Sharing but not Caring – Performance of TCP BBR and TCP CUBIC at the Network Bottleneck

Saahil Claypool, Mark Claypool
Worcester Polytechnic Institute
Worcester, USA
Email: {smclaypool,claypool}@wpi.edu

Jae Chung
Viasat
Marlborough, USA
Email: jaeWon.chung@viasat.com

Feng Li
Verizon Labs
Waltham, USA
Email: feng.li@verizon.com

Abstract—**Loss-based congestion control protocols, such as TCP CUBIC, can unnecessarily fill router buffers, adding delays that degrade application performance, particularly streaming video. Newcomer Transport Control Protocol (TCP) BBR (Bottleneck Bandwidth and Round-trip propagation time) uses estimates of the bottleneck bandwidth and Round-Trip Time (RTT) to try to operate at the theoretical optimum – just enough data to fully utilize the network without excess queuing. We present detailed experimental results showing that, in practice, BBR can misestimate the bottleneck bandwidth and RTT, causing high packet loss for shallow buffer routers and massive throughput variations when competing with TCP CUBIC flows. We suggest methods for improving BBR's estimation mechanisms to provide more stability and fairness.**

Keywords–TCP; congestion control; Raspberry Pi.

I. Introduction

TCP is the most commonly used Internet protocol and, for video, will account for 80% of all global traffic in 2019 [1]. Despite its prevalence, TCP is not optimized for video – video streaming has real-time constraints such that the next packet is often more important than a lost or in-flight packet. In contrast, TCP blocks sender window progression until each packet is delivered in order, possibly incurring playout delay at the client [2]. This is especially true when TCP is configured to use a loss-based congestion control protocol, such as CUBIC [3]. With TCP CUBIC, networks that would have Round-Trip Times (RTTs) of milliseconds when uncongested bloat buffers and can instead have RTTs of seconds. This increased RTT can reduce video streaming Quality of Service (QoS) because it can increase the time to detect and retransmit lost packets or to switch video to lower encoding rates [4].

TCP BBR, a new congestion control protocol developed by Google [5], aims to combat bufferbloat by estimating the minimum Round-Trip propagation time (RTprop) and the maximum Bottleneck Bandwidth (BtlBw) for a given connection to compute the Bandwidth Delay Product (BDP). BBR then caps its inflight packets to a small multiple of the BDP and paces its sending rate to match the estimated bottleneck bandwidth. In theory, having a single BDP of packets inflight is optimal for a network flow as it minimizes delay while maximizing a connection's throughput [6], and BBR's inflight cap is set to operate close to this optimal value.

In practice, BBR has been successfully deployed in Google's YouTube edge servers, increasing QoS [5]. Spotify AB, an audio streaming platform, has found that BBR helps reduce streaming stutters [7] and Dropbox, a file hosting service, has run preliminary BBR experiments that show an increase in file download speeds [8].

Despite promising performance, BBR may not operate well on pathways with shallow router buffers and when in direct competition with loss-based congestion controlled flows, such as TCP CUBIC [5], [9], [10]. In shallow buffers, BBR creates high loss due to its too-high Congestion Window (CWND) cap and persistence despite lost packets. This is especially problematic when it shares a bottleneck with loss-based congestion control protocols. Even when router queues are not shallow, BBR can induce significant throughput variation when sharing a bottleneck with CUBIC.

This study seeks to answer the following questions:

A) Can an inexpensive hardware testbed be effective for evaluating state of the art congestion control protocols?

B) Does BBR exhibit high loss rates when run over shallow router buffers?

C) Is BBR unfair when run with competing CUBIC flows?

We set up a hardware testbed for controlled experiments using the latest network OS code and create custom tools to conduct a wide variety of network performance tests. These tests vary: link capacities, network latencies, router queue lengths, TCP congestion control algorithms, and number of flows competing at a bottleneck.

Analysis of the results verifies prior work regarding BBR's performance in shallow buffers and in competition with CUBIC. It is crucial for experimental results to be reproduced by others within the scientific community in order to generalize the knowledge beyond the experience of the individual scientist. Our work goes further than previous research since our wide variety of tests allows us to more precisely define and identify BBR's behaviors over a range of network conditions. Specifically, we find that BBR's high throughput variation and high loss in shallow buffers are due to a static CWND cap and erroneous minimum RTprop estimates.

The contributions of this work include:

- Configuration, software framework and validation of an inexpensive testbed for conducting congestion control research using the latest Linux kernel code.
- Confirmation of base BBR performance under known, controlled conditions, validating Cardwell et al. [5].

- Extension to understanding BBR's behavior in shallow buffers, validating Hock et al. [11], with new observations on loss rate versus buffer size over a range of link conditions.
- Extension to understanding of BBR's cyclic behavior with deeper buffers and competing with CUBIC flows, validating Scholz et al. [9] and Miyazawa et al. [10], contributing new details on the BBR protocol features that cause this cyclic behavior, and quantification of fluctuation and unfairness versus buffer size.
- Suggestions for improving BBR by adjusting BBR's RTT estimation and adding a feedback loop to dynamically adjust BBR's CWND.

The rest of this paper is organized as follows: Section II discusses prior work in TCP congestion control and BBR; Section III describes our experimental setup to evaluate BBR in a hardware testbed; Section IV analyzes the experiment results; and Section V summarizes our conclusions and presents possible future work.

II. Related Work

This section defines the optimal network operating point (Section II-A), and describes recent TCP congestion control protocols (Section II-B), including BBR (Section II-C).

