BBR-ACD: BBR with Advanced Congestion Detection

Imtiaz Mahmud, Geon-Hwan Kim, Tabassum Lubna and You-Ze Cho *

School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea;
imtiaz.tee@gmail.com (I.M.); kgh76@ee.knu.ac.kr (G.-H.K.); juthy.cse@gmail.com (T.L.)
* Correspondence: yzcho@ee.knu.ac.kr

Abstract: With the aim of improved throughput with reduced delay, Google proposed the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm in 2016. Contrasting with the traditional loss-based congestion control algorithms, it operates without bottleneck queue formation and packet losses. However, we find unexpected behaviour in BBR during testbed experiments and network simulator 3 (NS-3) simulations. We observe huge packet losses, retransmissions, and large queue formation in the bottleneck in a congested network scenario. We believe this is because of BBR's nature of sending extra data during the bandwidth probing without considering the network conditions, and the lack of a proper recovery mechanism. In a congested network, the sent extra data creates a large queue in the bottleneck, which is sustained due to insufficient drain time. BBR lacks a proper mechanism to detect such large bottleneck queues, cannot comply with the critical congestion situation properly, and results in excessive retransmission problems. Based on these observations, we propose a derivative of BBR, called "BBR with advanced congestion detection (BBR-ACD)", that reduces the excessive retransmissions without losing the merits. We propose a novel method to determine an actual congestion situation by considering the packet loss and delay-gradient of round-trip time, and implement a proper recovery mechanism to handle such a congestion situation. Through extensive test and NS-3 simulations, we confirmed that the proposed BBR-ACD could reduce the retransmissions by about 50% while improving the total goodput of the network.

Keywords: BBR; congestion control; actual congestion detection; retransmissions; TCP

---

1. Introduction

Despite rigorous research on the packet loss problem during data transmission [1,2], a significant amount of packet losses are observed, especially in the wireless network. Packet losses can occur due to both network congestion and network failures. While bottleneck buffer overflow is the primary cause of network congestion, network failures can occur for multiple reasons, such as link disruptions, bit errors, routing failures, etc. [3,4]. To handle packet loss issues, the end hosts have to actively control the flow of data and retransmit the lost packets. In the 1980s, the Transmission Control Protocol (TCP) introduced congestion control by interpreting packet loss as an indication of congestion [5]. TCP maintains a congestion window (CWND) at the sender which determines the allowed amount of data to be sent. If there is no packet loss event, the CWND increases, and whenever there is a packet loss event, it is halved to slow down the flow of data. Even though packets can be lost for network failures too, the current TCP congestion controls are mostly loss-based, interpreting each loss as an indication of congestion [6–10]. In the case of the large bottleneck buffer, these loss based congestion controls arise bufferbloat by keeping the buffer full, whereas for a small bottleneck buffer, they cause low throughput due to false recognition of congestion.

Google proposed a new congestion based congestion control algorithm named bottleneck bandwidth and round-trip time (BBR) with a view to avoid congestion [11]. In its essence, it prevents

persistent queue formation in the bottleneck by controlling the sending rate by sequentially measuring the minimum round-trip time (minRTT), bottleneck bandwidth (BtlBW), and delivery rate. As a result, BBR keeps the RTT to a minimum level while utilizing the full bandwidth. This allows sending more data in a short time, thus ensuring better network utilization. According to Google [11], BBR has improved B4's [12] bandwidth by 133 times, reduced YouTube's median round-trip time (RTT) by more than 80%, and converges to the Kleinrock's [13] optimal operating point for the first time. Therefore, BBR is expected to provide high goodput with fewer retransmissions.

Captivated by its promising performance improvements, several recent works have investigated BBR's performance in different scenarios [14–17]. We also tested BBR in our testbed environment and found unexpected but significant performance degradation in a congested network scenario with moderate bottleneck buffer size. To investigate in detail, we generated a fairly congested scenario in network simulator 3 (NS-3) [18] and tested a modified version of Jain et al.'s BBR implementation [19] in compliance with BBR v1.0 [20]. We found that a large persistent queue is formed in the bottleneck during bandwidth probing due to BBR's data overshooting nature irrespective of the network condition. Because of insufficient drain time and absence of a proper recovery mechanism, this queue remains and causes huge packet losses and retransmissions. Surprised by the results, we investigated further and tried to answer the following three questions in this paper:

(1) How extreme is the performance degradation? To investigate the performance degradation in BBR, we performed the experiments in NS-3 simulation environment with seven sources, seven destinations, and two routers. The BtlBW was set as 1 Mbps between the two routers. The experiments reveal that enormous packets are lost which tends to increase with the increase in simulation time. In a 300 s simulation time, about 42% of the packets were lost. While the average throughput was 0.87 Mbps, the average goodput (the received number of original/not-duplicate packets per second) was only 0.5 Mbps. In support of this finding, we found that Google has recently admitted the issue in IETF'98 [21] that, when BBR replaced CUBIC in YouTube, the global average retransmission rate doubled, from approximately 1% to around 2%. Several recent studies have also reported similar phenomenon [14,17,22].

