An Emulation-based Evaluation of TCP BBRv2 Alpha for Wired Broadband

Elie F. Kfoury[a], Jose Gomez[a], Jorge Crichigno[a], Elias Bou-Harb[b]

[a]*Integrated Information Technology Department, University of South Carolina, USA.*
[b]*The Cyber Center For Security and Analytics, University of Texas at San Antonio, USA.*

Abstract

Google published the first release of the Bottleneck Bandwidth and Round-trip Time (BBR) congestion control algorithm in 2016. Since then, BBR has gained a widespread attention due to its ability to operate efficiently in the presence of packet loss and in scenarios where routers are equipped with small buffers. These characteristics were not attainable with traditional loss-based congestion control algorithms such as CUBIC and Reno. BBRv2 is a recent congestion control algorithm proposed as an improvement to its predecessor, BBRv1. Preliminary work suggests that BBRv2 maintains the high throughput and the bounded queueing delay properties of BBRv1. However, the literature has been missing an evaluation of BBRv2 under different network conditions.

This paper presents an experimental evaluation of BBRv2 Alpha (v2alpha-2019-07-28) on Mininet, considering alternative active queue management (AQM) algorithms, routers with different buffer sizes, variable packet loss rates and round-trip times (RTTs), and small and large numbers of TCP flows. Emulation results show that BBRv2 tolerates much higher random packet loss rates than loss-based algorithms but slightly lower than BBRv1. The results also confirm that BBRv2 has better coexistence with loss-based algorithms and lower retransmission rates than BBRv1, and that it produces low queuing delay even with large buffers. When a Tail Drop policy is used with large buffers, an unfair bandwidth allocation is observed among BBRv2 and CUBIC flows. Such unfairness can be reduced by using advanced AQM schemes such as FQ-CoDel and CAKE. Regarding fairness among BBRv2 flows, results show that using small buffers produces better fairness, without compromising high throughput and link utilization. This observation applies to BBRv1 flows as well, which suggests that rate-based model-based algorithms work better with small buffers. BBRv2 also enhances the coexistence of flows with different RTTs, mitigating the RTT unfairness problem noted in BBRv1. Lastly, the paper presents the advantages of using TCP pacing with a loss-based algorithm, when the rate is manually configured a priori. Future algorithms could set the pacing rate using explicit feedback generated by modern programmable switches.

Keywords: Active queue management (AQM), bandwidth-delay product (BDP), Bottleneck Bandwidth and Round-trip Time (BBR), BBRv2, congestion control, Controlled Delay (CoDel), CUBIC, round-trip time unfairness, router's buffer size.

1. Introduction

The Transmission Control Protocol (TCP) [1] has been the standard transport protocol to establish a reliable connection between end devices. One of the key functions of TCP is congestion control, which throttles a sender node when the network is congested and attempts to limit the TCP connection to its fair share of network bandwidth [2].

The principles of window-based congestion control were described in late 1980s by Jacobson and Karels [3]. Since then, many enhancements have been proposed [4]. Traditional congestion control algorithms rely on the additive increase multiplicative decrease (AIMD) control law [5] to establish the size of the congestion window, which in turn regulates the sending rate. These algorithms include Reno and its multiple variants such as CUBIC [6] (the default algorithm used in multiple Linux distributions and in recent versions of Windows and MacOS), HTCP [7], and others. Most traditional algorithms are loss-based, because a packet loss is used as a binary signal of congestion. The well-known TCP macroscopic model [8] demonstrated that traditional algorithms cannot achieve high throughput in the presence of even a modest packet loss rate and large round-trip times (RTTs); the model states that the throughput of a TCP connection is inversely proportional to the RTT and the square root of the packet loss rate. Essentially, the time needed for TCP to recover from a packet loss is significant, as the congestion window is only increased by approximately one Maximum Segment Size (MSS) every RTT [9].

In a TCP connection, congestion occurs at the bottleneck link. Usually, the router with the bottleneck link multiplexes packets received from multiple input links to an output link. When the sum of arrival rates exceeds the capacity of the output link, the output queue grows large and the router's buffer is eventually exhausted, causing packet drops. Loss-based congestion control algorithms only indirectly infer congestion.

The router's buffer plays an important role in absorbing traffic fluctuations, which are present even in the absence of congestion. The router avoids losses by momentarily buffering

Email addresses: ekfoury@email.sc.edu (Elie F. Kfoury), gomezgaj@email.sc.edu (Jose Gomez), jcrichigno@cec.sc.edu (Jorge Crichigno), elias.bouharb@utsa.edu (Elias Bou-Harb)

packets as transitory bursts dissipate. When the router has a small buffer, packets may be dropped even though the link may be largely uncongested. With traditional loss-based algorithms such as Reno and CUBIC, packet losses lead to a low throughput as a consequence of the AIMD rule. While traditional loss-based algorithms were adequate in the past for applications requiring low throughput (e.g., Telnet, FTP), they face limitations for applications demanding high throughput, such as high-resolution media streaming, grid computing / Globus' gridFTP [10], and big science data transfers [9].

The Bottleneck Bandwidth and Round-Trip Time (BBRv1) algorithm [11] has been the first scheme to estimate the bottleneck bandwidth. BBRv1 is a rate-based congestion control algorithm that periodically estimates the bottleneck bandwidth and does not follow the AIMD rule [11]. It uses pacing to set the sending rate to the estimated bottleneck bandwidth. The pacing technique spaces out or paces packets at the sender node, spreading them over time. This approach is a departure from the traditional loss-based algorithms, where the sending rate is established by the size of the congestion window, and the sender node may send packets in bursts. Thus, traditional algorithms rely on routers to perform buffering to absorb packet bursts.

While BBRv1 has improved the throughput of a TCP connection, the literature [12] has reported some behavioral issues, such as unfairness and high retransmission rates. Recently, BBR version 2 (BBRv2) [13] has been proposed to address these limitations. BBRv1 uses two estimates to establish the sending rate of a connection: the bottleneck bandwidth and the RTT of the connection. BBRv2 enhances this approach by also incorporating Explicit Congestion Notification (ECN) and estimating the packet loss rate to establish the sending rate.

1.1. Contribution and Findings

Although some preliminary evaluations of BBRv2 have been reported [13, 14], the literature is still missing a comprehensive evaluation of it. Thus, this paper presents an experimental evaluation of BBRv2 under various scenarios designed to test the features of BBRv2. Scenarios include alternative active queue management (AQM) algorithms, routers with different buffer sizes, variable packet loss rates and RTTs, and small and large numbers of TCP flows. Experiments are conducted with Mininet using a real protocol stack in Linux, and provide the following findings:

1. BBRv2 tolerates much higher random packet loss rates than CUBIC but slightly lower than BBRv1.

2. In networks with small buffers, BBRv2 produces high Jain's fairness index [15] while maintaining high throughput and link utilization. On the other hand, in networks with large buffers, the fairness index is significantly reduced. Thus, BBRv2 works better with small buffers. This observation also applies to BBRv1 and indicates that small buffers reduce the delay in the TCP control loop, which produces better outcomes.

3. The retransmission rate of BBRv2 is very low when compared with that of BBRv1. Although the retransmission rate of CUBIC is lower than that of BBRv2, the throughput is also lower. Additionally, the high retransmission rate noted in BBRv1 can be reduced by using an AQM algorithm such as Flow Queue Controlled Delay (FQ-CoDel) [16].

4. The excessive delay or bufferbloat problem [17], noted in traditional congestion control algorithms when the buffer size is large, is not manifested in BBRv2. Instead, BBRv2 exhibits a queueing delay that is loosely independent of the buffer size, even when routers implement a simple Tail Drop policy.

5. BBRv2 shows better coexistence with CUBIC than BBRv1, measured by the fairness index. Additionally, as the number of flows in the network increases, the aggregate throughput observed in BBRv2 is more consistent than that of BBRv1, independently of the buffer size. Thus, selecting the correct buffer size is no longer essential when using BBRv2.

6. BBRv2 enhances the coexistence of flows with different RTTs, mitigating the RTT unfairness problem observed in BBRv1. Such problem can be also corrected by using FQ-CoDel [16], regardless of the buffer size.

7. A simple Tail Drop policy leads to unfair bandwidth allocation among BBRv2 and CUBIC flows, in particular when the buffer size is large. On the other hand, advanced AQM policies such as FQ-CoDel [16] and Common Applications Kept Enhanced (CAKE) [18] produce fair bandwidth allocations.

8. Pacing [19] helps improve the fairness and the performance of CUBIC when the rate is known a priori. Although dynamically applying pacing with CUBIC to predefined rates was solved in newer Linux versions, results suggest that future congestion control algorithms could consider the use of pacing and more explicit feedback signals from routers. As new-generation programmable switches are now becoming accessible, network operators could implement in-band network telemetry and other dataplane features to provide explicit feedback control to adjust the pacing rate.

The rest of the paper is organized as follows: Section 2 presents a brief background on BBRv2 and the AQM algorithms used in the experiments. Section 3 discusses related work. Section 4 presents the experimental setup and the reported metrics, and Section 5 presents the evaluation results. Section 6 concludes the paper.

2. Background

2.1. BBRv2

BBRv2 is a rate-based, model-based congestion control algorithm. Fig. 1 depicts its high-level architectural design. The algorithm measures the bandwidth, the RTT, the packet loss rate, and the ECN mark rate. The measurements are used to estimate the bandwidth-delay product (BDP) and to model the end-to-end path across the network, referred to as the network

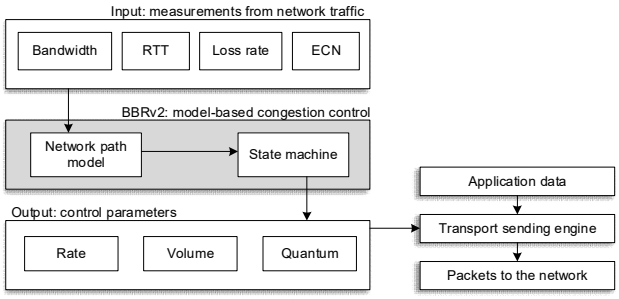


Figure 1: High-level architectural design of BBRv2.

path model. Based on the current network path model, the algorithm transitions among states of a finite state machine, which includes probing for bandwidth and RTT. The state machine generates three control parameters: rate, volume, and quantum. The rate is the pacing rate that will be used by the sender. The volume is the amount of data or bits that can be inside the path as they propagate from the sender to the receiver, also referred to as in-flight volume. The quantum is the maximum burst size that can be generated by the sender. The control parameters are used by the transport protocol sending engine, which segments the application data into bursts of quantum size before sending the data to the network as packets. Although packet loss rate and ECN signals are inputs of the model, BBRv2 does not simply always apply a multiplicative decrease for every round trip where packet loss occurs or an ECN signal arrives.

