When to use and when not to use BBR: An empirical analysis and evaluation study

Yi Cao
Stony Brook University
yicao1@cs.stonybrook.edu

Arpit Jain
Stony Brook University
arjain@cs.stonybrook.edu

Kriti Sharma
Stony Brook University
krisharma@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
arunab@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

ABSTRACT

This short paper presents a detailed empirical study of BBR's performance under different real-world and emulated testbeds across a range of network operating conditions. Our empirical results help to identify network conditions under which BBR outperforms, in terms of goodput, contemporary TCP congestion control algorithms. We find that BBR is well suited for networks with shallow buffers, despite its high retransmissions, whereas existing loss-based algorithms are better suited for deep buffers.

To identify the root causes of BBR's limitations, we carefully analyze our empirical results. Our analysis reveals that, contrary to BBR's design goal, BBR often exhibits large queue sizes. Further, the regimes where BBR performs well are often the same regimes where BBR is unfair to competing flows. Finally, we demonstrate the existence of a loss rate "cliff point" beyond which BBR's goodput drops abruptly. Our empirical investigation identifies the likely culprits in each of these cases as specific design options in BBR's source code.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network performance analysis**; **Network measurement**.

ACM Reference Format:
Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to use and when not to use BBR: An empirical analysis and evaluation study. In *Internet Measurement Conference (IMC '19), October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3355369.3355579

1 INTRODUCTION

TCP congestion control algorithms have continued to evolve for more than 30 years [36]. As the internet becomes more and more complex, researchers have designed different TCP congestion control algorithms to serve different scenarios. For example, the legacy TCP Reno [25] uses an AIMD (additive-increase/multiplicative-decrease) algorithm to quickly respond to losses while slowly recovering from congestion. TCP Cubic [28] responds more aggressively to recover from losses — rather than a linear increase, TCP Cubic grows its congestion window in a cubic manner. Instead of using packet loss as a congestion signal, TCP Vegas [19] treats increasing round trip time (RTT) as evidence of congestion. To achieve high throughput and low latency in data centers, Alizadeh et al. implemented DCTCP [16], which uses explicit congestion notification (ECN) [35] as the congestion signal to prevent the packet loss from occurring before the buffer becomes too congested.

A fundamental issue with regards to designing a congestion control algorithm is: *what is the optimal operating point for doing congestion control?* Should we keep sending packets until the buffer becomes full and use packet loss as the congestion signal (e.g. Reno, Cubic), or should we treat packet delay as the congestion evidence (e.g. Vegas, Copa [17, 19]), or should we implement sophisticated algorithms via learning-based techniques (e.g. PCC, Indigo [24, 38])?

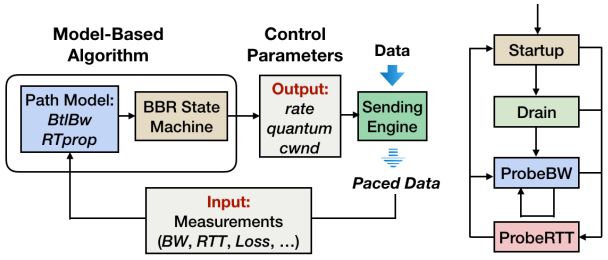
In 1979, Kleinrock showed that the optimal operating point for the network was when the bandwidth was maximized while minimizing the delay [33]. However, it was not until 2016 that this design point was explicitly used for congestion control. Google's BBR (Bandwidth Bottleneck and Round-trip propagation time) algorithm aims to operate at this optimal point by probing the current bandwidth and delay sequentially in the network [20], as we discuss in Section 2. BBR has since been employed at Google, and continues to be actively developed. To avoid bufferbloat [27], BBR regulates its congestion window size such that the amount of in-flight packets is a multiple of the bandwidth-delay product (BDP); ideally, this should result in small buffer sizes.

Despite the rising popularity of BBR, it is not fully clear *when BBR should be employed in practice*, that is, when does BBR outperform other congestion control algorithms. Prior work has typically focused on BBR's fairness properties [34, 37] and its throughput and queueing delay [30]; we discuss prior work on BBR in detail in Section 5.

The goal of this short paper is to conduct a comprehensive empirical study to investigate BBR's performance under different network conditions and determine when to employ BBR. In doing so, we also aim to identify the root causes of BBR's sub-optimal performance.

To this end, we conduct extensive experiments in both Mininet [6] and real-world networks to analyze the performance of BBR. To span the range of network operating conditions, we vary the bandwidth, RTT, and bottleneck buffer sizes by employing a router

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IMC '19, October 21–23, 2019, Amsterdam, Netherlands
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6948-0/19/10…$15.00
https://doi.org/10.1145/3355369.3355579



(a) BBR high level design. (b) State machine.

Figure 1: BBR congestion control algorithm design.

between the client and server machines. For each scenario, we contrast BBR's performance with that of Cubic [28], which is the default TCP variant on Linux and Mac OS, to determine the operating conditions under which BBR is helpful.

