ProbeRTT so RTT_{est} includes some queueing delay:

$$RTT_{est} = \max(l, \frac{4N - cl}{c} + l) \quad (6)$$

Thus, the the in-flight cap when N BBR flows compete is dependent on the BDP. Further, if the queue is smaller than $4N - cl$ when $4N > cl$, then the BBR flows will consume the entire queue and hence 100% link capacity.



(a) RTT$_{est}$. **(b) Inflight$_{cap}$.**

Figure 6: Comparisons between model and observation for RTT$_{est}$ and in-flight$_{cap}$ at 40ms × 15Mbps and 64 BDP queue.

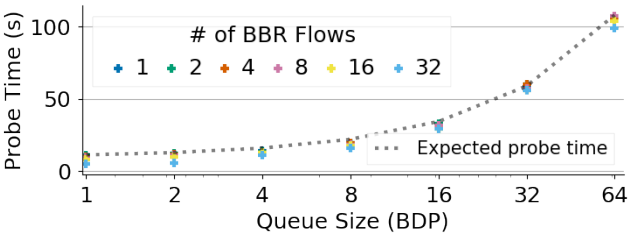


Figure 7: Probe$_{time}$ model for 40ms × 10 Mbps link vs. measured probe time for BBR flows competing with 1 Cubic flow in varying queue sizes.

Validation: Fig. 6a shows the measured median RTT estimate across a varying number of BBR flows versus (6). The estimate increases linearly, similar to our prediction. Here, the BDP is only 75KB, so the queue will not completely drain during ProbeRTT when there are 13 or more BBR flows. Fig. 6b shows how this corresponds to the inflight cap. If the BDP were larger, the flows would have been able to measure the correct $RTTest$.

Multiple BBR Flows vs Loss-Based Flows: We now return to multiple BBR flows vs loss-based flows. As we saw when BBR flows were only competing with each other, if the BDP is not large enough to accommodate $4N$ packets during ProbeRTT, BBR's RTT estimate will be too large. If we assume $4N$ additional packets are in the queue during ProbeRTT, then,

$$RTT_{est} = \frac{pq + 4N}{c} + l. \tag{7}$$

Here, we also include l, no longer assuming it is negligible compared to queueing delay. Plugging (7) and (2) into (1), in aggregate all N BBR flows will have:

$$\text{inflight}_{cap} = 2(1-p)c\left(\frac{pq + 4N}{c} + l\right). \tag{8}$$

To compute the BBR flows' aggregate fraction of the link, we set inflight$_{cap}$ equal to the amount of data BBR flows have in-flight and solve for p:

$$\begin{aligned} 2(1-p)c\left(\frac{pq + 4N}{c} + l\right) &= (1-p)q + (1-p)cl \\ p &= \frac{1}{2} - \frac{1}{2X} - \frac{4N}{q} \end{aligned} \tag{9}$$

If p were a negative number, this would mean BBR's inflight cap exceeded the total capacity (BDP + the queue size) and hence BBR's share of the link would be 100%.

In the next section, we complete our extended model by computing the amount of time BBR operates at its in-flight cap.

5.5 Extended Model: ProbeRTT Duration

During ProbeRTT, BBR stops sending data while it waits for its in-flight data to fall to 4 packets. You can see this behavior impacting goodput in Fig. 1c. If the queue is large and also full when BBR goes into ProbeRTT, this results in long intervals where BBR is not sending any data. [4] This results in BBR on average consuming a lower fraction of link capacity than if it were sending constantly at a rate proportional to its inflight cap.

Model: If the total duration of time the flows are competing (after convergence) is d, the fraction of the link BBR flows will use when competing with loss-based CCAs is:

$$BBR_{frac} = (1-p) \times \left(\frac{d - \text{Probe}_{time}}{d}\right), \tag{10}$$

where p is computed using (9). During Probe$_{time}$ throughput is nearly zero.