BBRv2 maintains short-term and long-term estimates of the bottleneck bandwidth and maximum volume of in-flight data. This is analogous to CUBIC, which has a short-term slow start threshold estimate (`ssthresh`) and long term maximum congestion window (`W_max`). BBRv2 spends most of the connection time in a phase where it can quickly maintain flow balance (i.e., the sending rate adapts to the new bottleneck bandwidth / bandwidth-delay product), while also attempting to leave unutilized capacity in the bottleneck link. This spare capacity, referred to as *headroom* capacity, enables entering flows to grab bandwidth. Therefore, BBRv2 maintains a short-term `bw_lo` and `inflight_lo` estimates to bound the behavior using the last delivery process (loss, ECN, etc.). The basic intuition in this case is to maintain reasonable queueing levels at the bottleneck bandwidth by inspecting the recent delivery process.

BBRv2 also periodically probes for additional bandwidth beyond the flow balance level. It maintains a long-term tuple (`bw_hi` and `inflight_hi`) that estimate the maximum bandwidth and in-flight volume that can be achieved, consistent with the network's desired loss rate and ECN mark rate.

BBRv2 Research Focus and Goals. According to the IETF proposal [13], BBRv2 has the following goals:

1. Coexisting with loss-based congestion control algorithms sharing the same bottleneck link. The literature [12], [20] has shown that when the router's buffer size is below 1.5BDP, BBRv1 causes constant packet losses in the bottleneck link, leading to successive multiplicative decreases in flows using loss-based algorithms. When the router's buffer size exceeds 3BDP, loss-based algorithms steadily claim more bandwidth in the bottleneck link, leading to a decrease in the bandwidth allocated to BBRv1. BBRv2 aims at improving the fairness index [21] when competing with loss-based algorithms, in particular in the above scenarios.

2. Avoiding the bufferbloat problem. BBRv2 aims at having no additional delay when a router's buffer is large (up to 100BDP). At the same time, BBRv2 aims at having low packet loss rate when a router's buffer is small.

3. Minimizing the time to reach an equilibrium point where competing flows fairly share the bandwidth. BBRv2 attempts to solve the RTT bias problem observed in BBRv1: when a router's buffer is large, flows with large RTTs have large bandwidth-delay product / in-flight volume and use much of the buffer, thus claiming more bandwidth than flows with small RTTs.

4. Reducing the variation of the throughput by making the "PROBE_RTT" phase less drastic.

Life Cycle of a BBRv2 Flow. The state machine of BBRv2 is similar to that of BBRv1, alternating between cycles that probe for bandwidth and for round-trip time. A simplified life cycle of a flow is shown in Fig. 2. Initially, a flow starts at the `STARTUP` phase (a), which is similar to the traditional slow-start phase. During the `STARTUP` phase, the algorithm attempts to rapidly discover the bottleneck bandwidth by doubling its sending rate every round-trip time. If the packet loss rate or ECN mark rate increases beyond their respective thresholds, the `inflight_hi` is set as an estimation of the maximum in-flight volume. The flow exits the `STARTUP` phase when continuous bandwidth probes reach either a stable value (*plateau*) or the `inflight_hi` is set. The flow then enters the `DRAIN` phase (b) which attempts to drain the excessive in-flight bits and the queue, which may have been formed at the bottleneck link during the previous phase, by setting a low pacing (sending) rate. The flow exits this phase when the in-flight volume is at or below the estimated bandwidth-delay product.

The flow spends most of the remainder of its life into the `CRUISE` phase (c), where the sending rate constantly adapts to control queueing levels. The {bw, inflight}_lo tuple is updated every round-trip time, using also packet loss and ECN signals. Afterwards, during the `PROBE_BW:REFILL` phase (d), the flow probes for additional bandwidth and in-flight capacity. The goal of this phase is to increase the in-flight volume (which was previously reduced) by sending at the estimated capacity (bottleneck bandwidth) for one round-trip time. Note that the expectation in this phase is that queues will not be formed, as the sending rate is not beyond the estimated bottleneck bandwidth. Also, if the routers have small buffers, packet losses that are not due congestion may occur. To avoid a reduction in the sending rate, which would negatively impact the throughput, the algorithm tolerates up to `loss_thresh` losses every round-trip time.

During bandwidth probing `PROBE_BW:UP` phase, if `inflight_hi` is fully utilized then it is increased by an

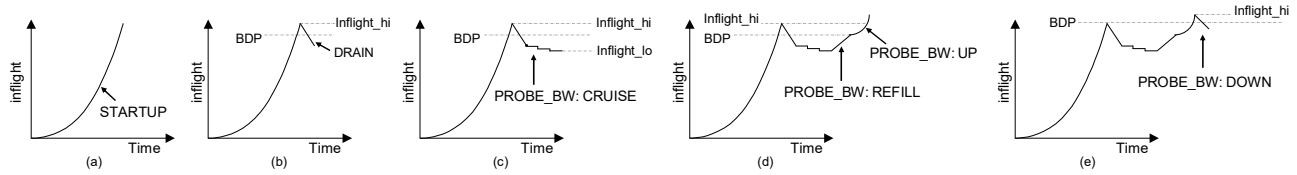


Figure 2: Life cycle of a BBRv2 flow and its five phases: (a) STARTUP, (b) DRAIN, (c) PROBE_BW: CRUISE, (d) PROBE_BW: REFILL and PROBE_BW: UP, and (e) PROBE_BW: DOWN.

amount that grows exponentially per round (1, 2, 4, 8 ... packets). On the other hand, if the loss rate or the ECN mark rate is too high, then the `inflight_hi` is reduced to the estimated maximum safe in-flight volume. The flow exits the `PROBE_BW:UP` phase when the `inflight_hi` is set, or the estimated queue is high enough (in-flight volume $> 1.25 \cdot$ estimated_BDP). Finally, the flow enters the `PROBE_BW:DOWN` phase (e) to drain the queue recently created, and leaves unused headroom. The flow exits this phase when both of the following conditions are met: the in-flight volume is below a headroom margin from the `inflight_hi` and the in-flight volume is at or below the estimated bandwidth-delay product.

2.2. Active Queue Management and Bufferbloat

AQM encompasses a set of algorithms to reduce network congestion and queueing delays by preventing buffers from remaining full. AQM policies help to mitigate the bufferbloat problem [17, 22–24], which not only excessively increases the latency but also decreases the aggregate network throughput and increases the jitter.

Several schemes have been proposed to mitigate the bufferbloat problem, such as regulating the sending rate at end devices (e.g., BBRv1/BBRv2) and implementing AQM policies at routers to signal imminent congestion. AQM policies include Controlled Delay (CoDel) [25], Flow Queue CoDel (FQ-CoDel) [16] and Common Applications Kept Enhanced (CAKE) [18].

FQ-CoDel. CoDel [25] is a scheduling algorithm that prevents bufferbloat by limiting the queue size. CoDel measures the packet delay in the queue, from ingress to egress through timestamps. It recognizes two types of queues: 1) good queue, which has a small sojourn time; and 2) bad queue, which has a high sojourn time. When the bad queue is manifested, the bufferbloat problem emerges. CoDel does not rely on estimating RTT or link rate; instead it uses the actual delay experienced by each packet as a criterion to determine if a queue is building up. Thus, CoDel can adapt dynamically to different link rates without impacting the performance. FQ-CoDel [16] combines fair queueing scheduling and CoDel, attempting to keep queues short and to provide flow isolation. The queue used for a flow is selected by using a hashing function that maps the packets' flow to the selected queue. The scheduler selects which queue to be dequeued based on a deficit round-robin mechanism.

CAKE. This scheduling algorithm is an extension of FQ-CoDel, designed for home gateways [18]. CAKE uses an eight-way set-associative hash instead of a direct hash function as in FQ-CoDel. CAKE provides bandwidth shaping with overhead compensation for various link layers, differentiated service handling, and ACK filtering.

3. Related Work

Previous studies focused on BBRv1 and provided an in-depth analysis of its behavior under various network conditions. Scholz et al. [26] explored features such as bottleneck bandwidth overestimation, inter-protocol behavior with CUBIC, inter-flow synchronization, RTT unfairness, and others. Ma et al. [27] conducted an experimental evaluation and analysis of the fairness of BBRv1. The researchers focused on the RTT unfairness and proposed BBQ, a solution that provides better RTT fairness without deviating from Kleinrock's optimal operating point. Fejes et al. [28] performed experiments that combined different AQMs and congestion control algorithms to analyze fairness and coexistence of flows. The authors concluded that fairness is poor as the assumptions used during the development of AQMs do not typically hold in real networks. While the authors tested some congestion control algorithms and AQMs, they did not report results with FQ-CoDel [16] and Cake [18]. Their experiments did not consider metrics and network conditions such as RTT, queueing delays, flow completion time (FCT), retransmission rates, and RTT unfairness. Tahiliani et al. [29] studied the differences in perceiving congestion, from the viewpoint of end systems, between networks that do not provide explicit feedback and AQMs, and those that provide feedback and AQMs. They demonstrated that simple feedback generated at the point of congestion eliminates the congestion ambiguities faced by end-systems. The authors tested BBRv2 and DCTCP-style ECN [30], and showed that BBRv2 achieves higher throughput when ECN is enabled in routers. Zhang [14] conducted experiments on BBRv1 variations, controlling parameters used to improve the throughput. The tests focused on the life cycles of BBRv1 variations.

Recent work includes in-network TCP feedback with new-generation switches [19, 31]. While these protocols cannot be used without replacing the legacy routers currently used in the Internet, their results suggest future research directions. Kfoury et al. [19] proposed a scheme based on programmable switches to dynamically adjust the rates of competing TCP flows, according to the number of flows sharing the bottleneck link. The

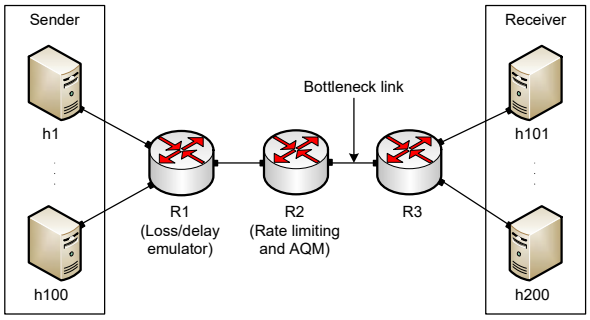


Figure 3: Topology used for evaluations.

scheme uses a custom protocol that is encapsulated inside the IP Options header field and requires programmable switches to parse such header. Li et al. [31] developed High Precision Congestion Control (HPCC), a new congestion control protocol that leverages in-network telemetry (INT) to obtain precise link load information. The authors showed that by using programmable switches and INT, HPCC quickly converges to an equilibrium point while avoiding congestion and maintaining small queues.