(2) What is the reason behind such performance degradation? By comprehensive analysis, we find that the poured extra data during the bandwidth probing generates a persistent queue in the bottleneck and that the drain time is insufficient to release that queue. As a result, bufferbloat occurs, which gradually leads to packet losses. TCP informs BBR about the packet losses or bufferbloat events by the three-duplicate acknowledgments (3-dupACKs). However, BBR does not back-off properly and continues to send data at almost the same rate, the queue sustains and results in enormous packet losses with a significant increase in latency, which ultimately leads to a timeout event. In such a timeout event, BBR resets its CWND to one MSS (maximum segment size) and starts the recovery process. As a result, it cannot achieve the better goodput that it would have achieved with a proper recovery scheme.

(3) How to solve the issues? In-depth analysis shows that whenever a persistent queue is formed, first, bufferbloat occurs, then packets are lost. Inclusively, these two together can be regarded as a clear indication of actual congestion. In addition, the relationship between the RTT and delivery rate in respect to the amount of inflight data (Figure 1) shows that, when the buffer becomes full and operating point B of the loss-based congestion control is crossed, the RTT and delivery rate remains identical, i.e., does not change anymore; at this point the packets are simply discarded. Although RTT remains identical before operating point A, BBR always operates beyond point A as reported by [14]. Therefore, packet loss events during RTT remaining identical can be considered as a clear indication of actual congestion too. Thus, rather than being too conservative to reduce the CWND in loss events, if we can implement a method that can recognize the actual congestion events by considering these two factors and take proper actions, we might be able to mitigate the issues in BBR. Moreover, Google is actively developing BBR v2.0, where the early presentations suggest that they are also considering responding to loss events to reduce retransmissions [23]. However, responding to every loss event

may degrade the performance when competing BBR flows are present, especially for large bottleneck buffers as can be observed in the performance of current BBR v2 [24].

Therefore, we propose BBR with advanced congestion detection (BBR-ACD), a simple but effective solution to mitigate BBR's excessive retransmission problem without losing any benefits. BBR-ACD detects actual congestion events by considering the delay-gradient of RTT, and packet losses; and slows down the flow of data in the case of actual congestion events by halving the CWND. This allows the bottleneck to get rid of the excess queue, reduces increased latency due to bufferbloat, and ensures high link utilization.

We implemented BBR-ACD into BBR v1.0's implementation in Linux [20] and extensively experimented on the testbed. The evaluation results show that BBR-ACD reduces the retransmissions by about 50%. Moreover, during simulation evaluation, BBR-ACD loses only about 3.3% of the actual data packets, which is about 92% reduction in packet losses in comparison with BBR. Moreover, BBR-ACD does not reduce the throughput, rather it enhanced it to 0.89 Mbps while increasing the goodput to 0.86 Mbps because of the reduced packet losses.

The rest of the paper is organized as follows: Section 2 gives an overview of BBR, Section 3 provides details on the extents and causes of the excessive retransmission problem in BBR by both the testbed and simulation experiments. Section 4 specifies BBR-ACD in details. Section 5 gives a detailed performance analysis of BBR-ACD by the same testbed and simulation experiments. Finally, Section 6 concludes the paper.
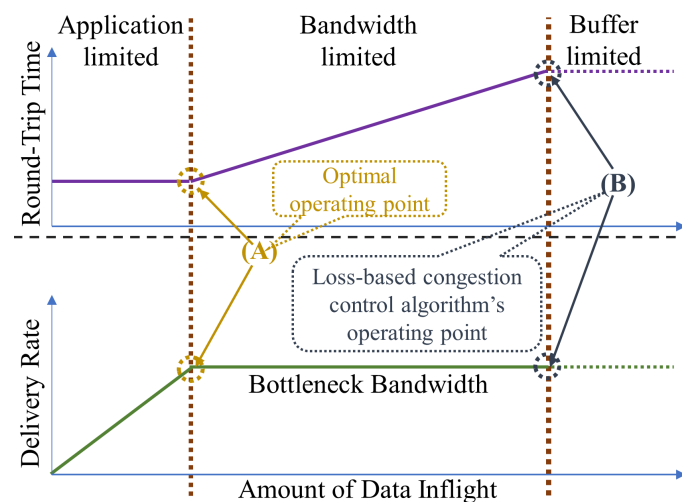

Figure 1. Relationship of delivery rate and round-trip time with the amount of inflight data, and the operating points of the congestion control algorithms according to [14].

