Received January 19, 2021, accepted February 5, 2021, date of publication February 24, 2021, date of current version March 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3061696

Understanding of BBRv2: Evaluation and Comparison With BBRv1 Congestion Control Algorithm

YEONG-JUN SONG[1], GEON-HWAN KIM[1], IMTIAZ MAHMUD[1], WON-KYEONG SEO[2], AND YOU-ZE CHO[1], (Senior Member, IEEE)

[1]School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, South Korea
[2]Department of Military Electronic Communication, Yeungjin University, Daegu 41527, South Korea

Corresponding author: You-Ze Cho (yzcho@ee.knu.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2018R1A6A1A03025109, in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant NRF-2019R1A2C1006249, and in part by the BK21 FOUR Project funded by the Ministry of Education, South Korea, under Grant 4199990113966.

ABSTRACT Google proposed a bottleneck bandwidth round-trip propagation time (BBR) for TCP to replace the loss-based congestion control algorithms, such as Reno and CUBIC. Unlike the loss-based algorithms, BBR models a network path from source to destination and dynamically controls the sending rate using control parameters, such as pacing rate, congestion window, and quantum to achieve high throughput and low latency. However, many studies have reported performance issues in BBR operation, such as excessive packet loss in shallow buffers, the unfairness among different RTT flows, the unfairness with loss-based algorithms, and so on. Google is developing a new version of BBR, BBRv2, to resolve these performance issues. In this study, we evaluate and compare two versions of BBR on a Mininet emulator and a physical testbed, focusing on whether the BBRv2 alpha can alleviate the performance issues of BBRv1 and whether other novel issues will arise in BBRv2. The experiment results show that BBRv2 improves the unfairness and aggressiveness in small buffer less than 1 BDP. Moreover, multiple BBRv2 flows not only show better fairness in bandwidth sharing, but also reduce the amount of packet retransmissions. However, we observed that the challenging issues such as RTT unfairness, coexistence with loss-based algorithms, and synchronization between BBRv2 flows still exist. This study explores BBRv2's current behavior in various network scenarios and compares the performance of BBRv2 with the BBRv1 congestion control algorithm.

INDEX TERMS TCP congestion control, BBRv1, BBRv2, fairness, retransmission.

I. INTRODUCTION

The TCP congestion control was introduced in 1980s, and it interpreted packet loss as a network congestion [1]. However, on the current Internet, as the bandwidth and buffer sizes have continuously evolved, the relationship between packet loss and congestion has become more tenuous. In particular, when the bottleneck buffer sizes are large, the loss-based congestion control, such as Reno [2] and CUBIC [3], causes longer queueing delay due to bufferbloat [4]. When the bottleneck buffer sizes are small, the loss-based congestion control misinterprets short-term loss as a signal of congestion, leading to throughput degradation [5]. In 2016, Google proposed a bottleneck bandwidth and round-trip propagation time version 1 (BBRv1) to replace the loss-based congestion control algorithms, thus attempting to achieve high throughput while limiting queueing delay [6]. In contrast to traditional congestion control algorithms, BBRv1 measures the maximum delivery rate (`BtlBw`) and the minimum round-trip time (`RTprop`), and models the network path using these parameters. Based on the bandwidth delay product (BDP), which matches the product of `BtlBw` and `RTprop`, three control parameters are calculated to control the sending rate: pacing rate, congestion window, and quantum. Therefore, BBRv1 is called a model-based or rate-based congestion control algorithm. However, many studies have found the following issues in BBRv1 behaviors:

- **Aggressive startup operation:** In startup phase, the consecutive increase in the sending rate creates

The associate editor coordinating the review of this manuscript and approving it for publication was Rentao Gu.

a long-standing queue; thus, it completely suppresses existing TCP flows. [7].

- **Aggressive bandwidth probing:** Hock *et al.* [8] reported that each BBRv1 flow overestimates the delivery rate in the coexistence of multiple BBRv1 flows, resulting in the creation of a standing queue link of more than 1.5 BDP. In particular, BBRv1 flows cause excessive packet loss when the buffer sizes are smaller than 1 BDP and they cannot avoid this congestion. This aggressiveness in bandwidth probing suppresses other TCP flows, resulting in unfairness in bandwidth sharing.
- **round-trip time (RTT) unfairness:** Hock *et al.* [8] also reported that if BBRv1 flows with different propagation delays share a common bottleneck link, a long RTT flow takes more bandwidth compared to a short RTT flow. This is because a long RTT flow calculates a higher BDP, and therefore, it injects more data into the network than a low RTT flow.
- **Unfairness with loss-based congestion control:** Hock *et al.* [8] and Scholz *et al.* [7] observed that the throughput between BBRv1 and loss-based congestion control algorithms depends on the bottleneck buffer size.

To alleviate these operating issues in BBRv1, Google introduced a second version of BBR, called BBRv2, in 2018 [9], which is still under development at the time of writing. Table 1 summarizes the differences between the two versions of BBR. BBRv2 calculates the packet loss and ECN [10] rates and uses them as the congestion signal. For example, BBRv2 limits the inflight data using the maximum boundary, called `inflight_hi`, when it detects packet loss and ECN signals over a predefined threshold (`loss_threshold` = 2%, `ECN_threshold` = 50%) [11]. In addition, BBRv2 modifies the size of `cwnd` in ProbeRTT phase. Drastic `cwnd` change in BBRv1's ProbeRTT phase caused fluctuation in throughput. Therefore, BBRv2 reduced the `cwnd` by BDP/2 to alleviate the throughput fluctuation.

Google reported that BBRv2 improves the fairness with the loss-based congestion control algorithm and reduces the number of packet retransmissions when buffer sizes are small. In particular, BBRv2 shows better performance in bandwidth sharing and in reducing the standing queue using ECN than that without using ECN [12]. Moreover, several studies observed that BBRv2 mitigates the unfairness that BBRv1 experienced, such as inter-protocol fairness and RTT fairness [13]–[15]. However, recent studies have focused on the BBRv2's improvements compared to BBRv1 without considering deterioration. Therefore, we investigate the improvement of RTT fairness and the inter-protocol fairness in coexistence with the loss-based congestion control algorithm, and observed the issues of the performance degradation of BBRv2 compared to that of BBRv1.

We configure a Mininet emulator and a physical testbed environment to perform the experiment, evaluate the throughput, and trace the internal parameters, such as estimated bottleneck bandwidth, congestion window, inflight data, packet retransmission, and buffer backlog, to observe the behavioral

TABLE 1. Comparison of BBRv1 and BBRv2 [11].

	BBRv1	BBRv2
Model Parameters	`BtlBw`, `RTprop`	`BtlBw` `RTprop` loss_rate Aggregation
Packet loss as congestion signal	N/A	loss_rate exceeds loss_threshold
ECN	N/A	DCTCP/L4S style ECN
Exit startup	throughput plateaus	throughput plateaus or loss/ECN rate exceeds threshold
cwnd in ProbeRTT	4 packets	BDP/2

characteristics in the two versions of BBR. Based on the experimental results, we analyze whether the performance issues in BBRv2 have been resolved and investigate the new problems arising in BBRv2. The following are the key observations from our experiments:

- BBRv2 controls the amount of inflight data using the packet loss rate and ECN signal rate, resulting in a significantly reduced the number of packet losses compared to BBRv1.
- BBRv2 fails to quickly probe the available bandwidth in a network environment where the bandwidth dramatically changes, thus, leading to a low link utilization.
- If two identical BBRv2 flows enter the same bottleneck link at different times, the convergence between the throughput of two flows relies on the bottleneck buffer size. With small buffers below 0.3 BDP, the throughput of BBRv2 flows shows better intra-protocol fairness. However, when buffer sizes are large enough, the flow that started later cannot achieve fair bandwidth sharing.
- With small buffers of less than 1 BDP, the unfairness between BBRv2 flows with different RTTs is alleviated compared to that for BBRv1 and the number of packet retransmissions greatly decreases. However, in case where buffer sizes are large enough not to be affected by the buffer overflow, a long RTT flow still occupies more bandwidth than a short RTT flow.
- With buffers below 2 BDP, BBRv2 shows better inter-protocol fairness with CUBIC than BBRv1. However, as the buffer sizes increase, CUBIC still occupies more bandwidth, resulting in decreased inter-protocol fairness.

The remainder of this paper is organized as follows. Section II reviews the related work, considering BBRv1 issues and the current state-of-the art studies on BBRv2. The BBRv2's behavior is discussed in Section III.

Section IV presents and analyzes the experimental results. Finally, Section V summarizes and concludes the paper.

II. BBR CONGESTION CONTROL EVOLUTION

A. STANDARDIZATION OF BBR

At Internet Engineering Task Force-97 (IETF-97) [6], Google presented BBR behavior, attempting to find the optimal operating point and providing experimental results to compare BBR with CUBIC. Google noted that BBR outperforms the traditional algorithms, such as CUBIC and Reno, in terms of throughput and latency.

At IETF-98 [16], Google introduced three significant changes that are considered to improve the performance of BBR and stated various research topics and opportunities of BBR. Furthermore, an experiment of fairness with CUBIC and Reno demonstrated contradictory results based on the bottleneck buffer size. In this meeting, Google sought to modify the behavior of BBR to reduce latency and packet loss, to coexist with loss-based congestion control in the shallow buffers, and to mitigate the problem of RTT fairness between BBR flows.

At IETF-99 [17], Google reported that the quick user datagram protocol Internet connection (QUIC) on YouTube is used by applying the BBR, and they published an Internet draft for the estimation of the delivery rate [18] and BBR congestion control algorithm [19].

At IETF-100 [20], Google reported that testing for BBRv2, which changed the behavior of the algorithm, was in progress to improve in problems reported in studies that followed BBRv1's release. The testing focused on improving the performance of BBR by reducing the high retransmission rate in the shallow buffer pointed out in BBRv1 and improving inter-protocol fairness.

The changes to BBRv2 on IETF-102 [9] and IETF-104 [11] were ways to adjust bandwidth probe cycles to improve the inter-protocol fairness, to leave extra space to reduce packet loss and latency, to exit the startup phase faster, to limit an interval of the ProbeRTT phase to 2.5 s, and to reduce a `cwnd` to BDP/2 not the four packets during the ProbreRTT phase.

At IETF-105 [12], Google released BBRv2 alpha/preview open-source code to encourage researchers to dive in and help evaluate and improve BBRv2. They described the improvements to BBRv2 from BBRv1. BBRv2 improved coexistence when the bottleneck was shared with Reno or CUBIC, and it reduced the loss rate for the bottleneck buffer sizes of less than 1.5 BDP. Moreover, it achieved high throughput for a path with high degrees of aggregation and reduced the throughput reduction in ProbeRTT.