A. *The Optimal Operating Point*

Regardless of the number of hops, TCP views the network path as a single link characterized by its Bottleneck Bandwidth (BtlBw) and the minimum RTT (RTprop) it takes a packet to propagate through the network in the absence of queuing delay. The theoretical optimal operating point for a network flow using TCP is when exactly one bandwidth × delay (the Bandwidth-Delay Product, or BDP) of packets is in flight at all times, and the arrival rate of packets at the bottleneck router is close to the service rate (the limiting factor of the bandwidth) of that router [6], [12]. If these conditions are met, then the bottleneck router link is fully utilized with no extra queuing delay. This operating point has been coined *Kleinrock's operating point* after researcher Leonard Kleinrock.

This relationship can be seen in Figure 1, with the packets inflight depicted on the x-axes and the RTT (including queuing) on the y-axis of the top graph and the delivery rate on the y-axis of the bottom graph. The minimum RTT is labeled RTprop, achieved when the inflight is less than or equal to the BDP. When there is less than a BDP of packets in the network, the bottleneck link is underutilized, seen with a delivery rate less than the link capacity (labeled BtlBw). If there is more than one BDP of packets in the network, the delivery rate stays maxed out at the bottleneck link capacity, but the packets incur queuing delay and the RTT increases beyond the minimum. Thus, the optimal operating point achieves the maximum throughput *and* the minimum latency. Note that loss-based congestion control protocols, such as TCP CUBIC, operate at the right side of this graph, increasing the amount of inflight packets until the bottleneck router queue saturates and packets are dropped.

B. *Recent Congestion Control Protocols*

Unfortunately, Kleinrock's operating point has been proven to be practically impossible to converge to for a distributed algorithm [13]. Thus every congestion control protocol seeks a "good" operating point, often favoring throughput over delay. The different congestion control approaches include: loss-based protocols (Section II-B1) that maximize throughput, utility-based protocols (Section II-B2) that adjust their operating points based on measures of utility, and measurement-based protocols (Section II-B3) that explicitly set parameters in an attempt to match the theoretical optimal operating point.

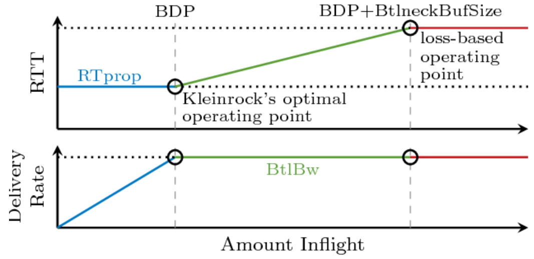


Figure 1. Optimal operating point [9].

1) Loss-based congestion control: Loss-based congestion control protocols treat lost packets as congestion signals, increasing the number of inflight packets until packet loss occurs (indicating the network is saturated) whereupon they decrease their inflight packets, and repeat the cycle. This behavior means loss-based protocols generally operate to the right of Kleinrock's operating point as depicted in Figure 1, resulting in high throughput, but also high latency.

TCP CUBIC [3], the de facto standard loss-based congestion control protocol, aims to maximize network utilization by controlling the Congestion Window (CWND) with a cubic function. The cubic function's convex nature allows the CWND to quickly grow to utilize available capacity. In the case of a congestion event (such as packet loss), the cubic function adjusts such that the CWND increases in a concave manner as it approaches the previous maximum, avoiding overshooting the maximum network capacity and excessive packet loss. In the absence of loss, the cubic function returns to the convex profile to rapidly fill available network capacity. Together, these two profiles seek high utilization and low loss.

2) Utility-based congestion control: Utility-based congestion control protocols evaluate their performance based on a utility function, and adjust parameters to maximize utility over time. Unlike loss-based congestion control protocols (such as TCP CUBIC), utility-based congestion control protocols do not have explicit responses for different congestion events, but rather have just a general set of actions to take to maximize their utilities. In general, using a utility function can simplify protocol design since not every condition needs to be explicitly handled, but rather only needs algorithm parameter adjustments to improve utility.

The utility-based Performance-oriented Congestion Control (PCC) [14] works under the assumption that networks are too complicated to deterministically predict the effects of a given action. The PCC premise is that it is infeasible for a protocol to tune performance with a predefined action for every congestion event, as does CUBIC. Thus, PCC treats the underlying network as a "black box" and empirically observes which actions provide better utility by continuously conducting mini experiments. In a mini experiment, PCC reduces or increases the sending rate and CWND and observes the utility

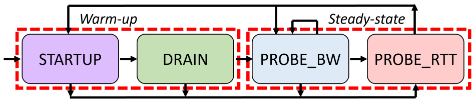


Figure 2. BBR state transition diagram.

for this action. The observation informs its next action through a gradient-ascent algorithm to adjust sending rates towards the optimal. This decoupling of congestion events and actions allows PCC to perform well even in networks with high random loss, such as in some wireless networks.

Copa [15], another utility-based congestion control protocol, uses a simple set of rules to update the sending rate and CWND towards an optimal utility value, creating little to no queuing at the bottleneck router if all flows are using a similar CWND updating method. Copa measures the queuing delay as the difference between the observed and minimum RTTs, and increases its sending rate until small queues are created at the bottleneck. Copa implements a "competitive mode" if it detects a competing buffer-filling flow. This addresses the problem with delay-based utility functions, such as in TCP Vegas [16], where full buffers cause a reduction in inflight packets to reduce congestion, leading to unfairness when competing with buffer-filling protocols.

3) Measurement-based Congestion Control: Loss-based congestion control protocols are reactive, waiting until loss is observed before taking an action to reduce congestion, and utility-based congestion control protocols treat the network as a black box, just taking actions to based on some utility function. Conversely, measurement-based protocols attempt to estimate the current network operating point and explicitly adjust protocol parameters to try to meet the optimal. Examples include TCP Vegas, discussed in this section, and TCP BBR, discussed in Section II-C.