2. BBR Overview

In this section, we give a brief description of BBR, the motivation behind it, how it works, and what are its advantages. For further details, we would like to refer the interested readers to Cardwell et al. [11].

2.1. Motivation behind BBR: The Optimal Operating Point

A bottleneck is the network cluster which has the lowest available bandwidth on a path. In every network, there is a bottleneck which determines the throughput of a network. This is also the place where the persistent queue builds up [16]. Kleinrock et al. [13] have proved that, if the data inflight—the unacknowledged data—matches to exactly one bandwidth-delay product (BDP), i.e., `inflight = BtlBW × RTprop`, it will maximize throughput, minimize RTT, and act as the optimal operating point (point A in Figure 1) for both individual connection and the entire network. Here, BtlBW stands for the bottleneck bandwidth, and RTprop is the round-trip propagation time which is almost equivalent to minRTT. In simple words, if we can achieve a delivery rate equal to the BtlBW while maintaining the RTT identical to the minRTT, we would be able to fully utilize the network and

get maximum throughput off it. Motivated by this idea, BBR claims to operate at the optimal operating point and tries to deliver maximum throughput.

2.2. BBR Operation

To operate at the optimal operating point, BBR requires to measure the BtlBW and minRTT continuously. In order to measure the BtlBW, the pipe needs to be overfilled, whereas all the queues need to be drained empty to measure minRTT. Therefore, BBR separately measures the BtlBW and minRTT in a sequence. Based on the measured values of BtlBW and minRTT, BBR determines its sending rate by two control parameters: Pacing rate and CWND. Pacing rate determines the rate at which a sender sends data. CWND puts an upper limit on the data inflight, i.e., the maximum allowed amount of inflight unacknowledged data. BBR operates in four states:

(1) Startup: In the startup phase, BBR starts with an exponential search for BtlBW. The sending rate increases by a rate of `2/ln2` for the reception of every acknowledgment (ACK) until the delivery rate stops growing. In the meantime, BBR maintains a maximum CWND/inflight-cap of $3 \times$ BDP.

(2) Drain: During the startup bandwidth probing, a queue forms in the bottleneck due to extra data pouring. To free up the excess queue, BBR slows down its sending rate by an inverse function of startup rate, `ln2/2`, until the delivery rate matches BDP.

After draining, BBR steps into a steady-state phase where it cruises while sequentially measuring the BtlBW and minRTT by the following two states.

(3) ProbeBW: BBR spends most of its time in this state. It consists of eight cycles; each cycle lasts for one RTT. In one cycle, the pacing gain is set to 1.25, thus allowing BBR to pour extra data to overfill the pipe in order to probe for extra bandwidth. In the very next cycle, the pacing gain is reduced to 0.75 so that BBR can drain out the extra data that it poured in the previous cycle. For the next six cycles, BBR maintains a pacing gain of 1 so that it can cruise while maintaining the measured bandwidth.

(4) ProbeRTT: While in the ProbeBW state, if BBR cannot find any minRTT value less than the existing minRTT for a 10 seconds timespan, BBR enters the ProbeRTT state. Here, it reduces its CWND to only four packets so that all the queues become free and BBR can re-probe the minRTT.

2.3. Benefits of BBR

BBR has gained widespread attention over its other competitors because of the following benefits:

1. By operating in the optimal operating point, BBR claims to achieve high throughput while maintaining RTT close to minRTT. As a result, better network utilization is assured.
2. BBR can be implemented quite easily by executing it only on the sender side. It does not require any modification in the middleboxes or at the receiver end.
3. In the multitude of different congestion control protocols, after measuring the capacity of the network, BBR claims a fair share and actively defends that share from being crowded out. As a result, albeit being a congestion avoiding protocol that tries to keep the queue free, BBR sustains in the battle with loss-based protocols which try to keep the queues fully occupied.

3. Excessive Retransmission Problem in BBR

In this section, we discuss in detail the retransmission issue in BBR with experimental results and explain the reasons behind it. We start by experimenting with BBR v1.0 [20] in our testbed environment and give a simple view of the retransmission problem observed in it. Then, in order to investigate the extent of the problem and the cause, we experiment in a simulation environment.

