Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement

GEON-HWAN KIM AND YOU-ZE CHO, (Senior Member, IEEE)
School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea
Corresponding author: You-Ze Cho (yzcho@ee.knu.ac.kr)

This work was supported in part by the Next-Generation Information Computing Development Program through NRF funded by the Ministry of Science and ICT under Grant NRF-2017M3C4A7083676, 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, and in part by the National Research Foundation of Korea (NRF) funded by the Korea Government (MSIT) under Grant NRF-2019R1A2C1006249.

ABSTRACT In late 2016, Google proposed the Bottleneck Bandwidth and Round-trip propagation time (BBR) congestion control algorithm to achieve high bandwidth and low latency for Internet traffic. Unlike loss-based congestion control algorithms, BBR works without filling the bottleneck buffer. Consequently, BBR can reduce packet loss and minimize end-to-end packet delay, which has attracted the attention of many researchers in recent years. However, some studies have reported the creation of persistent queues that cause unintended problems, resulting in a serious fairness issue between TCP BBR flows with different round-trip times (RTTs). Although existing congestion control algorithms also exhibit fairness issue between different RTT flows, BBR has a more serious problem in that the imbalance is considerable even with small RTT difference between the two flows, and the long RTT flow uses most of the bandwidth. The preponderance of long RTT flows is a serious problem because a particular user may cause imbalance by maliciously increasing the delay. Therefore, we propose a Delay-Aware BBR (DA-BBR) congestion control algorithm to mitigate the RTT fairness issue between BBR flows. In a network emulation experiment using the Mininet, the DA-BBR increased the fairness index by 1.6 times that of the original BBR, and the retransmission was greatly reduced. DA-BBR flow with short RTT demonstrated fair throughput even in competition with DA-BBR flows where RTT is five times higher.

INDEX TERMS BBR, congestion control, fairness, round-trip time, TCP.

I. INTRODUCTION

The TCP congestion control algorithm, introduced in the 1980s, uses packet loss to indicate congestion [1]. This loss-based congestion control algorithm has worked well without throughput degradation for many years under conditions such as insufficient link bandwidth, small buffers in the network infrastructure, and Ethernet communications where packet loss is typically caused by buffer overflows. However, loss-based congestion control algorithms, which reduce the transmission window when a packet loss occurs, suffer throughput degradation in wireless environments because of packet loss unrelated to congestion. Moreover, the continuous development of buffer capacity has resulted in a large delay because congestion is only detected when the bottleneck buffer is full. Thus, new congestion control algorithms based on various approaches have been proposed to satisfy the requirements of today’s Internet traffic.

Google announced a new concept for congestion control, the Bottleneck Bandwidth and Round-trip propagation time (BBR) congestion control algorithm [2], to replace the loss-based congestion control algorithms. The end-to-end throughput of the TCP flow is determined by the bottleneck bandwidth (BtlBw), which is the data rate of the bottleneck link. Furthermore, the minimum round-trip delay for data transmission is the round-trip propagation time (RTprop) of the end-to-end link, as illustrated in Fig. 1. Therefore, until the Bandwidth-Delay Product (BDP) is reached, the value of RTprop is constant and then increases beyond the BDP. However, BtlBw increases until the BDP is reached and then no longer increases. BBR is a new paradigm of congestion control algorithms that dynamically controls sending rates by periodically measuring the end-to-end RTprop and the BtlBw to operate at the optimum operating point of Kleinrock [3].
Google has already introduced BBR as a congestion control algorithm and applied it to its YouTube server and cloud platform; Google’s experiments demonstrate that TCP BBR’s throughput shows an improvement of 2 to 20 times that of the traditional TCP CUBIC algorithm [1], [4]. TCP BBR is currently deployed and released on Linux’s network stack and is being improved to BBR v2 [4]. Improvements from BBR v1 to BBR v2 include increased fairness when sharing a bottleneck with Reno/CUBIC, reduced loss rates when the bottleneck buffer size is less than $1.5 \times BDP$, and increased throughput of the Wi-Fi path. Also, the DCTCP/L4S-style Explicit Congestion Notification (ECN) signal is used to cope with packet loss, and some behavior of the algorithm is modified. Because BBR is a work in progress, many issues [5]–[9] were reported for the initial BBR v1 announced by Google. One of the serious issues is throughput imbalance due to the RTT differences of BBR flows. A long RTT flow had sufficient throughput up to a $3 \times RTT$ difference. However, the throughput imbalance problem began with a small RTT difference of only 5 ms.