TCP Vegas [16] measures the observed throughput and RTT to estimate how much bloat (excess queued packets) it has created in the network. Vegas sets the expected throughput to $CWND/RTprop$ and compares this value to the actual observed throughput. If the actual throughput is lower than the expected throughput by some threshold α, Vegas assumes it has too small a congestion window, and increases the CWND linearly for one RTT. Similarly, if the actual throughput is greater than the expected throughput by some threshold β, Vegas assumes the CWND is too large and decreases the CWND accordingly.

In theory, this scheme should let Vegas operate near Kleinrock's operating point, but in practice Vegas tends to be too conservative and cedes capacity to loss-based congestion control protocols such as CUBIC because Vegas decreases its CWND to minimize buffer bloat while loss-based protocols increase CWNDs until loss occurs.

C. Bottleneck Bandwidth and Round-trip Propagation Time

Bottleneck Bandwidth and Round-trip propagation time (BBR) [5] is a congestion control protocol designed to replace loss-based algorithms. BBR [5] aims to operate near Kleinrock's operating point (see Section II-A) by estimating the BtlBw and RTprop parameters and setting the sending rate and inflight packets accordingly. BBR operates using a series of states shown in Figure 2. After slow start, BBR looks for increases to the bandwidth during *PROBE_BW* every 8 RTTs, increasing the CWND and setting a rate multiplier to send at a rate greater than the current BtlBw. The new estimated BtlBw is set to the maximum delivery rate observed during this probe. The RTprop estimation expires after 10 seconds, causing BBR to enter *PROBE_RTT* to re-estimate the minimum RTT. In Probe_RTT, BBR briefly reduces its inflight packets, draining any queue that it had built up. The new RTprop estimate is set to the minimum RTT observed. BBR always paces its sending rate at the estimated BtlBw and caps its inflight CWND to two times the estimated BDP. The CWND inflight cap is set to two BDP rather than the theoretically optimal one BDP to accommodate delayed and stretched ACKs in wireless networks. However, as we show in Section IV, the larger CWND can cause high packet loss, instability, and unfairness.

Because BBR relies on an estimated RTprop and BtlBw, BBR has inconsistent behavior when it misestimates one or both of these values. Hock et al. [11] find that when multiple BBR flows share a bottleneck, BBR pathologically over-estimates its fair-share of the bandwidth since each flow measures the maximum available bandwidth over a time period. The sum of throughput's derived from these estimates is then *always* greater than the bottleneck's actual maximum bandwidth, causing persistent queues to build at the bottleneck router until the inflight cap of 2 BDP is reached [11]. This persistent queue is especially problematic when the bottleneck router queue is smaller than a single BDP, whereupon BBR attempts to build a queue of 1 BDP and ignores the massive packet loss caused by overwhelming the bottleneck queue.

Scholz et al. [9] and Miyazawa et al. [10] show that BBR also produces inaccurate RTprop estimates when it shares the bottleneck with buffer filling protocols, such as TCP CUBIC. When BBR over-estimates the RTprop, it drastically changes its CWND and thus creates large amounts of loss. This loss and mis-measurement leads to a cyclic behavior where BBR and CUBIC each have constantly fluctuating throughput.

We confirm these prior BBR results but in a broader range of network scenarios and provide additional explanations in Section IV. We incorporate these new insights into our suggestions to improve BBR's performance in Section IV-E.

III. Experiments

We set up a hardware testbed and develop a set of custom tools that enable a variety of network experiments for evaluating TCP CUBIC and TCP BBR in a controlled environment. Because our testbed relies on off-the-shelf hardware and software, it provides an attractive environment for congestion control research that can be recreated at relatively low cost.

Our testbed, named "Panaderia" which translates to "bakery" or "bread shop" in Spanish, consists of 8 Raspberry Pi computers, two network switches, and one Linux PC named "Horno" (oven) functioning as a router. The use of hardware (versus simulation) allows us to run the mainline Linux kernel versions of BBR and CUBIC, thus ensuring adherence to implemented protocol behavior. Further, the low cost allows us to afford separate hosts for each flow, thus removing any confounding factors from multiple flows originating from a single machine.

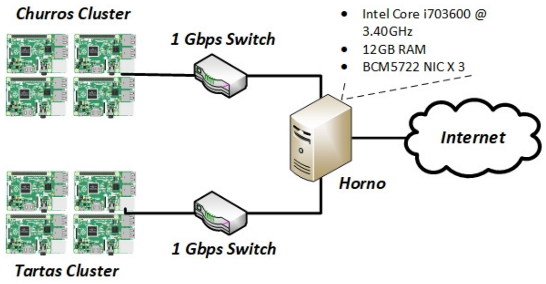


Figure 3. Hardware testbed topology.

TABLE I. EXPERIMENT PARAMETERS.

Parameters	Values
Number of Flows	2, 4, 8
Network Capacity	40, 80, 120 Mb/s
Congestion Control Protocol	CUBIC, BBR
Router Queue Size	0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 4.00, 8.00 BDP

The hardware is configured in a traditional dumbbell topology shown in Figure 3 – the Raspberry Pis are split into two subnets of four machines (“churros” and “tartas” clusters).