4. Experimental Setup

Fig. 3 shows the topology used to conduct the experiments. The topology consists of 100 senders (h1, h2, ..., h100), each opening a TCP connection to a corresponding receiver (h101, h102, ..., h200). The hosts in the experiments are network namespaces in Linux. The AQM policy used in routers is the simple Tail Drop, unless another policy is explicitly stated. The emulation is conducted on Mininet [32], a lightweight mechanism for isolating network resources. The virtual machine used for the experiment runs a light Ubuntu (Lubuntu 19.04) distribution and the kernel version is 5.2.0-rc3+. The scripts used for the emulation are available in the following GitHub repository [33]. The emulation scenario was carefully designed, and sufficient resources were allocated (8 Xeon 6130 cores operating at 2.1 GHz, with 8GB of RAM) to avoid over-utilization of resources (e.g., the usage of CPUs was at all times below prudent levels), thus avoiding misleading results. The version of BBRv2 used in the experiments can be found at [34]. Specifically, the "v2alpha-2019-07-28" release was used. The Fair Queue (`fq`) queuing discipline (`qdisc`) was used on the sending hosts to implement pacing.

Latency/Loss Emulation. The router R1 is used to configure the propagation delay and random packet loss rate, on the link connected to router R2. The Network Emulator (NetEm) tool [35] is used to set these values. All tests are configured with a total propagation delay of 20ms, unless otherwise specified.

Rate Limitation and Buffer Size. Another important factor studied in this paper is the impact of the buffer size on the bottleneck link. The topology of Fig. 3 uses the Linux's Token Bucket Filter (TBF) for limiting the link rate and hence for emulating a bottleneck in the network. TBF is also used to specify the buffer size on the interface of the router R2, facing the router R3. The bottleneck bandwidth (link R2-R3) is set to 1Gbps, unless otherwise specified. All other links have a capacity of approximately 40Gbps.

Metrics Collection. The tool used to perform the measurements between devices is iPerf3 [36]. Performance metrics and variables include throughput, retransmission rate, size of congestion window, RTT, and others. The queue occupancy on the interface connecting to the bottleneck link is measured using Linux's traffic control (tc). The estimated bottleneck bandwidth in BBRv2, pacing rate, and other internal variables are measured using Linux's *ss* command. The retransmission rate is calculated using `netstat`. The Jain's fairness index is computed to measure fairness, as described in RFC 5166 [15]:

$$\mathcal{F} = \frac{\left(\sum_{i=1}^{n} T_i\right)^2}{n \cdot \sum_{i=1}^{n} (T_i)^2}, \tag{1}$$

where T_i is the throughput of flow i. For example, in an scenario with 100 simultaneous flows, $n = 100$ and $i = 1, 2, ..., 100$. The results report the fairness index in percentage, which is given by multiplying Eq. (1) by 100.

Each test was executed for 300 seconds, unless otherwise specified. The size of the TCP send and receive buffers (`net.ipv4.tcp_wmem` and `net.ipv4.tcp_rmem`) on the end-hosts (senders and receivers) was set to 10 times the bandwidth-delay product.

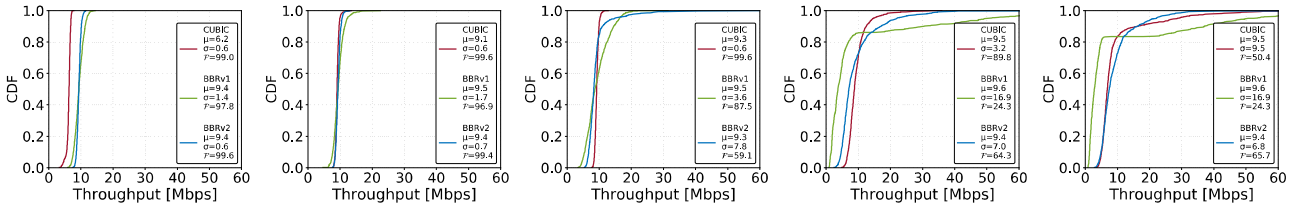
5. Results and Evaluation

This section presents the results obtained by running tests in different network conditions. Every experiment is repeated 10 times and results are averaged for more accuracy.

5.1. Multiple Flows and Buffer Sizes

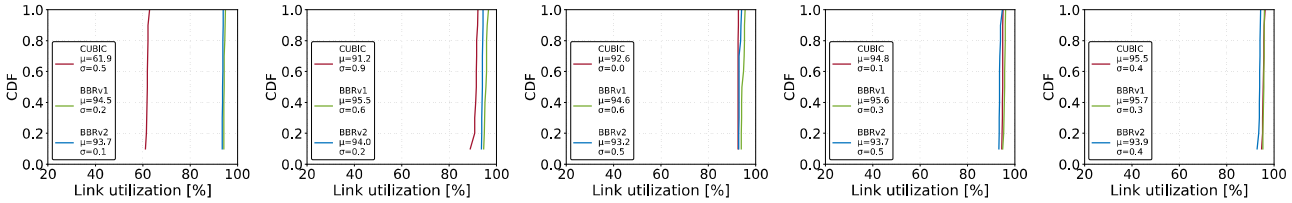
These tests measure the throughput of a flow and the link utilization of the bottleneck link. Each test consists of 100 flows running the same congestion control algorithm: CUBIC, BBR, or BBRv2. The throughput of each flow over the duration of the test (300 seconds) is calculated, which produces 100 samples. As the test is repeated 10 times, the Cumulative Distribution Function (CDF) of the throughput is constructed with the 1000 samples. The link utilization for a test is obtained by adding the throughput of the 100 flows corresponding to that test and dividing this aggregate throughput by the capacity of the link, 1Gbps. The CDF of the link utilization is constructed with the 10 samples.

Fig. 4 shows the CDFs of the throughput when no random packet losses are emulated, considering various buffer sizes. Fig. 5 shows their corresponding CDFs of the link utilization. Figs. 4(a)-(b) and 5(a)-(b) show the results when the buffer size is small, 0.01BDP and 0.1BDP, respectively. BBRv2's average throughput exceeds CUBIC, and is closely similar to that



(a) Buffer size: 0.01BDP. (b) Buffer size: 0.1BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 4: Cumulative distribution function of the throughput of CUBIC, BBRv1, and BBRv2, with various buffer sizes and no random packet loss.



(a) Buffer size: 0.01BDP. (b) Buffer size: 0.1BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 5: Cumulative distribution function of the link utilization of CUBIC, BBRv1, and BBRv2, with various buffer sizes and no random packet loss.

of BBRv1. CUBIC presents lowest throughput and link utilization due to the constant AIMD cycles emerging from frequent packet losses, caused by the small buffer. For BBRv1 and BBRv2, the standard deviation of the throughput is small and the bottleneck link's bandwidth is evenly distributed among the simultaneous flows. Thus, the fairness index is high.

Fig. 4(c) and Fig. 5(c) show the results when the buffer size is increased to 1BDP. Although the average throughput of BBRv2 is similar to that of CUBIC, its standard deviation is increased, which indicates that some flows are grabbing more bandwidth than others, and hence, the fairness index decreases.

Fig. 4(d) shows the results when the buffer size is increased to 10BDP. The throughput and the link utilization of CUBIC slightly increase while its fairness index decreases to 89.8%. Although the throughput and link utilization of BBRv1 and BBRv2 remain high, their fairness indices are only 24.3% and 64.3%. Thus, for BBRv1 and BBRv2, increasing the buffer size to 1BDP and above only reduces the fairness.

Figs. 6 and 7 show the results obtained with 1% packet loss rate. Initially, the same tests were executed with 0.01% packet loss rate instead of 1%, but the results were very similar to Figs. 4 and 5 due to having a small loss rate distributed among the 100 flows. Thus, 1% loss rate was chosen. With packet losses, BBRv1 and BBRv2 achieve much higher throughput and link utilization than CUBIC, independently of the buffer size. However, when the buffer size is large, 10BDP and 100BDP, see Fig. 6(d)-(e), the fairness index produced by BBRv1 and BBRv2 decreases to low levels.

Summary: with small buffers, BBRv1 and BBRv2 produce a fair bandwidth allocation and high throughput and link utilization. With large buffers, the high throughput and the link utilization remain but the fairness decreases significantly. These two observations indicate that BBRv1 and BBRv2 work better with small buffers, as the TCP control loop becomes faster. Both BBRv1 and BBRv2 have better performance than CUBIC, especially with random packet losses and small buffers.

5.2. *Retransmissions with Multiple Flows*

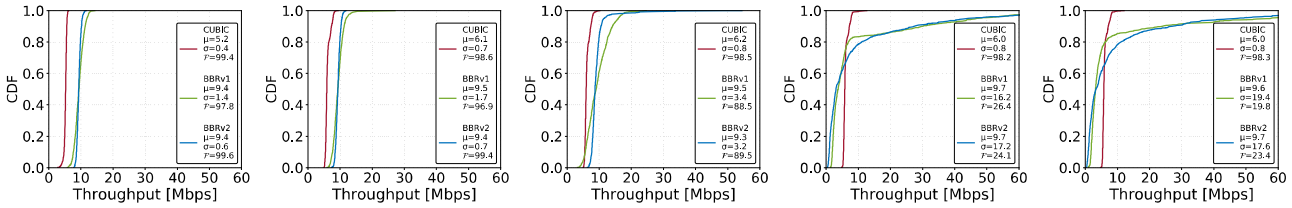
These tests compare the retransmission rate of CUBIC, BBRv1, and BBRv2 as a function of the number of competing flows. The buffer size is 0.02BDP and no random packet losses are introduced. The duration of each test is 300 seconds and the emulated propagation delay is 100ms.

Fig. 8 shows that the retransmission rate of BBRv1 is significantly high with any number of flows. With a single flow, the retransmission rate of BBRv1 is around 2.5%, while CUBIC and BBRv2's retransmission rates are below 0.1%. When the number of flows increases to 10, the retransmission rate of BBRv1 increases to approximately 18%, while BBRv2's retransmission rate is approximately 2.5%. CUBIC's retransmission rate remains low at 0.5%. As the number of flows increases, BBRv1's retransmission rate continues to increase and is significantly higher than that of BBRv2. Although the retransmission rate of CUBIC is lower than that of BBRv2, the throughput is also lower, see Figs. 4 and 6.

Summary: the retransmission rate of BBRv2 is significantly lower than that of BBRv1. Thus, BBRv2 successfully reduces the excessive number of retransmissions of BBRv1. Although the retransmission rate of CUBIC is low, its throughput is also low.

