electronics

MDPI

Article

BBR-CWS: Improving the Inter-Protocol Fairness of BBR

Yeong-Jun Song, Geon-Hwan Kim and You-Ze Cho *

School of Electronics Engineering, Kyungpook National University, Daegu 41556, Korea; syj5385@knu.ac.kr (Y.-J.S.); kgh76@ee.knu.ac.kr (G.-H.K.)

* Correspondence: yzcho@ee.knu.ac.kr

Received: 24 April 2020; Accepted: 14 May 2020; Published: 22 May 2020

Abstract: TCP congestion control adjusts the sending rate in order to protect Internet from the continuous traffic and ensure fair coexistence among multiple flows. Especially, loss-based congestion control algorithms were mainly used, which worked relatively well for past Internet with low bandwidth and small bottleneck buffer size. However, the modern Internet uses considerably more sophisticated network equipment and advanced transmission technologies, and loss-based congestion control can cause performance degradation due to excessive queueing delay and packet loss. Therefore, Google introduced a new congestion control in 2016, Bottleneck Bandwidth Round-trip propagation time (BBR). In contrast with traditional congestion control, BBR tries to operate at the Kleinrock's optimal operating point, where delivery rate is maximized and latency is minimized. However, when BBR and loss-based congestion control algorithms coexist on the same bottleneck link, most of bottleneck bandwidth is occupied by flows that use a particular algorithm, and excessive packet retransmission can occur. Therefore, this paper proposes a BBR congestion window scaling (BBR-CWS) scheme to improve BBR's inter-protocol fairness with a loss-based congestion control algorithm. Through Mininet experiment results, we confirmed that fairness between BBR-CWS and CUBIC improved up to 73% and has the value of 0.9 or higher in most bottleneck buffer environments. Moreover, the number of packet retransmissions was reduced by up to 96%, compared to the original BBR.

Keywords: TCP congestion control; BBR; CUBIC; inter-protocol fairness; retransmission

1. Introduction

Many Internet traffics use TCP as the transport layer protocol for reliable data transmission [1,2]. TCP congestion control, a main TCP feature, protects the Internet from persistent overload and ensures that multiple hosts use bottleneck bandwidth evenly [3–5]. In this context, congestion refers to a state where data sent from a host is queued at the router and packet loss occur due to the router buffer being filled with excessive inflight data, causing network problems such as increased latency and goodput degradation. Therefore, each sender should adjust their sending rate using congestion window (`cwnd`) according to network congestion, to coexist with other flows fairly.

Loss-based congestion control algorithms such as Reno [6] and CUBIC [7] have been widely used since congestion control was introduced in the 1980s, and they have worked relatively well on past networks that had low bandwidth and small bottleneck buffers. However, the increase in network transmission capacity, driven by the advance in transmission technologies and the development of network devices, makes the existing congestion control algorithms experience buffer-bloat [8] and unexpected performance degradation.

In 2016, Google introduced a new TCP congestion control algorithm called Bottleneck Bandwidth Round-trip propagation time (BBR) to replace existing congestion control algorithms [9,10]. In contrast

with traditional congestion control algorithms, BBR attempts to operate at the Kleinrock's optimal operating point [11], i.e., maximized delivery rate and minimized latency, by dynamically adjusting transfer data, using measured maximum bandwidth and minimum round-trip time. Figure 1 shows the relationship between delivery rate and round-trip time, based on the amount of inflight, for small and large bottleneck buffer. In Figure 1a,b, the delivery rate increases linearly until the inflight reaches Point (A), the Bandwidth Delay Product (BDP), but round-trip time has constant value. After that, if the amount of inflight exceeds the BDP, the standing queue begins to form, resulting in the delivery rate stagnating, while round-trip time increases. At this time, when the inflight is beyond Point (B), where the buffer is full, resulting in packet loss without RTT increasing. Each congestion control algorithm operates at a different point according to its operating characteristics. For instance, loss-based congestion control algorithms persistently increase its inflight in order to fill the bottleneck buffer, so that it puts its operating point at Point (B). Unlike loss-based algorithm, BBR aims to operate at Point (A), Kleinrock's optimal operating point, by estimating the BDP, which is the product of the available maximum bandwidth and minimum round-trip time. To achieve this operating goal, BBR keeps the `target_cwnd` as 2 BDP in order to limit the aggressive growth of the `cwnd`; therefore, its actual operating point lies at Point (C) in Figure 1. As a result, BBR creates a standing queue of up to 1 BDP. This behavior of BBR does not matter if the buffer size is larger than 1 BDP, as shown in Figure 1a. However, as shown in Figure 1b, if the buffer size is larger than 1 BDP, Point (C), BBR's operating point, is located to the right of Point (B), resulting that excessive packet loss occurs.

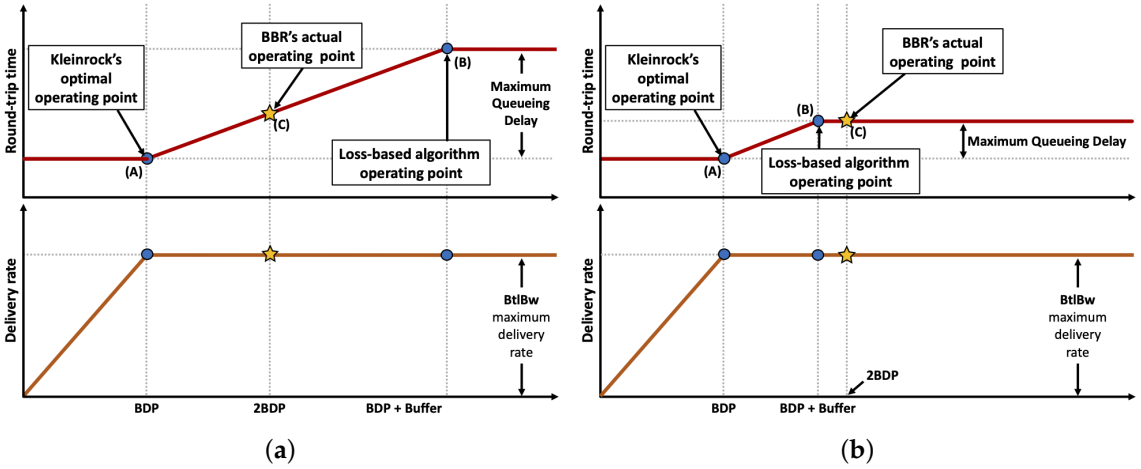


Figure 1. Delivery rate and round-trip time based on the inflight: (**a**) buffer size > 1 BDP (deep); and (**b**) buffer size < 1 BDP (shallow).

After BBR was introduced, many researchers claimed the problems that arise when BBR operates in a variety of network environments [12–19], which are RTT fairness [20–22] and inter-protocol fairness [23–28]. In particular, the inter-protocol fairness problem between BBR and existing loss-based congestion control algorithms is considered to be a serious issue, because the throughput between two algorithms can vary significantly with bottleneck buffer size. For example, if flows operate on the link with shallow buffer, the BBR flow takes up much more bandwidth than the loss-based algorithm, whereas there is cyclic performance variation between two-group algorithms in deep bottleneck buffer. That is because BBR has inflexible operating characteristics, which means that it does not actively reduce the congestion window by packet loss, but only determines the sending rate depending on BDP, estimated by Bottleneck Bandwidth (`BtlBw`) and Round-trip propagation time (`RTprop`). If BBR sends out much more data than the link's capacity, it causes severe network congestion and excessive packet loss. However, despite the packet loss, BBR does not reduce the inflight-cap at all, and therefore the vicious cycle is constantly repeated.

In this paper, we propose a congestion window scaling mechanism for BBR to improve inter-protocol fairness. We call the proposed algorithm as BBR Congestion Window Scaling (BBR-CWS). Unlike the original BBR, BBR-CWS considers packet loss to be a signal that the operating point was set incorrectly. When BBR-CWS detects packet loss, it first reduces the congestion window size. Next, it observes some parameters and then analyzes why the packet loss occurred. Finally, depending upon the cause of packet loss, BBR moves to an appropriate operating point where it gets a fair share without continuous packet retransmission.