We synthesize the results of our 640 different experiments in the form of a *decision tree* highlighting the choice between BBR and Cubic, to aid practitioners. In general, we find that when the bottleneck buffer size is much smaller than BDP, BBR can achieve 200% higher goodput than Cubic. However, in the case of deep buffers, Cubic can improve goodput by 30% compared to BBR; here, buffer refers to the bottleneck buffer size. While we find that BBR achieves high goodput in shallow buffers, we observe that BBR's packet loss can be several orders of magnitude higher than that of Cubic when the buffers are shallow.

Our analysis of BBR's source code reveals that the high packet loss under shallow buffers is because of BBR's configuration parameters that maintain 2× BDP of data in flight. Decreasing the 2× multiplier or increasing the bottleneck buffer size significantly lowers the packet losses.

Our empirical results also suggest the existence of a "cliff point" in loss rate for BBR above which BBR's goodput decreases significantly. We find that, empirically, this cliff point is at around 20% loss rate. By modifying the BBR parameters, we find that the cliff point is largely dictated by the maximum *pacing_gain* value BBR uses when probing for more bandwidth. Interestingly, we find that BBR exhibits the highest amount of packet retransmissions at this cliff point.

Finally, we investigate the behavior of BBR in the presence of other flows. We find that the goodput share of BBR primarily depends on the bottleneck buffer size — BBR utilizes more bandwidth when the buffers are shallow, despite the high losses, whereas Cubic does better when the buffers are deep.

2 BACKGROUND ON BBR

Overview of BBR's design: We illustrate the design of BBR via Figure 1(a). BBR periodically obtains network information via measurements, including bandwidth, RTT and loss rate. BBR then models the bandwidth by using a max filter (the maximum value of the observed bandwidth in the last few RTTs), *BtlBw*, and the network delay by using a min filter, *RTprop*. BBR works according to a state machine which decides BBR's next state, as shown in Figure 1(b). The *BtlBw* and *RTprop* values are treated as input to this state machine.

Based on the current state, BBR calculates the *pacing_gain* (a dynamic gain factor used to scale *BtlBw*) and *cwnd_gain* (a dynamic gain factor used to scale BDP), and uses these values to derive *pacing_rate* (which controls the inter-packet spacing) and congestion window size, *cwnd*, respectively. BBR then regulates the *pacing_rate* between $1.25 \times BtlBw$ and $0.75 \times BtlBw$ to explore the achievable bandwidth and to drain the subsequently inflated queues. Finally, BBR sends *cwnd* packets at the inter-packet speed of *pacing_rate*.

The algorithm continues iteratively with the next round of network measurements. BBR transitions between different states of the state machine based on the observed *BtlBw*, *RTprop*, amount of packets in flight, etc. BBR periodically enters the *ProbeRTT* state to reduce its *cwnd* and drain the queue to reset itself.

BBR vs other congestion control algorithms: BBR differs from other major congestion control algorithms in the following aspects:

1) BBR does not explicitly respond to losses. Reno and Cubic regard packet loss as a congestion event, and subsequently reduce their *cwnd* value by a certain factor. Similarly, the delay-based algorithm TCP Vegas decreases its *cwnd* when observing increasing RTT. BBR, however, does not use explicit congestion signals to reduce *cwnd*. Rather, BBR decides the amount of packets to be sent based on past bandwidth and RTT measurements. Thus, in contrast to other event-driven algorithms, BBR is feedback driven.

2) BBR uses pacing_rate as the primary controller. Most congestion control algorithms, like Reno and Cubic, use *cwnd* to determine the number of packets in flight. However, *cwnd* does not directly control the sending rate, resulting in traffic bursts or an idle network [21]. To solve this issue, BBR uses pacing rate to control the inter-packet spacing. Implementing the pacing rate in TCP congestion control algorithms is known to have benefits for throughput and fairness [15].

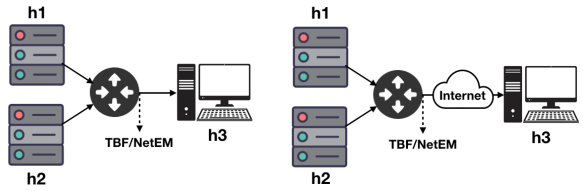
3) BBR actively avoids network congestion, whereas loss-based algorithms passively decrease their sending rate in response to congestion. BBR is designed to have low latency and high throughput by maintaining (typically) 2× BDP packets in flight. One BDP is budgeted for the network capacity, and the other is to deal with delayed/aggregated ACKs [20]. BBR thus avoids congestion by limiting the number of packets in flight. By contrast, Reno and Cubic keep increasing the packets in flight until the bottleneck buffer is full and a packet loss is detected. This is problematic when the bottleneck buffer is deep, in which case Reno and Cubic queue up too many packets in the buffer, causing bufferbloat [27].

While in principle BBR should outperform other TCP variants due to the above design decisions, a detailed empirical study is necessary to evaluate the performance of BBR.