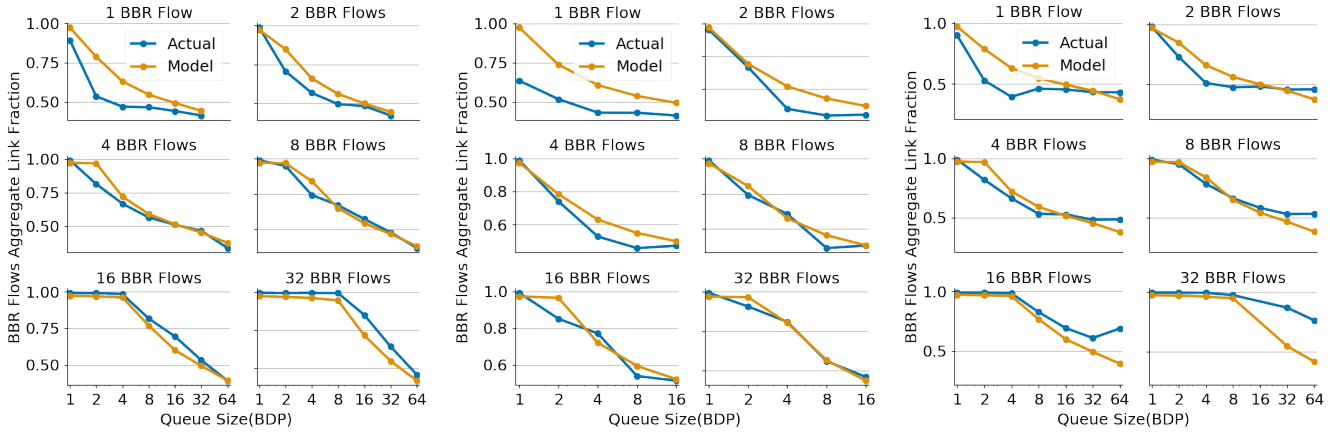
We compute Probe$_{time}$ by computing the length of time spent in ProbeRTT state, and multiply by how many times BBR will go into ProbeRTT state. Assuming the queue is full before BBR enters ProbeRTT state, BBR will have to wait for the queue to drain before its data in-flight falls to 4 packets. Once it reaches this in-flight cap, BBR also waits an additional 200ms and a packet-timed round trip before exiting ProbeRTT. Assuming synchronized flows and the queue is typically full, BBR flows should rarely measure a smaller RTT outside of ProbeRTT state so it should enter ProbeRTT about every 10 seconds. Altogether, this means probe time increases linearly with queue size:

$$\text{Probe}_{time} = \left(\frac{q}{c} + .2 + l\right) \times \frac{d}{10} \tag{11}$$

Validating Probe$_{rtt}$: First, we measure the probe time from experiments with competing BBR flows in for a 40ms×15 Mbps network for experiments run for 400 seconds after convergence (d=400) for Cubic We compare this to our prediction computed from (11). Fig. 7 compares (11) to measured probe time—the model fits the observations well. Most commonly the predicted probe time for experiments with Cubic is 1-3 seconds larger than the expectation and is at most about 8 seconds too large.

Validating the Extended Model: We measure the average throughput for BBR competing against Cubic or Reno after convergence (d = 400 for Cubic, d = 200 for Reno). We use (10) to compute BBR's expected fraction of the link versus our measurements. Our expectations closely follow empirical results in most cases, validating our model. Fig. 8 compares (10) to the BBR flows aggregate fraction of the link when competing with Reno or Cubic. The median error competing against Cubic 5%, and against Reno 8%.

[4] In fact, BBR authors have even noted that this is a significant limitation on BBR's performance, and in BBRv2 design change ProbeRTT so that it reduces BBR's inflight cap to 50% of it's BDP$_{est}$ instead of 4 packets [7].



(a) 40ms × 10 Mbps, vs 1 Cubic Flow **(b) 30ms × 50 Mbps, vs 1 Cubic Flow** **(c) 40ms × 10 Mbps, vs 1 Reno Flow**

Figure 8: Model compared to observed aggregate fraction of the link.

For Cubic, the model fits the observations best with large queue sizes and large numbers of flows. In this case, our assumptions that the queue is typically full, and $4N$ BBR packets will be in the queue during ProbeRTT, inflating RTT_{est}, are more likely to be true. However, Reno reveals an opposite trend: the model does *worse* as the queue becomes larger. We suspect this is due to Reno's slower (relative to Cubic) additive increase failing to take advantage of the available capacity and hence leaving a larger share of throughput for BBR.

6 RELATED WORK

The first independent study of BBR was presented by Hock et al. [11]. Their analysis of BBR identifies the important property that multiple BBR flows operate at their in-flight cap in buffer-bloated networks. Further, they present experiments for 1 BBR flow and 1 Cubic flow, noting that in large buffers, they oscillate around equally sharing the bottleneck. They also observe that when 2 BBR flows compete with 2 Cubic flows in a shallow-buffered network, BBR flows will starve the Cubic flows. Several additional empirical studies have reproduced and extended these results [9, 14, 16]. Scholz et al. [14] run tests for up to 10 BBR flows competing with up to 10 Cubic flows in a large buffer and conclude that, "independent of the number of BBR and Cubic flows, BBR flows are always able to claim at least 35% of the total bandwidth." Dong et al. [9] also note that as 1 BBR flow competes with an ever increasing number of Cubic flows, BBR's fraction of the bandwidth remains the same.