At IETF-106 [22], Google ensured that BBRv2 suitable as a general-purpose congestion control for LAN, WAN, and data center. In particular, they tuned its performance to enable full-scale rollout at Google and to improve the algorithm to scale for large numbers of TCP flows

At IETF-109 [23], they introduced the BBR.Swift approach, that uses delay as a congestion signal. BBR.Swift is a congestion control algorithm scheme for a data center applying a mechanism to accurately measure and separate the network RTT and host delay of the Swift congestion control algorithm [24]. They mentioned that BBR.Swift achieves higher fairness among flows that use the same congestion control algorithm and reduces the retransmission rate. Moreover, they stated that BBRv2 is under experimentation for various proportional rate reduction (PRR) [25] responses in loss recovery.

B. RELATED WORK

After the BBRv1 algorithm was released by Google, various studies to evaluate it were presented. Hock *et al.* [8] evaluated the performance of BBR in terms of intra-protocol, RTT fairness, inter-protocol fairness, and packet loss in various network scenarios. Scholz *et al.* [7] implemented an emulation-based reproducible experimental framework and verified it. It was evident that BBRv1 cannot achieve a maximized delivery rate and minimized latency.

Atxutegi *et al.* [26] evaluated BBR on emulation and real mobile networks. Parichehreh *et al.* [27] systemically evaluated and analyzed BBR in an LTE uplink with multiple concurrent flows. They observed a lack of fairness among simultaneous flows and excessive packet loss. Dai *et al.* [28] argued that BBR does not achieve the professed fairness when competing with loss-based flows, and it does not perform well over real-world Ethernet and 4G mobile networks. Nguyen *et al.* [29] investigated TCP variants' performance on the IEEE 802.11ad wireless link, finding that BBR had excellent performance under a high-loss scenario.

Crichigno *et al.* [30] compared the performance of BBR against loss-based algorithms in a 10 Gbps network in the presence of packet loss and latency. Empirical results demonstrated that BBR reacts better than a loss-based congestion control algorithm (CUBIC, Reno, and HTCP) to a large maximum segment size (MSS). Wang *et al.* [31] evaluated the performance of QUIC with BBR via geostationary orbit (GEO) satellite Internet access on a dedicated network emulator. Moreover, QUIC with BBR helps shorten the transmission delay and maintain the transmission speed when the packet-loss rate grows. Zhang *et al.* [32] evaluated the performance of BBR based on network simulator 3 (NS-3).

From the question of whether TCP will work in a mmWave cellular system, Zhang *et al.* [33] showed that the performance of TCP, including BBR, on mmWave links highly depend on different parameter combinations. Kumar *et al.* [34] experimented on an mmWave point-to-point link operating at 60 GHz and observed significant throughput loss with BBRv1. This performance degradation was caused by the link RTT variation that results in incorrect estimates of the bandwidth and minimum RTT.

Based on studies evaluating BBR in various environments, a BBR-ACD to reduce packet loss occurring in a shallow buffer [36] and a BBRp for improving the performance of BBR in a wireless local area network (WLAN) have been proposed [37].

Several papers [38]–[42], [45] deal with problems that can occur when comparing between CUBIC and BBR, and studies to solve them have been introduced [43], [44], [46]–[49]. Furthermore, several studies introduced the causes and solutions to the RTT fairness problem in BBR where high RTT flows occupy more bandwidth [50]–[55]. Moreover, efforts have been made to apply BBR, which is based on the model, not the loss, to multipath TCP [56]–[58].

Recently, Zhang *et al.* [13] discussed variants of BBRv1, designed to resolve the operating problems of BBRv1. They evaluated and compared the variants, including BBRv2. They reported that BBRv2 shows improvement in RTT fairness when compared with BBRv1 and that it achieves better coexistence with CUBIC and Reno in terms of inter-protocol fairness. However, with a 5% random loss rate, BBRv2 experiences low channel utilization.

Furthermore, Gomez *et al.* [14] presented an experimental evaluation of BBRv2 using a Mininet emulation. They demonstrated that the coexistence between BBRv2 and CUBIC is better than that between BBRv1 and CUBIC. They also reported that BBRv2 mitigates the RTT unfairness problem and achieves a better fair share of the bandwidth compared to BBRv1 when network conditions, such as bandwidth latency, dynamically change.

Nandagiri *et al.* [15] presented an experimental evaluation of BBRv1 and BBRv2 in terms of the fairness, queueing delay, and link utilization. They reported that the performance of BBRv1 is limited only by its inflight cap, however, BBRv2 can overcome the limitation of BBRv1 in networks with small buffers using ECN.

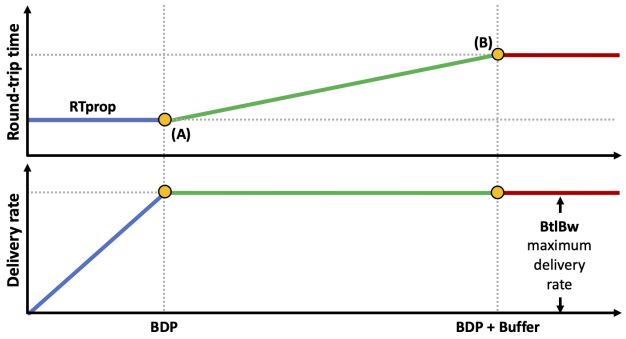


FIGURE 1. Delivery rate and round-trip time according to the inflight data [5].

III. ANALYSIS OF BBRv1 AND BBRv2 BEHAVIORS

A. MODEL-BASED CONGESTION CONTROL

Fig. 1 shows the delivery rate and the round-trip time according to the inflight data. The loss-based congestion control algorithms interpret the packet loss as a signal of network congestion. They continuously increase the congestion window to increase the sending rate, and reduce the congestion window rate when detecting packet loss. Therefore, they operate at point (B), where they can achieve the maximum delivery rate, although they cannot provide the minimum latency due to queuing delay. However, on the current Internet, where the bandwidth and the buffer sizes have continuously increased, the relationship between packet loss and network congestion has become more tenuous. In particular, the loss-based algorithms experience performance degradation, such as bufferbloat and the resulting long latency.

Therefore, BBR congestion control algorithm, proposed by Google, strives to achieve high throughput but low latency by limiting the queue growth. This algorithm tries to operate on the point (A), called Kleinrock's optimal operating point [59], which was proven to provide the maximum throughput and the minimum latency on the bottleneck.

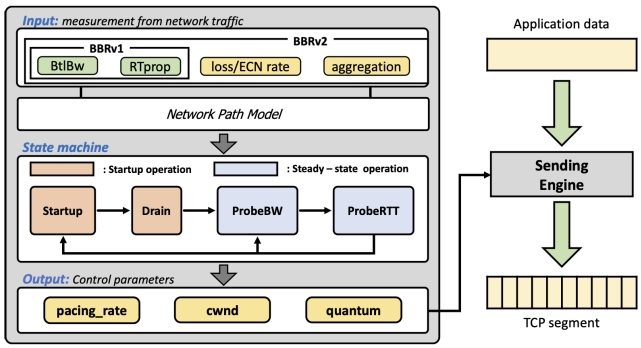


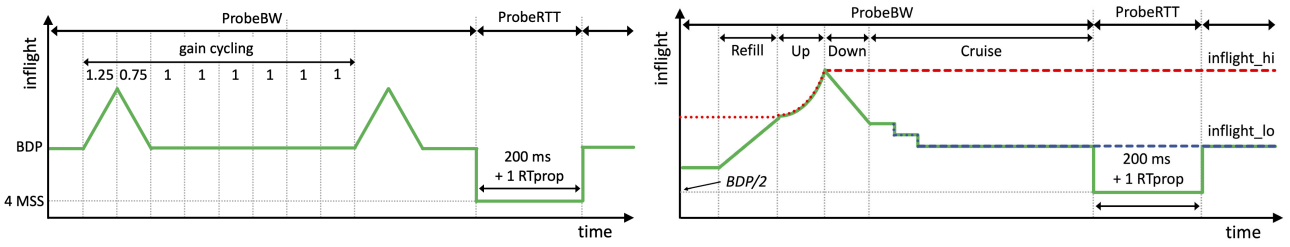
FIGURE 2. State machine and network path model in BBR congestion control.

Fig. 2 depicts how two versions of BBR work in TCP congestion control. They commonly measure the parameters of the network path model through state machines comprising four operating states. BBRv1 uses only `BtlBw` and `RTprop` as components of the network path model; in contrast, BBRv2 utilizes loss/ECN rate and aggregation aiming to build a more accurate link model. The two versions of BBR control the transmission rate by calculating three output parameters: `pacing_rate`, `cwnd`, and `quantum` based on the configured network path model.

Next, the operating modes in BBRv1 and BBRv2 are discussed. They commonly have two operating states; the startup and steady-state operations. The startup operation is divided into Startup and Drain phases, and the steady-state operation is divided into ProbeBw and ProbeRTT phases.

B. STARTUP OPERATION

[Startup] In this phase, both BBRv1 and BBRv2 exponentially increase the sending rate to quickly measure the available bandwidth. Therefore, they use a `pacing_gain` of 2/ln2 to double the `pacing_rate` for each RTT (for each unit of time interval). Then, if the delivery rate sample does not increase by more than 25% for the three ACK samples, both versions of BBR consider that the available bandwidth has been found and terminate the startup phase. However, if BBRv1 performs the startup phase on a small buffer of below 1 BDP, the excess data of up to 2 BDP can cause



(a) BBRv1 (b) BBRv2

FIGURE 3. Steady-state operation in BBR congestion control.

the buffer to overflow, which can cause excessive packet retransmission and unfairness with other TCP flows.

To reduce the aggressiveness in bandwidth probing, BBRv2 uses loss and ECN rate as the congestion signal. If BBRv2 detects packet loss or ECN-marked rate higher than the predefined threshold, it stops increasing the inflight data and exits the startup phase. Then, BBRv2 sets the `inflight_hi` where it prevents inflight data from exceeding this maximum boundary until the new `inflight_hi` is updated.

[Drain] After terminating the startup phase, both BBRv1 and BBRv2 enter the drain phase to empty the excess queue filled during the startup phase. The drain phase uses the inverse of the startup's `pacing_gain`, and both BBRs end this phase when the inflight data are equal or less than 1 BDP.

C. STEADY-STATE OPERATION

[ProbeBW] Fig. 3(a) shows that BBRv1 circulates through eight cycles {1.25, 0.75, 1, 1, 1, 1, 1, 1}, called ProbeBW gain cycles, to periodically probe the available bandwidth. First, BBRv1 increases the sending rate using the `pacing_gain` of 1.25 to probe for more bandwidth. It maintains this phase during `RTprop`, gradually, increasing the amount of inflight data. However, if the inflight data reaches 1.25 BDP or packet loss is detected, BBRv1 immediately stops increasing the inflight data regardless of the elapsed time. Second, BBRv1 decreases the sending rate using a `pacing_gain` of 0.75 to drain the excess queue. This value of `pacing_gain` lasts until either a full `RTprop` elapses or the amount of inflight data decreases below 1 BDP. Finally, for the remaining six cycles, with a `pacing_gain` of 1, BBRv1 sends out packets at a constant `pacing_rate`, which is calculated by `pacing_gain` × `BtlBw` × `RTprop`, and each `pacing_gain` lasts for 1 `RTprop`.