3 EXPERIMENTAL SETUP

3.1 Testbeds

We use Mininet [6], LAN, and WAN networks for our experiments. Mininet is a network emulator which creates a virtual network running multiple hosts, links and switches on a single machine. For Mininet or LAN experiments, we use a simple dumbbell topology as shown in Figure 2(a). The minimum RTT between the various machines shown in the figure, $h1/h2$ and $h3$ in Mininet and LAN testbed is $40\mu s$.



(a) Mininet/LAN Testbed (b) WAN Testbed

Figure 2: Testbeds employed in our study.

Figure 2(b) shows our WAN testbed, where two senders are located at Stony Brook University (New York), and the receiver is located at Rutgers University (New Jersey). The minimum RTT between the sender and the receiver is 7ms. The network interfaces of all hosts have a 1Gbps peak bandwidth.

3.2 Setting the network parameters

We use Linux TC to configure different network conditions for both real and emulated network links. Specifically, we use TC-NetEm [29] to set link delay, and TC-tbf [14] to set link bandwidth and bottleneck buffer size. Note, we do *not* set network parameters on end hosts, since doing so can result in a negative interaction with TCP Small Queues [1, 4]. Instead, in LAN and WAN networks, we employ TC on a separate Linksys WRT1900ACS router (running OpenWRT Linux OS) between the end hosts. In Mininet, we set another node as a router between the end hosts. We investigate BBR in Linux 4.15, where the TCP layer can handle the pacing requirements of BBR, thus fq (Fair Queue) [2] is not needed [5].

TC-tbf: Figure 3 shows how Linux TC-tbf is used to limit the bandwidth. When the application wants to send data, the packets will first go through the IP stack to obtain headers. Then, the packets are enqueued to a queue named *qdisc* (queueing discipline) [11]. When we use TC-tbf to limit the bandwidth to, say *rate*, a bucket holding tokens is created. During each second, the system adds *rate* tokens into the bucket. The bucket size is pre-configured and can only hold limited tokens. If the bucket is already full, then the new tokens being added will be discarded. Each token allows 1 byte of data to go through the network. As a result, a queued packet of size L can move to the NIC only if there are L tokens in the bucket. Note that we can also configure the network buffer size by changing the *qdisc* length.

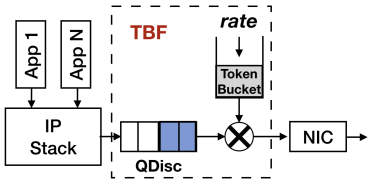


Figure 3: Using TC-tbf to limit network bandwidth.

4 EVALUATION

This section discusses our experimental results on evaluating BBR's performance in terms of goodput, packet loss, and fairness using our above-described testbeds.

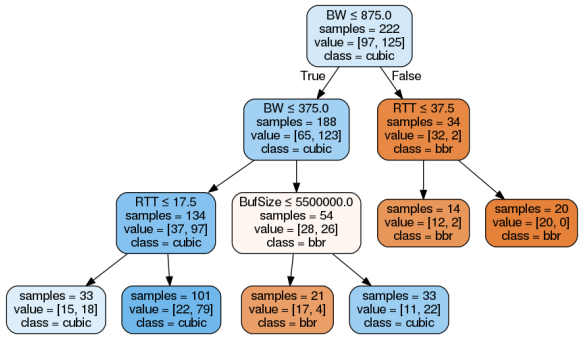


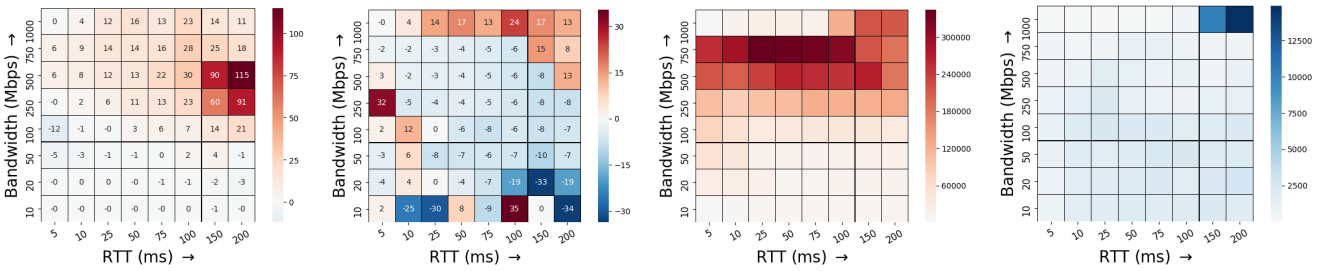
Figure 4: Decision tree for employing BBR versus Cubic under different network conditions.

4.1 BBR versus Cubic

Our first evaluation aims to answer a practical question with respect to BBR – "given a certain network condition, should we employ BBR to maximize goodput?" To answer this question, we empirically study the goodput of BBR and Cubic under 640 different configurations in our LAN testbed. We generate traffic using iPerf3 [12] in our LAN topology (Figure 2(a)) from $h1$ to $h3$. For each network configuration, we run the iPerf3 experiment 5 times, each for 60 seconds. On the LAN router we configure:
8 RTT values: 5, 10, 25, 50, 75, 100, 150, and 200ms;
8 BW values: 10, 20, 50, 100, 250, 500, 750, and 1000Mbps;
5 Buffer sizes: 0.1, 1, 10, 20, and 50 MB.
The range of our chosen parameters is based on values commonly employed in modern networks [30, 31].