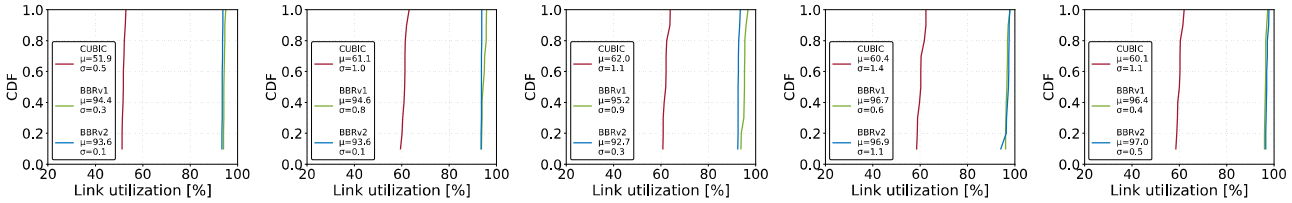
5.3. *Queueing Delay with Large Buffers*

These tests measure the queueing delay when the router's buffer size is large. Experiments are analogous to those reported by the BBRv2's IETF presentation [13]. The duration of each test is 300 seconds and the total propagation delay and the bandwidth are 30ms and 1Gbps respectively. Various buffer sizes are



(a) Buffer size: 0.01BDP. (b) Buffer size: 0.1BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 6: Cumulative distribution function of the throughput of CUBIC, BBRv1, and BBRv2, with various buffer sizes and a random packet loss rate of 1%.



(a) Buffer size: 0.01BDP. (b) Buffer size: 0.1BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 7: Cumulative distribution function of the link utilization of CUBIC, BBRv1, and BBRv2, with various buffer sizes and a random packet loss rate of 1%.

used and samples of RTTs are averaged into a smoothed round-trip time (SRTT).

Fig. 9(a) shows the RTT results, which is the sum of the queueing delay plus the propagation delay, when two flows are present. When the buffer size is 1BDP, the queueing delay is small and packets experience a slight increase above the propagation delay (30ms) with BBRv2. When the buffer size is 10BDP, the RTT of CUBIC increases to approximately 300ms. On the other hand, the RTTs of BBRv1 and BBRv2 are significantly lower, approximately 60ms. As the buffer size increases to 50BDP and 100BDP, the RTT of CUBIC increases to approximately 1000ms and 2000ms, respectively. The RTTs of BBRv1 and BBRv2 remain approximately constant at 60ms.

Figs. 9(b)-(d) show the latency results, when 10, 25, and 50 flows use the same congestion control algorithm. The RTT of CUBIC decreases as the number of flows increases, especially with large buffers. With more simultaneous flows, CUBIC's congestion window is relatively small and the throughput per flow decreases, which reduces the buffer occupancy. BBRv1 and BBRv2 maintain low RTTs, between 50-75ms.

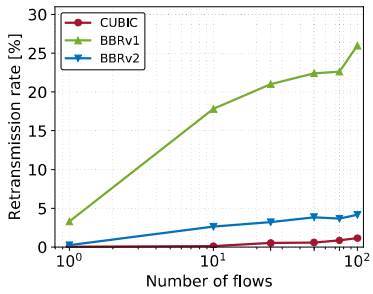


Figure 8: Retransmission rate as a function of the number of flows. The buffer size is 0.02BDP.

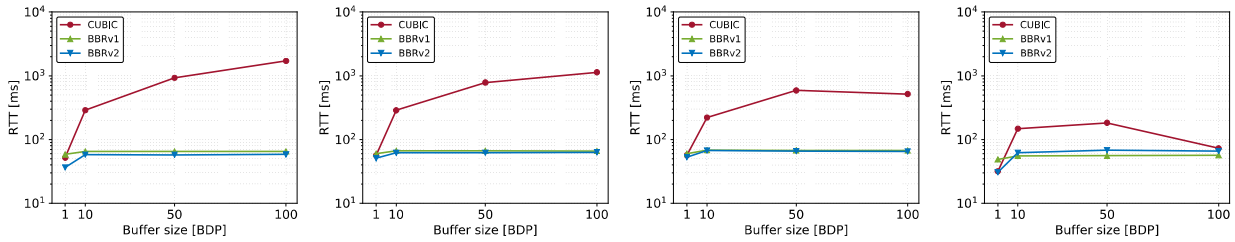
Summary: BBRv1 and BBRv2 maintain relatively low queueing delay, independently of the number of flows and the buffer size. CUBIC has a high queueing delay with large buffers, in particular when the number of simultaneous flows in the network is small.

5.4. *Throughput and Retransmissions as a Function of Packet Loss Rate*

These tests measure the throughput and the retransmission rate of a single TCP flow as a function of the packet loss rate, using CUBIC, BBR, and BBRv2. Tests with buffer sizes of 0.1BDP and 1BDP are conducted. The total propagation delay is 100ms.

Fig. 10(a) shows that BBRv1 and BBRv2 achieve higher throughput than CUBIC when the buffer size is 0.1BDP. When the packet loss rate is lower than 1%, BBRv1 and BBRv2 have a steady throughput of approximately 900Mbps. At that rate, the throughput of BBRv2 decreases to 600Mbps while that of BBRv2 remains at 900Mbps. When the packet loss rate increases to 10%, the throughput of BBRv2 collapses while that of BBRv1 decreases to 600Mbps. Although BBRv1 has a higher throughput than BBRv2, its retransmission rate is also higher, as shown in Fig. 10(b). BBRv1 is loss-agnostic, which leads to a higher retransmission rate, in particular when the buffer size is small. On the other hand, BBRv2 uses the packet loss rate as a signal to adjust the sending rate, as shown in Fig. 1.

Fig. 10(c) shows the throughput when the buffer size is 1BDP. The throughput of BBRv2 decreases to 800Mbps when the packet loss rate is 1%, and collapses as the rate increases further. BBRv1 achieves a better throughput than BBRv2 when the packet loss rate exceeds 0.1%. Both BBRv1 and BBRv2 significantly outperform CUBIC in throughput. Fig. 10(d)



(a) Round-trip time, 2 flows. (b) Round-trip time, 10 flows. (c) Round-trip time, 25 flows. (d) Round-trip time, 50 flows.

Figure 9: Round-trip time experienced by packets with: (a) 2 simultaneous flows; (b) 10 simultaneous flows; (c) 25 simultaneous flows; and (d) 50 simultaneous flows in the network. The total propagation delay is 30ms.

shows the retransmission rate when the buffer is 1BDP. BBRv1 and BBRv2 have similar retransmission rates.

Summary: BBRv2 can tolerate up to approximately 1% of random packet loss rate without a significant performance degradation, which is much higher than the packet loss rate tolerated by CUBIC but lower than that of BBRv1. On the other hand, with a small buffer, the retransmission rate of BBRv2 is significantly lower than that of BBRv1.

5.5. Coexistence and Fairness with CUBIC

These tests investigate the capabilities of BBRv1 and BBRv2 to coexist with CUBIC, with different buffer sizes. Four types of experiments are considered: 1) a single BBRv1 flow competing with a single CUBIC flow; 2) 50 BBRv1 flows competing with 50 CUBIC flows; 3) a single BBRv2 flow competing with a single CUBIC flow; and 4) 50 BBRv2 flows competing with 50 CUBIC flows. Experiments with and without random packet losses are conducted.

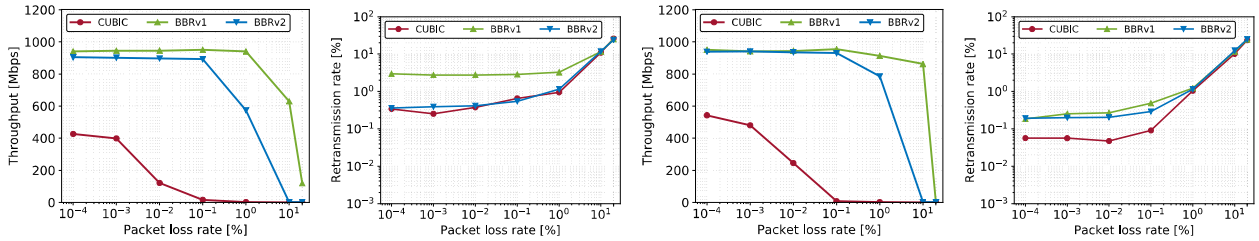
The middle and bottom graphs of Fig. 11(a) show the throughput as a function of the buffer size. The top graph shows the corresponding fairness index. When the buffer size is below 1BDP, BBRv1 consumes approximately 90% of the bandwidth. On the other hand, BBRv2 shows a better coexistence with CUBIC, in particular with a small buffer. When the buffer size exceeds 1BDP, CUBIC starts consuming more bandwidth. When the buffer is significantly large, 100BDP, the fairness index decreases to approximately 60%.

Fig. 11(b) shows the results with a random packet loss rate of 0.01%. When the buffer size is below 1BDP, BBRv2 has higher throughput than CUBIC, yet both achieve comparable throughput. This result is reflected in the fairness index, which is always above 80%. In contrast, when the buffer size is below 1BDP, BBRv1 consumes most of the bandwidth, leading to a poor performance of the CUBIC flow. The corresponding fairness index is below 60%. This index increases to approximately 100% only when the buffer size is very large, 100BDP.

Figs. 11(c) and 11(d) show the results with 50 BBRv1/BBRv2 flows competing with 50 CUBIC flows, with and without random packet losses. BBRv2 achieves better throughput than CUBIC, independently of the buffer size, and the fairness index is always above 80%. On the other hand, BBRv1 consumes most of the bandwidth, in particular when the buffer size is below 1BDP. When no random packet losses are configured and when the buffer size is above 4BDP, the throughput of BBRv1 and CUBIC are similar and the fairness index increases.

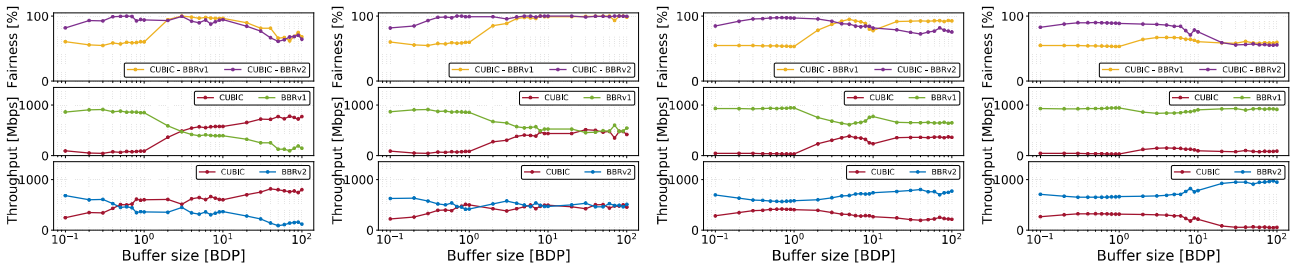
Fig. 12 evaluates the impact of the number of competing flows on the throughput share per congestion control algorithm. The dashed lines represent the ideal fair share, while the solid lines represent the measured share. It can be seen that generally BBRv2 flows claim more bandwidth than their fair shares when competing against CUBIC flows. Nevertheless, as the number of BBRv2 flows increases, CUBIC's bandwidth share approaches the ideal share; for example, with 10 BBRv2 flows and two CUBIC flows, the achieved share is exactly the fair share.