The main contributions of the BBR-CWS are as follows:

- BBR-CWS improves inter-protocol fairness when it shares the link with loss-based congestion control algorithms and reduces excessive retransmissions.
- The main difference between BBR-CWS and the original BBR is that BBR-CWS uses `RTprop`'s changes to determine packet loss cause, and then, performs the corresponding operation to improve inter-protocol fairness, reducing aggressiveness and packet retransmission.
- BBR-CWS was implemented on Ubuntu with Linux Kernel version 4.14 [29], adding a single function, "bbr_set_upper_scaling_factor", and a mechanism to reduce the congestion window by packet loss on current BBRv1 implementation, hence BBR-CWS has lightweight implementation.

We implemented BBR-CWS and conducted performance evaluation on a physical testbed and Mininet network emulation to evaluate the inter-protocol fairness and the number of packet retransmissions. In the experimental results, the inter-protocol fairness between BBR-CWS and loss-based algorithms was improved about 70% and the number of retransmitted packets was reduced by about 90% compared to the original BBR.

The remainder of paper is organized as follows. Section 2 reviews related work, in particular, considering BBR fairness issues when coexisting with loss-based congestion control. Causes for this unfairness are discussed in Section 3. In Section 4, the BBR congestion window scaling method is proposed. Section 5 presents Mininet experimental results. Finally, Section 6 summarizes and concludes the paper.

2. Related Works

2.1. BBR Operation with Shallow Buffer

Hock et al. evaluated BBR performance for various network scenarios and identified some design problems [12]. They found excessive packet retransmissions when BBR operated on bottleneck links with shallow buffers, and subsequently investigated the cause. BBR aims to operate at Kleinrock's optimal operating point by measuring `BtlBw` and `RTprop`. In the BBR implementation, BBR increases `cwnd` by the number of ACKs received from the receiving host. At this time, BBR keeps the `target_cwnd` so that the growth of `cwnd` is limited. Because of this behavior, BBR operates at Point (C) (Figure 1b). However, when the buffer size is smaller than 1 BDP, BBR's operating point lies to the right of Point (A) (Figure 1b), resulting that the maximum size of the standing queue exceeds the bottleneck buffer size, and buffer overflow occurs. The more serious problem is that BBR does not reduce inflight enough for the packet loss; therefore, excessive packet loss cannot be avoided.

2.2. Inter-Protocol Fairness with Loss-Based Algorithm

B. Turkovic et al. classified various TCP congestion control algorithms, including BBR, into loss-based, delay-based. and hybrid algorithms according to the network congestion recognition method [16]. Based on these groups, they detailed operating characteristics of each algorithm groups. Moreover, to depict the interactive operation of TCP congestion control algorithms, they constructed physical testbed, so that the fairness between algorithms in the same groups and the fairness between different groups were evaluated. Especially, they classified BBR as a hybrid algorithm and depicted several operating issues in detail when BBR and other groups of congestion control algorithm compete.

Hock et al. also found problems with interactions between BBR and CUBIC [12]. They conducted an experimental competition between BBR and CUBIC with shallow (0.8 BDP) and deep buffers (8 BDP). For the shallow buffer, BBR occupies most of the bandwidth in competition with CUBIC because CUBIC operating point lies to the left of the BBR's operating point. For the deep buffer, whereas BBR limits the amount of inflight data to 2 BDP, CUBIC increases the `cwnd` until the buffer is full, resulting that it can get enough bandwidth.

Jaeger et al. [13] and Miyazawa et al. [28] identified cyclic performance fluctuation between BBR and CUBIC when two algorithms compete on the same link with large buffer and short end-to-end latency. This problem is caused by incorrect `RTprop` estimation for BBR. CUBIC's aggressive probing for bandwidth fills the queues, causing BBR to measure higher `RTprop` and set a higher inflight cap. Thus, BBR flow becomes more than CUBIC flow. After the `RTprop` is expired, BBR drains the queue, hence it can measure a lower `RTprop` than before. This process is repeated during entire data transfer time, and the oscillation causes constantly low fairness, regardless of whether both flows reach the same average throughput.

2.3. RTT Fairness and Its Solutions

M. Hock et al. revealed RTT-fairness problem that arises when the BBR flows with different RTT coexist on the same bottleneck link [12]. They focused on the bottleneck buffer size as the dependency of RTT fairness among different RTTs' flows. With large buffers, a flow with long RTT occupied more bandwidth than flow with short RTT. That was because two flows were limited by inflight-cap that was calculated as the product of `BtlBw` and `RTprop`. However, when the buffer size is shallow, long RTT flow occupied 40% of the bandwidth, as two flows were not limited by the inflight-cap.

Moreover, Y. Tao et al. studied BBR's RTT-fairness problem from a theoretic perspective. From this analysis, they showed that the degree of unfairness is dictated by the RTT ratio between two flows irrespective of the other network parameters, especially when the RTT between two flows differs by more than two times, the flow with short RTT is completely overwhelmed [20].

To resolve RTT-fairness problem, S. Ma et al. proposed BBQ algorithm [21]. BBQ includes the mechanism for detecting the excess queue continuously and limiting the probing time, so that it prevents long RTT flow from sending a large amount of traffic to the pipe. Moreover, G. H. Kim et al. introduced Delay-Aware BBR (DA-BBR), which improves RTT-fairness problem between different RTT flows. In DA-BBR method, each flows calculate regulating factor (γ). Using this factor, each flow scales the amount of data to be injected to the pipe. Generally, long RTT flow has a lower γ value, thus it reduces the buffer size, thus improving RTT-fairness problem.

2.4. BBRv2

Google started designing BBRv2 in July 2018 [30]. BBRv2 was developed to solve excessive packet retransmission when the original BBR operating in shallow buffers and mitigate inter-protocol fairness between BBR and loss-based congestion control algorithms. BBRv2 operates with the same goal as BBRv1, i.e., minimize the latency and maximize the delivery rate, but the difference from BBRv1 is that it operates by subdividing ProbeBW phase into Up, Down, Cruise and Refill. Especially, it uses packet loss rate and DCTCP/L4S style ECN [31] signal so that BBRv2 limits the maximum amount of inflight. In ProbeBW:Up phase, if the loss rate exceeds the predefined threshold value (2%) or the filtered ECN rate is more than the predetermined threshold (50%), BBRv2 estimates the max safe inflight volume by setting the `inflight_hi`, and then it leaves unused headroom. Moreover, the reduction of congestion window size in ProbeRTT phase was modified to be half of the inflight so that BBRv2 intended to make the `RTprop` less drastic and reduce the fluctuation in competition with loss-based congestion control algorithms. At the IETF 105 meeting held in July 2018, Google presented BBRv2 preview/alpha version code as well as illustrated some lab results from BBRv2 [32,33]. Figure 2 shows experimental results presented at IETF 105, exhibiting improved inter-fairness with loss-based congestion control in a shallow buffer smaller than 2 BDP. However, when the buffer is larger than 2 BDP, inter-protocol

fairness is similar to or slightly lower than that of BBRv1. However, our BBR-CWS can ensure an inter-protocol fairness regardless of the bottleneck buffer size. When we started working on BBR-CWS, only the concept of BBRv2 was introduced and recently the preview implementation was released. That was why we implemented BBR-CWS based on BBRv1 instead of BBRv2.

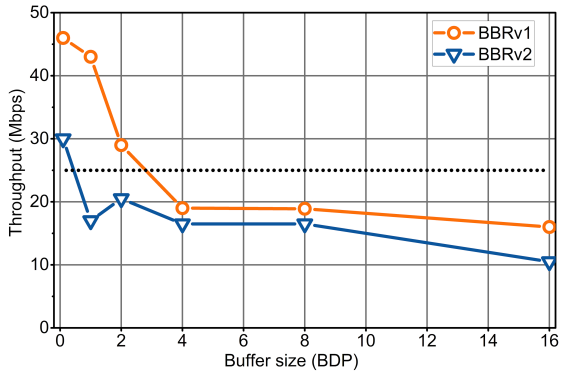


Figure 2. Coexistence with usable throughput for CUBIC [32].

2.5. Solutions for Inter-Protocol Fairness of BBR

Modest BBR: Zhang et al. proposed moderately aggressive BBR congestion control, Modest BBR, to alleviate retransmission and achieve better inter-protocol fairness [26]. They showed that CUBIC performance degraded sharply when competing against the original BBR. Their experimental results show that Modest BBR achieved comparable throughput compared with the original BBR, improving overall throughput by 7.1% and offering better fairness when coexisting with loss-based algorithms. However, they did not specify the bottleneck buffer size they configured. Comparing their results with other papers, we estimated the experiment was conducted in a shallow bottleneck buffer, because the BBR flow suppressed CUBIC flow. Furthermore, they did not address BBR and CUBIC fairness for various buffer environments. The proposed BBR-CWS improves BBR and CUBIC fairness regardless of bottleneck buffer size, hence can be applied to more general network environments than Modest BBR.