4.1.1 Decision Tree. We summarize our LAN results using a *decision tree*, in Figure 4, which shows whether BBR or Cubic achieves higher goodput under different network conditions. The decision tree is generated by using the *DecisionTreeClassifier* API provided in Python3 scikit-learn package [13]. The input data consists of the goodput values of BBR and Cubic from all 640 LAN experiments. The median and mean classification accuracy for the decision tree is 81.9% and 81.3%, respectively, under 5-fold cross validation. Note that we set the TCP read and write memory to the maximum allowable value in Ubuntu 18.10, (2^{31}-1 bytes); this is done so that the data transfer is not limited by small memory sizes. Under the OS's default TCP memory sizes, the median and mean classification accuracy for the decision tree increase to 90.2% and 90.0%. This is because the Linux default TCP read and write memory sizes are usually quite small (usually 4MB to 6MB), which results in lower variation in bandwidth, thus reducing prediction outliers.

For each node in the tree in Figure 4, except for the leaf nodes, the first row shows the condition; nodes to the left are *True* and nodes to the right are *False* with respect to the condition. The second row shows how many cases (out of the 75% training data) fall under this node. The third row shows the number of cases that are classified as BBR or Cubic, based on the condition in the first row. Finally, the last row indicates the general decision for this node. If the node's color is orange, the decision is BBR; if the node's color is blue, then the decision is Cubic. The intensity of the color is determined using the Gini impurity, and indicates how confident we are in our decision. The leaf nodes provide the final classification output. To leverage the tree for a given network condition, we start at the root



(a) $GpGain^{bbr}_{cubic}$, *buffer*=100KB (b) $GpGain^{bbr}_{cubic}$, *buffer*=10MB (c) BBR's # retransmits, *buffer*=100KB (d) Cubic's # retransmits, *buffer*=100KB

Figure 5: Analysis of BBR and Cubic in terms of improvement in goodput and number of retransmissions under shallow and deep buffers for the LAN setup.

and traverse until we reach a leaf node; the decision in the leaf node is the predicted optimal choice (BBR versus Cubic).

The decision tree can be reasoned as follows. The left branch indicates the region where BDP is small and buffer is large, under which Cubic results in higher goodput. The right branch indicates the region where BDP is large and buffer size is small, under which BBR has higher goodput. The key to explaining these findings is that BBR maintains $\mathbf{2 \times BDP}$ number of packets in flight, and so the BDP number of packets are queued in the buffer (while the remaining BDP are on the wire). Now, if the buffer size is larger than *BDP*, which is the case in most of the left branch, then BBR is unable to fully utilize the buffer, resulting in likely inferior goodput. When the buffer size is small, both BBR and Cubic experience losses. However, Cubic responds to losses by significantly shrinking its *cwnd*, whereas BBR does not directly respond to the loss, resulting in higher goodput.

We also generated a decision tree using Mininet experiments. Results are qualitatively similar, with the Mininet-based decision tree providing an accuracy of about 90%.

4.1.2 Deconstructing the decision tree results. To further analyze the decision tree results based on the LAN experiments, we focus on two metrics: goodput and packet loss. Goodput characterizes how well the congestion control algorithm utilizes the network, while packet loss indicates the extent of network resource wastage incurred by the algorithm.

Goodput: We use the following metric to evaluate BBR's goodput percentage gain over Cubic:

$$GpGain^{bbr}_{cubic} = \frac{goodput|_{BBR} - goodput|_{Cubic}}{goodput|_{Cubic}} \times 100 \qquad (1)$$

We use *heatmaps* to visualize $GpGain^{bbr}_{cubic}$ for different network settings — for each metric, we show one heatmap for shallow buffer (100KB) and one for deep buffer (10MB). Note that we refer to 10MB as "deep" buffer since it is larger than most of the BDP values in our experiments. For example, a 500Mbps bandwidth and 100ms RTT results in 6.25MB BDP value, which is smaller than 10MB. In each heatmap, we show the metric value under different RTT and bandwidth settings. Red shaded regions and positive values indicate that BBR outperforms Cubic, whereas blue shaded regions and negative values indicate that Cubic outperforms BBR.