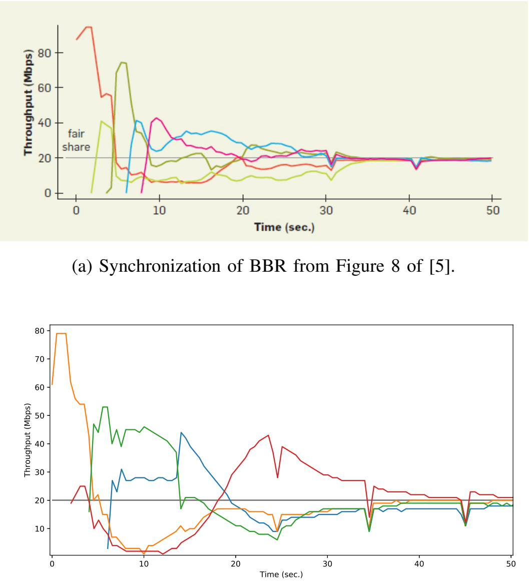
Each Raspberry Pi is a model 3B+ running the Linux kernel v4.17. Pilot tests (not shown) reveal an individual Raspberry Pi has a maximum sending rate of about 225 Mb/s, limited by the USB 2.0 bus speed. Below this throughput, we verified that the network behavior of the Raspberry Pi performs similarly to the network behavior of a Linux PC. We developed a series of Python scripts to allow us to nimbly run experiments, vital for comparing BBR’s behavior over a wide range of network conditions. Details on the setup specifics, as well as access to our configuration scripts, are available via our git repository [17].

The router is a Linux PC, configured with an Intel i7 CPU, 12 GB of RAM, and Broadcom BCM5722 Gigabit Ethernet PCI cards. The router uses NetEm [18] to add a fixed propagation delay of 24 ms giving a total round-trip propagation time of 25 ms to align with previous research [5]. The router also uses `tc-tbf` token bucket filters to control the bottleneck bandwidth [19]. Since accumulated tokens can cause bursts of traffic interfering with BBR’s bottleneck bandwidth estimate, the router is configured with two token bucket filters with small buckets to limit instantaneous bursts [20]. Receiving and sending devices each collect `pcap` network traces, which are subsequently analyzed for RTT, throughput, and inflight packets. Additionally, the router records the number of bytes queued and number of packets dropped.

Our experiments vary flows, protocols, network capacities and router queue sizes, as shown in Table I.

Validation: When creating a novel testing environment, it is important to validate testbed behavior against known results. Similarly, for science, it is vital to confirm that prior work is reproducible to ensure a knowledge foundation before making improvements. We validate the Raspberry Pi’s TCP networking by, among other methods, confirming that BBR follows the behavior seen by Cardwell et. al [5]. Specifically, we ensure that when multiple BBR flows share a single bottleneck, each flow converges to a fair share of the maximum bandwidth, and that the flows synchronizes during their PROBE_RTT phases.

Figure 4a depicts the throughputs of 5 simultaneous BBR flows with staggered start times competing for a 100 Mb/s bottleneck with a RTprop of 10 ms, as evaluated by Cardwell et al. [5]. Each of the flows obtains a fair share of the bottleneck capacity, 20 Mb/s, and after about 30 seconds, each flow enters PROBE_RTT at the same time, seen by the throughput dips at about times 30 and 40 seconds. Figure 4b depicts our Panaderia testbed’s behavior in similar conditions with 4 simultaneous BBR flows with staggered start times competing for a 80 MB/s link. Similar to Cardwell et al., our BBR flows synchronize at a fair share of 20 Mb/s within about time 30 seconds, repeatedly doing so every 10 seconds. Further validation tests are not shown due to lack of space, but can be found in the full report [21]. In total, this both confirms Cardwell et al.’s measured BBR behavior (a first-class scientific contribution) as well as supports the validity of the Panaderia as a network testbed for evaluation of TCP congestion control protocols.



(a) Synchronization of BBR from Figure 8 of [5].

(b) Synchronization of BBR in the Panaderia.

Figure 4. Comparison of Cardwell et al. [5] to our Panaderia testbed.

IV. Results

This section: validates BBR’s behavior (Section IV-A); extends previous work in shallow buffers (Section IV-B) and router queue sizes (Section IV-C); evaluates BBR’s interplay with CUBIC (Section IV-D); and proposes mechanisms to improve BBR’s performance (Section IV-E).

A. Standard BBR Behavior

Prior work has shown that when there is more than one flow competing for the same bottleneck, BBR tends to create a 1 to 1.5 BDP standing queue [11]. We verify this by running 2, 4, and 8 BBR flows for 5 minutes at 40, 80, and 120 Mb/s with a 25 ms RTprop and a large bottleneck queue.

Figure 5 depicts 4 BBR flows competing for an 80 Mb/s link with the maximum router queue size set to 2 MBytes ($8BDP$). Only the steady-state behavior is analyzed (1 minute

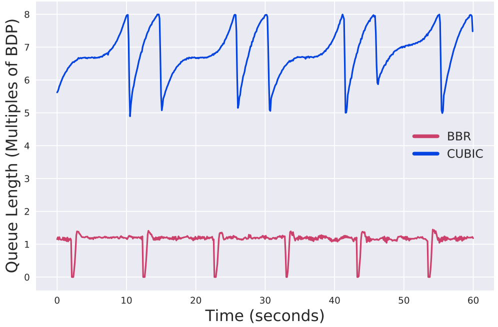


Figure 5. BBR and CUBIC in a large router queue.