Fig. 13 shows the heatmaps of the fairness index. The tests consider two competing flows (CUBIC and BBRv2), different buffer sizes, without random packet losses and with 0.01% random packet loss rate. Each entry in a heatmap was generated by



(a) Buffer size: 0.1BDP. (b) Buffer size: 0.1BDP. (c) Buffer size: 1BDP. (d) Buffer size: 1BDP.

Figure 10: Throughput and retransmission rate as functions of the packet loss rate. The buffer size is: (a), (b): 0.1BDP, and (c), (d): 1BDP.



(a) No packet loss, 2 flows. (b) 0.01% packet loss rate, 2 flows. (c) No packet loss, 100 flows. (d) 1% packet loss rate, 100 flows.

Figure 11: Throughput and fairness index as functions of the buffer size. (a) 2 flows: one BBRv1 (middle graph) / BBRv2 (bottom graph) flow and one CUBIC flow, with no random packet losses; (b) 2 flows: one BBRv1/BBRv2 flow and one CUBIC flow, with a random packet loss rate of 0.01%; (c) 100 flows: 50 BBRv1/BBRv2 flows and 50 CUBIC flows, with no random packet losses; and (d) 100 flows: 50 BBRv1/BBRv2 flows and 50 CUBIC flows, with a random packet loss rate of 1%.

running 10 experiments, calculating the average fairness value, and coloring the entry according to that value. Additionally, an entry includes three values: the link utilization of the bottleneck link in percentage (center top value), the percentage of bandwidth used by CUBIC (bottom left value), and the percentage of bandwidth used by BBRv2 (bottom right value). The rows and columns correspond to the bandwidth of the bottleneck link and the propagation delay respectively. For simplicity, the propagation delay is referred to as delay hereon.

Fig. 13(a)-(c) show the heatmap with no random packet losses. When the buffer size is small, see Fig. 13(a), the heatmap illustrates a low fairness index. Even with small BDP values (upper left entries of the matrix), BBRv2 is allocated more bandwidth than CUBIC. The latter presents low bandwidth allocation because of the constant AIMD cycles emerging from frequent packet losses due to the small buffer. When the bandwidth (lower left entries), the delay (upper right entries), or both increase, the fairness index decreases further. This is clearly noted for large BDP values (lower right entries), where the bandwidth allocated to CUBIC collapses as that of BBRv2 surges. As a traditional loss-based congestion control algorithm, CUBIC throughput is inversely proportional to the RTT and the square root of the packet loss rate [37]. Because of this relationship, its performance is poor with large BDP values. On the other hand, the above relation does not apply to BBRv2. Moreover, BBRv2 actively measures the propagation delay and the available bandwidth to estimate the number of packets that can be in-flight, leading to a better utilization of the bandwidth. Note that when the bandwidth is 20Mbps and the delay is 20ms, the buffer can only hold one packet, which leads to a low link utilization.

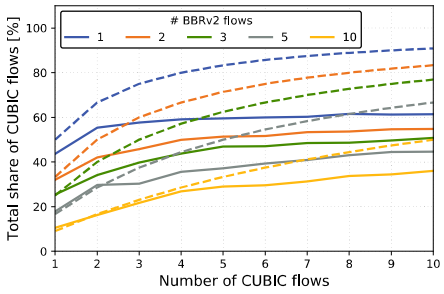


Figure 12: Bandwidth share for competing CUBIC and BBRv2 flows. The dashed lines show the ideal fair share and the solid lines show the measured share.

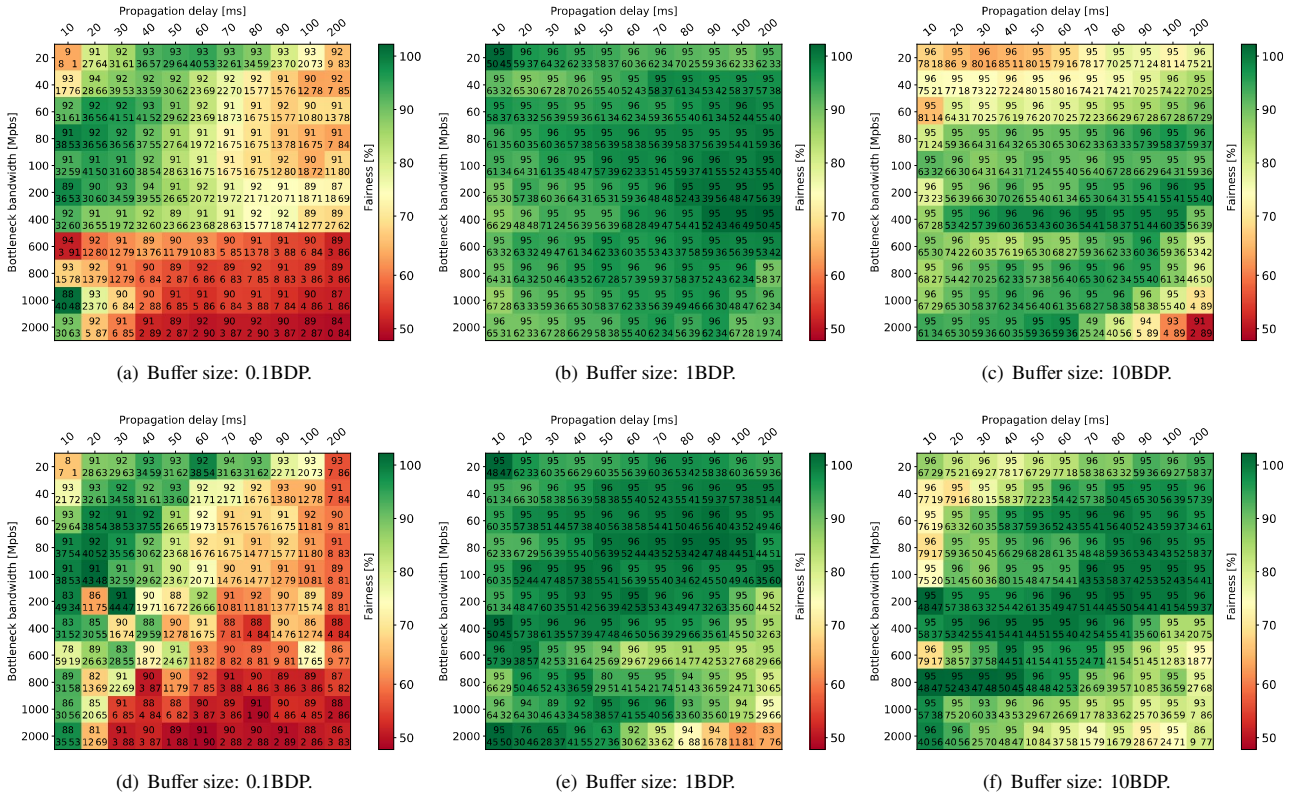
When the buffer size is 1BDP, see Fig. 13(b), packet losses due to buffering at the router decrease. Consequently, CUBIC utilizes more bandwidth and a better fairness index is observed. Increasing the buffer size further to 10BDP, however, results in lower fairness index, see Fig. 13(c). When the BDP is small (upper right entries), CUBIC fills the buffer and uses most of the bandwidth, while BBRv2 is allocated the remaining capacity. As the bandwidth (lower left entries) or the delay (upper right entries) increases, the bandwidth is more evenly allocated. When the BDP is large (lower right corner), the performance of CUBIC deteriorates and BBRv2 utilizes the available bandwidth.

When random packet losses are injected, see Fig. 13(d)-(f), the fairness index is similar to the results observed in Fig. 13(a)-(c). A larger bandwidth allocation for BBRv2 is more evident as the BDP increases (lower right corner) and CUBIC is unable to use its share of the bandwidth.

Summary: BBRv2 shows a better coexistence with CUBIC than BBRv1. Additionally, as the number of flows increases, the aggregate throughput observed in BBRv2 is more consistent than that of BBRv1, independently of the buffer size. Therefore, when BBRv2 is used, selecting the correct buffer size is not as important as it is when loss-based congestion control algorithms are used. Furthermore, as the number of BBRv2 flows increases when competing against CUBIC, the bandwidth share of CUBIC approaches the ideal share. When the buffer size is 1BDP, BBRv2 demonstrates good coexistent with CUBIC. BBRv2 is also able to use the available bandwidth in networks with small buffer size and in networks with large BDP. In such scenarios, additional bandwidth is available because of the inability of CUBIC to use its share.

5.6. Round-trip Time Unfairness

The literature has reported that BBRv1 suffers from RTT unfairness [27]. Unlike traditional congestion control algorithms,



(a) Buffer size: 0.1BDP. (b) Buffer size: 1BDP. (c) Buffer size: 10BDP.

(d) Buffer size: 0.1BDP. (e) Buffer size: 1BDP. (f) Buffer size: 10BDP.

Figure 13: Fairness index visualized in a heatmap for CUBIC and BBRv2 flows. Each entry in a heatmap includes three numbers: the percentage of the link utilization at the bottleneck link (center top) and the percentage of bandwidth used by the CUBIC flow (bottom left) and the BBRv2 flow (bottom right). (a)-(c) No packet loss. (d)-(f) 0.01% packet loss rate.

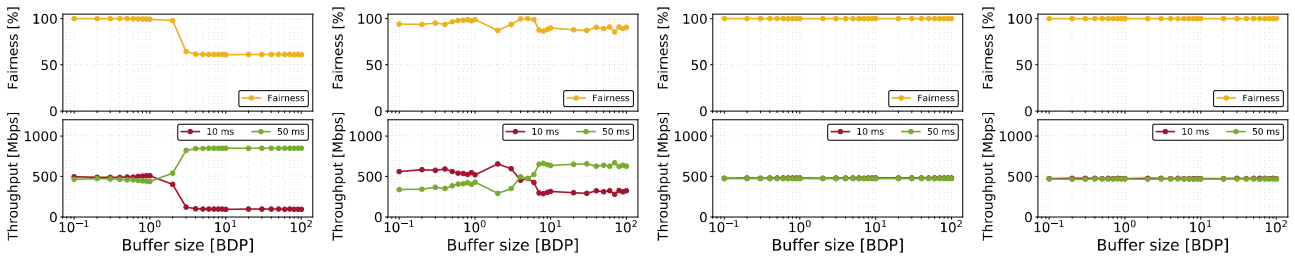
which favor flows with small RTTs, BBRv1 allocates more bandwidth to flows with large RTTs. According to [27], this bias presents a tradeoff between low latency and high delivery rate and therefore violates the engineering efforts of minimizing latency. It also disrupts the concept of finding a route with the minimum RTT.