Congestion Window Scaling Method: We first introduced congestion window scaling concept for BBR in [34]. In this study, we only compared congestion window scaling method with the original BBR as well as only results of throughput and the number of packet retransmission were shown. Therefore, in this study, we analyzed the detailed operation of BBR-CWS which competes with CUBIC by tracing the `cwnd` and `RTprop` changes over time. Moreover, we selected Modest BBR as benchmark algorithm which focused on the same issues, so that we compared our proposed algorithm with both the original BBR and Modest BBR.

3. Unfairness with Loss-Based Algorithm

We constructed a simple topology based on physical testbed to evaluate inter-protocol fairness between the original BBR and CUBIC in a network environment with various buffer sizes, as shown in Figure 3. The bottleneck link was configured between switch 1 and 2. The bottleneck bandwidth was 100 Mbps, and the propagation round-trip time between two end-to-end hosts was set to 10 ms. The sending host created two TCP flows with BBR and CUBIC, respectively, which shared a common bottleneck link. Figure 4 shows throughput change between BBR and CUBIC according to the bottleneck buffer size.

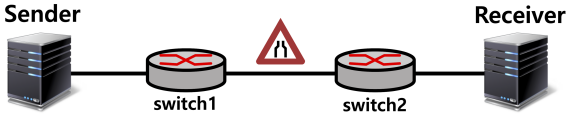


Figure 3. Physical testbed topology.

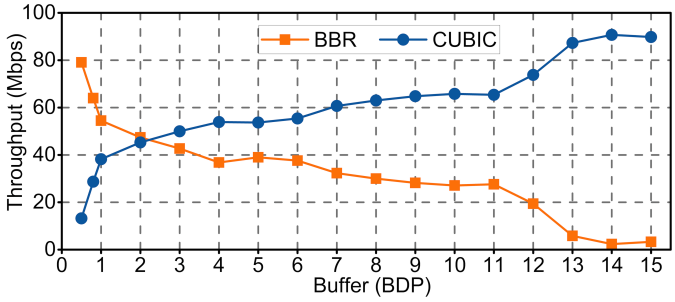


Figure 4. Throughput of BBR and CUBIC according to the buffer size.

As shown in Figure 4, the throughput between BBR and CUBIC clearly depends on the bottleneck buffer size. When the bottleneck buffer size is smaller than 1 BDP, most of the bandwidth is occupied by BBR. However, when the buffer size is larger than 1 BDP, CUBIC occupies more bandwidth as the buffer size gets larger.

We revealed the two causes of the inter-protocol fairness problem in this study. First, when the BBR and the loss-based algorithm coexist on a link with a buffer smaller than 1 BDP, the imbalance between the two flows is attributed to the fixed `cwnd_gain` of the BBR. Second, when two algorithms compete on a link with sufficient buffer size, a variation in throughput occurs. That is because BBR measures `RTprop` higher than the actual propagation round-trip time due to the queueing delay generated by the CUBIC flow.

3.1. Shallow Buffer: Static Cwnd_gain

In the implementation of BBR, the congestion window is used to limit the growth of inflight, and its maximum size is set to 2 BDP. However, updating the BBR's congestion window directly whenever an ACK is received can cause a drastic change in the inflight, which can have significant adverse effects on data transmission performance. Therefore, BBR uses `target_cwnd` that represents the maximum upper limit to which the congestion window can grow. BBR sets the `target_cwnd` as `cwnd_gain` $\times$ `BtlBw` $\times$ `RTprop`, and increases the congestion window each time an ACK is received, until `cwnd` reaches the `target_cwnd`. However, the `cwnd_gain` is fixed to 2 in BBR, so that a serious problem occurs when BBR competes with CUBIC on the same link with a buffer of 1 BDP or less. The static `cwnd_gain` makes BBR fill both the bottleneck pipe and the buffer, resulting in BBR creating excess queues up to 1 BDP. The size of the excess queue which BBR creates is much larger than the link can accommodate, and therefore buffer overflow continues. As a result, CUBIC cannot increase the congestion window at all, and it is completely overwhelmed by the BBR (Figure 5a).

3.2. Deep Buffer: Minimum RTT Overestimation

If BBR and a loss-based algorithm share the same link with a buffer above 1 BDP, throughput between the two algorithms fluctuates significantly. Figures 5b and 6 show the throughput, congestion window size, and round-trip time of BBR flow when one BBR and one CUBIC flow send data over a path with a 5 BDP buffer. Figure 5b shows that the throughput between BBR and CUBIC flow fluctuates once every 10 s. The reason for this variation is that BBR measures the `RTprop` completely different from the actual propagation round-trip time of the path. From 0 to 10 s in Figure 6, CUBIC continuously increases the congestion window size, so that it completely fills the buffer. At 10 s, BBR sets the congestion window size to four packets in order to measure the new `RTprop`. However, BBR cannot see the actual propagation round-trip time due to the fully filled buffer. Consequently, BBR sets a higher inflight cap and generates excess queue in the 10 to 20 s window, resulting in overwhelming the CUBIC flow. Most importantly, BBR sends out much more data than the link can accommodate, resulting in excessive packet loss by buffer overflow. At 20 s, the expiration of `RTprop` makes BBR enter a new ProbeRTT phase again, so that BBR drains the buffer entirely and measures `RTprop` close to the actual propagation round-trip time. At the same time, CUBIC actively increases the sending rate with

aggressive growth of the congestion window and occupies most of bandwidth. The above process is repeated, and the throughput variation continues over time.

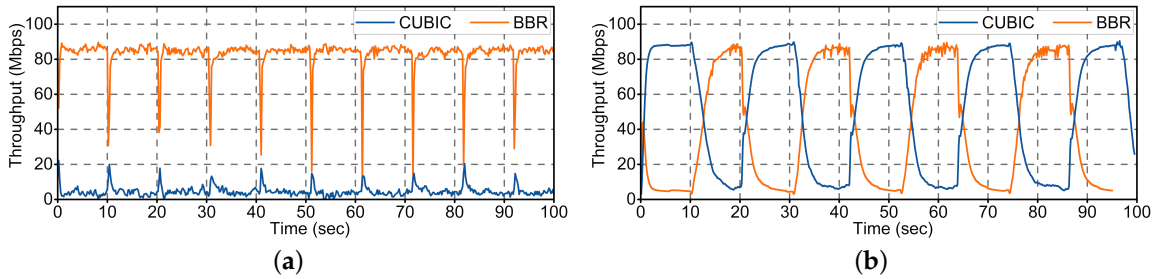


Figure 5. Throughput between BBR and CUBIC: (**a**) shallow buffer (0.8 BDP); and (**b**) deep buffer (5 BDP).

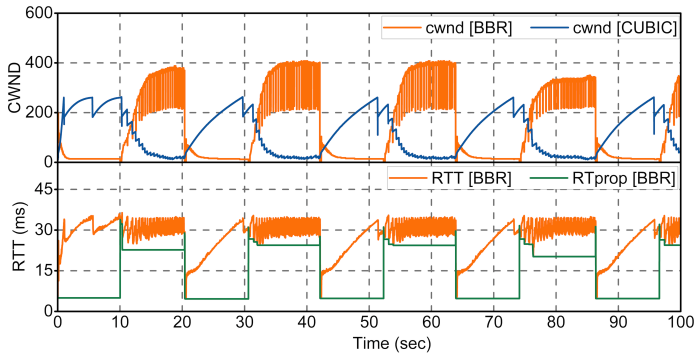


Figure 6. The congestion window and round-trip time of BBR and CUBIC (deep buffer: 5 BDP).

4. BBR-CWS: BBR Congestion Window Scaling

In this section, we provide details of the BBR congestion window scaling method. Algorithm 1 shows the BBR-CWS scheme. The "bbr_main" function is called when BBR gets an ACK from the receiver host, and it executes the following five functions on Lines 43–47 of Algorithm 1. Using these functions, BBR measures the available maximum bandwidth and minimum round-trip time of the network path model, and calculates output parameters (`pacing_rate`, `congestion_window`, and `send_quantum`) in order to control the amount of inflight data.