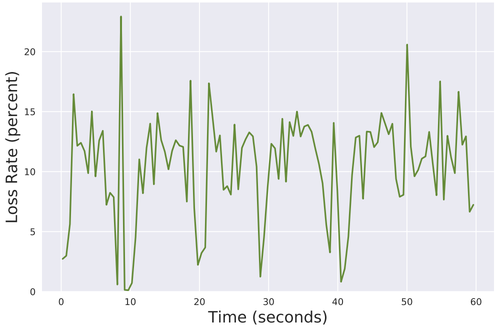


Figure 6. BBR's loss rate in a small router queue.

of a 5 minute trial). For reference, we independently ran and show on the same graph a TCP CUBIC flow to compare behaviors. CUBIC, as a loss-based protocol, continues to fill the queue until the 8 BDP maximum is reached. In contrast, BBR creates a persistent queue of about 1.1 BDP, increasing to roughly 1.5 BDP during bandwidth probing.

Over the full range of flows, capacities, and queue sizes listed in Table I, analysis confirms that BBR creates a standing queue of 1 to 1.5 BDP, similar to what is shown in Figure 5. Results are not shown due to lack of space, but can be found in the full report [21]. Note that while BBR's RTT is low (roughly 0.25 CUBIC's), it is around double the RTprop since the 1 BDP of packets queued at the router takes a full RTT to process ($BDP = BtlBw \times RTprop$).

B. BBR in Shallow Buffers

Since multiple BBR flows competing at a bottleneck create about a BDP queue at the bottleneck router, when routers have shallow buffers (i.e., a small router queue) BBR has the potential to saturate the queue and create loss.

We evaluate a shallow buffer scenario with a bottleneck bandwidth of 80 Mb/s and a RTprop of 25 ms, but with a bottleneck router queue of just 0.5 BDP. Figure 6 depicts the loss rate at the router averaged over 500 ms intervals. All further loss rate graphs are calculated in the same manner. From the graph, loss varies considerably over time, but persists and averages over 10 percent.

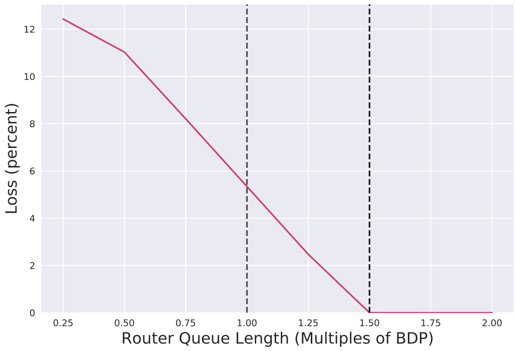


Figure 7. BBR's loss versus router queue size.

C. BBR over Different Router Queue Sizes

We evaluate BBR's loss rate over a range of router queue sizes. Specifically, we run 3 identical trials at 40, 80, and 120 Mb/s, all at 25 ms RTprop for 5 minutes for each given queue size: 0.25, 0.50, 0.75, 1.25, 1.50, 2.00, 4.00, and 8.00 BDP. The recorded packet captures at each host and the queue statistics at the bottleneck router determine the aggregate behavior of BBR given the bottleneck queue size.

Figure 7 depicts the results. The x-axis is the maximum router queue length in terms of the BDP and the y-axis is the loss rate during steady state (minutes 2 to 4 of a 5 minute connection) averaged over the 3 trials.

Since BBR causes an extra 1.5 BDP of packets to be enqueued at the bottleneck router, the loss rate is extremely high when the buffer size is less than a BDP and is high unless the router buffer size is at least 1.5 BDP. This result confirms the findings of Hock et al. that in practice BBR's inflight is 2.5 BDP [11], while providing specific analysis of the actual loss rates realized versus queue size.

Since BBR does not respond to loss as a congestion signal, the queue always grows to 1.5 BDP, even though BBR's throughput remains relatively high despite the loss (not shown due to lack of space, but shown in the full report [21]). From here on, we refer to buffers less than 1.5 BDP as shallow, buffers greater than 4 BDP as large and buffers in-between as medium. Note analysis of router queue size is relative to the BDP – with high throughput, high RTT, a shallow buffer of 1 BDP could have a lot of bytes.

D. BBR's Interplay with CUBIC

Another concern with BBR is when it competes with CUBIC and other loss-based congestion control. BBR's mechanism for controlling the bottleneck bandwidth is at odds with CUBIC's – CUBIC adjusts its CWND to minimize loss, while BBR mostly ignores loss as a congestion signal. This difference presents itself uniquely in each of the conditions discussed above – shallow, medium and large buffers.

1) BBR and CUBIC in Shallow Buffers: By itself in shallow buffers, BBR creates high amounts of ambient loss by growing its CWND beyond what the network can handle (see Figure 6). Because CUBIC treats such loss as congestion in the same scenario, CUBIC shrinks its CWND in response. Figure 8 depicts the relative throughputs of BBR and CUBIC competing over a shallow (0.5 BDP) buffer in an 80 Mb/s

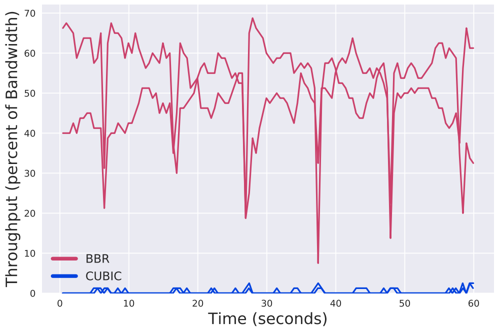


Figure 8. Throughput utilization.