These tests compare the RTT unfairness of BBRv1 and BBRv2, and investigate whether this limitation is mitigated by BBRv2. The tests incorporate one flow with a propagation delay of 10ms competing with another flow with a propagation delay of 50ms. The tests are executed 10 times and the average is reported. Four types of experiments are considered: 1) two competing BBRv1 flows. Router R2 implements a simple Tail Drop policy; 2) two competing BBRv2 flows. Router R2 implements a simple Tail Drop policy; 3) two competing BBRv1 flows. Router R2 implements FQ-CoDel policy; and 4) two competing BBRv2 flows. Router R2 implements FQ-CoDel policy. The severity of RTT unfairness is also related to the buffer size. Thus, the throughput and fairness are reported as a function of the buffer size. Fig. 14(a) shows the throughput of BBRv1 flows, for experiment 1. When the buffer size is below 1BDP, the two flows receive similar amounts of bandwidth and the fairness index is approximately 100%. When the buffer size increases above 1BDP, the flow with 50ms RTT receives more bandwidth than the flow with 10ms RTT; at 3BDP and above, the flow with 50ms RTT receives approximately 85% of the bandwidth and the fairness index remains at approximately 60%.

Fig. 14(b) shows the throughput of BBRv2 flows, for experiment 2. At the tested buffer sizes, the two flows receive similar amounts of bandwidth and the fairness index is always above 85%. Figs. 14(c) and 14(d) show that by using FQ-CoDel, the RTT unfairness is eliminated and the fairness index is 100% for both BBRv1 and BBRv2.

Fig. 15 shows the average retransmission rates generated by BBRv1 and BBRv2 (i.e., the average of the retransmission rates observed in both flows, the one with propagation delay of 10ms and the one with propagation delay of 50ms), with Tail Drop and FQ-CoDel policies. The results indicate that the retransmission rate generated by BBRv2 is low, independently of the queue management algorithm and the buffer size. On the other hand, the retransmission rate generated by BBRv1 is high when the Tail Drop policy is used and when the buffer size is below 1BDP, see Fig. 15(a). Applying FQ-CoDel helps to reduce the retransmission rate of BBRv1, see Fig. 15(b).

Summary: BBRv2 enhances the coexistence of flows with different RTTs, mitigating the RTT unfairness problem. Also, by using an AQM algorithm such as FQ-CoDel to manage the buffer delay, oversized buffers are made irrelevant, without negatively impacting the bottleneck link utilization. The AQM al-



(a) BBRv1 with Tail Drop policy. (b) BBRv2 with Tail Drop policy. (c) BBRv1 with FQ-CoDel policy. (d) BBRv2 with FQ-CoDel policy.

Figure 14: Throughput and fairness index as functions of the buffer size, for two competing flows: one flow has 10ms RTT and another has 50ms RTT. (a) Flows use BBRv1 and the router R2 implements a simple Tail Drop policy. (b) Flows use BBRv2 and the router R2 implements a simple Tail Drop policy. (c) Flows use BBRv1 and the router R2 implements FQ-CoDel policy. (d) Flows use BBRv2 and the router R2 implements FQ-CoDel policy.

gorithm also helps to reduce the retransmission rate of BBRv1.

5.7. Flow Completion Time (FCT)

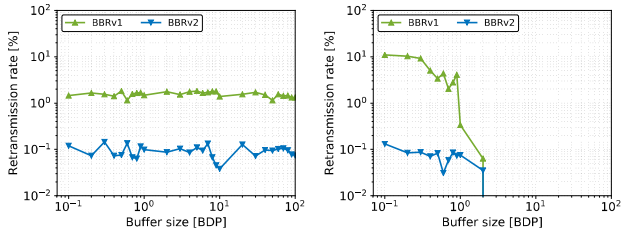
These tests compare the flow completion times (FCTs) using different congestion control algorithms, considering several buffer sizes. Five types of experiments are conducted: 1) 100 flows using the same congestion control algorithm (CUBIC, BBRv1, and BBRv2); 2) 50 flows using CUBIC and 50 flows using BBRv2; 3) 50 flows using CUBIC and 50 flows using BBRv1; 4) the same composition of flows as in experiment 2, but incorporating a random packet loss rate of 1%; and 5) the same composition of flows as in experiment 3, but incorporating a random packet loss rate of 1%. Each experiment is repeated ten times and the average is reported.

Fig. 16 shows the results for experiment 1, where each flow completes a data transfer of 10 Gigabytes (GB). Consider first the case when no random packet losses are introduced, Fig. 16(a). When the buffer size is 0.1BDP, the completion time of flows using CUBIC increases considerable with respect to those flows using BBRv1 and BBRv2, as frequent packet losses occur at the router and the bottleneck link is poorly utilized. When the buffer size is 0.2BDP and larger, the completion time is approximately 90s in all cases, independently of the congestion control algorithm. Note that although relatively small when compared with the rule-of-thumb, 1BDP, a buffer size of 0.2BDP is acceptable for these experiments, because the number of flows is large, 100. This result is aligned with the buffer size rule proposed by Appenzeller et al. [38], who stated that a buffer size of (RTT·bottleneck bandwidth)/ $\sqrt{N}$ bits is sufficient for a full utilization of the bottleneck link, where N is the number of flows. Consider now Fig. 16(b). In the presence of a packet loss rate of 1%, the completion time of flows using BBRv1 and BBRv2 is still approximately 90s. Meanwhile, the completion time of flows using CUBIC increases substantially. Thus, BBRv1 and BBRv2 have a better performance than CUBIC, as they are less sensitive to random packet losses.

Fig. 17 shows the results for experiment 2, where each flow completes a data transfer of 100 Megabytes (MB). When the buffer size is 0.1BDP, see Fig. 17(a), the completion time of flows using BBRv2 is approximately between 50s and 60s, whereas the completion time of flows using CUBIC is approximately between 80s to 90s. As the buffer size increases to 0.5BDP and 1BDP, see Figs. 17(b) and 17(c), the gap between the completion time of flows using BBRv2 and CUBIC is slightly reduced. As the buffer size increases to 10BDP and 100BDP, see Figs. 17(d) and 17(e), the variability of the completion time of flows using both BBRv2 and CUBIC increases, and values vary between 20s and 70s for BBRv2, and between 18s and 80s for CUBIC.

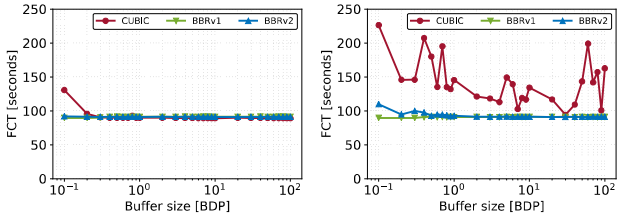
Fig. 18 shows the results for experiment 3, where each flow completes a data transfer of 100 Megabytes (MB). Similar to BBRv2, BBRv1 has a smaller FCT than CUBIC, independently of the buffer size. BBRv1's FCT is smaller than BBRv2 due its aggressiveness when competing with CUBIC.

Fig. 19 shows the results for experiment 4, where the sce-



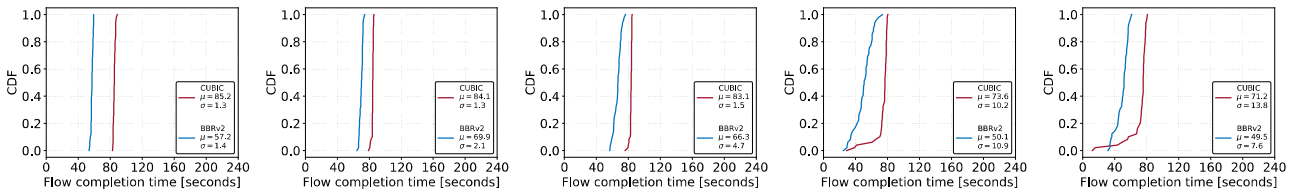
(a) Retransmissions, Tail Drop. (b) Retransmissions, FQ-CoDel.

Figure 15: Retransmission rate generated by BBRv1 and BBRv2, with: (a) a simple Tail Drop policy implemented at router R2, and (b) with FQ-CoDel policy implemented at router R2.



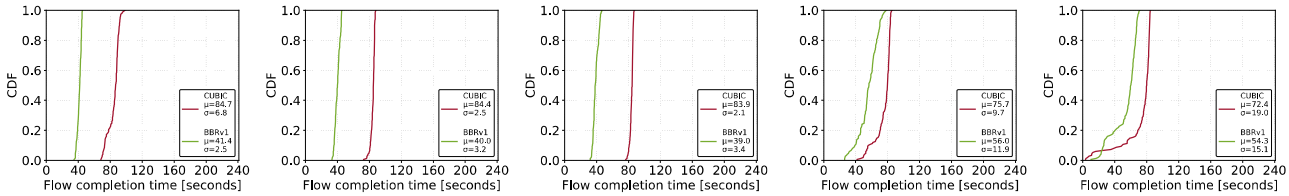
(a) No random packet losses. (b) 0.01% random packet loss rate.

Figure 16: Flow completion time as a function of the buffer size with: (a) no random packet losses; and (b) random packet loss rate of 0.01%.



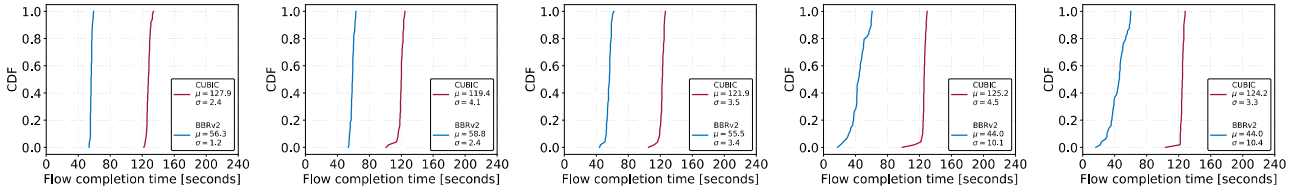
(a) Buffer size: 0.1BDP. (b) Buffer size: 0.5BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 17: Cumulative distribution function of the flow completion time, for 50 flows using CUBIC and 50 flows using BBRv2, with various buffer sizes and no random packet losses.