1) Why is RTT fairness a serious issue? In traditional TCP congestion control algorithms [1], [10]–[12], a shorter RTT corresponds with increased performance. Efforts to reduce RTT in both hardware and software are ongoing. However, BBR experiences a throughput fairness problem that disagrees with conventional wisdom. As illustrated in Fig. 2, Reno demonstrated that the throughput of a 10 ms RTT flow increased with an increase in the RTT gap, and a long RTT flow had sufficient throughput up to a $3 \times RTT$ difference. However, BBR exhibited a considerable increase in the throughput of the long RTT flow with an increase in the RTT gap. More serious was the problem that the throughput imbalance began with a small RTT difference of only 5 ms. This bias toward a long RTT flow in TCP BBR could cause critical fairness problems because certain users could acquire significant bandwidth by maliciously increasing the delays.

2) How can the RTT fairness problem be solved? Contrary to its intent, BBR transmits excessive data when multiple flows exist [5]. Excessive data transmission causes BBR to operate at the point where it is moved to the right from the initially-intended Kleinrock’s operating point presented in Fig. 1. If two BBR flows with different RTTs exist, both flows transmit an additional 1.5 BDP of inflight data [6]. Because the BDP is determined by the $RTprop$ of each flow, the long RTT flow has a larger BDP than the short RTT flow. Thus, the long RTT flow transmits more excess data than the short RTT flow, making more of the buffer backlog. The difference in the occupancy in the buffer results in a throughput fairness problem between the end hosts. Thus, we predict that RTT fairness can be improved by reducing the excessive data transmission of each BBR flow.

Therefore, we propose a Delay-Aware BBR (DA-BBR) algorithm to mitigate RTT imbalance and improve BBR throughput. DA-BBR uses RTT and $RTprop$ to reduce the excessive data transmission of the original BBR and varies the reduction of the sending rate depending on the $RTprop$ of each BBR flow. The larger the $RTprop$, the greater the reduction in the sending rate, thus eliminating a larger amount of buffer backlog.

We implemented DA-BBR based on BBR v1 [13] in Linux kernel v4.14 [14] and experimented on a network emulator using the Mininet. The emulation results demonstrated that DA-BBR improved the fairness index by approximately 1.6 times that of the original BBR algorithm. Consequently, the short RTT flow can use sufficient bandwidth when competing with the long RTT flow. Together with the improvement of the throughput fairness problem provided by the RTT difference, the reduced buffer backlog and appropriate countermeasures against the packet loss significantly reduced the number of retransmissions.
validating BBR v2 alpha since it was released. Because the RTT fairness problem between BBR v2 flows has not been verified yet, the approach proposed in this paper can be used if similar fairness problems arise in BBR v2.

The rest of the paper is organized as follows. We discuss related studies in Section II. In Section III, we present an overview of the BBR algorithm. In Section IV, we investigate the throughput fairness problem in BBR based on the emulation experiment results. We introduce the DA-BBR algorithm details in Section V. In Section VI, we evaluate our DA-BBR algorithm against the benchmark algorithm based on the emulation experiment results. Finally, we conclude this paper in Section VII.

II. RELATED WORK
After the release of the BBR algorithm by Google [2], many papers evaluating the performance of TCP BBR were presented [5], [6], [15]. Hock et al. [5] identified the design problem of the BBR algorithm: multiple BBR flows can result in the creation of a persistent queue in the bottleneck. Scholz et al. [6] performed a performance evaluation of BBR using the Mininet emulation rather than a testbed experiment. They analyzed the flow-to-flow synchronization behavior of BBR. Atxutegi et al. [15] compared the performance of BBR, NewReno, and CUBIC over emulation and real mobile networks. They concluded that BBR outperforms other algorithms in a mobile environment but does not always provide consistent results depending on the operator’s network state or flow type.

A study focusing on interactions with each other, rather than performance evaluation with other algorithms, was also proposed [16]–[18]. Ware et al. [16] argued that BBR’s inflight cap is a key element of BBR behavior and demonstrated that BBR becomes window-limited and ACK-clocked—similar to traditional congestion control—when BBR flows compete with other algorithms. Sasaki et al. [17] Miyazawa et al. [18] examined fairness between CUBIC and BBR and found extreme inter-protocol throughput imbalances and periodic throughput variations when coexisting with CUBIC. Zhang et al. [9] proposed Modest BBR to mitigate the aggressiveness of BBR when the bottleneck buffer is small. BBR was modified to be sensitive to packet loss, reducing the sending rate based on loss so the CUBIC flow could use the proper bandwidth.