In this study, we implemented the "bbr_set_upper_scaling" function in the existing BBR implementation of the Linux kernel code to perform congestion window scaling. Moreover, we slightly modified the "bbr_set_cwnd" and "bbr_update_model" functions to apply the scaling variable as shown on Line 21, and configured the amount of reduction in ProbeRTT phase and packet conservation mode.

The variables that we additionally declared in BBR code to implement the BBR-CWS scheme are as follows:

- `loss_detected_time`: This variable indicates the time stamp to which BBR-CWS's `tcp_ca_state` is set to *TCP_CA_Recovery*, which indicates that BBR-CWS has detected packet loss.
- `is_rtt_min_updated`: This variable checks whether the lower `RTprop` is measured within a certain time after BBR reduces the congestion window in order to decrease inflight.
- `last_action_time`: This variable denotes the time when BBR performed the last action of increasing or decreasing *upper_scaling_factor*.
- `upper_cwnd`: This variable is defined as the product of `upper_scaling_factor` and `target_cwnd`, and it represents the actual maximum size of the congestion window scaled by the BBR-CWS scheme.

Algorithm 1 BBR congestion window upper scaling.

```
1:  bbr_set_cwnd_to_recover_or_restore ():
2:    if current_ca_state == TCP_CA_Recovery
3:          and prev_ca_state != TCP_CA_Recovery :
4:       if loss_detected_time == 0 :
5:          loss_detected_time = tcp_jiffies32
6:          upper_cwnd = current_inflight
7:          cwnd = upper_cwnd / 3
8:          packet_conservation = 1
9:    if current_ca_state < TCP_CA_Recovery
10:         and prev_ca_state >= TCP_CA_Recovery :
11:      packet_conservation = 0
12:   if packet_conservation == 1 :
13:      return true
14:   return false
15: bbr_set_cwnd:
16:   if bbr_set_cwnd_to_recover_or_restore() == true
17:      goto done
18:   target_cwnd =cwnd_gain × BtlBw × RTprop
19:   upper_cwnd = target_cwnd ×upper_scaling_factor
20: done:
21:   cwnd = min(cw nd + acked, upper_cwnd)
22: bbr_set_upper_scaling_factor:
23:   if loss_detected_time > 0 :
24:      if current_time>loss_detected_time+ current_rtt×3:
25:         if not is_rtt_min_updated :
26:            upper_scaling_factor = upper_scaling_factor×β
27:            last_action_time = current_time
28:            if upper_scaling_factor < 0.2 :
29:               upper_scaling_factor = 0.2
30:         loss_detected_time = 0
31:         is_rtt_min_updated = 0
32:   else:
33:      if current_time > last_action_time + 3 × current_rtt :
34:         upper_scaling_factor = upper_scaling_factor + δ
35:         if upper_scaling_factor > 1 :
36:            upper_scaling_factor = 1
37:         last_action_time = current_time
38: bbr_main: 0,0.3,0.6/* on acked */
39:   bbr_update_model ()
40:   bbr_set_pacing_rate () 0.5 /* not modified */
41:   bbr_set_tso_segs_goal () 0.5 /* not modified */
42:   bbr_set_cwnd ()
43:   bbr_set_upper_scaling ()
44:
45: * is_rtt_min_updated is set when RTprop is updated in bbr_update_model function
```

4.1. Minimum Round-Trip Time Update

The throughput between two flows fluctuates periodically when BBR and loss-based congestion control algorithm coexist on the same link with sufficient buffer size. When BBR overwhelms CUBIC flow, continuous packet loss occurs due to buffer overflow. BBR-CWS considers this packet loss to be due to abnormal data transmission due to an incorrect operating locality. In BBR-CWS, this packet loss is used as an indicator for moving toward appropriate operating point.

Before understanding our mechanism, we needed to be aware how BBR found packet loss in its implementation. TCP congestion control implementation uses a state machine to keep and switch between different states of a connection for recovery purposes. These different states are defined in a enum type in "ipv4/tcp.h" [35]. If packet loss is detected, the `tcp_ca_state` is converted to TCP_CA_Recovery (= 3) or TCP_CA_Loss (= 4). After the lost packet was retransmitted, `tcp_ca_state` is converted to TCP_CA_Open (= 0) again. Using this, BBR checks packet loss in "bbr_set_cwnd_to_recover_or_restore" function.

On Line 3 of Algorithm 1, `current_ca_state` represents the current `tcp_ca_state` in the tcp socket, and `prev_ca_state` represents `tcp_ca_state` when the previous ACK was received. When the `tcp_ca_state` is converted to TCP_CA_Recovery, BBR-CWS reduces the size of the congestion window by one-third of the amount of current inflight (Algorithm 1, Lines 3–9). The congestion window size is maintained until the lost packet is retransmitted and the TCP_CA_Recovery state is exited (Algorithm 1, Lines 10–12). Consequently, the excess queue filled by the BBR-CWS flow reduces, hence BBR-CWS sees a lower `RTprop` than before. We set the congestion window reduction to 1/3 of the current inflight so that BBR-CWS can reduce the congestion window size by more than half to eliminate excess queues.

After measuring the lower `RTprop`, BBR-CWS calculates a lower BDP than previously, i.e., maximum congestion window limitation is lower than previously. Packet loss again will occur if a single `RTprop` update does not reduce the inflight cap enough. In this situation, BBR repeats the above process until excessive packet loss no longer occurs, so that it finds a stable operating point.

4.2. Upper Congestion Window Scaling

When using a shallow buffer, only trying to find a lower `RTprop` might not be a good solution for moving the BBR's inflight cap, because BBR transmits more data than the link's capacity, no matter how low the `RTprop` measured by BBR, resulting in continuous buffer overflow and domination by BBR. Therefore, we can be aware that the biggest difference between operating with a large and a small buffer was whether `RTprop` could be updated when reducing the amount of inflight. If the buffer is large enough, long queue created by other flows makes BBR flow overestimate `RTprop`, so that BBR determines higher inflight cap resulting in being aggressive. In this case, if BBR reduces the congestion window sufficiently, it drains most of the buffer and the resulting `RTprop` will be updated. However, if the buffer is smaller than 1 BDP, BBR already has a value for `RTprop` which is close to the actual propagation round-trip time, and therefore cannot update the lower `RTprop`, although reducing the congestion window by packet loss. In the proposed scheme, BBR-CWS uses `upper_scaling_factor` in order to reduce the inflight data and sets `upper_cwnd`, which is the product of the existing `target_cwnd` and `upper_scaling_factor`, to adjust the inflight efficiently in shallow buffer.

(1) `upper_scaling_factor` **decrease:** When BBR-CWS detects packet loss, it temporarily reduces the congestion window size. Additionally, BBR-CWS then waits for the lower `RTprop` measured during up to 3 RTT. If the new `RTprop` is estimated within 3 RTT, BBR-CWS resets the `upper_scaling_factor` to 1 so that a new `RTprop` is entirely applied to the inflight-cap determination. Otherwise, BBR-CWS assumes that it has been exporting more data than the link can accept, and reduces the maximum inflight-cap by multiplying β (default: 0.9) in `upper_scaling_factor` (Algorithm 1, Line 29). We set the wait time for the new `RTprop` estimation to 3 RTT so as to give the BBR-CWS enough time to reduce inflight after cutting the congestion window. BBR-CWS performs the above process repeatedly, resulting in moving below the point where packet loss occurs and finding an appropriate operating point according to the bottleneck buffer size.

(2) `upper_scaling_factor` **increase:** To ensure fair competition with the loss-based congestion control algorithm which increases the congestion window continuously and to probe more available bandwidth, BBR-CWS increases the `upper_scaling_factor` by δ if no packet loss is detected for 3 RTT, so that BBR-CWS increases the inflight a little (Algorithm 1, Line 37). In this study, the δ value was set to 0.01. At this point, the maximum of `upper_scaling_factor` was fixed to 1. If packet loss

occur again following a recurring increase in `upper_scaling_factor`, BBR-CWS again reduces the congestion window in order to control the inflight. Through this method, BBR-CWS operates below the operating point at which packet loss occurs; moreover, it can compete fairly with the loss-based congestion control algorithms without long lasting queue.