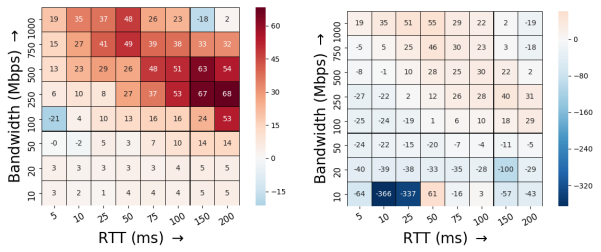
Figure 5(a) shows $GpGain^{bbr}_{cubic}$ under the shallow buffer (100KB). We observe that BBR outperforms Cubic when either bandwidth or RTT is high, i.e. the BDP is high. On the other hand, for a deep buffer (10MB), Figure 5(b) shows that Cubic has higher goodput except for very large bandwidth and RTT values. However, Cubic's goodput gain in deep buffers is not as high as BBR's gain in shallow buffers. For example, under 100KB buffer size, 200ms RTT, and 500Mbps bandwidth, Cubic's average goodput is 179.6Mbps, while BBR has a significantly higher average goodput of 386.0Mbps (115% improvement). However, for a 10MB buffer, Cubic only sees a maximum goodput improvement of 34%. We also tried a 50MB buffer size, but the results were similar to that for 10MB buffer size.

Loss: Although BBR sees significantly higher goodput values in shallow buffers, there is a caveat here – high number of losses. Figures 5(c) and 5(d) show the number of packet retransmissions for both BBR and Cubic in our LAN experiments under the 100KB bottleneck buffer size and different bandwidth and RTT values; note that retransmissions are initiated after a loss is detected [23]. We see that BBR often incurs 10× more retransmissions than Cubic. This is largely because BBR sets *cwnd_gain* to 2 most of the time, thus requiring a buffer size of at least BDP to queue its outstanding requests in flight; when the buffer size is smaller than BDP, BBR will continually have losses. On the other hand, although Cubic also hits the buffer capacity for a shallow buffer, it responds by lowering its *cwnd*, thus avoiding continued losses. In terms of losses, for 100KB buffer size, the average loss percentage for BBR and Cubic is 10.1% and 0.9%, respectively. For 10MB buffer size, the corresponding loss percentage for BBR and Cubic is 0.8% and 1.3%, respectively.

When the bottleneck buffer size increases, we find that the number of retransmissions decreases significantly for both BBR and Cubic. For example, in our 25ms RTT and 500Mbps bandwidth LAN experiment, when we increase the bottleneck buffer size from 100KB to 10MB, BBR and Cubic's retransmissions decrease from 235798 to 0 and 1649 to 471, respectively. To better understand this non-trivial loss behavior, we further analyze the relationship between goodput and losses in the next subsection.

Latency: To investigate TCP latency, we now consider finite flow sizes. Specifically, we use iPerf3 to generate 10MB and 100MB flows under the same 640 network configurations as for our previous set of experiments in the LAN testbed. We use the following metric to evaluate BBR's latency improvement percentage over Cubic:

$$LatDec^{bbr}_{cubic} = \frac{latency|_{Cubic} - latency|_{BBR}}{latency|_{Cubic}} \times 100 \qquad (2)$$



(a) $LatDec^{bbr}_{cubic}$, *buffer*=100KB (b) $LatDec^{bbr}_{cubic}$, *buffer*=10MB

Figure 6: BBR's latency decrease compared to Cubic under shallow and deep buffers for a finite flow size (100MB).

Our latency results for 100MB flows are shown in Figure 6, which agrees with with our goodput results in Figure 5. We observe in Figure 6(a) that in the shallow buffer case, BBR typically has lower latency. For a deep buffer, Figure 6(b) shows that Cubic has lower latency when the BDP is small. We also experimented with 10MB flows, and found that BBR has lower latency in almost all cases.

4.2 BBR's goodput vs packet losses

In the current Linux implementation, BBR does not actively react to packet losses. However, we observe in our experiments that BBR exhibits an abrupt drop in goodput when the loss rate becomes very high, about 20% in our case. This suggests a "***cliff point***" in loss rates beyond which BBR inadvertently reacts to losses.

On further analysis, we find that the cliff point has a close relationship with BBR's *pacing_rate* parameter that determines its probing capabilities (see Section 2). If the packet loss probability is p, then during bandwidth probing, BBR paces at $pacing_rate \times BW$. However, due to losses, its effective pacing rate is $pacing_rate \times BW \times (1-p)$. Thus, if this value is less than the bandwidth, BBR will not probe for additional capacity, and will in fact infer a lower capacity due to losses. We determine the cliff point by solving:

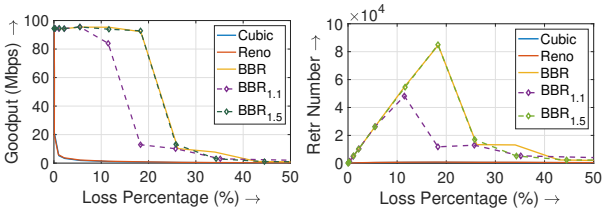
$$pacing_gain \times BW \times (1-p) = BW \quad (3)$$

Consequently, the cliff point is $p = 1 - 1/pacing_gain$. In current implementations, the maximum *pacing_gain* is 1.25, so the cliff point should be at $p = 0.2$, or 20% loss rate.