These studies address the issue of fairness between BBR and other congestion control algorithms. However, the inter-protocol fairness problem of BBR is also evident, with a very large degree of imbalance. Unlike traditional congestion control, where short RTT flows increase the congestion window (cwnd) rapidly and use more bandwidth, short RTT BBR flows are significantly suppressed by long RTT BBR flows.

Several studies evaluating BBR have also presented experimental results for RTT fairness problem [5], [6]. Ma et al. [8] analyzed the causes of BBR’s RTT fairness problem and proposed the BBQ algorithm as a solution. BBQ improved RTT fairness by reducing probing time for long RTT flows. Tao et al. [19] demonstrated that the bandwidth occupancy between different RTT flows depends on the RTT ratio and built a theoretical model to characterize BBR bandwidth dynamics.

In our previous paper [20], the inflight cap was limited to less than 2 BDP in ProbeBW to suppress queue creation. If the link is not fully utilized, it behaves similar to the original BBR; otherwise, it is limited to 1 BDP when sending data. However, this method was slow and unstable in convergence, and the limitation to 1 BDP caused a decrease in total throughput. Therefore, we modified the previous method to increase total throughput and stabilize behavior [21]. However, because this modified method simply reduced the amount of data transferred over the original BBR, inter-protocol fairness problem was still a concern.

To this end, this paper provides a new approach and solution to improve BBR’s RTT fairness by explaining the relationship between RTT and RTprop. It also improves the throughput of short RTT flows that were suppressed. Because RTT fairness between BBR v2 flows has not yet been verified, the approach proposed in this paper can be used for similar RTT fairness problems.

III. BBR OVERVIEW
This section discusses the operating points of TCP congestion control and describes the features and states of the BBR algorithm.

A. OPERATING POINT OF THE TCP CONGESTION CONTROL ALGORITHM
The traditional TCP loss-based congestion control algorithm uses packet loss as an indicator of congestion. The sending host continuously increases the sending rate until a packet loss occurs in the bottleneck, and if a packet loss occurs, it judges this loss as congestion and reduces the sending rate. Transmitting data beyond the BDP creates a queue at the bottleneck and results in an increase in the end-to-end propagation delay. When the amount of excess inflight data reaches the maximum buffer size of the bottleneck link, the buffer is full and the newly arrived packets are dropped. Point (B) in Fig. 1 is the point at which a packet loss occurs because of the buffer overflow, which is the operating point of the loss-based congestion control. Therefore, RTT is highly relative to the size of the buffer because of its characteristic of filling the bottleneck buffer fully.

The end-to-end throughput of the TCP flow is determined by the delivery rate of the bottleneck link. The BDP is obtained by multiplying BtlBw (the delivery rate of the bottleneck) and RTprop (the round-trip propagation time of the link). To use 100% of the available capacity of the link, the amount of inflight data delivered across the entire link should be equal to the BDP.

In 1979, Leonard Kleinrock [3] proved that point (A) in Fig. 1 is the optimum operating point for obtaining the maximum throughput and minimum transmission delay. BBR is a congestion control algorithm that periodically measures...
RTprop and BtlBw to operate at Kleinrock’s optimum operating point and dynamically controls the sending rate while keeping the bottleneck full but not fuller.

B. DATA DELIVERY OF BBR

Fig. 3 illustrates the data delivery process of BBR. It builds an explicit model for the available BtlBw and RTprop of the pipes by using periodically-measured bandwidth and end-to-end propagation delay samples. The BtlBw and RTprop measurements are fed to a type of probing state machine that increases or decreases the amount of data in transit (inflight) to maintain an adequate number of packets in the pipe. Then, the data delivered from the higher layer are reduced to a quantum size by a transport layer protocol such as TCP and forwarded at a given rate so that they do not exceed the maximum amount of data (cwnd).

C. FOUR STATES OF BBR

The BBR algorithm operates through four states as illustrated in Fig. 4.

1) Startup: At the beginning of the BBR flow, the sending rate is considerably increased by an exponential increase in each RTT. If the increase in the sending rate does not exceed 25% for three consecutive attempts at increasing the sending rate, the bottleneck bandwidth is considered to be found and the Startup state is terminated. The pacing_gain in the Startup state is approximately 2.885, and the maximum amount of data that can be delivered is limited to 3 BDP. Thus, an excess queue of 2 BDP can be created except for 1 BDP that fills the pipe.