The ProbeBW phase has the following characteristics: one is the randomization of `pacing_gain` and the other is max_filter for the bandwidth estimation.

First, BBRv1 randomizes the initial `pacing_gain` of 1.25 in the gain cycling of ProbeBW. This is followed by the `pacing_gain` of 0.75 to drain the excess queue. Randomized behavior improves the fairness with other BBR flows and reduces the queues when multiple BBR flows share the same bottleneck link.

Second, BBRv1 uses the max_filter to estimate the available bandwidth and calculates the delivery rate by dividing the amount of data delivered by the time elapsed.

However, the delivery rate samples are normally below the actual bandwidth of the link because of the several noises, such as the random variation in physical processes and the excess queue along the path. Thus, BBRv1 adopted this max_filter to estimate the `BtlBw` close to the actual bandwidth. It estimates the available bandwidth using a windowed maximum recent delivery rate sample over the past 10 round-trip times.

However, in multiple BBRv1 flows, each sender overestimates the available bandwidth because of the max_filter that is immediately applied as new sending rate; therefore, the total amount of data sent from BBR hosts exceeds the pipe's capacity, resulting in the standing queue creation. If the buffer sizes are not large enough to store excess data, the packet loss occurs. Moreover, this packet loss persists throughout the entire bandwidth sharing of multiple BBRv1 flows because BBRv1 does not reduce the maximum boundary of the inflight data, no matter how much packet loss occurs. This maximum boundary of the above-mentioned inflight data is the inflight cap, calculated as 2 BDP, and it is used to maintain the maximum throughput and low latency.

To resolve the BBRv1's limitation, BBRv2 uses additional parameters, such as the loss/ECN signal rate, to recognize the entrance into a treacherous region in the inflight data. As depicted in Fig. 3(b), BBRv2 probes the available bandwidth by periodically increasing the amount of inflight data; however, BBRv2 differently limits the growth of the inflight data. First, when BBRv2 starts probing the bandwidth, it linearly increases the amount of inflight data during 1 `RTprop` in ProbeBW:Refill phase, which fills the pipe. Thereafter, BBRv2 performs the ProbeBW:Up phase during which it exponentially increases the amount of inflight data to quickly probe the bandwidth. This growth of the inflight data is terminated when the amount of inflight data reaches 1.25 times of BDP or when the packet loss or the ECN rate exceeds the threshold predefined in the implementation. In particular, BBRv2 sets the `inflight_hi` when it experiences higher loss/ECN rate than the threshold, so that ensures that it does not exceed the operating point where the excessive packet loss may occur. After exiting the ProbeBW:Up, BBRv2 enters the ProbeBW:Down phase to compensate the queue, and cruises

at a constant delivery rate until the next bandwidth probing begins.

Therefore, two versions of BBR behave differently when new TCP flows enter the bottleneck link or the available bandwidth rapidly decreases. BBRv1 does not directly reduce the sending rate. If the delivery rate decreases due to the bandwidth occupancy by other TCP flows, then BBRv1 flow measures the lower bottleneck bandwidth and calculates the lower BDP, resulting in reduction of the sending rate after 10 RTTs. On the other hand, BBRv2 uses more information, such as loss rate and ECN-marked rate, to recognize the link's congestion. When other TCP flows enter, packet loss may occur. In this case, BBRv2 has safeguards to limit the inflight data, such as `inflight_hi` and `inflight_lo`. When the packet loss rate exceeds the threshold in the ProbeBW:Up, BBRv2 sets the `inflight_hi`, so that it provides the bandwidth for other flows to share the bottleneck. When BBRv2 experiences packet loss in the ProbeBW:Cruise, it uses `inflight_lo` to cope with the temporary packet loss, and the amount of inflight data is limited by `inflight_lo` until the next ProbeBW:Up phase. This allows BBRv2 to adapt to the changing network environment by temporarily restricting inflight data during one cruising period before updating a new path model.

[ProbeRTT] Both BBRv1 and BBRv2 always maintain the timestamp when `RTprop` was last measured, and if the newly measured RTT is lower than the existing `RTprop`, they immediately update the `RTprop` and record the timestamp when a new `RTprop` is measured. However, if the new `RTprop` is not measured for a predefined time (BBRv1: 10 s, BBRv2: 5 s), BBR is forced to enter the ProbeRTT phase. As soon as BBR enters the ProbeRTT phase, it initializes and sets the shortest value of RTT measured during this ProbeRTT phase. Initially, BBRv1 sets the `cwnd` to only four packets in the ProbeRTT phase. However, this approach causes a drastic change in `RTprop` resulting in throughput fluctuation in coexistence with loss-based congestion control algorithms. Therefore, in the BBRv2 implementation, the `cwnd` size in ProbeRTT is set to BDP/2.

IV. EXPERIMENTAL RESULTS AND EVALUATION

A. EXPERIMENTAL ENVIRONMENT SETUP

We configured a Mininet emulator and a physical testbed using the host system (6-core Intel 3.6 GHz CPU and 32 GB memory, 2.5G NIC) and the switch system (Intel core i5 3.0 GHz CPU, 16 GB memory). In the host systems, we installed the 5.4.0-rc6 version of the Linux kernel including the implementation of BBR v2 alpha/preview, updated on Nov. 22, 2019 [21].

The Mininet emulation environment was configured only the host system. We implemented the Mininet framework to perform the experiment, to trace the parameters, and to analyze the results, referring to the measurement framework by Scholz *et al.* [60]. To evaluate the behavior in various network scenarios, we configured a dumbbell topology, in which multiple TCP hosts shared the same bottleneck link, as shown in Fig. 4. This framework was used only in the experiments where the bottleneck bandwidth was below 100 Mbps. We used three switches to configure the bottleneck for clearly separating the functions of the network configuration and evaluating the performance.

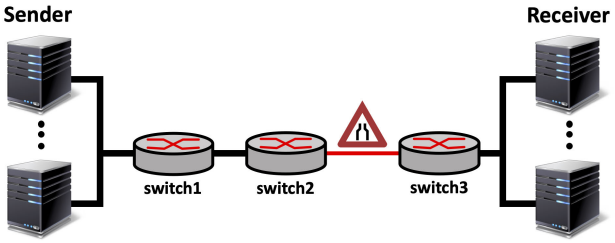


FIGURE 4. Experimental topology setup.

Furthermore, we constructed a physical testbed to perform the experiment with gigabit rate. We set the bottleneck bandwidth to 1 Gbps using *ethtool* and emulated the bottleneck buffer size and delay using *netem*. Two sending hosts measured the throughput over time using *TCPlog* [64].

In both test environments, all hosts set the TCP memory size to the maximum so that the performance was not constrained by the host memory size used in the system (`rmem_max`=250,000,000, `tcp_rmem`='4,096, 131,072, 250,000,000', `wmem_max`=250,000,000, `tcp_wmem`='4,096, 16,384, 250,000,000', referring to Google's test configuration. Finally, each sending host on the left side in Fig. 4 sent a TCP segment to each receiving host using the *iperf* [65] application with different durations and start times that depended on each test scenario.

B. PERFORMANCE EVALUATION FOR SINGLE FLOW

1) ACCORDING TO BOTTLENECK BUFFER SIZE

To compare the single flow operation between BBRv1 and BBRv2 according to the bottleneck buffer size, we constructed an experiment in which the bottleneck bandwidth and the end-to-end round-trip propagation time were set to 50 Mbps and 30ms, respectively. The bottleneck buffer size was varied from 0.1 to 16 BDP. Each congestion control algorithm flow sends data for 50 s. Fig. 5 shows the throughput,

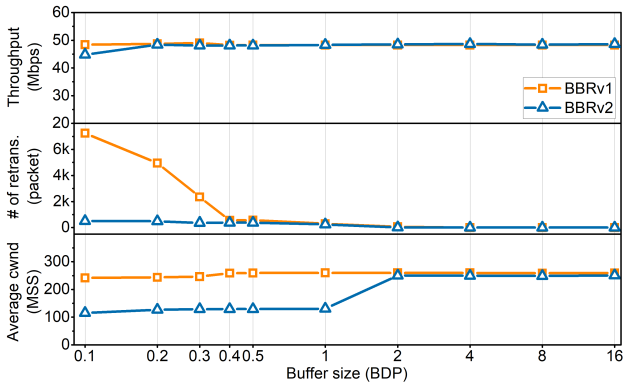
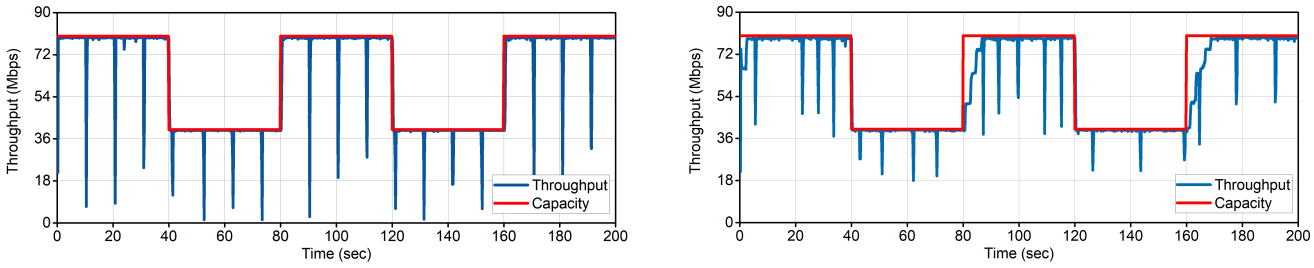
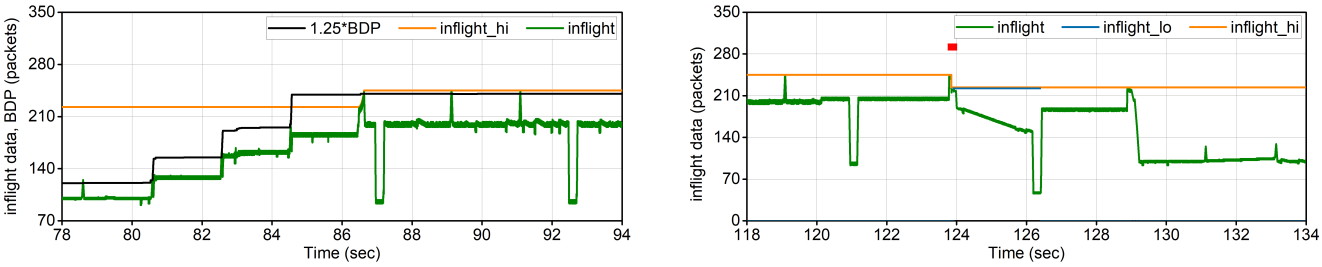


FIGURE 5. Throughput, number of retransmissions, and average `cwnd` of single BBR flow according to bottleneck buffer size.