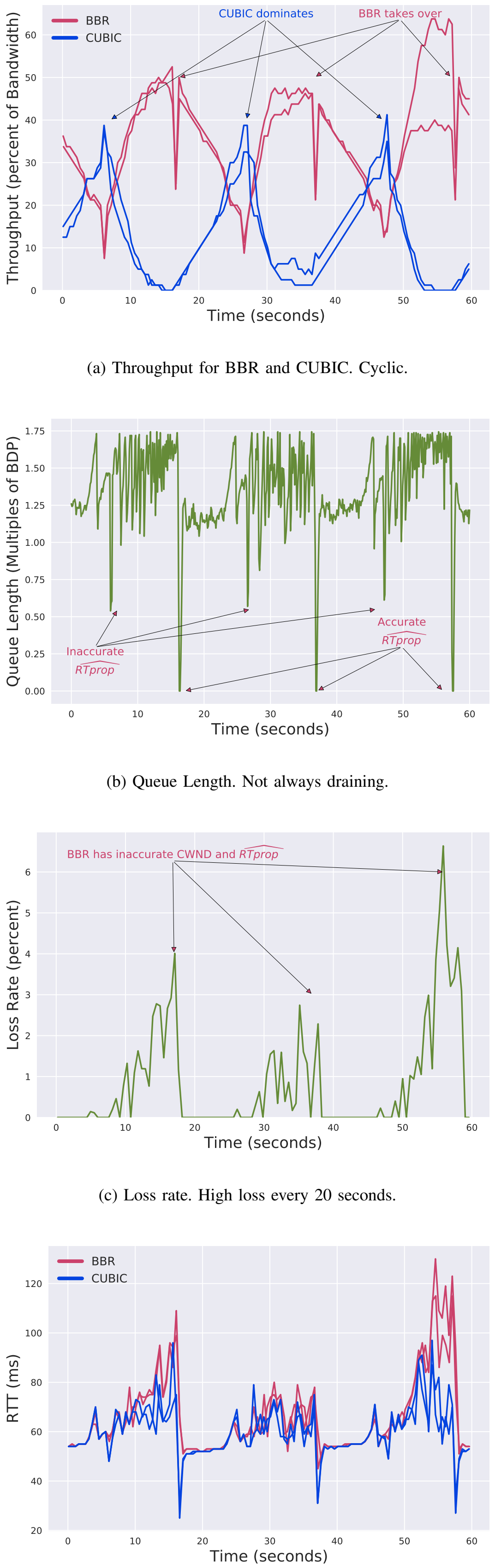
and 25 ms bottleneck. The loss rates are similar to Figure 6, averaging about 10 percent. As seen in Figure 12, the relative throughput for CUBIC is *much* lower than BBR because, again, CUBIC responds to the loss whereas BBR does not. Similar trials with 40, 80 and 120 Mb/s and 2, 4 and 8 flows show similar results. These results are omitted due to lack of space, but can be found in the full report [21].

2) BBR and CUBIC in Medium to Large Buffers: In medium buffers, where BBR is *not* persistently inducing loss, BBR and CUBIC display a cyclic behavior. We evaluate this by running 2 BBR and 2 CUBIC flows through a 80 Mb/s capacity, 25 ms RTprop bottleneck and a router queue of 1.75 BDP. Figure 9 shows the results, annotated to match the explanations that follow. BBR and CUBIC exhibit cyclic performance – they alternate which flows dominate the connection over a regular 20 second period, confirming prior results by Scholz et al. [9] and Miyazawa et al. [10]. We build upon this work by explaining the factors that cause the cycles in detail.

CUBIC Dominates. When BBR has an accurate estimate for the BtlBw and RTprop, it caps its inflight at 2 $\widehat{BDP}$. This means that BBR allows just 1 BDP of packets to enqueue at the bottleneck router for 8 RTTs. During this time, CUBIC expands its CWND by an additional 0.75 BDP to fill the router queue before encountering loss.

Since a flow's throughput is proportional to the queue share at the bottleneck router, as CUBIC gets more packets enqueued, BBR observes a lower throughput and thus further decreases its CWND, which is derived from the observed throughput. This creates a positive feedback loop allowing CUBIC to continue to increase its CWND in response to BBR's decrease of its CWND as it observes a reduced throughput. This behavior can be seen from time 0 to 10 sec. in Figure 9.

BBR takes over. Every 10 seconds without observing a new minimum RTT ($\widehat{RTprop}$), BBR probes for RTprop by reducing its inflight packets to just 4 packets to drain the router queue [5] (e.g., time 28 seconds of Figure 9a). BBR uses the minimum observed RTT as the new $\widehat{RTprop}$. However, the queue length, shown in Figure 9b, does not change significantly because most of the packets in the queue are from the CUBIC flows. In other words, even when BBR decreases its inflight packets, the queue stays relatively filled. Figure 9d depicts the RTT over this period, where around time 28 seconds, the RTT is still much higher than the true RTprop – around 60 ms rather than 25 ms. This causes the $\widehat{RTprop}$ to



(a) Throughput for BBR and CUBIC. Cyclic.

(b) Queue Length. Not always draining.

(c) Loss rate. High loss every 20 seconds.

(d) RTT. Cyclic.

Figure 9. BBR and CUBIC show cyclic performance in medium buffers.

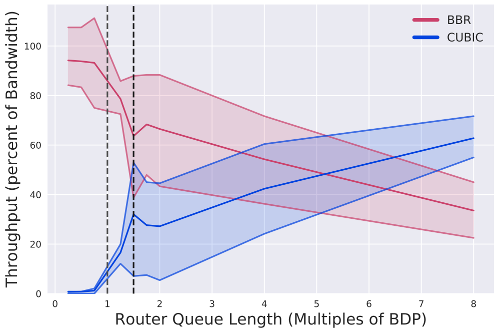


Figure 10. BBR and CUBIC's interplay versus router queue length.

be too large, and, because BBR's $\widehat{BDP}$, and thus CWND, is derived from this RTprop, BBR greatly increases its CWND.