2) Drain: If the bottleneck bandwidth is found in the Startup state, the Drain state is performed. A process to eliminate a queue created from rapid increases in the previous state is executed. In the Drain state, the reciprocal of

the pacing_gain in the Startup state is used to remove the excess queue. This value is approximately 0.35, and if the amount of data being transferred decreases to 1 BDP, the Drain state is terminated and the ProbeBW state is performed.

3) ProbeBW: BBR stays in ProbeBW most of the time to probe the available bandwidth. This state consists of eight cycles with a pacing_gain of [1.25, 0.75, 1, 1, 1, 1, 1, 1], each cycle lasting for RTprop. First, when the pacing_gain is 1.25, it sends more data to check whether there is any available bandwidth. For the subsequent value of 0.75, BBR removes any excess queue generated from the previous probing. In the remaining six cycles, the maximum bottleneck bandwidth obtained by probing is used to send the data at a constant speed of pacing_gain=1.

4) ProbeRTT: If the new RTprop (lower measured value) is not measured for 10 s during the ProbeBW state, BBR enters the ProbeRTT state. In ProbeRTT, to measure the actual RTprop of the pipe, the maximum number of packets is reduced to four to remove all possible queues from the bottleneck. To ensure the measurement of the minimum RTprop, this state lasts for 200 ms + RTprop, and when a new peak value is measured, it is updated to the RTprop and maintained for 10 s.

IV. SUPPRESSED SHORT RTT FLOW

In this section, we investigate the RTT fairness issue of BBR based on the experimental results and analyze the cause of the bias towards the long RTT BBR flow. We conducted a test with BBR v1 and found that the short RTT flow was excessively suppressed by the long RTT flow during the entire test duration.

A. EMULATION ENVIRONMENT

The experimental environment for evaluating the RTT fairness problem was constructed as illustrated in Fig. 5. We used the framework implemented by Jaeger et al. [22]. Mininet was run on Ubuntu with Linux kernel version 4.14, and both the senders and destinations used BBR v1. Based on the considered scenario, network parameters such as RTT, bottleneck buffer size, and bandwidth were configured through Netem (network emulator) [23]. Furthermore, we changed the queuing discipline to fair queueing (fq) [24] to enable pacing. The data were transmitted by each sender to the destination by using iPerf3 [25]. Because the RTT fairness problem of BBR always occurs in sufficient buffer environments, we set the bottleneck buffer size to 7 BDP on the basis of the 10 ms RTT flow when the bottleneck bandwidth is 100 Mbps.

B. TEST RESULTS

Figs. 6(a)–(b) illustrate the throughput of the BBR flows, one with 10 ms RTT and the other with 50 ms and 12 ms RTT each, competing on a 100 Mbps bottleneck link. When the 50 ms 10 ms TCP BBR flows compete, the 50 ms flow uses most of the bandwidth, as illustrated in Fig. 6(a). The 10 ms BBR flow exhibits an increase in throughput immediately
The bias toward the long RTT flow of TCP BBR also occurs when the RTT difference between the two BBR flows is only 2 ms. A large throughput gap between the two flows is apparent in Fig. 6(b) when a 10 ms BBR flow competes with a 12 ms BBR flow. The result of the long RTT flow overwhelming the short RTT flow is opposite of the bias towards the short RTT flow in the traditional TCP congestion control algorithm. The bias toward the long RTT flow in TCP BBR might cause serious fairness problems because certain users could acquire high bandwidth by maliciously increasing the delays. Also, as illustrated in Fig. 2, the RTT fairness worsens as the RTT difference between the two TCP BBR flows increases. However, the large throughput gap due to only the small RTT differences (10 ms to 15 ms) is more remarkable than the imbalance caused by the large RTT differences. The long RTT flow uses $2 \times$ bandwidth, even when the RTprop is only 2 ms larger than the 10 ms RTT flow.

### C. CAUSES OF RTT FAIRNESS PROBLEM

There are two causes of BBR’s RTT fairness problem [8]. First, probing more bandwidth with $pacing\_gain = 1.25$ during the ProbeBW state creates an excessive persistent queue on the bottleneck. Second, the BDP, which determines the amount of data to be delivered, is calculated with the measured $BtlBw$ and $RTprop$; because the long RTT BBR flow has a higher BDP, it sends a larger amount of data and uses more of the bottleneck buffer. In contrast, the short RTT flow does not use sufficient bandwidth or bottleneck buffer. Moreover, the short RTT flow is limited by the $cwnd$, which is set to 2 BDP and fails to find available bandwidth during probing. Accordingly, the short RTT flow does not find additional bandwidth despite more frequent probing.