In summary, BBR-CWS dynamically adjusts the excess queue size within the range of from 0 to 1 BDP through the `upper_scailing_factor` in buffer of less than 1 BDP. Furthermore, in a buffer of 1 BDP and above, BBR-CWS measures the appropriate `RTprop` to fairly coexist with other flows, resulting in leading to fair competition with loss-based congestion control algorithm.

5. Experimental Results

5.1. Mininet Experimental Topology

We constructed a Mininet emulation experiment environment as shown in Figure 7 in order to evaluate the inter-protocol fairness of the BBR-CWS algorithm. The Mininet experimental environment was implemented on a Ubuntu operating system (6 Core Intel 3.6 GHz CPU and 20-GB memory) with Linux kernel version 4.14. In Figure 7, Switches 1 and 3 generate pcap files for packets passing through the switch using "tcpdump" [36], respectively. Switch 2 configures the bottleneck bandwidth, buffer size, and latency, which are determined for each experimental scenario, using "netem" [37]. Each sender on the left-hand side of the diagram uses the "iperf" [38] application to send TCP segments to a receiver. We enabled the queueing discipline "fq" [39] of the network interfaces of the BBR hosts, and disabled it in one of the CUBIC hosts, since BBR determines the pacing rate of the outgoing packets carried out by "fq". We used two pcap files created by "tcpdump" and "tcpprobe", which are Linux basic tracing tools, in order to trace TCP parameters. In particular, "tcpprobe" was modified as needed, so that we could trace variables added in the BBR-CWS implementation.

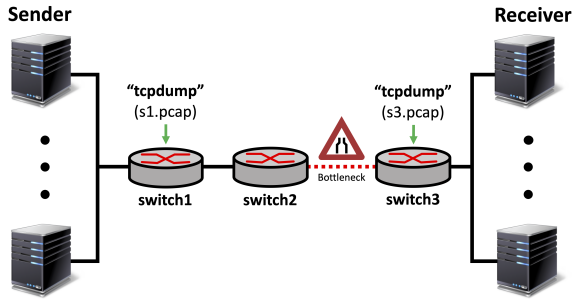


Figure 7. Mininet experimental topology.

5.2. Modest BBR: Benchmark Algorithm

We used Modest BBR [26] as a benchmark congestion control algorithm, which focused on the inter-protocol fairness issues with a loss-based congestion control algorithm. Modest BBR was implemented as a modification of BBRv1. Its pseudo code is shown in Algorithm 2, moreover the Table 1 describe the variables used in its implementation. Modest BBR adapted binary search phase and linear phase similar to binary increase congestion (BIC) algorithm [40]. We implemented Modest BBR only depending on its pseudo code, hence our Modest BBR results may not be entirely consistent with Zhang et al's. However, our Modest BBR implementation follows their main idea, and it showed similar results with the performance improvement that they presented in [26]. Therefore, we compared Modest BBR with our BBR-CWS.

Algorithm 2 Modest BBR congestion control.

```
if retransmission occurs and bin_p =0 then
    bin_p = 1, add_p = 0
    last_cwnd = cwnd
    min_win = cwnd× w_max
    max_win = (cwnd+min_win) / 2
    cwnd = max_win
else if bin_p = 1 then
    if retransmission occurs then
        max_win = max_win = (max_win + min_win) / 2
        cwnd = max_win
    else
        min_win = (max_win + min_win) / 2
        cwnd = min_win
    end if
    if (max_win - min_win) <= S_min then
        add_p = 1
        bin_p = 1
    end if
end if
if add_p = 1 and cwnd <= last_cwnd then
    for each ACK do
        cwnd = cwnd + S_min
    end for
end if
```

Table 1. Modest BBR congestion control parameters.

Parameter	Description
W_{min}	Minimum window scale factor
W_{max}	Maximum window scale factor
max_win	Upper bound of window size in binary search phase
min_win	Lower bound of window size in binary search phase
bin_p	Indicating in binary search phase
add_p	Indicating in linear increase phase
S_{min}	the minimum increase step

5.3. Single Operation Evaluation

We configured the BBR sending host to send data for 60 s, in order to evaluate the behavior of the original BBR, Modest BBR, and BBR-CWS hosts operated with a buffer of 1 BDP or less, respectively. In the Mininet topology, the bottleneck bandwidth was set to 50 Mbps, and the end-to-end propagation round-trip time was set to 10 ms. Figure 8 shows the results of three BBR flows on links with 0.5, 0.8, and 1 BDP buffers.

The original BBR creates significant packet retransmission, as shown in Figure 8a, because BBR static `cwnd_gain` causes buffer overflow and is passive to packet loss. Evidence that the original BBR passively deals with packet loss can be found in the average congestion window. In general, smaller buffers require smaller average congestion windows to avoid continuous packet loss. However, the original BBR show that average congestion window sizes are similar, regardless of buffer size. That is why BBR causes excessive loss in shallow buffer.

Modest BBR reduces packet retransmission compared to the original BBR, since Modest BBR recognizes packet loss as network congestion and has a mechanism to reduce congestion window size by packet loss. Consequently, Modest BBR has smaller average congestion window size than the original BBR for shallow buffer environments, reducing retransmission by up to 75%.

Furthermore, BBR-CWS has a smaller average congestion window when operating on links with a smaller buffer, although they have similar throughput to the original BBR. For example, BBR-CWS has 50 MSS average congestion window when operating on the 0.5 BDP buffer, whereas the original BBR average congestion window size is 82 MSS for the same buffer environment. However, when buffer size is 1 BDP, BBR-CWS has congestion window size of 71 MSS. BBR-CWS dynamically adjusts the `upper_scaling_factor` depending on the bottleneck buffer size, reducing average congestion window for the entire transfer time, as shown in Figure 8b. As a result, the `upper_scaling_factor` has a lower value when the buffer size is smaller. Moreover, the maximum of `upper_scaling_factor` is fixed at 1. Therefore, it prevents BBR-CWS from generating an excessive queue more than 1 BDP when it operates in a deep buffer of 2 BDP or more.

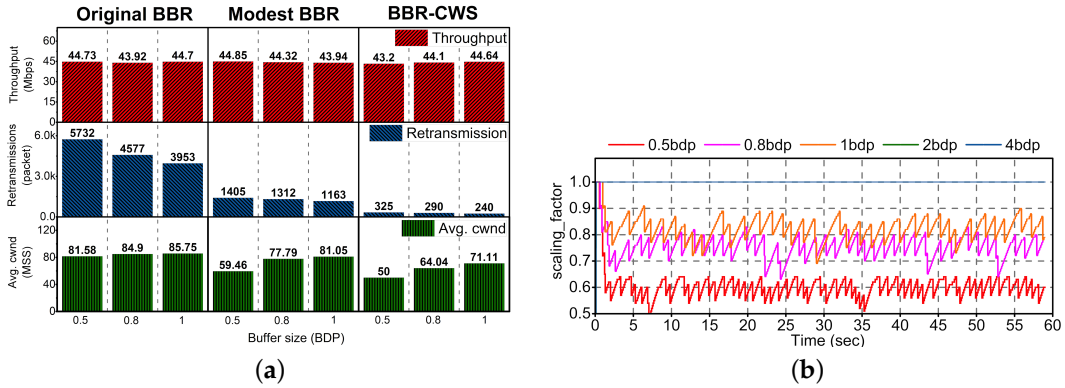


Figure 8. Single BBR-CWS operations: (**a**) `Throughput, loss and average cwnd of BBR:`; and (**b**) `upper_scaling_factor` of BBR-CWS.

5.4. Interactive Operation Evaluation

To evaluate the inter-protocol fairness performance of BBR-CWS when competing with the loss-based congestion control algorithm, we set up an experimental environment in which BBR and CUBIC transmitted data in four scenarios, as shown in Table 2. We set the shallow buffer to 0.5 BDP and 5 BDP for the deep buffer.

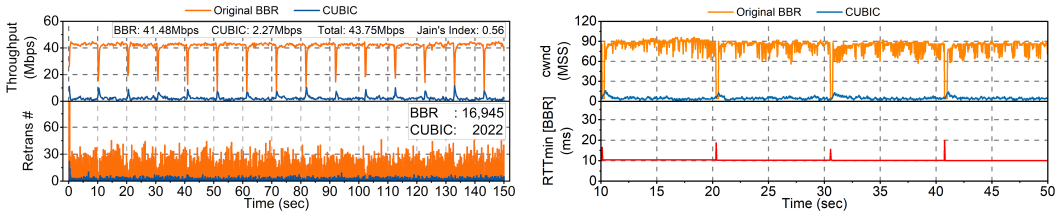
Table 2. Interaction experimental scenario.