(a) BBRv1 (b) BBRv2

FIGURE 6. Throughput according to dynamic change of bottleneck bandwidth.



(a) increasing bandwidth (b) decreasing bandwidth

FIGURE 7. The inflight data of BBRv2 according to bandwidth change.

total number of packet retransmissions, and average `cwnd` for BBRv1 and BBRv2.

BBRv1 achieves the maximum throughput regardless of the bottleneck buffer size. However, it experiences excessive packet loss when the buffer size is smaller than 0.3 BDP. This is because BBRv1 periodically increases the inflight data in the ProbeBW phase to probe the available bandwidth. If the buffer size is larger than 0.25 BDP, this excess data can be stored in the bottleneck buffer without buffer overflow. Otherwise, the excess data will be dropped, resulting in continuous packet loss.

BBRv2 not only achieves high bandwidth utilization except when operating on the bottleneck buffer of 0.1 BDP, but it also reduces the number of packet retransmissions compared to the BBRv1. When BBRv2 detects a high packet loss rate, it sets the `inflight_hi`, the maximum boundary where the inflight data can increase. As a results, the average `cwnd` of BBRv2 are lower than those of BBRv1 only when the buffer sizes are small.

2) ACCORDING TO BOTTLENECK BANDWIDTH CHANGE

To evaluate how quickly BBRv1 and BBRv2 adapt to the environment where the bottleneck bandwidth is dramatically changed, we configured an environment where the bandwidth switched between 40 and 80 Mbps every 40 s. In this experiment, the end-to-end round-trip propagation time was 30 ms, and the bottleneck buffer size was fixed to 100 packets (= 1 BDP on links with 40 Mbps bandwidth and 30 ms latency).

The evaluation results are presented in Fig. 6. BBRv1 exhibits a high link utilization by rapidly adapting to both the bandwidth increase (t=80 s) and the decrease (t=120 s) scenarios. BBRv1 frequently probes the bandwidth by updating the `pacing_gain` every `RTprop` within eight pacing cycles, therefore, it can quickly update the BDP according to the changing bandwidth. When the bandwidth increases, BBRv1 measures more bandwidth of about 25% through one pacing cycle, and it consumes three pacing cycles to find the full bandwidth in the scenario where the bandwidth doubles. As soon as the bandwidth decreases, the inflight data previously sent from BBRv1 creates a long-standing queue, resulting in packet loss. However, BBRv1 can quickly avoid self-inflicted congestion by updating the BDP through fast bandwidth measurement, thus reducing the amount of inflight data injected into the network.

In contrast, BBRv2, which operates on the link where the bandwidth is dramatically changed, takes longer than BBRv1 to fully probe the doubled bandwidth, as shown in Fig. 6(b). In particular, when the bandwidth increase from 40 to 80 Mbps, BBRv2 takes about 8 s to reach the full bandwidth. Fig. 7 presents the `cwnd`, the `inflight_hi`, and the amount of inflight data. After BBRv2 performs ProbeBW:Refill, it linearly increases the inflight data during one `RTprop` and tries to probe the available bandwidth by exponentially increasing the inflight data in ProbeBW:Up phase. However, as in Fig. 7(a), the ProbeBW:Up phase is terminated without the exponential growing of the inflight data. Therefore, it fails to find the full bandwidth in one cycle and repeats the same cycle three times because the

ProbeBW:Up phase is terminated immediately after the inflight data exceeds the previously set 1.25 × BDP, even though a new BDP has not been updated as shown in Fig. 7(a). In addition, BBRv2 repeated the ProbeBW:Up phase once after random intervals (2 to 3 s), so that it takes quite a long time to probe the full bandwidth.

Fig. 6(b) shows that BBRv2 quickly adapts to the new test environment where the bandwidth decreases from 80 to 40 Mbps in terms of throughput. However, BBRv2 takes about 9 seconds to operate at the optimal operating point that provides the maximum throughput and the minimum latency as shown in Fig. 7(b). When BBRv2 detected the packet loss, it sets the `inflight_lo` to temporarily reduce the amount of inflight data and waits for the lower BDP to be calculated. That is, BBRv2 controls the amount of inflight data depending on the `inflight_lo` until it estimates the lower bottleneck bandwidth close to the actual bottleneck bandwidth.

3) ACCORDING TO RANDOM PACKET LOSS

We measured the average throughput of BBRv1, BBRv2, and CUBIC that operated on the link where the random packet loss rate ranging from 0.000001% to 10% occurred, and Fig. 8 shows the results of the experiments. CUBIC shows a significant performance degradation despite a small packet loss rate of 0.01%. That is because CUBIC recognizes the packet loss as the network congestion signal and repeatedly reduces the congestion window. Unlike the loss-based congestion control algorithm, BBRv1 does not directly reduce the congestion window size no matter how much packet loss occurs. Hence, BBRv1 achieves a high data rate despite the high packet loss rate. However, this operating characteristic caused BBRv1's aggressiveness in the coexistence with other TCP flows. To compromise between the aggressiveness and robustness, BBRv2 reduces the inflight data if it detects the packet loss that exceeds the predefined threshold (`loss_threshold`=2%). Therefore, BBRv2 shows high link utilization when packet loss rate is less than 2%, and the throughput rapidly decreases in the environment where the packet loss rate is more than 2%.

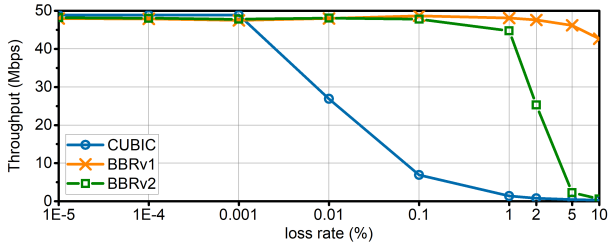


FIGURE 8. Throughput according to random packet loss rate.

C. INTRA-PROTOCOL CONVERGENCE

We configured an experimental environment in which two BBR flows transmit data for 100 s on the common bottleneck link to evaluate how fast the throughput for two flows with different start times converge for the existence of two BBR flows. We set the bottleneck bandwidth to 50 Mbps and 1 Gbps and the round-trip propagation time to 30 ms. The bottleneck buffer sizes were varied from 0.1 to 16 BDP according to the test scenarios. One flow (Flow 1) first started sending data at 0 s, then the other (Flow 2) entered the bottleneck link after 2 s. In addition, Fig. 9 shows the average throughput for each flow repeated 10 times in the same scenario, and Fig. 10 presents the change in throughput, the buffer backlog of Switch 2, and the timestamp of packet retransmissions when each version of BBR flows operated on links of 0.2, 2, and 4 BDP bottleneck buffers.

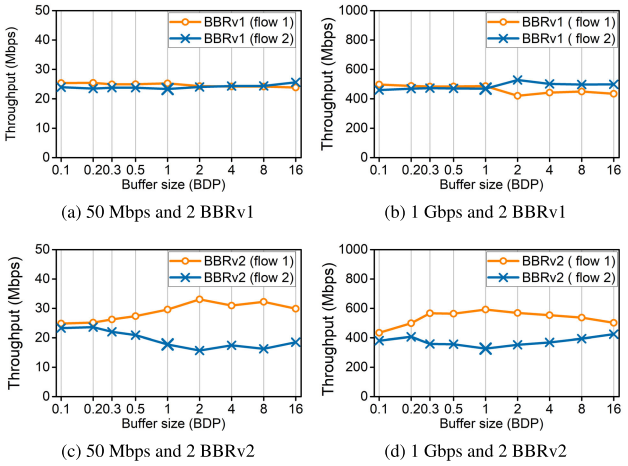
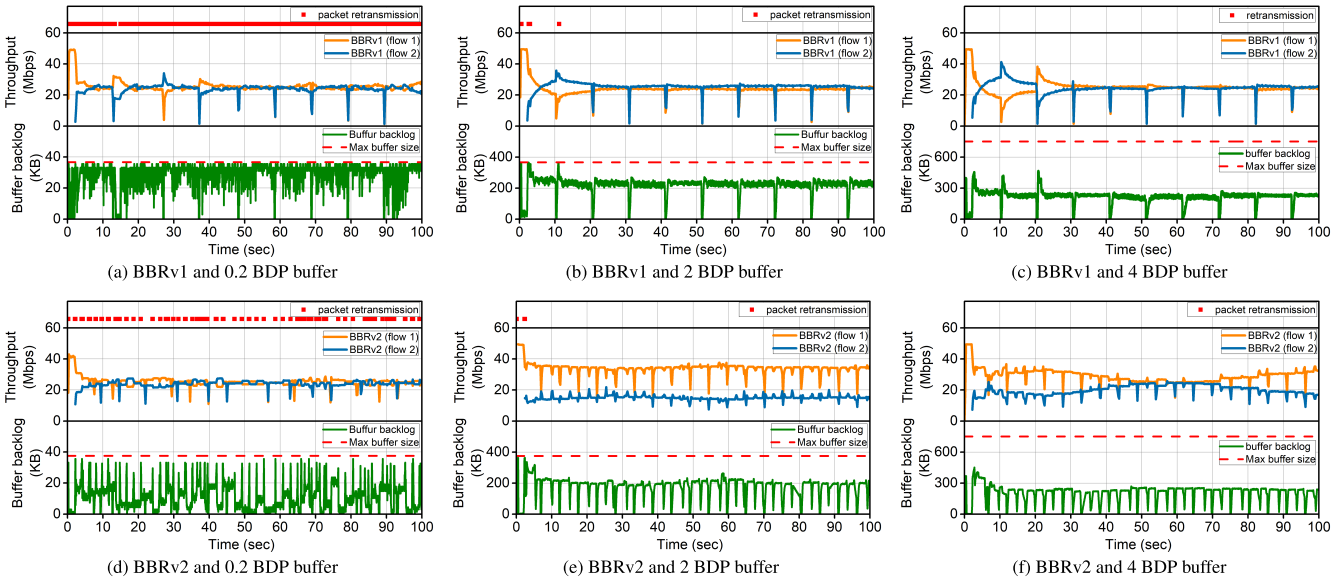


FIGURE 9. Throughput according to bottleneck buffer size when two identical BBR flows start at different times. (Flow 1: 0 second, Flow 2: 2 second).