We analyzed the changes in $cwnd$ and the amount of inflight data for 30–40 s for the two BBR flows in Fig. 6(a) to investigate the limitation caused by $cwnd$ that decreased performance. As illustrated in Fig. 7(a), the 10 ms BBR flow no longer increases after reaching its upper limit ($cwnd$) and does not deliver additional packets. The dominant 50 ms BBR flow in Fig. 7(b) reveals that probing for additional bandwidth was performed because it had sufficient free space even though it was sending considerably more data than the 10 ms BBR flow. Thus, the short RTT flow limits itself and does not have a chance to reverse.

### V. DA-BBR: DELAY-AWARE BBR ALGORITHM

To solve the two main causes of the RTT fairness problem, the inherent issue of the BBR algorithm must be addressed. According to [5], when multiple BBR flows exist, the queuing delay that occurs is approximately 1–1.5 times the RTT. Therefore, if multiple BBR flows share a common bottleneck link, BBR does not operate at the operating point of Kleinrock but rather at the point where it has moved slightly to the right.

![FIGURE 5. Emulation topology with the Mininet.](image_url)

![FIGURE 6. Throughput of two BBR flows: (a) 10 ms vs. 50 ms. (b) 10 ms vs. 12 ms.](image_url)

![FIGURE 7. cwnd and the amount of inflight data of two BBR flows: (a) 10 ms BBR flow is limited by cwnd. (b) 50 ms BBR flow has free space for probing.](image_url)
We have illustrated the delivery rate and the round-trip time graphs in Fig. 1 for BBR and with different RTTs in Fig. 8. The intended operating point is slightly different for the two flows: the long RTT flow had a BDP larger than that of the short RTT flow. Because of the additional 1.5 BDP data, the long RTT flow sends more data into the pipe. Thus, if two BBR flows with different RTTs compete, the gap between the actual operating points of the two flows becomes wider and the long RTT flow uses significantly more of the buffer and bandwidth, resulting in throughput imbalance.

To address this inherent design issue, we propose DA-BBR to identify the changes in RTT and RTprop and reduce the amount of inflight data based on the RTT values of each BBR flow. Two modifications are added to improve the throughput of BBR, along with the improvement of RTT fairness.

### Algorithm 1 Improvement of RTT Fairness

To solve the problem of BBR transmitting excessive data unlike its intended behavior, the measured RTT and RTprop are used to reduce the amount of inflight data from each flow. Although all BBR flows send more data than intended, long RTT flows transmit significantly larger amounts, and therefore differential reduction is required based on the RTT of each flow. We investigated the relationship between RTT and RTprop, and defined a regulating factor with a value less than 1, and this regulating factor was observed as illustrated in Fig. 9 based on the RTT differences. The regulating factor has a value of 0.7 in the competition between BBR flows with the same RTT, and as the difference of RTT between the two flows increases, the regulating factor of the short RTT flow increases. Simultaneously, the regulating factor of the long RTT flow decreases, and the competition between the 10 ms RTT flow and the 50 ms RTT flow produces values of approximately 0.8 and 0.6. The BDP calculation multiplies the regulating factor \( \gamma \) to reduce the amount of inflight data differently depending on the RTT. The regulating factor \( \gamma \) is defined as follows:

\[
\gamma = \frac{RTT}{RTT + RTprop}
\]

The long RTT flow has a lower \( \gamma \) than the short RTT flow, and the reduction of the long RTT flow is greater than the BDP reduction of the short RTT flow. Therefore, as it sends less data than the original BBR that generated a queuing delay by sending excessive data, the buffer share of the long RTT flow is considerably reduced, thus improving RTT fairness.

However, sending less data than the original BBR may lead to a different result in the competition with the loss-based congestion control algorithm. The original BBR that competes with CUBIC in a large bottleneck buffer shows periodic throughput variation due to the measured RTprop. CUBIC is dominant when the measured RTprop of BBR is similar to the actual propagation time, and BBR is dominant when the measured RTprop is high because of the queuing delay created by CUBIC. To ensure that co-existence with the existing CUBIC is maintained, DA-BBR behaves similar to the original BBR when the current RTprop is greater than the threshold:

\[
RTprop > \alpha \times RTprop_{prev}
\]
B. IMPROVEMENT OF THROUGHPUT

Besides the RTT fairness improvement, additional changes have been made to increase the throughput of the original BBR and to react appropriately to packet loss.