Because the router queue is already filled by CUBIC, BBR's increased CWND causes a large amount of packet loss for both CUBIC and BBR. BBR mostly ignores the loss, but CUBIC backs off, decreasing its inflight packets. This loss can be seen at around time 10, 30, and 50 seconds of Figure 9c, each of which corresponds to just after BBR increases its CWND after an inaccurate RTprop probe.

This continues for 10 seconds, whereupon BBR again probes for RTprop. This time, the probe obtains an accurate $\widehat{RTprop}$ of 25 ms, as seen at time 38 seconds of Figure 9b, because the queue is fully drained, and thus BBR reduces its CWND accordingly. This allows CUBIC to grow its CWND, as discussed above, and the cycle repeats.

We analyzed this behavior over the full range of test parameters in Table I, visualizing this interplay for 80 Mb/s, 25 ms, 2 BBR & 2 CUBIC flows with queue sizes: 0.25, 0.50, 0.75, 1.00, 1.75, 2.00, 4.00, and 8.00 BDP in Figure 10. The x-axis depicts the queue size for each trial, and the y-axis the throughput utilized by the flows grouped by congestion control protocol. The thick lines depict the mean throughput of BBR (red) and CUBIC (blue), with the thinner colored lines showing the 25th and 75th percentiles, averaged over half second intervals. From the figure, the behavior differs drastically for router queue sizes below 1.5 BDP where BBR creates persistent loss. Above 1.5 BDP, throughput spread greatly increases from the cyclic performance described earlier.

As the bottleneck queue gets larger, BBR becomes more limited by its 2 BDP CWND cap. This causes CUBIC to progressively obtain more of the throughput as its CWND grows beyond BBR's CWND limits.

E. Improving BBR's Performance

We have identified two performance weaknesses in BBR: BBR's static 2 BDP CWND, and BBR's inaccurate RTprop estimation. This section suggests fixes to these issues, with a limited proof of concept evaluation. Note that the evaluation is meant as an inspiration, not as a vigorous implementation.

1) CWND: Currently, BBR caps the inflight packets at 2 BDP, causing there to be 1 to 1.5 BDP of packets queued at the bottleneck router and a high amount of loss in shallow buffers. These results indicate that this 2 BDP CWND cap is sometimes too large to suit the network conditions. However, when the bottleneck router queue is large and BBR competes with a loss-based congestion control protocol, BBR's CWND cap limits its queue share, and, hence, it's share of the bottleneck bandwidth. These results indicate the CWND cap is sometimes too small to suit the network conditions. We suggest a feedback loop where BBR dynamically adjusts its CWND cap based on the loss, throughput and RTT measurements. This idea is similar to the approach of Copa [15] which generally keeps a small CWND, but increases it in response to competition from buffer-filling congestion control protocols.

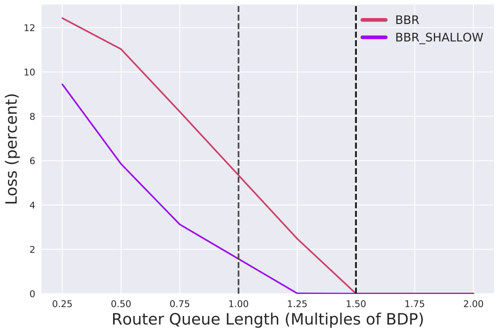


Figure 11. Modified BBR_Shallow's loss versus router queue length.

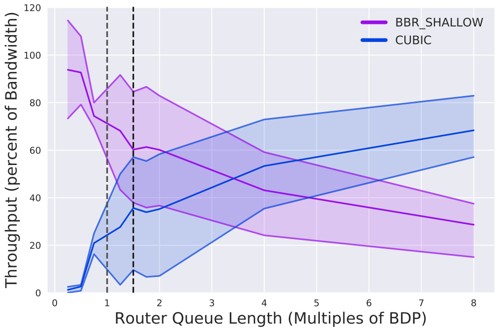


Figure 12. BBR_Shallow and CUBIC's interplay versus router queue length.

When encountering persistent, high loss, BBR can infer that it is over saturating, and in response BBR can decrease the CWND cap. However, if the observed throughput is less than the measured bottleneck bandwidth (BtlBw), BBR can infer it is underutilizing its share of the bottleneck capacity (or that there are delayed and/or aggregated ACKs) and increase the CWND cap. When the measured RTT is greater than the RTT min, then the router queue must be saturated and the CWND cap should be decreased.

We demonstrate that BBR's inflight cap is responsible for its capacity and shallow buffer loss by manually adjusting BBR's inflight cap to 1.5 BDP down from 2 BDP. We then rerun our experiments with this modified "BBR_Shallow" [22].

Figure 11 depicts loss rate at the router calculated with a 500 ms sliding window as in Figure 7. BBR_Shallow induces less loss in buffers smaller than 1.5 BDP because it attempts to queue fewer packets at the bottleneck router, helping with inter-protocol fairness by reducing loss.

When competing with CUBIC, the router queue size that provides for a fair share is smaller, as shown in Figure 12, following the format of Figure 10. Instead of having a fair share when the bottleneck router queue size is around 5 BDP, the fair share is instead when the maximum bottleneck router queue size is around 3 BDP. Thus, the CWND cap helps determine inter-protocol fairness by limiting throughputs.

2) Improving BBR's RTprop estimation: The RTprop should be nearly constant for the same network path, only changing after a route change. However, when BBR competes with CUBIC, BBR *increases* the estimated RTprop when it is unable to drain the queue, causing BBR to greatly overestimate the BDP, and thus create high queuing and loss.