Validation of cliff points: We validate our above analysis by varying the maximum *pacing_gain* value in BBR's source code, and conducting experiments in both our WAN and Mininet testbeds. We experiment with two different *pacing_gain* values, in addition to the default value of 1.25: *pacing_gain* = 1.1, denoted as *BBR_1.1*, and *pacing_gain* = 1.5, denoted as *BBR_1.5*. Via Eq. (3), we expect the cliff point of BBR_1.1 and BBR_1.5 to be at 9% and 33%, respectively.

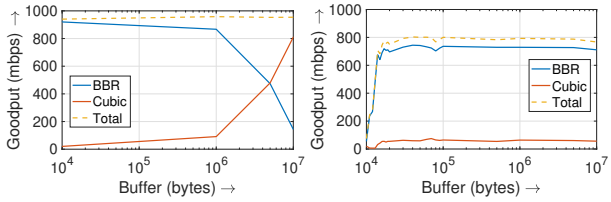
Figure 7 shows how different TCP algorithms react to packet losses in the Mininet testbed (100Mbps bandwidth and 25ms RTT). We vary loss rate from 0 to 50% using TC-NetEm to emulate lossy networks; TC-NetEm introduces random losses, which are common in WiFi and RED routers [26]. For each loss rate, we run iPerf3 experiments for 60s with different congestion control algorithms.

Figure 7(a) shows how goodput is affected by loss rates. We see that for loss-based algorithms, Reno and Cubic, goodput decreases significantly even under moderate loss rates since they proactively reduce *cwnd* when encountering losses. On the other hand, BBR and BBR_1.1 exhibit a drop in goodput around 20% and 9% loss rates respectively, validating our prior cliff point analysis. However,



(a) Goodput vs loss rate. (b) Retransmits vs loss rate.

Figure 7: Mininet: 100Mbps *BW*, 25ms *RTT*, 10MB buffer.



(a) Mininet results. (b) WAN results.

Figure 8: BBR and Cubic's bandwidth share under 1Gbps *BW*, 20ms *RTT*, and different buffer sizes.

BBR_1.5 exhibits its cliff point much before the predicted 33% loss rate. This is because BBR considers a > 20% loss rate a signal of policing [8], and so uses the long-term average bandwidth instead of updating its maximum bandwidth estimate *BtlBw*.

Figure 7(b) shows the number of retransmissions under different loss rates. We see that BBR reaches its peak retransmissions around the loss rate cliff point. This is because, before this cliff point, the BBR goodput is stable but the loss rate is increasing, resulting in increasing number of retransmits. However, after the cliff point is reached, BBR's goodput decreases, resulting in fewer packets sent and, subsequently, fewer retransmissions. We also conducted experiments in WAN to verify this behavior. We obtained similar results as Figure 7(b), thus confirming our cliff point analysis.

4.3 Analyzing BBR's fairness

Given BBR's aggressive behavior, evidenced by its high retransmissions, we now investigate the fairness of BBR when it coexists with other flows in our Mininet and WAN testbeds.

Mininet results: For our Mininet testbed setup for fairness (Figure 2(a)), nodes *h*1 and *h*2 send iPerf3 traffic to *h*3 using BBR and Cubic, respectively. On the link from the router to *h*3, we add a 20ms network delay and vary the buffer size between 10KB and 100MB. Under each buffer size, the iPerf3 experiment lasts for 60s.

Figure 8(a) shows the results of our Mininet experiments. We see that the bandwidth share of BBR and Cubic depends on the bottleneck buffer size. For a small buffer size (10KB), BBR utilizes 94% of the network goodput value. When the buffer size is large (10MB), Cubic utilizes 3× more bandwidth than BBR. Under moderate buffer sizes (~ 5MB), BBR and Cubic evenly share the bandwidth.

In terms of retransmissions, we find that BBR has a high retransmission rate when coexisting with Cubic flows in shallow buffers. Table 1 shows the number of retransmissions for BBR and Cubic under different buffer sizes in our Mininet experiments. Clearly, BBR has orders of magnitude higher retransmits than Cubic for small buffer sizes. For example, under 100KB buffer size, BBR has

Buffer (bytes)	1e4	1e5	1e6	5e6	1e6	5e7	1e8
BBR Retr#	26746	305029	68741	1324	204	0	0
Cubic Retr#	908	1398	3987	1145	794	7	16

Table 1: Number of retransmissions for BBR and Cubic for different buffer sizes when they co-exist under the 1Gbps bandwidth and 20ms RTT Mininet network.

200× more retransmits than Cubic. In deep buffers, BBR's retransmissions drops to zero since its packets in flight (*cwnd*) is now much smaller than the buffer capacity.

WAN results: We also conduct fairness experiments in our WAN network. For configuring the network conditions, we apply the same Mininet parameters to our WAN testbed. Our WAN results in Figure 8(b) show a different behavior – even at high buffer sizes, Cubic's bandwidth share does not increase, unlike that in Figure 8(a). This results suggests the existence of a shallow buffer on the WAN between our router and the receiver $h3$. In Figure 8(b), the goodput of BBR and Cubic stabilizes when our router buffer size reaches 20KB; this indicates that the bottleneck buffer in the wild in our WAN setup is around 20KB. In terms of retransmissions, while we also see a large number of retransmits for BBR in shallow buffers, we find that BBR's retransmits stabilize at around 500 packets/min as we increase our router buffer size beyond 20KB; this further confirms the 20KB bottleneck buffer in the wild for our WAN setup.