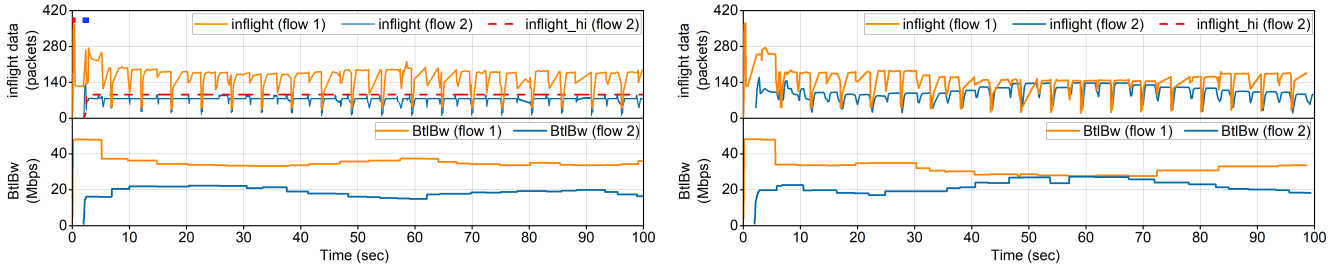
Considering only the throughput, in Fig. 9(a),(b), two BBRv1 flows that originate at different times fairly share the bottleneck link regardless of the bottleneck bandwidth and buffer size because two BBRv1 flows measure similar `BtlBw` and `RTprop` and calculate a similar BDP. Therefore, they inject a similar amount of inflight data into the network. However, the behaviors of the two BBRv1 flows show significant differences depending on the size of the bottleneck buffer when analyzed from the standing queue viewpoint. During the coexistence of two BBRv1 flows, each BBRv1 host overestimates the bottleneck bandwidth to send out about 200 KB more data than the actual BDP of the link as shown in Fig. 10. If the bottleneck buffer is sufficiently large to prevent buffer overflow like Fig. 10(b),(c), no packet retransmissions occur except in the startup phase. Otherwise, BBRv1 flows experience excessive packet retransmissions as described in Fig. 10(a). Moreover, the duration of the throughput fluctuation that occurs when the second flow enters increases with the bottleneck buffer size increases in Fig. 10(b),(c). As the buffer size increases, the size of the `RTprop` increased when the second flow commenced, resulting in a superior delivery rate and long-standing queue until the first flow enters the next ProbeRTT phase. The first flow



(a) BBRv1 and 0.2 BDP buffer (b) BBRv1 and 2 BDP buffer (c) BBRv1 and 4 BDP buffer

(d) BBRv2 and 0.2 BDP buffer (e) BBRv2 and 2 BDP buffer (f) BBRv2 and 4 BDP buffer

FIGURE 10. The timestamp of packet retransmission, throughput and buffer backlog when two identical BBR flows started at different times (flow 1: 0 second, flow 2: 2 second).



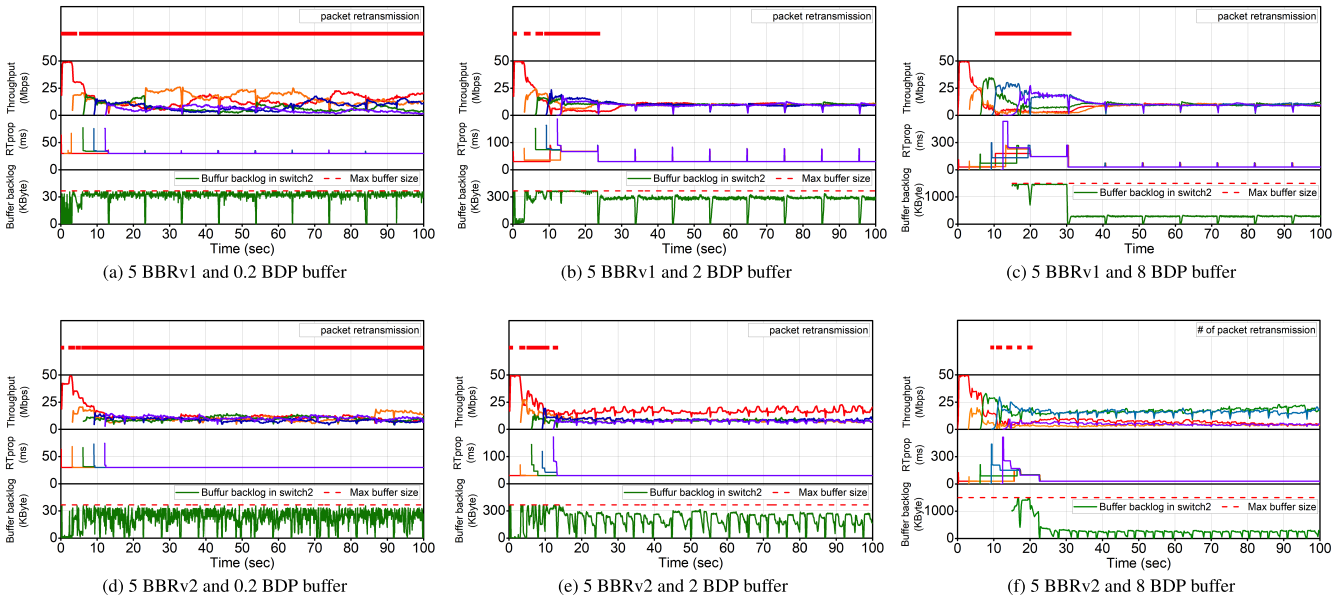
(a) buffer size: 2 BDP (b) buffer size: 4 BDP

FIGURE 11. The amount of inflight data, `inflight_hi`, and `BtlBw` when two BBRv2 flows started at different times.

also measures a higher `RTprop` than the actual round-trip propagation time because the long-standing queue is created by the second flow. This induces the fluctuation between the two BBRv1 flows, which fluctuation is terminated when the two flows measure a similar `RTprop`.

In contrast, when two BBRv2 flows enter the bottleneck link at different times, the throughput between the two flows varied depending on the bottleneck buffer size, as shown in Fig. 9(c),(d). When the buffer sizes are extremely small, such as 0.1 or 0.2 BDP, two BBRv2 flows achieve high fairness and reduce the standing queue compared to BBRv1. For instance, in Fig. 10(d), where the buffer size is 0.2 BDP, they achieve high fairness, as well as reduce the standing queue size and number of packet retransmissions. In other words, with a small buffer of less than 0.25 BDP, they experience packet loss before reaching 1.25 $\times$ BDP. This makes BBRv2 flows behave like loss-based congestion control algorithms.

In buffers greater than 0.25 BDP, the throughput of two flows does not converge during the bandwidth sharing as shown in Fig. 10(e),(f), where the buffer sizes are 2 and 4 BDP. In particular, in case buffer sizes is 2 BDP, the throughput of flow 2 is lower than that of flow 1. The difference between the two results for 2 and 4 BDP buffers is whether the second flow sets the `inflight_hi` in the startup phase. Fig. 11 shows the amount of inflight data, `inflight_hi`, and `BtlBw` for two BBRv2 flows when the buffer sizes are 2 and 4 BDP. In Fig. 11(a), the second-started flow experiences more than 2% of packet loss in the startup phase; therefore, it sets the `inflight_hi`, limiting the increase in inflight data by terminating the ProbeBW:Up phase. Thus, flow 2 cannot measure the higher bandwidth. Consequently, the amount of inflight data of flow 2 is maintained on a similar level for the entire data transmission time, as shown in Fig. 10(e). In Fig. 11(b), where the second flow does not experience packet loss in the startup phase, flow 2 does not set the `inflight_hi`. Therefore, it can probe the available bandwidth more freely than when the buffer size is small. Furthermore, two BBRv2 flows create a length of standing queue similar to that of BBRv1 because each BBRv2 flow measures a bandwidth sample larger than the actual available capacity due to the operating characteristic



(a) 5 BBRv1 and 0.2 BDP buffer (b) 5 BBRv1 and 2 BDP buffer (c) 5 BBRv1 and 8 BDP buffer

(d) 5 BBRv2 and 0.2 BDP buffer (e) 5 BBRv2 and 2 BDP buffer (f) 5 BBRv2 and 8 BDP buffer

FIGURE 12. The timestamp of packet retransmissions, throughput, `RTprop` and buffer backlog when five BBR flows started at different times.

of periodically increasing the sending rate to check whether bandwidth is available.

To evaluate the convergence of multiple BBR flows, we configured the test environment where the bottleneck bandwidth was 50 Mbps and five BBR flows entered the same bottleneck link successively starting every 3 s. Fig. 12 presents the throughput, `RTprop`, and the timestamps for retransmitted packets.

From Fig. 12(a), multiple BBRv1 flows exhibit unfair data transmission. That is because the five flows measure different delivery rates, leading to calculation of different levels of BDP. Moreover, the standing queue they create completely exceeds the bottleneck buffer size, causing the excessive retransmission of 117,097 packets. When sharing the bottleneck link with buffers above 2 BDP, the newly entered flow measures a higher `RTprop` than the actual round-trip propagation time in the startup phase because of the standing queue created by the previous flows. After the synchronization of `RTprop`, the five flows evenly share the bottleneck link, successfully achieving the convergence. Moreover, the time for convergence between flows tends to increase with the bottleneck buffer size. In Fig. 12(b),(c), where the buffer size is 2 and 8 BDP, the flows on the 8 BDP buffer take more time than those on the 2 BDP buffer to completely converge.

As presented in Fig. 12(d), five BBRv2 flows coexisting on the link with a buffer of 0.2 BDP achieve fair bandwidth sharing and reduce the average standing queue and the number of packet retransmissions, only 5,441 packets. This is because five BBRv2 flows set the inflight cap due to packet loss, limiting the inflight data. With 2 and 8 BDP buffers, five BBRv2 flows do not converge, although it takes less time to measure `RTprop` close to the actual round-trip propagation time, as illustrated in Fig. 12(e),(f). In Fig. 12(e), the first flow occupies the most bandwidth among the flows, and the other flows cannot increase the delivery rate because the flows commenced later set the `inflight_hi`, which prevents BBRv2 flows from measuring higher delivery rate samples.

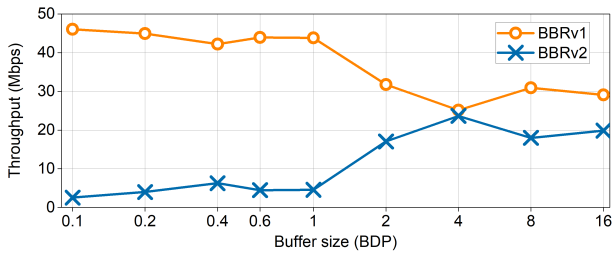
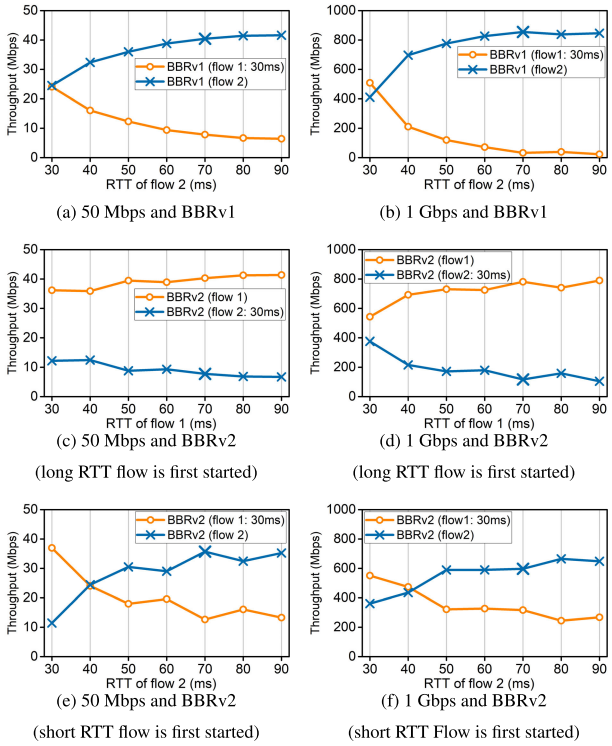


FIGURE 13. Throughput according to the bottleneck buffer size when BBRv1 and BBRv2 share the same bottleneck link.