Scenario #	`BtlBw`	End-to-End Delay
1	50 Mbps	10 ms
2		40 ms
3	20 Mbps	10 ms
4		40 ms

Suppose two hosts sent data for 150 s, starting at the same time. Figures 9 and 10 show the interactive operation results between the original BBR and CUBIC when the bottleneck buffer size is shallow (=0.5 BDP) and deep (5 BDP) in Scenario 1, respectively. In addition, Figures 11 and 12 depict the result of BBR-CWS and CUBIC in the same buffer and network scenario. As shown in each figure, we measured the throughput and the number of packet retransmissions at 0.1 intervals, and the `cwnd` and `RTprop` were measured each time the sending host received an ACK. The throughput and the number of retransmission are plotted over the entire transfer time; however, the `cwnd` and `RTprop` shows only a range between 10 and 50 s in order to closely analyze the BBR's behavior over time. Moreover, in these figures, we use Jain's fairness index [41] to evaluate how fairly multiple flows use the network resource.

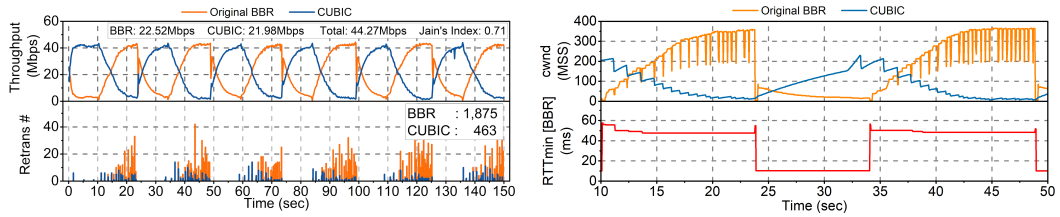
$$\mathcal{F} = \frac{1}{n} \times \frac{[\sum_{i=0}^{n} x_i]^2}{\sum_{i=0}^{n} x_i{}^2} \quad (1)$$

where n is the number of TCP flows, and x_i is throughput for the i-th flow.



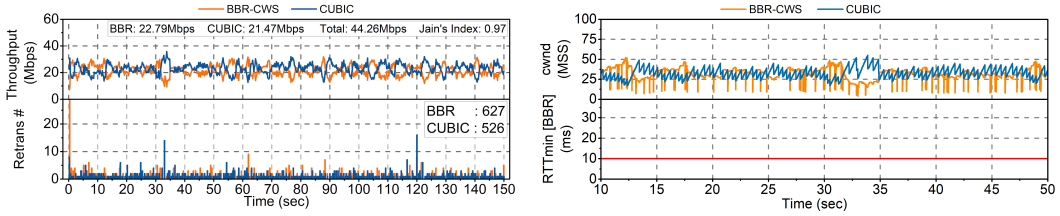
(**a**) (**b**)

Figure 9. Original BBR vs. CUBIC (shallow buffer: 0.5 BDP): (**a**) throughput and retransmission; and (**b**) `cwnd` and `RTprop` of the original BBR.



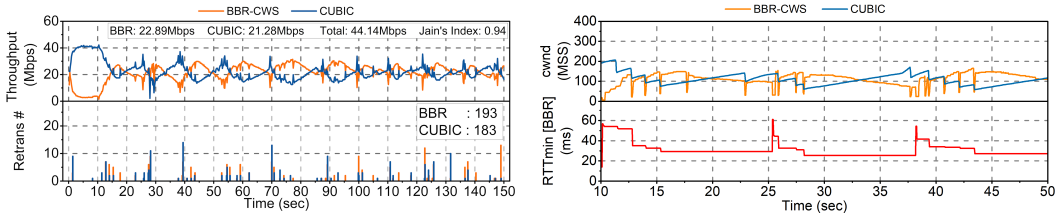
(**a**) (**b**)

Figure 10. Original BBR vs. CUBIC (deep buffer: 5 BDP): (**a**) throughput and retransmission; and (**b**) `cwnd` and `RTprop` of the original BBR.



(**a**) (**b**)

Figure 11. BBR-CWS vs. CUBIC (shallow buffer: 0.5 BDP): (**a**) throughput and retransmission; and (**b**) `cwnd` and `RTprop` of BBR-CWS.



(**a**) (**b**)

Figure 12. BBR-CWS vs. CUBIC (deep buffer: 5 BDP): (**a**) throughput and retransmission; and (**b**) `cwnd` and `RTprop` of BBR-CWS.

Figure 9a shows that the original BBR and CUBIC are seriously unfair when they operate on a shallow buffer. Especially, BBR takes up most of the bandwidth, and causes 16,945 excessive packet retransmissions. That is because the original BBR sets up an inflight-cap that exceeds the link's capacity, causing network congestion for the entire transfer time, which causes the CUBIC flow's congestion window to never grow as shown in Figure 9b. It can be seen that the `cwnd` of the original BBR temporarily decreases due to the continuous packet loss, but the congestion still continues due to the operation quickly returning to the previous `cwnd` after the recovery mode is terminated. When the buffer size was 5 BDP, cyclic performance fluctuation between the original BBR and CUBIC was observed, as shown in Figure 10a. Moreover, when BBR overwhelmed the CUBIC flow, it caused excessive packet retransmission. That is because the original BBR measures the longer `RTprop` than the actual link's propagation delay. At 10 s in Figure 10b, the original BBR measures the new `RTprop`, which is larger than the actual propagation round-trip time; hence, BBR's `cwnd` experiences drastic growth, which causes unfairness between two flows as well as excessive packet retransmissions.

However, when BBR-CWS and CUBIC operate on the same bottleneck link with 0.5 BDP buffers, we can see the fair data transmission in Figure 11a. Moreover, the number of packet retransmissions was reduced by 96%. Initially, in the shallow buffer, BBR-CWS sets high inflight-cap, so that it sends out much more data than CUBIC flow. This aggressive behavior of BBR-CWS causes continuous packet loss. In this case, BBR-CWS temporarily reduces the `cwnd` in order to measure the lower `RTprop`. However, BBR-CWS fails to measure the lower `RTprop` because it already has the `RTprop` similar to actual propagation round-trip time. Therefore, BBR-CWS reduces the `upper_scaling_factor` to adjust the inflight data. BBR-CWS repeats this process and varies its `cwnd` dynamically, so that it can lead to fair data transmission in competition with CUBIC, as shown in Figure 11b. With deep buffers, we can see that BBR-CWS and CUBIC fairly share the bottleneck link after the BBR-CWS's first ProbeRTT, as shown in Figure 12a. Because BBR-CWS measures the actual `RTprop` before the first ProbeRTT phase, it limits the inflight data and makes CUBIC flow occupy most of the bandwidth. After the `RTprop` is expired, BBR tries to measure the new `RTprop`; however, it sees `RTprop` which is completely different from the actual propagation delay because of the fully filled buffer by CUBIC flow. This abnormal measurement makes BBR-CWS aggressive, so that BBR-CWS sets higher inflight-cap and experiences excessive packet retransmission. In this case, BBR-CWS temporarily reduces its congestion window in order to show the lower `RTprop`. BBR-CWS drains the excess queue; therefore, it measures the lower minimum round-trip time. as shown in Figure 12b. By repeating this process, BBR-CWS can find the appropriate `RTprop` where it compete with CUBIC fairly.

Finally, Figure 13 shows the fairness index between BBR and CUBIC according to the bottleneck buffer size when end-to-end round-trip time is set to 10 and 40 ms, respectively. In this result, we compared the fairness performance of BBR-CWS with both the original BBR and Modest BBR. From the result of short latency (Figure 13a), we can be aware that the fairness between the original BBR and CUBIC entirely depends on the bottleneck buffer size. When the buffer size is smaller than 1 BDP, the fairness index between the two flows has a value from 0.5 to 0.6, which means that the two flows are seriously unfair. Moreover, when the buffer size is larger than 1 BDP, the fairness between the original BBR and CUBIC gets lower as the buffer grows larger because of the throughput variation. On the other hand, in environments with long-end-to-end latency of 40 ms (Figure 13b), the fairness between the original BBR and CUBIC still very low in shallow buffer environment, even though it has a good fairness about 0.9 when the buffer is deep.