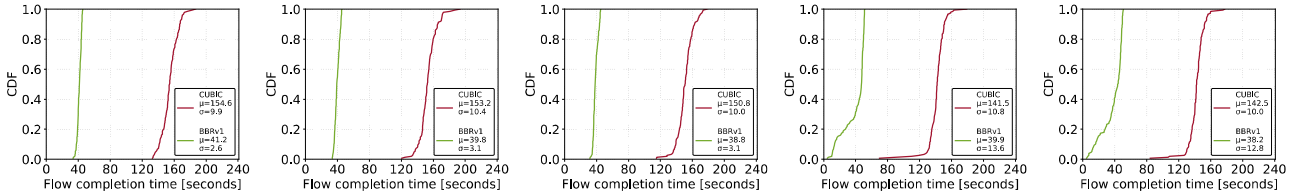
(a) Buffer size: 0.1BDP. (b) Buffer size: 0.5BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 18: Cumulative distribution function of the flow completion time, for 50 flows using CUBIC and 50 flows using BBRv1, with various buffer sizes and no random packet losses.



(a) Buffer size: 0.1BDP. (b) Buffer size: 0.5BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 19: Cumulative distribution function of the flow completion time, for 50 flows using CUBIC and 50 flows using BBRv2, with various buffer sizes and a random packet loss rate of 1%.



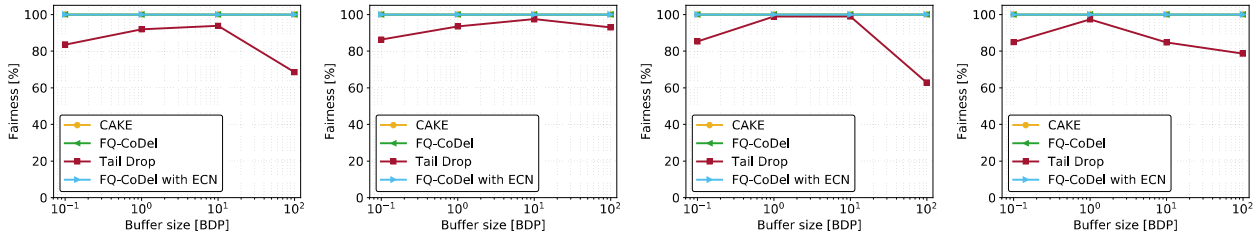
(a) Buffer size: 0.1BDP. (b) Buffer size: 0.5BDP. (c) Buffer size: 1BDP. (d) Buffer size: 10BDP. (e) Buffer size: 100BDP.

Figure 20: Cumulative distribution function of the flow completion time, for 50 flows using CUBIC and 50 flows using BBRv1, with various buffer sizes and a random packet loss rate of 1%.

nario is similar to experiment 2, with the addition of a packet loss rate of 1%. The completion time of flows using BBRv2 is similar to that observed in experiment 2, where no random packet losses were introduced. BBRv2 is not affected by such packet loss rate and correctly estimates the bottleneck bandwidth and sending rate. On the other hand, the flow completion time of flows using CUBIC substantially increases, independently of the buffer size, to values between approximately 100s and 135s.

Fig. 20 shows the results for experiment 5, where the scenario is similar to experiment 3, with the addition of a packet loss rate of 1%. Similar to BBRv2, BBRv1 has a shorter FCT than CUBIC, independently of the buffer size. BBRv2's FCT is larger than that of BBRv1, as it incorporates the packet loss rate in the network path model, see Fig. 1.

Summary: BBRv2 is not affected by random packet loss rates below 1-2% and correctly estimates the bottleneck bandwidth and sending rate, leading to a full utilization of the bottleneck link. The corresponding flow completion times are reduced with respect to those of CUBIC. On the other hand, as a loss-based algorithm, CUBIC erroneously decreases the sending rate when random packet losses occur, leading to a poor utilization



(a) 1 BBRv2 flow, 1 CUBIC flow. (b) 1 BBRv2 flow, 10 CUBIC flows. (c) 10 BBRv2 flows, 10 CUBIC flows. (d) 50 BBRv2 flows, 50 CUBIC flows.

Figure 21: Fairness index as a function of the buffer size, produced with an even number of flows using BBRv2 and CUBIC. The following AQM policies are implemented in the router R2: Tail Drop, CAKE, FQ-CoDel, and FQ-CoDel with ECN.

of the bottleneck link and high flow completion times.

5.8. Impact of AQM on Fairness

These tests compare the fairness index when flows using BBRv2 and CUBIC are present in the network, and when different AQM policies and buffer sizes are implemented on the routers. The tests consider the following AQM policies: simple Tail Drop, CAKE [18], and FQ-CoDel [16] with and without DCTCP-style ECN enabled [30]. The DCTCP-style ECN is enabled by setting the `ce_threshold` parameter of the `fq_codel` qdisc to 242us. Tests are executed with 2, 11, 20, and 100 flows, with even and uneven composition of flows using BBRv2 and CUBIC.

Fig. 21(a) shows the results for 2 flows: 1 BBRv2 flow and 1 CUBIC flow. Consider first the results with Tail Drop policy (red curve). When the buffer size is 0.1BDP, the fairness index is approximately 80%. This result is a consequence of the inability of CUBIC to fully utilize its bandwidth share when the buffer size is small. When the buffer size increases to values between 0.5BDP to 10BDP, the fairness index using Tail Drop improves to values close to 90%. However, when the buffer increases further to 100BDP, the Tail Drop policy leads to a poor fairness index of approximately 60%, because CUBIC fills the buffer, increases the delay, and consumes most of the bandwidth. Consider now the results with more advanced AQM policies, CAKE and FQ-CoDel (with and without ECN). These policies produce better fairness indices, which approach 100%. These AQM policies are designed to solve the bufferbloat problem and also prevent flows from consuming additional bandwidth beyond their share.

Fig. 21(b) shows the case when a BBRv2 flow and 10 CUBIC flows are present. In this case, the fairness index remains above 80% when the Tail Drop policy is used. Similarly, advanced AQM policies improve the fairness. Fig. 21(c) shows the results when 10 BBRv2 flow and 10 CUBIC flows are present. With the Tail Drop policy, the results are similar to those observed in Fig. 21(a), and the unfairness resulting from CUBIC consuming most of the bandwidth is noted when the buffer size is 100BDP. Fig. 21(d) shows the case when 50 BBRv2 flows and 50 CUBIC flows are present. The fairness index is approximately 80% for any buffer size different than 1BDP. On the other hand, advanced AQM policies produce better fairness indices, independently of the buffer size and the number of flows.

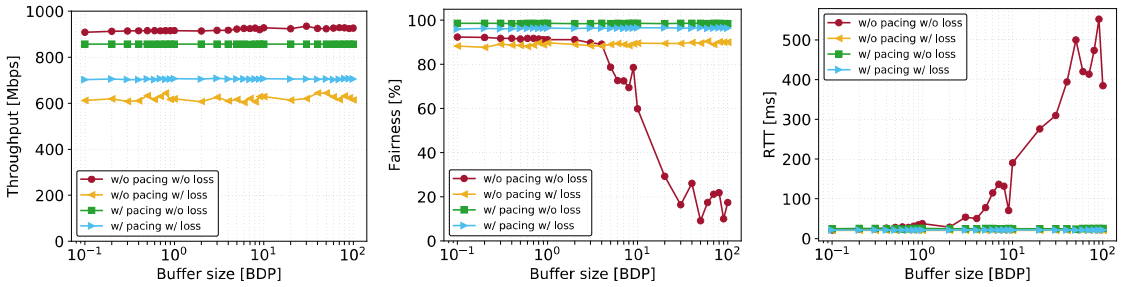
Summary: although very simple to implement, a simple Tail Drop policy leads to unfair bandwidth allocation among BBRv2 and CUBIC flows, in particular when the buffer size is large. On the other hand, advanced AQM policies such as FQ-CoDel and CAKE produce fair bandwidth allocations, independently of the buffer size and the number of flows.

5.9. The Effects of Fixed-rate Pacing on TCP CUBIC

TCP pacing is used by BBRv1 and BBRv2. It evenly spaces packets over time, so that packets are not sent in bursts. Pacing reduces the variance in the data rates at network bottlenecks, which in turn reduces the transient queues everywhere [8]. As a result, congestion control algorithms can still achieve high throughput with small buffers, as noted in previous sections. Motivated by such results, this section investigates whether pacing can have a positive impact on window-based congestion control algorithms.

These tests investigate the throughput, fairness, and RTT of flows using CUBIC, with and without TCP pacing. The number of flows is 100. The bottleneck bandwidth is 1Gbps and the total propagation delay of 20ms. The tests are executed without random packet losses and with a random packet loss rate of 1%. The experiments are executed 10 times and the average is reported. When pacing is used, the pacing rate is manually computed so that all flows have the same maximum bandwidth allocation. Thus, the pacing rate per flow is set to (1Gbps)/(100 flows) - headroom = 10Mbps - 1.5Mbps = 8.5Mbps. The headroom of 1.5Mbps is used to keep the link utilization at approximately 85%, which avoids filling the buffer. Pacing is applied through the `fq qdisc` in Linux using `fq pacing maxrate` parameter when the sending hosts are configured with CUBIC. Note that these experiments emulate ideal scenarios where the number of flows is known in advance and each flow is configured with the correct pacing rate. Nevertheless, the results provide insight into the success of new congestion control algorithms based on pacing.

Fig. 22(a) shows the aggregate throughput. When pacing is used and the network does not experience packet losses, the throughput is high, constant at 850Mbps. When pacing is not used, and the network does not experience packet losses, the throughput is slightly higher, peaking at 900Mbps. On the other hand, when the network experiences packet losses, enabling



(a) Throughput. (b) Fairness index. (c) Round-trip time.

Figure 22: Throughput, fairness index, and RTT as functions of the buffer size. The four curves represent the results of flows operating according to the following configurations: CUBIC without enabling TCP pacing and no random packet losses (red); CUBIC without enabling TCP pacing and with a packet loss rate of 1% (yellow); CUBIC with TCP pacing and no random packet losses (green); and CUBIC with TCP pacing and with a packet loss rate of 1% (blue).

pacing improves the total throughput: the throughput of paced flows is 700Mbps, while the throughput of non-paced flows is approximately 600Mbps. The buffer size does not affect the aggregate throughput.