In addition, Fig. 13 represents the throughput according to the bottleneck buffer size when a BBRv1 and a BBRv2 flow coexist on the same bottleneck link. With a small buffer less than 1 BDP, both BBR experience packet loss because they overestimate the `BtlBw` in startup phase, resulting in the creation of a standing queue and causing packet loss. However, BBRv2 reduces the inflight cap to limit the amount of inflight data by setting the `inflight_hi`; however, BBRv1 does not reduce the inflight cap. Therefore, BBRv1 exerts excess performance in coexistence with two BBR flows. In contrast, BBRv1 and BBRv2 exhibit fair bandwidth sharing in a large buffer of above 1 BDP because both of them only depend only on the BDP to determine the sending rate when packet loss does not occur.

D. RTT FAIRNESS

We configured the test topology where the round-trip propagation time of Flow 1 varied from 30 to 90 ms and that

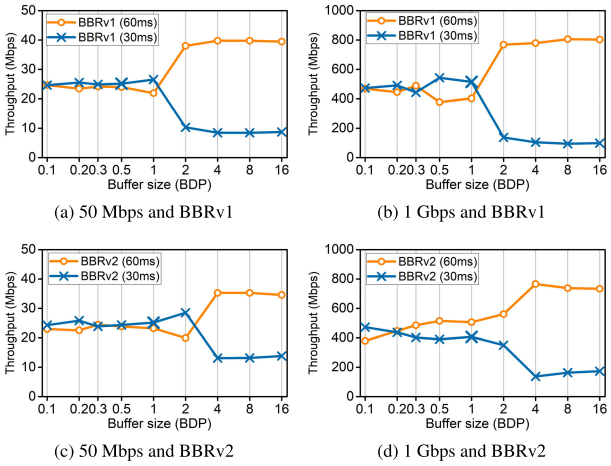


(a) 50 Mbps and BBRv1 (b) 1 Gbps and BBRv1

(c) 50 Mbps and BBRv2 (long RTT flow is first started) (d) 1 Gbps and BBRv2 (long RTT flow is first started)

(e) 50 Mbps and BBRv2 (short RTT flow is first started) (f) 1 Gbps and BBRv2 (short RTT Flow is first started)

FIGURE 14. Throughput according to the flow 2's RTT when two BBR flows with the different RTTs share the same bottleneck link (Flow 1: 30ms, Flow 2: 30-90ms).



(a) 50 Mbps and BBRv1 (b) 1 Gbps and BBRv1

(c) 50 Mbps and BBRv2 (d) 1 Gbps and BBRv2

FIGURE 15. Throughput according to the bottleneck buffer size when BBR flows with the different RTTs share the same bottleneck link (Flow 1: 60ms, Flow 2: 30ms).

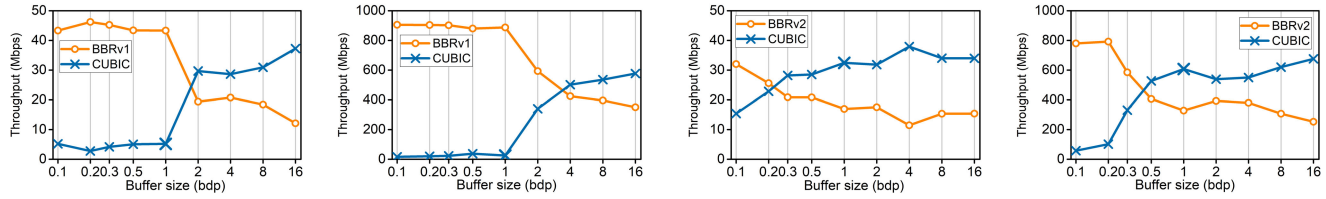
of flow 2 was fixed to 30 ms, to compare the RTT fairness between BBRv1 and BBRv2. The bottleneck bandwidth is set to 50 Mbps and 1 Gbps in the test scenarios. In this experiment, we set the bottleneck buffer to be large enough so that packet loss does not occur. Then, we evaluated the effect of the RTT difference between the two BBRv2 flows in the throughput. Flow 2 started after 2 s. Then, Flow 1 entered the bottleneck link, and the two flows transmitted data for 100 s.

In Fig. 14(a),(b), two BBRv1 flows with the same latency show similar throughput. However, as the latency of flow 2 increases, the long RTT flow occupies more bandwidth. The long RTT flow measures a higher `RTprop` than the short RTT flow; therefore, it calculates a larger BDP. This induces the long RTT flow to probe a higher `BtlBw` than before, and the short RTT flow loses `BtlBw`, resulting in serious unfairness in throughput between the two flows.

Similarly, the BBRv2 flow that has a long latency also occupies more bandwidth than the short RTT flow, as shown in Fig. 14(c),(d). As we described in Section IV.C, we already confirmed that two BBRv2 flows cannot evenly share the same bottleneck link in a sufficiently large buffer. Moreover, we experimented by changing the order in which flow 1 and 2 enter the bottleneck link to confirm that the unfairness between the two flows with different RTTs was not due to the order in which the flows started. The short RTT flow started first, and the long RTT flow entered the bottleneck link 2 s later. The results for the experiments are illustrated in Fig. 14(e),(f). If two flows have the same latency of 30 ms, the flow that started first exhibits higher throughput, which is consistent with the results of the intra-protocol fairness evaluation in Section IV.C. Furthermore, Flow 2, which has a long latency, tends to occupy more bandwidth as the round-trip propagation time of flow 2 increases. Thus, the throughput of the BBRv2 flows still depends on the difference in RTT among the flows.

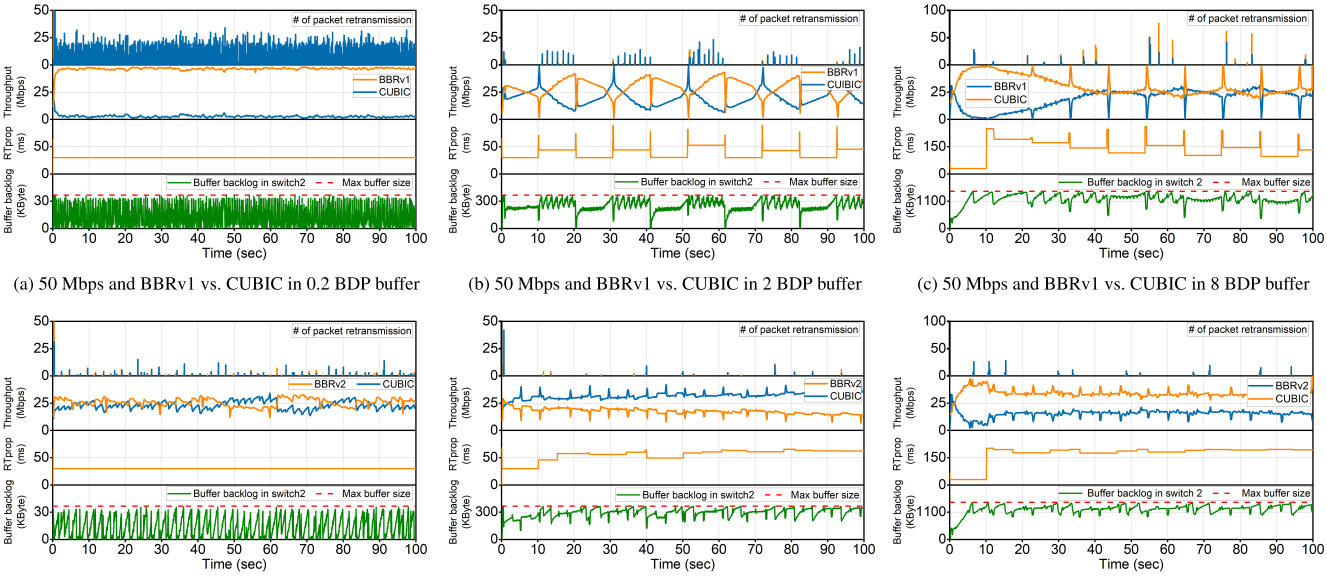
In addition, we configured the test topology to identify the effect of the bottleneck buffer size on RTT fairness, where the round-trip propagation time of flows 1 and 2 are 60 and 30 ms, respectively. The bottleneck buffer size is set from 0.1 to 16 BDP and the bottleneck bandwidth is set to 50 Mbps and 1 Gbps depending on the test scenarios. Fig. 15 shows the average throughput of BBRv1 and BBRv2. In Fig. 15(a),(b), two BBRv1 flows show better fairness but create excessive packet retransmission when the buffer size is smaller than 1 BDP. For example, when two BBRv1 flows coexist on the bottleneck link with a 0.2 BDP buffer and a 50 Mbps bandwidth, they cause 43,016 packet retransmissions in 100 s. Regardless of how fair the two flows are, the aggressiveness can cause network congestion and suppress other TCP flows. As soon as the buffer size increases above 2 BDP, the long RTT flow occupies most of the bandwidth, causing unfairness between the two flows.

During the coexistence of BBRv2 flows with different RTTs, the change in throughput according to the bottleneck buffer size is similar to that for BBRv1. When the buffer sizes are smaller than 1 BDP, each BBRv2 flow occupies half of the bottleneck bandwidth. However, the difference between BBRv1 and BBRv2 is that the number of packet retransmissions of BBRv2 is significantly reduced compared to BBRv1. For instance, in a 0.2 BDP bottleneck buffer and 50 Mbps bandwidth, they only create 1,100 packet retransmissions. This is because BBRv2 sets the inflight cap when it detects the packet loss that exceeds the predefined threshold;



(a) 50 Mbps and BBRv1 vs. CUBIC (b) 1 Gbps and BBRv1 vs. CUBIC (c) 50 Mbps and BBRv2 vs. CUBIC (d) 1 Gbps and BBRv2 vs. CUBIC

FIGURE 16. Throughput according to bottleneck buffer size when BBR and CUBIC coexist on the same bottleneck link.



(a) 50 Mbps and BBRv1 vs. CUBIC in 0.2 BDP buffer (b) 50 Mbps and BBRv1 vs. CUBIC in 2 BDP buffer (c) 50 Mbps and BBRv1 vs. CUBIC in 8 BDP buffer

(d) 50 Mbps and BBRv2 vs. CUBIC in 0.2 BDP buffer (e) 50 Mbps and BBRv2 vs. CUBIC in 2 BDP buffer (f) 50 Mbps and BBRv2 vs. CUBIC in 8 BDP buffer

FIGURE 17. The timestamp of packet retransmissions, throughput, `RTprop` and buffer backlog when BBR and CUBIC coexist on the same bottleneck link.

thus, it induces fair coexistence and reduces packet loss. Moreover, the long RTT flow still occupies more bandwidth when the buffer size increased beyond 2 BDP. In summary, the coexistence on the link with a small buffer of below 2 BDP greatly improved on BBRv2 compared to that on BBRv1, but unfairness still exists between different RTT flows when the buffer sizes are large enough.