Mitigation of cwnd reduction. Using Algorithm 1, DA-BBR transmits less data than the original BBR. This reduces the load on the bottleneck buffer, so the throughput can be improved by adjusting the cwnd reduction in the ProbeRTT state. Delivering four packets during the ProbeRTT state is a drastic change and causes instability in the inter-protocol competition. Consequently, BBR v2 is intended to reduce the cwnd by 50%. However, because DA-BBR is an algorithm based on BBR v1, if cwnd reduction is applied in the same way as BBR v2, then sufficient queues are not removed during the ProbeRTT state when competing with CUBIC. Therefore, we led a throughput improvement by decreasing the cwnd value by only one-third during the ProbeRTT state.

Reaction to packet loss. BBR v1 causes considerable packet loss because it does not react appropriately to the packet loss, whether it was caused by congestion or a network flaw. To reduce the high number of packet retransmissions, we modified the algorithm to reduce the pacing_gain when a packet loss occurred. Moreover, “restore” was not performed, as it was deemed unnecessary in the event of a packet loss. The detailed modifications are described in Algorithm 2.

VI. EVALUATION OF DA-BBR

The performance of DA-BBR is evaluated in this section. We conducted an emulation test based on Linux network namespaces with the Mininet [22], [26].

Algorithm 2 Improvement of Throughput

1: if state \( \equiv \) TCP_CA\(_{\text{recovery}} \) and state\(_{\text{prev}} \neq \) TCP_CA\(_{\text{recovery}} \) then
2: \( \text{cwnd} = \text{packet}_{\text{inflight}}(p) \)
3: \( \text{bbr} \rightarrow \text{pacing}_{\text{gain}} = 0.75 \)
4: else if state\(_{\text{prev}} \geq \) TCP_CA\(_{\text{recovery}} \) and state < TCP_CA\(_{\text{recovery}} \) then
5: \( \text{bbr} \rightarrow \text{restore}_{\text{cwnd}} = 0 \)
6: end if

FIGURE 10. Comparison of throughput and buffer backlog: (a) BBR with 1 BDP buffer. (b) BBQ with 1 BDP buffer. (c) DA-BBR with 1 BDP buffer. (d) BBR with 4 BDP buffer. (e) BBQ with 4 BDP buffer. (f) DA-BBR with 4 BDP buffer.

where \( \text{RTprop}_{\text{prev}} \) is the RTprop used in the previous ProbeBW state and \( \alpha = 1.2 \) to distinguish whether the opponent is CUBIC or not. Algorithm 1 describes the improvement of RTT fairness in DA-BBR.

Algorithm 2 Improvement of Throughput

1: if state \( \equiv \) TCP_CA\(_{\text{recovery}} \) and state\(_{\text{prev}} \neq \) TCP_CA\(_{\text{recovery}} \) then
2: \( \text{cwnd} = \text{packet}_{\text{inflight}}(p) \)
3: \( \text{bbr} \rightarrow \text{pacing}_{\text{gain}} = 0.75 \)
4: else if state\(_{\text{prev}} \geq \) TCP_CA\(_{\text{recovery}} \) and state < TCP_CA\(_{\text{recovery}} \) then
5: \( \text{bbr} \rightarrow \text{restore}_{\text{cwnd}} = 0 \)
6: end if
A. BENCHMARK ALGORITHM

The BBQ algorithm [8] and the original BBR algorithm were used as benchmark algorithms for the performance comparison. BBQ is the proposed algorithm to address the problem of RTT fairness of the original BBR. It continuously detected the excess queues and limited the time for probing when a queue was created to prevent long RTT flows from transmitting a considerable amount of traffic to the pipe. When the pipe was full, the two BBQ flows had a similar buffer backlog because they performed the probing with $pacing\_gain = 1.25$ simultaneously ($\alpha = 3\text{ ms}$). When the pipe had available space, the algorithm performed similarly to the original BBR to prevent performance degradation. Because the code was not disclosed, we implemented the BBQ algorithm ourselves based on the information provided in the related reference. The details of the BBQ algorithm are described in Algorithm 3.

Algorithm 3 BBQ Algorithm

1. bool $is\_full\_length = tcp\_stamp\_us\_delta > RTprop$
2. bool $is\_full\_length\_for\_BBQ = tcp\_stamp\_us\_delta > \alpha$
3. if $pacing\_gain > 1.0$ then
4. if $RTT \geq 1.5 \times RTprop$ and $RTprop > \alpha$ then
5. return $(is\_full\_length\_for\_BBQ)$ and $(rs \rightarrow losses$ or $inflight \geq bbr\_target\_cwnd)$
6. else
7. return $(is\_full\_length)$ and $(rs \rightarrow losses$ or $inflight \geq bbr\_target\_cwnd)$
8. end if
9. end if
B. EMULATION RESULTS
We obtained emulation results for DA-BBR, BBQ, and BBR for the throughput and the buffer backlog of the bottleneck link, as illustrated in Fig. 10.