Reason for using Mininet: While we show similar results in Section 4.1 for the LAN and Mininet testbeds, Mininet is more flexible than the LAN testbed in certain scenarios. In Section 4.3, we use Mininet to create a star topology for the fairness experiment – a router connecting 3 nodes. This was not possible in our LAN testbed given we only have two servers connected to a router. Also, we use Mininet in Section 4.2 and Section 4.3 due to its convenience and scalability to validate/reinforce our in-the-wild WAN results.

5 RELATED WORK

BBR's design was first published in a 2016 ACM article [20]. Since 2016, several BBR updates have been given at IETF conferences [7, 9, 21], in addition to internet drafts [3, 22].

Most of the prior work on evaluating the performance of BBR has focused on BBR's fairness. Hock et al. [30] study how BBR coexists with Cubic under shallow and deep buffers. They find that BBR gets a bigger share of the bandwidth in small buffers while Cubic gets a larger share in deep buffers. Ma et al. [34] show that the persistent queue that develops on the bottleneck buffer contributes to BBR's unfairness. However, these works either experiment with very few buffer sizes or only using a single testbed. Our paper not only analyzes BBR's fairness for a range of buffer sizes (10KB – 100MB) under multiple testbeds, but also highlights the non-trivial fairness behavior in our WAN setting (see Section 4.3).

The high loss rate under BBR has been discussed in some recent papers [30, 32, 37], but these works do not investigate the reasons (such as the cliff point, see Section 4.2) behind this observation.

There have also been some works that investigate BBR's performance for specific scenarios. Zhong et al. [39] investigate BBR in a mobile network, and analyze the impact of delayed ACKs on BBR's performance. Atxutegi et al. [18] study BBR's performance in live mobile networks, and contrast it with TCP NewReno and Cubic. Our work focuses on BBR performance in different wired settings, including LAN and WAN, in addition to Mininet.

6 LIMITATIONS AND FUTURE WORK

We now discuss the limitations of our study. First, all of our experimental testbeds, including LAN, Mininet, and WAN, use a simple dumbbell topology in order to easily control the bottleneck buffer size. However, the Internet consists of more complicated networks. For example, BBR has been used in Google's B4 network as well as in Youtube video servers [20]. We plan to extend our study to such real-world scenarios as part of the future work.

Second, our experiments thus far consider at most two concurrent TCP flows. Also, since the LAN and the Mininet testbeds are fully under our control, we deliberately eliminate the irrelevant (background) traffic in our experiments to focus on the fairness performance comparison between BBR and Cubic. However, in real networks, temporary flows can enter and leave the network at different times, which might affect the results. We plan to investigate the impact of more competing flows in our future work.

Third, this paper primarily focuses on empirical measurements. We have not investigated how we can use our empirical findings to optimize the performance of BBR. In our ongoing work, we are investigating the design flaws of BBR with the eventual goal of enhancing the design of BBR to improve its performance. Specifically, we are working on mitigating BBR's high retransmission and unfairness issues.

Finally, the key issues revealed by our study, such as cliff points, high retransmissions, and unfairness, are inherent in the current version of BBR. It is not entirely obvious whether or not these issues will persist in future versions of BBR, though there is some online discussion [9, 10] about addressing unfairness in subsequent versions of BBR. Nonetheless, the empirical findings and root cause analysis presented in this paper can help the community to identify and solve performance issues as BBR continues to evolve.

7 CONCLUSION

Despite the excitement around BBR, there is a dearth of studies that evaluate the performance of BBR on multiple real-world testbeds and across a range of parameter settings, especially studies that investigate *why* BBR performs the way it does. This paper conducts over 600 experiments under both emulated and real-world testbeds, and analyzes the network conditions under which BBR outperforms contemporary algorithms. Our analysis reveals that it is the relative difference between the bottleneck buffer size and BDP that typically dictates when BBR performs well. In fact, this finding also extends to BBR's unfair behavior when it coexists with Cubic; however, in such cases, we find that when BBR performs well, it can be very unfair to competing flows. In addition, our study reveals the existence of a "cliff point" in loss rate, beyond which BBR's goodput drops abruptly. Our analysis reveals that the *pacing_gain* parameter in BBR is partly to blame for this behavior.

ACKNOWLEDGMENT

This work was supported by NSF grants 1566260, 1717588, 1750109, and 1909356.

REFERENCES

[1] TCP Small Queues. https://lwn.net/Articles/507065/, 2012.

[2] Fair Queue Traffic Policing. http://man7.org/linux/man-pages/man8/tc-fq.8.html, 2015.

[3] Delivery Rate Estimation. https://tools.ietf.org/html/draft-cheng-iccrg-delivery-rate-estimation-00, 2017.