E. FAIRNESS WITH LOSS-BASED ALGORITHMS

This section reports on the evaluation of the inter-protocol fairness. We conducted an experiment in which the BBR shared a bottleneck link with CUBIC, one of the loss-based congestion control algorithm. In this experiment, the bottleneck bandwidth was 50 Mbps or 1 Gbps, and the round-trip propagation time was 30 ms. The bottleneck buffer size varied from 0.1 to 16 BDP according to the test scenarios. Fig. 16 shows the average throughput according to the bottleneck buffer size. Fig. 17 depicts the number of packet retransmissions, throughput, `RTprop` of BBR, and buffer backlog in Switch 2 when buffer sizes are 0.2, 2, and 8 BDP. Moreover, Fig. 19 shows the throughput when two BBR flows and two CUBIC flows coexist on the bottleneck link with a bandwidth of 50 Mbps and a round-trip propagation time of 20 ms.

From test results depicted in Fig. 16(a),(b), the throughput between BBRv1 and CUBIC completely depends on the bottleneck buffer size. As shown in Fig. 17(a), where the buffer size is 0.2 BDP, BBRv1 occupies the most of the bandwidth and fills the bottleneck buffer. BBRv1 uses the `cwnd` to limit the amount of inflight data to maintain the maximum sending rate and the minimum latency. The maximum value of the `cwnd` is determined by the `target_cwnd`, calculated as `cwnd_gain` × `BtlBw` × `RTprop`. However, `cwnd_gain` is fixed to 2 so that BBRv1 creates 1 BDP excess queue; therefore, excessive packet loss occurs when buffer sizes are smaller than 1 BDP. This prevents CUBIC from increasing the `cwnd`, therefore CUBIC cannot take up the bandwidth.

Moreover, BBRv1 and CUBIC flows show completely different behavior with a 2 BDP buffer; i.e., they periodically fluctuates as depicted in Fig. 17(b) because BBRv1 measures `RTprop` that completely differs from the actual round-trip propagation time due to the queuing delay. The excess queue created by CUBIC induces BBRv1 to measure the higher `RTprop`, causing BBRv1 to set a higher inflight

cap than before, making BBRv1 more aggressive than CUBIC and causing buffer overflow. When the `RTprop` expires, BBRv1 completely empties the excess queue so that BBRv1 can measure the `RTprop`, similar to the actual round-trip propagation time. Then, CUBIC increases the `cwnd` and achieves higher throughput than BBRv1. This process is repeated, and the throughput fluctuation continues over time.

Since the buffer sizes are large enough, as shown in Fig. 17(c), BBRv1 and CUBIC evenly share the buffer if the bottleneck buffer size is large enough. This is because the standing queue created by CUBIC allows the BBRv1 flow to measure the appropriate `RTprop`; thus, the amount of inflight data is similar, achieving convergence between two flows. However, the time for convergence between the two flows increases with the buffer size; hence, the CUBIC flow exhibits higher throughput than BBRv1.

Fig. 16(c),(d) shows that the dependency on the buffer size in the coexistence of BBRv2 and CUBIC is partially reduced compared to BBRv1. In a small buffer, as shown in Fig. 17(c), the BBRv2 and CUBIC flows evenly share the bottleneck link. Moreover, BBRv2 sets `inflight_hi` to limit the amount of inflight data when the packet loss rate exceeds 2%, which caused makes BBRv2 behave like a loss-based congestion control algorithm. Therefore, the fairness with CUBIC is enhanced compared to BBRv1, as shown in Fig. 17(d).

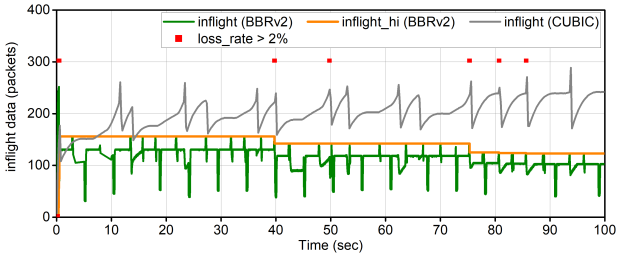
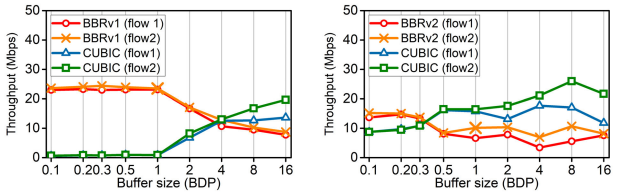


FIGURE 18. **The amount of inflight, `inflight_hi` and the timestamp when packet loss exceeds 2% in coexistence of BBRv2 and CUBIC (buffer size: 2 BDP).**

However, the coexistence between BBRv2 and CUBIC in a large buffer exhibits worse fairness compared to BBRv1, as shown in Fig. 17(e),(f). Moreover, BBRv2 was unable to estimate the more bandwidth over time because packet loss caused by CUBIC caused BBRv2 to set the `inflight_hi`, and this limits the growth of the inflight data in BBRv2. Fig. 18 depicts the `inflight`, `inflight_hi`, and the timestamp when packet loss exceeds 2% in the coexistence of BBRv2 and CUBIC on the bottleneck link with a 2 BDP buffer. Initially, BBRv2 sets the `inflight_hi` to 160 MSS. However, the packet loss caused by the continuous increase in the `cwnd` of CUBIC induces BBRv2 to set a lower `inflight_hi` than before. For example, at t=40 and t=75 s, BBRv2 sets the `inflight_hi` to a value significantly less than the previous value, continuously reducing the throughput in BBRv2.



(a) 2 BBRv1 vs. 2 CUBIC (b) 2 BBRv2 vs. 2 CUBIC

FIGURE 19. **Throughput according to the bottleneck buffer size when two BBR flows and two CUBIC flows coexist on the bottleneck link.**

Fig. 19(a) shows the performance changes according to the buffer size where two BBRv1 and two CUBIC flows share the bottleneck link. With a small buffer of less than 1 BDP, most of the bandwidth is occupied by two BBRv1 flows. With a large buffer fluctuates between the two groups of algorithms. However, as the buffer size increases, the fluctuation decreases but takes longer to converge. In Fig. 19(b) the throughput between two BBRv2 and two CUBIC flows represents similar behavioral characteristics of Fig. 16(c); i.e., they demonstrate improved throughput fairness in small buffer of below 1 BDP but worse throughput fairness in large buffers.

V. CONCLUSION

This study presented the comprehensive evaluation and comparison between BBR v1 and BBRv2 on a Mininet emulator and physical testbed comprising of dumbbell topology. The results showed that BBRv2 worked well on the bottleneck link with a small buffer. BBRv2 not only enhanced the fairness with other TCP flows, but also reduced the packet loss in comparison to BBRv1. However, with large buffers, BBRv2 still experienced significant issues such as coexistence between different RTT flows, competition with loss-based congestion control algorithms, and convergence between BBRv2 flows. In particular, we found that when the identical BBRv2 flows entered the bottleneck link with large buffers at different times, their throughputs did not converge at all. The second-started flow experienced the packet loss in Startup phase, therefore, setting the `inflight_hi` to limit the growth of the amount of inflight data. However, this limitation of inflight data rather prevented BBRv2 flow from probing the more bandwidth, causing unfairness between two BBRv2 flows.

At present, Google has released the BBRv2 alpha version, and is developing the final version of BBRv2. we hope that these experimental results can help to improve the performance of the BBRv2 congestion control algorithm.

REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," in *Proc. Symp. Proc. Commun. Archit. Protocols SIGCOMM*, 1988, pp. 314–329.
[2] T. Henderson, S. Floyd, and Y. Nishida, *The NewReno Modification to TCP's Fast Recovery Algorithm*, document IETF RFC 6582, 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6582.txt
[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.
[4] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Queue*, vol. 9, no. 11, p. 40, Nov. 2011.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 50-20–50-53, Oct. 2016.
[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR congestion control," in *Proc. IETF 97th Meeting*, Jul. 2016, pp. 1–36. [Online]. Available: https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf
[7] D. Scholz, B. Jaeger, L. Schwaighofer, L. Raumer, F. Geyer, and G. Carle, "Toward a deeper understanding of TCP BBR congestion control," in *Proc. IFIP Netw.*, May 2018, pp. 1–9.
[8] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.
[9] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, V. Vasiliev, J. Iyengar, V. P. V. Jha, Y. Seung, K. Yang, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control," in *Proc. IETF 102th Meeting*, 2018, pp. 1–36. [Online]. Available: https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-an-update-on-bbr-work-at-google-00
[10] K. Ramakrishnan, S. Floyd, and D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, document RFC 3168, IETF, 2001.
[11] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control," in *Proc. IETF 104th Meeting*, 2019, pp. 1–36. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00
[12] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control," in *Proc. IETF 105th Meeting*, 2019, pp. 1–36. [Online]. Available: https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00
[13] S. Zhang, "An evaluation of BBR and its variants," 2019, *arXiv:1909.03673*. [Online]. Available: http://arxiv.org/abs/1909.03673
[14] J. Gomez, E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "A performance evaluation of TCP BBRv2 alpha," in *Proc. 43rd Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2020, pp. 309–312.
[15] A. Nandagiri, M. P. Tahiliani, V. Misra, and K. K. Ramakrishnan, "BBRvl vs BBRv2: Examining performance differences through experimental evaluation," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN*, Jul. 2020, pp. 1–6.
[16] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR congestion control," in *Proc. IETF 98th Meeting*, 2017, pp. 1–36. [Online]. Available: https://www.ietf.org/proceedings/98/slides/slides-98-iccrg-an-update-on-bbr-congestion-control-00.pdf
[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev, and V. Jacobson, "BBR congestion control," in *Proc. IETF 99th Meeting*, 2017, pp. 1–49. [Online]. Available: https://www.ietf.org/proceedings/99/slides/slides-99-iccrg-iccrg-presentation-2-00.pdf
[18] Y. Cheng, N. Cardwell, S. H. Yeganeh, and V. Jacobson, "Delivery rate estimation," in *Draft-Cheng-ICCRG-Delivery-Rate-Estimation-00*. Fremont, CA, USA: IETF, 2017.
[19] N. Cardwell, Y. Cheng, S. H. Yeganeh, and V. Jacobson, "BBR congestion control," in *Draft-Cardwell-ICCRG-Bbr-Congestion-Control-00*. Fremont, CA, USA: IETF, 2017.
[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev, and V. Jacobson, "BBR congestion control," in *Proc. IETF 100th Meeting*, 2017, pp. 1–36. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers/
[21] *TCP BBRv2 Alpha/Preview Release*. Accessed: May 11, 2020. [Online]. Available: https://github.com/google/bbr/tree/v2alpha
[22] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control performance optimization," in *Proc. IETF 106th Meeting*, 2019, pp. 1–32. [Online]. Available: https://datatracker.ietf.org/meeting/106/materials/slides-106-iccrg-update-on-bbrv2
[23] N. Cardwell, Y. Cheng, K. Yang, S. H. Yeganeh, P. Jha, Y. Seung, L. Hsiao, M. Mathis, I. Swett, B. Wu, V. Vasiliev, N. Dukkipati, G. Kumar, and V. Jacobson, "BBR update: 1: BBR.Swift; 2: Scalable loss handling," in *Proc. IETF 109th Meeting*, 2020, pp. 1–21. [Online]. Available: https://datatracker.ietf.org/meeting/109/materials/slides-109-iccrg-update-on-bbrv2-00
[24] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 514–528.
[25] M. Mathis, N. Dukkipati, and Y. Cheng, *Proportional Rate Reduction*, document RFC 6937, IETF, 2013.
[26] E. Atxutegi, F. Liberal, H. K. Haile, K.-J. Grinnemo, A. Brunstrom, and A. Arvidsson, "On the use of TCP BBR in cellular networks," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 172–179, Mar. 2018.
[27] A. Parichehreh, S. Alfredsson, and A. Brunstrom, "Measurement analysis of TCP congestion control algorithms in LTE uplink," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2018, pp. 1–8.
[28] T. Dai, X. Zhang, and Z. Guo, "Poster abstract: Analysis and experimental investigation of BBR," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 1–2.
[29] K. Nguyen and H. Sekiya, "TCP behavior on multi-gigabit IEEE 802.11ad link," in *Proc. Int. Conf. Green Hum. Inf. Technol. (ICGHIT)*, Feb. 2020, pp. 58–61.
[30] J. Crichigno, Z. Csibi, E. Bou-Harb, and N. Ghani, "Impact of segment size and parallel streams on TCP BBR," in *Proc. 41st Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2018, pp. 641–645.
[31] Y. Wang, K. Zhao, W. Li, J. Fraire, Z. Sun, and Y. Fang, "Performance evaluation of QUIC with BBR in satellite Internet," in *Proc. 6th IEEE Int. Conf. Wireless Space Extreme Environ. (WiSEE)*, Dec. 2018, pp. 195–199.
[32] H. Zhang, H. Zhu, Y. Xia, L. Zhang, Y. Zhang, and Y. Deng, "Performance analysis of BBR congestion control protocol based on NS3," in *Proc. 7th Int. Conf. Adv. Cloud Big Data (CBD)*, Sep. 2019, pp. 363–368.
[33] R. Kumar, A. Koutsaftis, F. Fund, G. Naik, P. Liu, Y. Liu, and S. Panwar, "TCP BBR for ultra-low latency networking: Challenges, analysis, and solutions," in *Proc. IFIP Netw. Conf.*, May 2019, pp. 1–9.
[34] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi, "Will TCP work in mmWave 5G cellular networks," *IEEE Commun. Mag.*, vol. 57, no. 1, pp. 65–71, Jan. 2019.
[35] I. Kunze, J. Ruth, and O. Hohlfeld, "Congestion control in the wild—Investigating content provider fairness," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 1224–1238, Jun. 2020.
[36] I. Mahmud, G.-H. Kim, T. Lubna, and Y.-Z. Cho, "BBR-ACD: BBR with advanced congestion detection," *Electronics*, vol. 9, no. 1, p. 136, Jan. 2020.
[37] C. A. Grazia, M. Klapez, and M. Casoni, "BBRp: Improving TCP BBR performance over WLAN," *IEEE Access*, vol. 8, pp. 43344–43354, 2020.
[38] P. Hurtig, H. Haile, K.-J. Grinnemo, A. Brunstrom, E. Atxutegi, F. Liberal, and A. Arvidsson, "Impact of TCP BBR on CUBIC traffic: A mixed workload evaluation," in *Proc. 30th Int. Teletraffic Congr. (ITC 30)*, Sep. 2018, pp. 218–226.
[39] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda, and S. Yamaguchi, "TCP fairness among modern TCP congestion control algorithms including TCP BBR," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–4.
[40] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cycle and divergence of performance on TCP BBR," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–6.
[41] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cyclic performance fluctuation of TCP BBR," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 811–812.
[42] K. Miyazawa, S. Yamaguchi, and A. Kobayashi, "A study on cyclic performance fluctuation of CUBIC TCP and TCP BBR considering estimated RTT and bandwidth," in *Proc. 7th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Nov. 2019, pp. 478–480.
[43] Y.-J. Song, G.-H. Kim, and Y.-Z. Cho, "Enhanced loss-recovery mechanism for BBR to improve inter-protocol fairness," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 1181–1183.
[44] Y.-J. Song, G.-H. Kim, and Y.-Z. Cho, "Improvement of cyclic performance variation between TCP BBR and CUBIC," in *Proc. 25th Asia–Pacific Conf. Commun. (APCC)*, Nov. 2019, pp. 1–6.
[45] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Modeling BBR's interaction with loss-based congestion control," in *Proc. Internet Meas. Conf. (IMC)*, 2019, pp. 137–143.
[46] Y. J. Song, G. H. Kim, and Y. Z. Cho, "Congestion window scaling method for inter-protocol fairness," in *Proc. Annu. Consumer Commun. Netw. Conf.*, Jan. 2020, pp. 1–4.

[47] G.-H. Kim, Y.-J. Song, and Y.-Z. Cho, "Improvement of inter-protocol fairness for BBR congestion control using machine learning," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2020, pp. 501–504.
[48] Y. J. Song, G. H. Kim, and Y. Z. Cho, "BBR-CWS: Improving the inter-protocol fairness of BBR," *Electron.*, vol. 9, no. 5, p. 862, 2020.
[49] Y. Zhang, L. Cui, and F. P. Tso, "Modest BBR: Enabling better fairness for BBR congestion control," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, Art. no. 00646.
[50] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," 2017, *arXiv:1706.09115*. [Online]. Available: http://arxiv.org/abs/1706.09115
[51] Y. Tao, J. Jiang, S. Ma, L. Wang, W. Wang, and B. Li, "Unraveling the RTT-fairness problem for BBR: A queueing model," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.
[52] M. Yang, P. Yang, C. Wen, Q. Liu, J. Luo, and L. Yu, "Adaptive-BBR: Fine-grained congestion control with improved fairness and low latency," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–6.
[53] G.-H. Kim, Y.-J. Song, I. Mahmud, and Y.-Z. Cho, "Enhanced BBR congestion control algorithm for improving RTT fairness," in *Proc. 11th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2019, pp. 358–360.
[54] G. H. Kim, I. Mahmud, and Y. Z. Cho, "Fairness Improvement of BBR Congestion Control Algorithm for Difference RTT Flows," in Proc. International Conference on Electronics Information, and Communications, 2019.
[55] G.-H. Kim and Y.-Z. Cho, "Delay-aware BBR congestion control algorithm for RTT fairness improvement," *IEEE Access*, vol. 8, pp. 4099–4109, 2020.
[56] T. Zhu, X. Qin, L. Chen, X. Chen, and G. Wei, "WBBR: A bottleneck estimation-based congestion control for multipath TCP," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–5.
[57] K. Nguyen, M. G. Kibria, K. Ishizu, F. Kojima, and H. Sekiya, "An evaluation of multipath TCP in lossy environment," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 573–577.
[58] S. Zhang, W. Lei, W. Zhang, Y. Guan, and H. Li, "Congestion control and packet scheduling for multipath real time video streaming," *IEEE Access*, vol. 7, pp. 59758–59770, 2019.
[59] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *Proc. Int. Conf. Commun.*, vol. 43, 1979, pp. 1–10.
[60] *Measurement-Framework*. Accessed: May 15, 2020. [Online]. Available: https://gitlab.lrz.de/tcp-bbr/measurement-framework
[61] V. Jacobson, C. Leres, and S. McCanne. *Tcpdump—Dump Traffic on a Network*. Accessed: May 15, 2020. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man8/tcpdump.8.html
[62] *Intermediate Functional Block*. Accessed: May 15, 2020. [Online]. Available: http://linux-ip.net/gl/tc-filters/tc-filters-node3.html
[63] *Traffic Control*. Accessed: May 15, 2020. [Online]. Available: https://linux.die.net/man/8/tc
[64] *TCPlog*. Accessed: May 15, 2020. [Online]. Available: https://git.scc.kit.edu/CPUnetLOG/TCPlog
[65] *iPerf*. Accessed: May 15, 2020. [Online]. Available: https://iperf.fr

YEONG-JUN SONG received the B.S. degree in electronics engineering from Kyungpook National University, in 2019, where he is currently pursuing the M.S. degree in electronic and electrical engineering with the Telecommunication and Network Laboratory. His current research interests include TCP congestion control and artificial intelligence in networks.

GEON-HWAN KIM received the B.S. and M.S. degrees in electronics engineering from Kyungpook National University, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electronic and electrical engineering with the Telecommunication and Network Laboratory. His current research interests include FANETs and TCP congestion control algorithms.

IMTIAZ MAHMUD received the B.S. degree in telecommunication and electronic engineering from Hajee Mohammad Danesh Science and Technology University, in 2011, and the M.S. and Ph.D. degrees in electronics engineering from Kyungpook National University, in 2015 and 2019, respectively. He is currently working as a Postdoctoral Research Fellow with the Telecommunication and Network Laboratory, Kyungpook National University. His current research interests include multipath TCP, TCP congestion control algorithms, and UAVs path planning and network constraints.

WON-KYEONG SEO received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Kyungpook National University, South Korea, in 2005, 2007, and 2012, respectively. From September 2012 to August 2014, he had been with GYWELL Inc., South Korea, as a Chief Executive Officer. Since 2015, he has been with Yeungjin University, South Korea, where he is currently a Professor with the Department of Military Electronics. His research interests include mobility management and traffic engineering for wireless and mobile networks, and transport protocols for the future Internet.

YOU-ZE CHO (Senior Member, IEEE) received the B.S. degree in electronics engineering from Seoul National University, South Korea, in 1982, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, in 1985 and 1988, respectively. Since 1989, he has been with Kyungpook National University (KNU), South Korea, where he is currently a Professor with the School of Electronics Engineering and the Dean of the College of IT Engineering. From August 1992 to January 1994, he worked with the University of Toronto, Canada, as a Visiting Researcher. From March 2002 to February 2003, he worked with the National Institute of Standards and Technology, USA, as a Guest Researcher. In 2017, he has served as the President for the Korean Institute of Communications and Information Sciences (KICS). His research interests include mobility management and traffic engineering for wireless and mobile networks, and transport protocols for the future Internet.