The results of experiments conducted using small bottleneck buffers (1 BDP for a 50 ms RTT flow) are shown in the order BBR, BBQ, and DA-BBR in Fig. 10(a)–(c). BBR demonstrated a 50 ms RTT flow, overwhelming the 10 ms RTT flow during the entire test time, while the backlogs in bottleneck buffers averaged approximately 3 Mbit. After the ProbeRTT state, which occurred every 10 s, the 10 ms RTT flow increased the sending rate temporarily but did not reverse the 50 ms RTT flow. Instead, the aggressiveness of the 10 ms RTT flow at that time caused a considerable number of lost packets. For the BBQ in Fig. 10(b), the throughput gap between the two flows decreased, but the 50 ms RTT flow still had the upper hand. In the case of the buffer backlog, the occupancy in the ProbeBW state was reduced noticeably as compared to BBR, but the buffer backlog after the ProbeRTT state was similar to that of BBR. For DA-BBR (Fig. 10(c)), the gap between the two flows gradually decreased over time, demonstrating equitable throughput after the third ProbeRTT state. The buffer backlog also demonstrated considerably more stability and lower occupancy than the two aforementioned algorithms. This indicates the buffer backlog decreased significantly.

As illustrated in Figs. 10(d)–(f), we observed the performance of each algorithm for a large buffer (4 BDP for the 50 ms RTT flow). All three algorithms yielded similar results regardless of the buffer size, but packet loss was reduced because of the sufficient buffer size. DA-BBR demonstrated a slight gap compared with the previous results, but this gap was soon reduced.

C. EMULATION EVALUATION
We evaluated the emulation results for throughput, Jain’s fairness index [27], and the number of retransmissions.

1) THROUGHPUT
Figs. 11(a)–(c) illustrate the throughput of the three algorithms based on the RTT changes of the long RTT flow at the 1 BDP buffer. For a BBR in the 1 BDP buffer, the throughput difference is significant from the point where the RTT difference doubles. From the \(3 \times \text{RTT} \) difference, the long RTT flow uses approximately 90% of the bandwidth. The BBQ in Fig. 11(b) demonstrates sufficient throughput for the 10 ms flow, but it does not show a constant change depending on the RTT difference. For the DA-BBR algorithm, 10 ms DA-BBR flow uses sufficient bandwidth when competing with the 50 ms RTT flow, and overall, the DA-BBR algorithm demonstrates greater fairness than the BBQ algorithm. When the size of the bottleneck buffer is set to 4 BDP based on the long RTT flow, the throughput difference between the two flows increases. The long RTT BBR flow occupies 80% bandwidth, except when the RTT is the same (Fig. 11(d)). As illustrated in Fig. 11(e), BBQ demonstrates greater fairness than BBR, but the change is inconsistent, as in the 1 BDP buffer case. The 10 ms DA-BBR flow in Fig. 11(f) demonstrates dominance up to the competition with a 40 ms RTT flow and then reverses.

None of the algorithms have a fair share when competing with the 100 ms RTT flow. However, for the 10 ms DA-BBR flow, the improvement is noticeable because it uses four times more bandwidth than the BBR flow.

2) FAIRNESS INDEX
For a fairness comparison, we used Jain’s index [27] based on throughput to represent how flows share bandwidth fairly. For two flows, the fairness index is obtained as follows:

\[
F = \frac{(x_1 + x_2)^2}{2(x_1^2 + x_2^2)}
\]

If one flow does not occupy any bandwidth, a value of 0.5 is obtained, with the other flow at a value of 1. Fig. 12 illustrates the fairness index of each algorithm for (a) the bottleneck bandwidth of 10 Mbps and (b) the result of 100 Mbps. For the 10 Mbps bottleneck bandwidth, the 10 ms DA-BBR flow has a high fairness index in competition with the 100 ms RTT flow. With a fairness of at least 0.95, the RTT fairness problem is considerably improved. For BBQ, the competition until the 50 ms RTT flow demonstrates similar fairness to DA-BBR but does not use sufficient
VII. CONCLUSION

In this paper, we proposed DA-BBR to improve both the RTT fairness problem and the throughput of BBR. The original BBR had the problem of transmitting excessive data, which resulted in throughput imbalance for different RTTs. This RTT fairness became a critical problem because the long RTT flow prevailed. The bias toward the long RTT flow in TCP BBR could cause significant fairness problems by maliciously increasing delays. To solve this issue, a regulating factor using RTT and RTprop was defined and applied to the BDP calculation of BBR to adjust the amount of inflight data. The flow with long RTprop decreased more significantly, mitigating the throughput imbalance. The proposed DA-BBR was implemented in the Linux kernel and tested in the emulation environment using the Mininet.