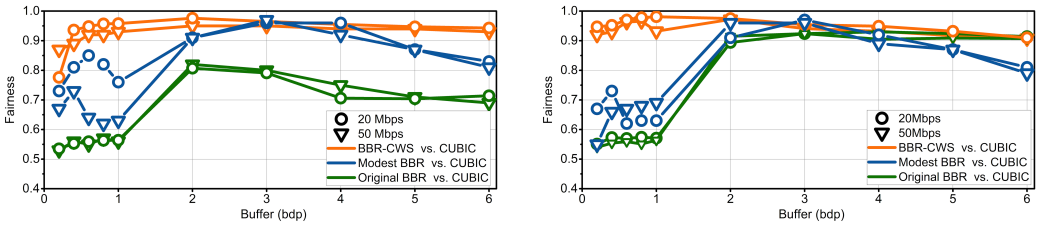


Figure 13. The fairness between BBR and CUBIC by the buffer size: (**a**) short latency (10 ms); and (**b**) long latency (40 ms).

There is no doubt that fairness between Modest BBR and CUBIC has improved over the original BBR, but, in the small buffer environment, it is still considered unfair given that the fairness index between Modest BBR and CUBIC is scattered across various distributions of 0.6 and 0.85. Moreover, when the buffer size is larger than 1 BDP, the fairness between Modest BBR and CUBIC is considerably improved compared to the original BBR. However, as the buffer size gets larger, the fairness between two flows sharply decreases.

In contrast, we can see that the fairness index between BBR-CWS and CUBIC represents an improved fairness index of 0.9 or higher in all latency environments without being affected by bottleneck buffer size. Although the fairness index between the two flows decreases with the buffer size when the buffer larger is than 1 BDP, the reduction is significantly lower than the original BBR and Modest BBR.

5.5. Multiple Flow Competition of BBR and CUBIC

We set up an emulation environment in which three BBR and three CUBIC hosts simultaneously transmitted data to a receiving host for 150 s to evaluate the fairness of multiple BBR and CUBIC flows. The experimental results are shown in Figures 14 and 15.

In Figure 14a, we can see serious unfair data transmission between the flows of the two groups of algorithms. Most of the bandwidth is occupied by the three original BBR flows. In the competition among BBR-CWSs and CUBIC flows, the two groups of algorithms take up similar bottleneck bandwidth (Figure 15a).

With a large buffer of 5 BDP (Figure 14b), the throughputs fluctuate continuously. However, Figure 15b shows that all six flows have reasonable bandwidth.

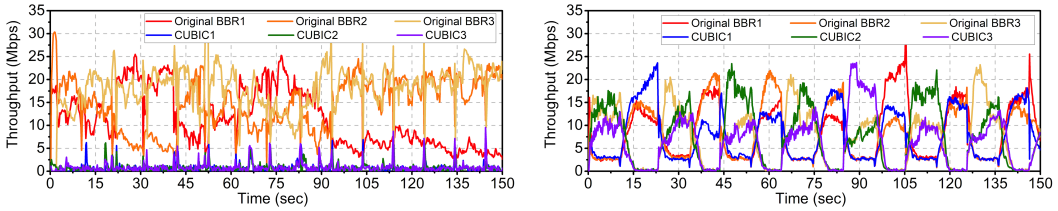


Figure 14. Three original BBR vs. three CUBIC: (**a**) shallow buffer (0.8 BDP); and (**b**) deep buffer (5 BDP).

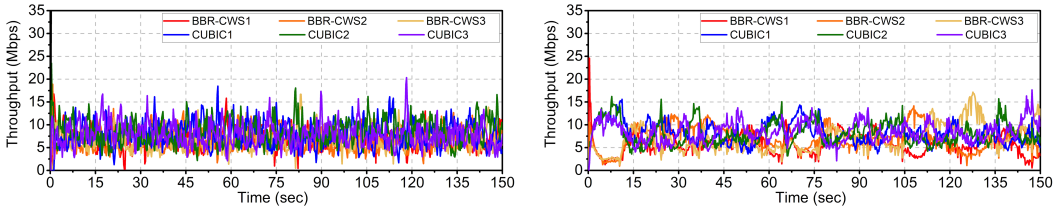


Figure 15. Three BBR-CWS vs. three CUBIC: (**a**) shallow buffer (0.8 BDP); and (**b**) deep buffer (5 BDP).

6. Conclusions

In this paper, we propose BBR-CWS to improve the inter-protocol fairness with loss-based congestion controls and reduce the excessive packet retransmissions. When the original BBR compete with loss-based congestion control algorithm such as CUBIC, it experiences unfair data transmission. Especially, the fairness between two algorithms completely depends on the bottleneck buffer size. With shallow buffers, which means that the buffer size is smaller than 1 BDP, BBR completely overwhelms CUBIC as well as creates excessive packet retransmissions. The throughput between BBR and CUBIC significantly fluctuates when the buffer size is larger than 1 BDP. This problem is caused by BBR's static operating point. To improve the inter-protocol fairness, we use congestion window scaling mechanism, that BBR-CWS dynamically adjusts its inflight-cap in loss recovery state. Therefore, BBR-CWS recognizes packet loss as the signal that it has estimated an abnormal operating point. By the packet loss, it moves the operating point resulting that finds the appropriate point where it gets a fair share with other TCP flows.

We performed experiment to evaluate the inter-protocol fairness of BBR. The experiment was conducted on a Mininet emulation. In the experiment results, BBR-CWS demonstrates a fairness index of 0.9 or higher in most buffer cases from 0.2 to 6 BDP, which means that BBR-CWS no longer depends on the buffer size. Moreover, active reaction to packet loss makes BBR-CWS reduce the excessive packet retransmission. This scheme is not harmful to the existing BBR behavior. While BBR-CWS achieves the improved inter-protocol fairness, it never creates a buffer that exceeds 1 BDP by the limitation of `upper_scaling_factor` growth, and it still shows a stable synchronization operation in multiple flow. Recently, BBRv2 tries to improve the inter-protocol fairness, but it still has inter-protocol fairness problem in a deep buffers, this approach can be used as a suitable solution.

Author Contributions: Y.-J.S. proposed the idea, conducted the experiments, and wrote the manuscript. G.-H.K. contributed by setting up the testbed and emulation environment. Y.-Z.C. supervised the entire research. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and ICT, 2017M3C4A7083676, and was funded by the Ministry of Education, 2018R1A6A1A03025109, and was funded by the Korea government (MSIT), 2019R1A2C1006249.

Acknowledgments: This research was supported by Next Generation Information computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF-2017M3C4A7083676) and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2018R1A6A1A03025109) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1006249.

Conflicts of Interest: The authors declare no conflict of interest

Abbreviations

The following abbreviations are used in this manuscript:

BBR	Bottleneck Bandwidth Round-trip propagation time
BBR-CWS	BBR Congestion Window Scaling
TCP	Transmission Control Protocol
BDP	Bandwidth Delay Product
RTT	Round-Trip Time
`BtlBw`	Bottleneck Bandwidth
`RTprop`	Round-Trip propagation time
`cwnd`	congestion window
IETF	Internet Engineering Task Force
MSS	Maximum Segment Size
DA-BBR	Delay-Aware BBR
Modest BBR	Moderately Aggressive BBR
DCTCP	Data Center TCP
ECN	Explicit Congestion Notification
BBRv2	BBR version 2
netem	network emulator