BBR could instead always use the historical minimum RTT, basically never increasing the estimated RTprop. With this change, BBR would no longer be responsive to route changes that result in a higher RTprop. Either BBR could ignore route changes completely (since they are uncommon, and most flows are short) or BBR could only accept a higher RTprop if it is *consistently* higher for many RTprop probes or drastically higher for a single probe. This could allow BBR to detect a route change without increasing its CWND erroneously.

V. Conclusions and Future Work

As TCP BBR becomes more widely adopted in streaming audio and video applications, it is important that BBR provides consistent behavior alongside existing protocols, such as TCP CUBIC, and efficient behavior over a range of router queue sizes. Currently, BBR's high loss rates in shallow buffers and throughput variations with CUBIC in deeper buffers can be inefficient for throughput and disastrous for streaming audio and video applications that rely on a consistent network throughput for smooth playback.

Contributions of our work include: a) description and software for an inexpensive testbed for congestion control research; b) validation of base BBR performance [5]; c) validation of BBR performance in shallow buffers, with new contributions on loss rate versus buffer size; d) validation of cyclic behavior when BBR and CUBIC compete [9], [10], with new details on the BBR protocol features, and quantification of variation and unfairness versus buffer size; e) original analysis quantifying variation and unfairness of CUBIC and BBR versus buffer size; and f) contemporary improvement suggestions to BBR to address the above shortcomings.

In response to the questions posed in the Section I:

A) Raspberry Pi 3B+ end-hosts and a Linux PC configured as a router can be used for congestion control research, having the benefits of a controlled, network testbed and the latest Linux kernel code as is used on the Internet.

B) BBR induces packet loss on congested links with a router buffer sized smaller than 1.5 BDP, with loss increasing linearly with a decrease in buffer size. Loss rates are 5.5% for a router buffer size of 1 BDP, doubling to about 11% when the router buffer size is 0.5 BDP, caused by BBR's 2 BDP CWND cap.

C) BBR is extremely unfair when competing with CUBIC in shallow buffers since CUBIC responds to the packet loss by reducing rates while BBR does not. In deeper buffers, BBR and CUBIC exhibit large 20-second throughput cycles, with average throughput fairness achieved only for specific router queue sizes based on the BDP. These cycles are caused by BBR's inaccurate estimate of the minimum RTT, with the unfairness caused by the tension between the difference between the BBR and CUBIC operating points.

We suggest heuristics to improve BBR that include a feedback loop to adjust BBR's CWND and minimum RTT measurements based on the network conditions. Future work is needed to implement and test these proposed changes. Future work could also verify that BBR's unstable behavior is present in competition with any buffer-filling congestion control. Additionally, BBR evaluation over broader network conditions such as 4G, 5G and satellite is important.

References

[1] Cisco, "VNI Global Fixed and Mobile Internet Traffic Forecasts," 2017.

[2] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," ACM Transactions on Multimedia Computing (TOMM), vol. 4, no. 2, 2008, p. 16.

[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," ACM SIGOPS OS Review, vol. 42, no. 5, Jul. 2008.

[4] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the Internet," Communications of the ACM, vol. 55, no. 1, Jan. 2012, p. 57.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and Van Jacobson, "BBR: congestion-based congestion control," Communications of the ACM, vol. 60, no. 2, Jan. 2017, pp. 58–66.

[6] L. Kleinrock, "Internet congestion control using the power metric: Keep the pipe just full, but no fuller," Ad Hoc Networks, vol. 80, Nov. 2018.

[7] E. Carlsson and E. Kakogianni, "Smoother streaming with BBR," Aug. 2018, Spotify Labs. Accessed: February 24, 2019. [Online]. Available: https://tinyurl.com/yyt5tbhd

[8] A. Ivanov, "Optimizing Web servers for high throughput and low latency," Sep. 2019, Dropbox Blogs. Accessed: April 7, 2019. [Online]. Available: https://tinyurl.com/y9qtt2yf

[9] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in Proceedings of IFIP Networking, Zurich, Switzerland, 2018.

[10] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cycle and divergence of performance on TCP BBR," in IEEE 7th International Conference on Cloud Networking (CloudNet), 2018.

[11] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in IEEE ICNP, Toronto, ON, Oct. 2017, pp. 1–10.

[12] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in ICC, Boston, MA, 1979.

[13] J. M. Jaffe, "Flow control power is nondecentralizable," IEEE Transactions on Communications, vol. 29, 10 1981, pp. 1301 – 1306.

[14] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in USENIX NSDI, 2015, pp. 395–408.

[15] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in Proceedings of the Applied Networking Research Workshop. Montreal, QC: ACM Press, 2018, pp. 19–19.

[16] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, Oct. 1995, pp. 1465–1480.

[17] S. Claypool, "The Panaderia," 2019. [Online]. Available: https://saahilclaypool.github.io/panaderia/

[18] S. Hemminger, "Network emulation with NetEm," in Proceedings of Linux Australia, Sydney NSW 2001, Australia, 2005, pp. 18–23.

[19] "tc-tbf (8): Token Bucket Filter - Linux man page," accessed: February 24, 2019. [Online]. Available: https://linux.die.net/man/8/tc-tbf

[20] "Dual Token Bucket Algorithms - TechLibrary," accessed: February 24, 2019. [Online]. Available: https://tinyurl.com/yy5owway

[21] S. Claypool, "Sharing but not caring - performance of TCP BBR and TCP CUBIC at the network bottleneck," WPI Major Qualifying Project MQP-CEW-1904, March 2019, (Advisor C. Wills). [Online]. Available: https://digitalcommons.wpi.edu/mqp-all/6728/

[22] S. Claypool, "BBR Shallow," 2019. [Online]. Available: https://tinyurl.com/bbrshallow-git