We performed experiments to investigate the RTT fairness issue of BBR, and a Mininet-based emulation was performed to evaluate the proposed DA-BBR. The emulation results revealed that the proposed DA-BBB demonstrates a fairness index of 0.95 or higher in most cases, and the bandwidth share of the short RTT flow increases by approximately four times as compared to the original BBR. Moreover, because of the prevention of excessive data transmission and the appropriate response to packet loss, the number of retransmissions in the environment where excessive retransmission occurred in the original BBR is significantly reduced. Future work will investigate the RTT fairness of BBR v2, and if the same RTT fairness problem exists, the approach proposed in this paper offers a suitable solution.

REFERENCES

[1] V. Jacobson, “Congestion avoidance and control,” ACM SIGCOMM Comput. Commun. Rev., vol. 18, no. 4, pp. 314–329, 1988.
[2] 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.
[3] 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.
[4] N. Cardwell, “BBR v2: A model-based congestion control,” in Proc. ICCRG IETF 104th Meeting, Mar. 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrг-an-update-on-bbrv0
[5] M. Hock, R. Biess, and M. Zitterbart, “Experimental evaluation of BBR congestion control,” in Proc. Int. Conf. Netw. Protocols (ICNP) Oct. 2017.
[6] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, “Towards a deeper understanding of TCP BBR congestion control,” in Proc. IFIP Netw. May 2018.
[7] R. J. Borgli and J. Misund. Comparing BBR and CUBIC Congestion Controls. [Online]. Available: https://www.uio.no/studier/emner/matnat/ifi/INF5072/v18/inf5072_example1.pdf
[8] S. Ma, J. Jiang, W. Wang, and B. Li, “Fairness of congestion-based congestion control: Experimental evaluation and analysis,” Jun. 2017, arXiv:1706.09115. [Online]. Available: https://arxiv.org/abs/1706.09115
[9] 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, pp. 00646–00651.
[10] S. Floyd and T. Henderson. The NewReno Modification to TCP’s Fast Recovery Algorithm’. Accessed: 1999. [Online]. Available: https://tools.ietf.org/html/rfc2582

[11] S. Floyd and T. Henderson. The NewReno Modification to TCP’s Fast Recovery Algorithm’. Accessed: 1999. [Online]. Available: https://tools.ietf.org/html/rfc2582

[12] L. Xu, K. Harfoush, and I. Rhee. “Binary increase congestion control (BIC) for fast long-distance networks.” in Proc. IEEE INFOCOM, Mar. 2004, pp. 2514–2524.

[13] N. Cardwell. BBR VI Release. Accessed: Sep. 9, 2019. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/tree/net/ipv4/tcp_bbr.c

[14] G.-H. Kim, I. Mahmud, and Y. Z. Cho, ”Fairness improvement of BBR congestion control algorithm for different RTT flows,” in Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet), Oct. 2018, pp. 1–6.

[15] G. H. Kim, I. Mahmud, and Y. Z. Cho, ”Enhanced BBR congestion control algorithm for improving RTT fairness,” in Proc. Int. Conf. Ubiquitous Future Netw. (ICUFN), Jul. 2019.

[16] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, ”Reproducible measurements of TCP BBR congestion control,” Comput. Commun., vol. 144, pp. 31–43, Aug. 2019.

[17] S. Hemminger, F. Ludovici, and H. Paal, NetEm—Network Emulator. Accessed: Sep. 9, 2019. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/man8/ncem.8.html

[18] L. Neukirchen. Fq—Job Queue Log Viewer. Accessed: Sep. 9, 2019. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/fq.1.html

[19] J. Dugan. Iperf3—Perform Network Throughput Tests. Accessed: Sep. 9, 2019. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/iperf3.1.html

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

YOU-ZE CHO (Senior Member, IEEE) received the B.S. degree in electronic 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 Information Technology Engineering. From August 1992 to January 1994, he was a Visiting Researcher with the University of Toronto, Canada. From March 2002 to February 2003, he was a Guest Researcher with the National Institute of Standards and Technology, USA. In 2017, he served as the President of the Korean Institute of Communications and Information Sciences (KICS). His research interests include mobility management and traffic engineering for wireless and mobile networks, and the transport protocols for future Internet.