Each of these studies touches on important aspects of BBR's behavior, but we are the first to model BBR's behavior in these scenarios rather than to simply observe it. Through our model, we are able to explain the missing parts of seemingly conflicting conclusions drawn in prior work.

Google is actively developing BBRv2 and very recently released a Linux kernel implementation of BBRv2 [2, 7, 8]. Early presentations [8] imply that it primarily resolves the fairness issues discussed by Hock et al [11], but does not touch on the fixed proportion of link capacity as discussed in this paper.

7 CONCLUSION

In this paper, we have shown that BBR's inflight cap – a 'safety mechanism' added to handle delayed and aggregated ACKs – is in reality central to BBR's behavior on the Internet. When BBR flows compete with other traffic (BBR, Cubic, or Reno), BBR becomes window-limited and ACK-clocked, sending packets at a rate entirely determined by its inflight cap.

When competing with loss-based TCPs such as Cubic and Reno, BBR's cap can be computed using the bottleneck buffer size, the number of concurrent BBR flows, and the baseline network RTT. However, the number of competing loss-based flows *are not* a factor in computing this cap. Hence, BBR does not reduce its sending rate even as more loss-based flows arrive on the network. This is the cause of reports arguing that BBR is 'unfair' to legacy TCPs.

8 ACKNOWLEDGEMENTS

We thank the Neal Cardwell, Yucheng Cheng, Soheil Hassas Yeganeh, and Jana Iyengar for the fruitful conversations surrounding BBR and our analysis, as well as our shepherd Srikanth Sundaresan for guiding the revision process. This research was funded by a Facebook Emerging Scholar Fellowship, NSF Grant #1850384, and a Google Faculty Research Award.

REFERENCES

[1] 2018. iperf3. https://software.es.net/iperf/. (2018).
[2] 2019. BBRv2 alpha Linux code. https://github.com/google/bbr/blob/v2alpha. (2019).
[3] N. Cardwell, Y. Chen, S. Hassas Yeganeh, and V. Jacobsen. 2017. BBR Congestion Control. IETF Draft draft-cardwell-iccrg-bbr-congestion-control-00. (2017).
[4] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR congestion control. In *IETF meeting*.
[5] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-based Congestion Control. *Commun. ACM* 60, 2 (Jan. 2017), 58–66. https://doi.org/10.1145/3009824
[6] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR Congestion Control: An update. In *Presentation in ICCRG at IETF 98th meeting*.
[7] N. Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. 2019. BBRv2: A Model-Based Congestion Control. In *Presentation at IETF104*.
[8] N. Cardwell, Yuchung Cheng, Soheil Hassas Yeganehand Priyaranjan Jha, , Yousuk Seung, Ian Swett, Victor Vasiliev, Bin Wu, Matt Mathis, and Van Jacobson. 2019.

BBRv2: A Model-Based Congestion Control IETF 105 Update. In *Presentation at IETF105*.

[9] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 343–356. https://www.usenix.org/conference/nsdi18/presentation/dong

[10] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. *SoftNIC: A Software NIC to Augment Hardware*. Technical Report UCB/EECS-2015-155. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html

[11] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[12] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. 2010. iNetalyzr: Illuminating the Edge Network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, New York, NY, USA, 246–259. https://doi.org/10.1145/1879141.1879173

[13] Rob Marvin. 2018. Netflix and YouTube Make Up Over a Quarter of Global Internet Traffic. *PC Magazine* (15 10 2018).

[14] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a Deeper Understanding of TCP BBR Congestion Control. In *IFIP Networking 2018*. Zurich, Switzerland.

[15] Scott Shenker, Lixia Zhang, and David D Clark. 1990. Some observations on the dynamics of a congestion control algorithm. *ACM SIGCOMM Computer Communication Review* 20, 5 (1990), 30–39.

[16] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. 2019. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv preprint arXiv:1903.03852* (2019).

[17] R. Ware, M. K. Mukerjee, J. Sherry, and S. Seshan. 2018. The Battle for Bandwidth: Fairness and Heterogeneous Congestion Control. In *Poster at NSDI 2018*.