References

1. Postel, J. Transmission Control Protocol. 1981; pp. 1–16. Available online: https://www.rfc-editor.org/rfc/rfc793.txt (accessed on 18 February 2020).
2. Stevens, W.R. *TCP/IP Illustrated, the Protocols*; Addison-Wesley Professional: Boston, MA, USA, 1994; Volume 1.
3. Jacobson, V. Congestion avoidance and control. In Proceedings of the SIGCOMM, Stanford, CA, USA, 16–18 August 1988; pp. 314–329.
4. Afanasyev, A.; Tilley, N.; Reiher, P.; Kleinrock, L. Host-to-host congestion control for TCP. *IEEE Commun. Surv. Tutor.* **2010**, *12*, 304–342. [CrossRef]
5. Allman, M.; Paxson, V.; Blanton, E. TCP Congestion Control. Standards Track. 2009; pp. 1–18. Available online: https://www.rfc-editor.org/rfc/rfc5681.txt (accessed on 18 February 2020).
6. Henderson, T.; Floyd, S.; Nishida, Y. The NewReno Modification to TCP's Fast Recovery Algorithm. Standards Track. 2012; pp. 1–16. Available online: http://www.ietf.org/rfc/rfc6582.txt (accessed on 18 February 2020).
7. Ha, S.; Rhee, I.; Xu, L. CUBIC: A new TCP-friendly high speed TCP variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*. [CrossRef]
8. Gettys, J.; Nichols, K. Bufferbloat: Dark buffers in the Internet. *ACM Queue* **2011**, *9*, 40:40–40:54. [CrossRef]
9. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *ACM Queue* **2016**, *14*, 50:20–50:53. [CrossRef]
10. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control. In Proceedings of the IETF 97th Meeting, Seoul, Korea, 13–18 November 2016. Available online: https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf (accessed on 18 February 2020).
11. Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in computer communications. In Proceedings of the International Conference on Communications, Boston, MA, USA, 10–14 June 1979; Volume 43, pp. 1–10.
12. Hock, M.; Bless, R.; Zitterbart, M. Experimental evaluation of BBR congestion control. In Proceedings of the International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017.
13. Jaeger, B.; Scholz, D.; Raumer, D.; Geyer, F.; Carle, G. Reproducible measurement of TCP BBR congestion control. *Comput. Commun.* **2019**, *144*, 31–43. [CrossRef]
14. Scholz, D.; Jaeger, B.; Schwaighofer, L.; Raumer, L.; Geyer, F.; Carle, G. Toward a Deeper Understanding of TCP BBR Congestion Control. In Proceedings of the IFIP Networking, Zurich, Switzerland, 14–16 May 2018.
15. Lin, J.; Cui, L.; Zhang, Y.; Tso, F.P.; Guan, Q. Extensive evaluation on the performance and behavior of TCP congestion control protocols under varied network scenarios. *Comput. Netw.* **2019**, *163*, 106872. [CrossRef]

16. Turkovic, B.; Kuipers, F.A.; Uhlig, S. Fifty shades of Congestion Control: A Performance and Interaction Evaluation. *arXiv* **2019**, arXiv:1903.03852.
17. Cao, Y.; Jain, A.; Sharma, K.; Balasubramanian, A.; Gandhi, A. When to use and when not to use BBR; An empirical analysis and evaluation study. In Proceedings of the Internet Measurement Conference, Toulouse, France, 28–31 October 2019; pp. 130–136.
18. Sasaki, K.; Miyazawa, K.; Oda, N.; Hanai, M.; Kobayashi, A.; Yamaguchi, S. TCP Fairness among modern TCP Congestion Control Algorithms including TCP BBR. In Proceedings of the IEEE International Conference Cloud Netw. (CloudNet), Tokyo, Japan, 22–24 October 2018.
19. Atxutegi, E.; Haile, F.L.H.K.; Grinnemo, K.; Brunstrom, A.; Arvidsson, A. On the use of TCP BBR in cellular networks. *IEEE Commun. Mag.* **2018**, *56*, 172–179. [CrossRef]
20. Tao, Y.; Jiang, J.; Ma, S.; Wang, L.; Wang, W.; Li, B. Unraveling the RTT-fairness Problem for BBR: A Queueing Model. In Proceedings of the IEEE Global Comm (GLOBECOM), Abu Dhabi, UAE, 9–13 December 2018.
21. Ma, S.; Jiang, J.; Wang, W.; Li, B. Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis. *arXiv* **2017**, arXiv:1706.09115, 2017.
22. Kim, G.H.; Cho, Y.Z. Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement. *IEEE Access* **2019**, *8*, 4099–4109. [CrossRef]
23. Kozu, T.; Akiyama, Y.; Yamaguchi, S. Improving RTT Fairness on CUBIC TCP. *Int. J. Netw. Comput.* **2014**, *4*, 291–306. [CrossRef]
24. Song, Y.J.; Kim, G.H.; Cho, Y.Z. Improvement of Cyclic Performance Variation between TCP BBR and CUBIC. In Proceedings of the 25th Asia Pacific Conference on Communication (APCC), Ho Chi Minh City, Vietnam, 6–8 November 2019.
25. Haile, H.; Hurting, P.; Grinnemo, K.; Brunstrom, A.; Atxutegi, E.; Liberal, F.; Arvidsson, A. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. In Proceedings of the International Teletraffic Congress (ITC), Vienna, Austria, 4–7 September 2018.
26. Zhang, Y.; Cui, L.; Tso, F.P. Modest BBR: Enabling Better Fairness for BBR Congestion Control. In Proceedings of the IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018.
27. Li, F.; Chung, J.W.; Jiang, X.; Claypool, M. TCP CUBIC versus BBR on the Highway. In Proceedings of the Passive and Active Measurement Conference (PAM), Berlin, Germany, 26–27 March 2018.
28. Miyazawa, K.; Sasaki, K.; Oda, N.; Yamaguchi, S. Cycle and Divergence of Performance on TCP BBR. In Proceedings of the IEEE International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018.
29. Linux 4.14. 2017. Available online: https://kernelnewbies.org/Linux_4.14/ (accessed on 18 February 2020).
30. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Swett, I.; Vasiliev, V.; Jha, P.; Seung, Y.; Mathis, M.; Jacobson, V. BBRv2: A Model-based Congestion Control. In Proceedings of the IETF 104th meeting, Prague, Czech Republic, 23–29 March 2019. Available online: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00 (accessed on 18 February 2020).
31. Bensley, S.; Thaler, D.; Balasubramanian, P.; Eggert, L.; Judd, G. Data Center TCP (DCTCP): TCP Congestion Control for Data Centers. 2017; pp. 1–17. Available online: https://www.rfc-editor.org/rfc/rfc8257.txt (accessed on 18 February 2020).
32. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Swett, I.; Vasiliev, V.; Jha, P.; Seung, Y.; Mathis, M.; Jacobson, V. BBRv2: A Model-based Congestion Control. In Proceedings of the IETF 105th Meeting, Montreal, QC, Canada, 20–26 July 2019. Available online: https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00 (accessed on 18 February 2020).
33. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Jha, P.; Seung, Y.; Yang, K.; Swett, I.; Vasiliev, V.; Wu, B.; Hsiao, L.; et al. BBRv2: A Model-based Congestion Control. In Proceedings of the IETF 106th Meeting, Singapore, 16–22 November 2019. Available online: https://datatracker.ietf.org/meeting/106/materials/slides-106-iccrg-update-on-bbrv2 (accessed on 18 February 2020).
34. Song, Y.J.; Kim, G.H.; Cho, Y.Z. Congestion Window Scaling Method for Inter-protocol Fairness of BBR. In Proceedings of the Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2020.
35. Arianfar, S. TCP's Congestion Control Implementation in Linux Kernel. In Proceedings of the Seminar on Network Protocols in Operating Systems, Espoo, Finland, 7 September 2012.

36. Jacobson, V.; Leres, C.; McCanne, S. Tcpdump—Dump Traffic on a Network. Available online: http://manpages.ubuntu.com/manpages/bionic/en/man8/tcpdump.8.html (accessed on 18 February 2020).
37. Hemminger, S.; Ludovici, F.; Paul, H. NetEm—Network Emulator. Available online: http://manpages.ubuntu.com/manpages/bionic/man8/tc-netem.8.html/ (accessed on 18 February 2020).
38. Dugan, J.; Elliott, S.; Mah, B.A.; Poskanzer, J.; Prabhu, K. Iperf. Available online: https://iperf.fr (accessed on 18 February 2020).
39. Neukirchen, L. FQ—Job Queue Log Viewer. Available online: http://manpages.ubuntu.com/manpages/bionic/en/man1/fq.1.html/ (accessed on 18 February 2020).
40. Xu, L.; Harfoush, K.; Rhee, I. Binary increase congestion control (BIC) for fast long-distance networks. In Proceedings of the IEEE INFOCOM, Hong Kong, China , 7–11 March 2004; Volume 4, pp. 2514–2524.
41. Jain, R.; Chiu, D.M.; Hawe, W.R. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*; Digital Equipment Corporation: Maynard, MA, USA, 1984.