[4] TC Configure Point. https://groups.google.com/d/topic/bbr-dev/8LYkNt17V_8, 2017.

[5] BBR Quick Start. https://github.com/google/bbr/blob/master/Documentation/bbr-quick-start.md, 2018.

[6] Mininet - An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/, 2018.

[7] BBR congestion control: IETF 102 Update: BBR Startup. https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-bbr-startup-behavior-01, 2019.

[8] BBR source code. https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/tree/net/ipv4/tcp_bbr.c, 2019.

[9] BBR v2: A Model-based Congestion Control: IETF 104 Update. https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00, 2019.

[10] BBR v2: A Model-based Congestion Control: IETF 105 Update. https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00, 2019.

[11] Components of Linux Traffic Control. http://tldp.org/HOWTO/Traffic-Control-HOWTO/components.html, 2019.

[12] iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/iperf-doc.php, 2019.

[13] Machine Learning in Python – scikit-learn. https://scikit-learn.org/stable/index.html, 2019.

[14] Token Bucket Filter. https://linux.die.net/man/8/tc-tbf, 2019.

[15] Aggarwal, A., Savage, S., and Anderson, T. Understanding the performance of tcp pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)* (2000), vol. 3, IEEE, pp. 1157–1165.

[16] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). *ACM SIGCOMM computer communication review 41*, 4 (2011), 63–74.

[17] Arun, V., and Balakrishnan, H. Copa: Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)* (2018), pp. 329–342.

[18] Atxutegi, E., Liberal, F., Haile, H. K., Grinnemo, K.-J., Brunstrom, A., and Arvidsson, A. On the use of tcp bbr in cellular networks. *IEEE Communications Magazine 56*, 3 (2018), 172–179.

[19] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. *TCP Vegas: New techniques for congestion detection and avoidance*, vol. 24. ACM, 1994.

[20] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. Bbr: Congestion-based congestion control.

[21] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., Swett, I., Iyengar, J., Vasiliev, V., and Jacobson, V. Bbr congestion control: Ietf 99 update. In *Presentation in ICCRG at IETF 99th meeting, Jul* (2017).

[22] Cardwell, N., Cheng, Y., Yeganeh, S. H., and Jacobson, V. Bbr congestion control draft-cardwell-iccrg-bbr-congestion-control-00. *Google, Inc Std.* (2017).

[23] Dempsey, B. J., Liebeherr, J., and Weaver, A. C. On retransmission-based error control for continuous media traffic in packet-switching networks. *Computer Networks and ISDN Systems 28*, 5 (1996), 719–736.

[24] Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. {PCC}: Re-architecting congestion control for consistent high performance. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (2015), pp. 395–408.

[25] Fall, K., and Floyd, S. Simulation-based comparisons of tahoe, reno and sack tcp. *ACM SIGCOMM Computer Communication Review 26*, 3 (1996), 5–21.

[26] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 4 (1993), 397–413.

[27] Gettys, J. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 3 (2011), 96.

[28] Ha, S., Rhee, I., and Xu, L. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review 42*, 5 (2008), 64–74.

[29] Hemminger, S., et al. Network emulation with netem. In *Linux conf au* (2005), pp. 18–23.

[30] Hock, M., Bless, R., and Zitterbart, M. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)* (2017), IEEE, pp. 1–10.

[31] Huffaker, B., Fomenkov, M., Plummer, D. J., Moore, D., Claffy, K., et al. Distance metrics in the internet. In *Proc. of IEEE international telecommunications symposium (ITS)* (2002).

[32] Hurtig, P., Haile, H., Grinnemo, K.-J., Brunstrom, A., Atxutegi, E., Liberal, F., and Arvidsson, Å. Impact of tcp bbr on cubic traffic: A mixed workload evaluation. In *2018 30th International Teletraffic Congress (ITC 30)* (2018), vol. 1, IEEE, pp. 218–226.

[33] Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications* (1979), vol. 43, pp. 1–43.

[34] Ma, S., Jiang, J., Wang, W., and Li, B. Towards rtt fairness of congestion-based congestion control. *arXiv preprint arXiv:1706.09115* (2017).

[35] Ramakrishnan, K., and Floyd, S. A proposal to add explicit congestion notification (ecn) to ip. Tech. rep., 1998.

[36] Schapira, M., and Winstein, K. Congestion-control throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (2017), ACM, pp. 122–128.

[37] Scholz, D., Jaeger, B., Schwaighofer, L., Raumer, D., Geyer, F., and Carle, G. Towards a deeper understanding of tcp bbr congestion control. In *IFIP Networking* (2018), pp. 109–117.

[38] Yan, F. Y., Ma, J., Hill, G. D., Raghavan, D., Wahby, R. S., Levis, P., and Winstein, K. Pantheon: the training ground for internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 731–743.

[39] Zhong, Z., Hamchaoui, I., Khatoun, R., and Serhrouchni, A. Performance evaluation of cqic and tcp bbr in mobile network. In *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)* (2018), IEEE, pp. 1–5.