Fig. 22(b) shows the fairness results. When pacing is used, the fairness index is high, close to 100%, independently of whether the network experiences packet losses or not. When pacing is not used and the buffer size is below 5BDP, the fairness index is similar in both cases, with packet losses and without packet losses. When the buffer size increases to 5BDP and above, the fairness index decreases rapidly. CUBIC without pacing is unable to avoid a very disproportionate bandwidth allocation, even when all flows use the same congestion control algorithm.

Fig. 22(c) shows the RTT results. When pacing is used, the RTT is almost constant and equal to the propagation delay. When pacing is not used and the network experiences packet losses, the RTT is also almost constant and equal to the propagation delay. In this regard, the random packet losses have a positive effect on CUBIC, as they prevent the congestion window to continue to increase, and sender nodes do not fill up the buffer. On the other hand, when no losses occur and pacing is not used, the bufferbloat problem is observed.

Summary: TCP pacing enhances the throughput of CUBIC when the network experiences packet losses. It also improves the fairness among CUBIC flows and avoids the bufferbloat problem when the buffer size is large. Although the scenarios described in this section are ideal (the pacing rate must be configured statically according to the number of flows. In production networks, estimating the number of flows in real time is challenging), future research directions may explore the capability of new-generation switches to obtain explicit feedback from routers to estimate the pacing rates dynamically [19, 31].

6. Conclusion

BBRv1 represented a major disruption to the traditional loss-based congestion control algorithms. BBRv1 departed from the use of binary signals -packet losses- to modify the sending rate. Instead, it relied on periodical measurements of the bottleneck bandwidth and the RTT to set the pacing rate and the number of bits to keep in-flight. Despite its success in improving the throughput of a connection, BBRv1 has shown some issues, including a poor coexistence with other congestion control algorithms and a high retransmission rate. In this context, BBRv2 has been recently proposed to address such issues.

This paper presented an emulation-based evaluation of BBRv2, using Mininet. Results show that BBRv2 tolerates much higher random packet loss rates than CUBIC but slightly lower than BBRv1, has better coexistence with CUBIC than BBRv1, mitigates the RTT unfairness problem, produces lower retransmission rates than BBRv1, and exhibits low queuing delay even with large buffers. The results also show that with small buffers, BBRv2 produces better fairness without compromising the high throughput and the link utilization. This observation was also noted in BBRv1 and suggests that model-based, rate-based congestion control algorithms work better with small buffers, which could have significant implications on networks. The paper also presented advantages of using AQM algorithms and TCP pacing with loss-based algorithms.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Numbers 1925484, 1829698, and 1907821, funded by the Office of Advanced Cyberinfrastructure (OAC). The authors are very grateful for the comprehensive review conducted by the anonymous reviewers. Their suggestions and corrections helped improve the quality of this manuscript.

References

[1] J. Postel, RFC 793: Transmission control protocol (TCP), September, 1981.
[2] J. Kurose, K. Ross, Computer networking: a top-down approach, 7th Edition, Pearson, 2017.
[3] V. Jacobson, M. Karels, Congestion avoidance and control, ACM SIGCOMM Computer Communications Review 18 (4) August, 1988.
[4] R. Al-Saadi, G. Armitage, J. But, P. Branch, A survey of delay-based and hybrid TCP congestion control algorithms, IEEE Communications Surveys & Tutorials, 21, (4), March, 2019.

[5] P. Hurley, J. Le Boudec, P. Thiran, Technical Report: A note on the fairness of additive increase and multiplicative decrease, Infoscience EPFL Scientific Publications, 1990.
[6] S. Ha, I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, ACM SIGOPS Operating Systems Review, 42, (5), July, 2008.
[7] D. J. Leith, R. N. Shorten, H-TCP protocol for high-speed long distance networks, Second International Workshop on Protocols for Fast Long-Distance Networks, Chicago, IL, USA, February, 2004.
[8] M. Mathis, J. Mahdavi, Deprecating the TCP macroscopic model, ACM SIGCOMM Computer Communication Review, 49, (5), November, 2019.
[9] J. Crichigno, E. Bou-Harb, N. Ghani, A comprehensive tutorial on science DMZ, IEEE Communications Surveys & Tutorials, 21, (2), October, 2018.
[10] K. Chard, I. Foster, S. Tuecke, Globus: Research data management as service and platform, 2017 Practice and Experience in Advanced Research Computing (PEARC) Conference, New Orleans, LO, USA, July, 2017.
[11] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, BBR: congestion-based congestion control, ACM Queue, 14, (5), September, 2016.
[12] M. Hock, R. Bless, M. Zitterbart, Experimental evaluation of BBR congestion control, 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, Canada, October, 2017.
[13] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, V. Jacobson, BBRv2: a model-based congestion control, Presentation in the Internet Congestion Control Research Group (ICCRG) at IETF 105 Update, Montreal, Canada, July, 2019.
[14] S. Zhang, An evaluation of BBR and its variants, In Proceedings of ACM Conference, Washington, DC, USA, July 2017.
[15] S. Floyd, RFC 5166: metrics for the evaluation of congestion control mechanisms, March, 2008.
[16] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, E. Dumazet, RFC 8290: The flow queue CoDel packet scheduler and active queue management algorithm, January, 2018.
[17] J. Gettys, Bufferbloat: dark buffers in the internet, IEEE Internet Computing, 15, (3), May, 2011.
[18] T. Høiland-Jørgensen, D. Täht, J. Morton, Piece of CAKE: a comprehensive queue management solution for home gateways, 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Washington, DC, USA, October, 2018.
[19] E. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, G. Srivastava, Enabling TCP pacing using programmable data plane switches, 42nd International Conference on Telecommunications and Signal Processing (TSP), Budapest, Hungary, July, 2019.
[20] S. M. Claypool, Sharing but not caring-performance of TCP BBR and TCP CUBIC at the network bottleneck, Worcester Polytechnic Institute, March, 2019.
[21] R. Jain, A. Durresi, G. Babic, Throughput fairness index: an explanation, ATM Forum contribution, 99, (45), February, 1999.
[22] M. Allman, Comments on bufferbloat, ACM SIGCOMM Computer Communication Review, 43, (1), January, 2013.
[23] Y. Gong, D. Rossi, C. Testa, S. Valenti, M. D. Täht, Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control, Computer Networks, 65, (1), June, 2014.
[24] C. Staff, Bufferbloat: what's wrong with the Internet?, Communications of the ACM, 55, (2), February, 2012.
[25] K. Nichols, V. Jacobson, Controlling queue delay, Communications of the ACM, 10, (5), July, 2012.
[26] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, G. Carle, Towards a deeper understanding of TCP BBR congestion control, 2018 IFIP Networking Conference (IFIP Networking) and Workshops, Zurich, Switzerland, May, 2018.
[27] S. Ma, J. Jiang, W. Wang, B. Li, Towards RTT fairness of congestion-based congestion control, Computing Research Repository (CoRR), June, 2017.
[28] F. Fejes, G. Gombos, S. Laki, S. Nádas, Who will save the internet from the congestion control revolution?, International Conference Proceeding Series (ICPS) Proceedings of the 2019 Workshop on Buffer Sizing, Palo Alto, CA, USA, December, 2019.
[29] M. P. Tahiliani, V. Misra, K. Ramakrishnan, A principled look at the utility of feedback in congestion control, International Conference Proceeding Series (ICPS) Proceedings of the 2019 Workshop on Buffer Sizing, Palo Alto, CA, USA, December, 2019.
[30] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, G. Judd, RFC 8257: Data center TCP (DCTCP): TCP congestion control for data centers, October, 2017.
[31] Y. Li, R. Miao, H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, M. Yu, HPCC: high precision congestion control, SIGCOMM 2019: Proceedings of the ACM Special Interest Group on Data Communication, Beijing, China, August, 2019.
[32] R. De Oliveira, C. Schweitzer, A. Shinoda, R. Prete, Using mininet for emulation and prototyping software-defined networks, 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, June, 2014.
[33] J. Gomez, E. Kfoury, Emulation scripts using BBRv2, [Online], Available: `https://github.com/gomezgaona/bbr2`, July, 2020.
[34] google/bbr, TCP BBR v2 Alpha/Preview Release, [Online], Available: `https://github.com/google/bbr/tree/v2alpha`, July, 2019.
[35] S. Hemminger, et al., Network emulation with NetEm, Linux conf au, April, 2005.
[36] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, K. Prabhu, iPerf3, tool for active measurements of the maximum achievable bandwidth on IP networks, [Online], Available: `https://github.com/esnet/iperf`.
[37] M. Mathis, J. Semke, J. Mahdavi, T. Ott, The macroscopic behavior of the TCP congestion avoidance algorithm, ACM SIGCOMM Computer Communication Review 27 (3) July, 1997.
[38] G. Appenzeller, I. Keslassy, N. McKeown, Sizing router buffers, ACM SIGCOMM Computer Communication Review, 34, (4), August, 2004.



Elie F. Kfoury is a PhD student in the College of Engineering and Computing at the University of South Carolina (USC). Prior to joining USC, he worked as a research and teaching assistant in the Computer Science department at the American University of Science and Technology in Beirut. He has published several research papers and articles in international conferences and peer-reviewed journals. His research focuses on telecommunications, network security, Blockchain, Internet of Things (IoT), Software Defined Networking (SDN), and data plane programming.



Jose Gomez is a PhD student in the College of Engineering and Computing at the University of South Carolina (USC) in the United States of America. Prior to joining USC, he worked as a researcher and teaching assistant in the School of Engineering at the Catholic University in Paraguay. His research focuses on P4 programming and congestion control protocols.



Dr. Jorge Crichigno is an Associate Professor in the College of Engineering and Computing at the University of South Carolina (USC) and the director of the Cyberinfrastructure Lab at USC. He has over 15 years of experience in the academic and industry sectors. Dr. Crichigno's research focuses on practical implementation of high-speed networks, network security, TCP optimization, offloading functionality to programmable switches, and IoT devices. His work has been funded by U.S. agencies such as the National Science Foundation (NSF) and the Department of Energy. He received his PhD in Computer Engineering from the University of New Mexico in 2009, and his bachelor's degree in Electrical Engineering from the Catholic University in Paraguay in 2004.



Dr. Elias Bou-Harb is the Associate Director of the Cyber Center for Security and Analytics at the University of Texas San Antonio (UTSA), where he leads, co-directs and co-organizes university-wide innovative cyber security research, development and training initiatives. He is also an Associate Professor at the department of Information Systems and Cyber Security specializing in operational cyber security and data science as applicable to national security challenges. Previously, he was a senior research scientist at Carnegie Mellon University (CMU) where he contributed to federally-funded projects related to critical infrastructure security and worked closely with the Software Engineering Institute (SEI). His research and development activities and interests focus on operational cyber security, attacks' detection and characterization, malware investigation, cyber security for critical infrastructure and big data analytics.