3.1. Testbed Environment

To test the performance of BBR, we set up a testbed scenario with two senders, one destination, and two routers as shown in Figure 2. Both the senders and destination used Linux Kernel v4.17 [25] and BBR v1.0 [20]. A Linux-based computer acted as the intermediate router. Each sender had five

flows of data connection (F1 to F5), where Sender #1 (S1) started the first flow at zero seconds, and sequentially started the rest of four flows each after one second intervals. Similarly, Sender #2 (S2) started its first flow at fifty-five seconds and the rest of the flows after one second intervals. The inset of Figure 2 illustrates the start of different flows for the two senders. To enable pacing, we enabled "fq" [26] as the queueing discipline. RTT and the bandwidth of the bottleneck were configured using "NetEm" [27] and "ethtool" [28], respectively. The sender transmitted data to the receiver using "iperf3" [29]. "Wireshark" [30] was used to measure the input/output packets per sender per second, the number of lost packets per sender per second, and the total number of transmitted packets by both the senders during the entire testbed experiment. "Netstat" [31] was used to measure the total number of retransmitted packets per sender during that experiment time. As discussed in Section 1, in a real network, there are multiple reasons for network failures, such as link disruptions due to mobility and interference in the wireless network, bit errors, routing failure, etc. These can occur in real-time without following any specific pattern. Therefore, to emulate packet drops due to network failures in the testbed, we adopted the random loss scheme available in the "NetEm" [27]. Now to investigate how BBR behaves for packet losses due to actual congestion and network failures, we need to have situations where either packet losses because of actual congestion or network failures are present. Therefore, a highly congested scenario with no network failures and a non-congested scenario with network failures are required to serve the two cases, respectively. In addition, it is preferable to observe BBR's behavior for a mix-up of the two types of packet losses in both scenarios. Therefore, we generate four scenarios by varying the queue size and loss ratio in Router #1 as follows:
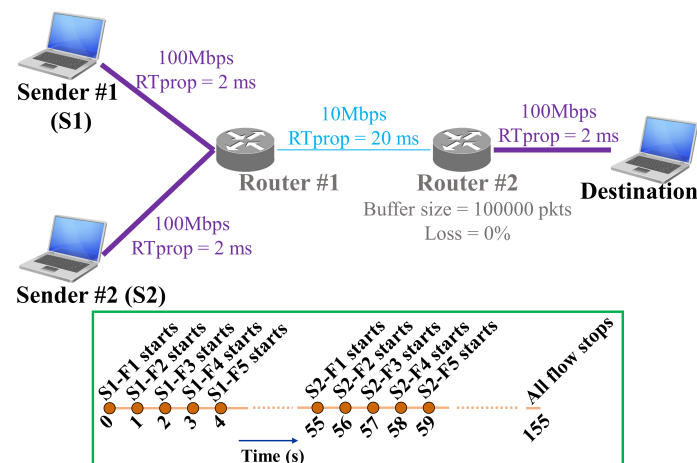

Figure 2. The setup and testbed scenario.

Testbed Scenario #1—large bottleneck buffer without packet loss: Here, we set the buffer-size of Router #1 to 100,000 packets and the loss ratio to zero. As the buffer is too large, there is no chance for packet losses due to buffer overflow, i.e., no loss of packets because of network congestion. Moreover, no network failure is present due to the zero loss ratio. This scenario allows us to observe the performance of BBR in a comparatively unbounded network environment. Moreover, it enables to compare the performance of BBR in congested and uncongested scenarios.

Testbed Scenario #2—large bottleneck buffer with packet loss: Again, we set the buffer-size of Router #1 to 100,000 packets but with a loss ratio of 0.1%, i.e., one packet is lost during the transmission of 1000 packets. This allows us to observe how BBR performs when there is packet loss only due to network failures.

Testbed Scenario #3—small bottleneck buffer without packet loss: We set the buffer-size of Router #1 to 100 packets and the loss ratio to zero. Therefore, there is no loss of packets due to network failures. Thus, in any case, if a packet is lost, we can assume that the packet is lost only because of buffer overflow, i.e., actual network congestion.

Testbed Scenario #4—small bottleneck buffer with packet loss: Again, we set the buffer-size of Router #1 to 100 packets but with a loss ratio of 0.1%. Therefore, besides the packet losses due to actual congestions, packet losses because of network failures are present in this scenario.

3.2. Testbed Results

In this section, we compare the testbed experiment results for BBR in terms of input/output packets per sender per second and the number of lost packets per sender per second as can be observed in Figure 3.
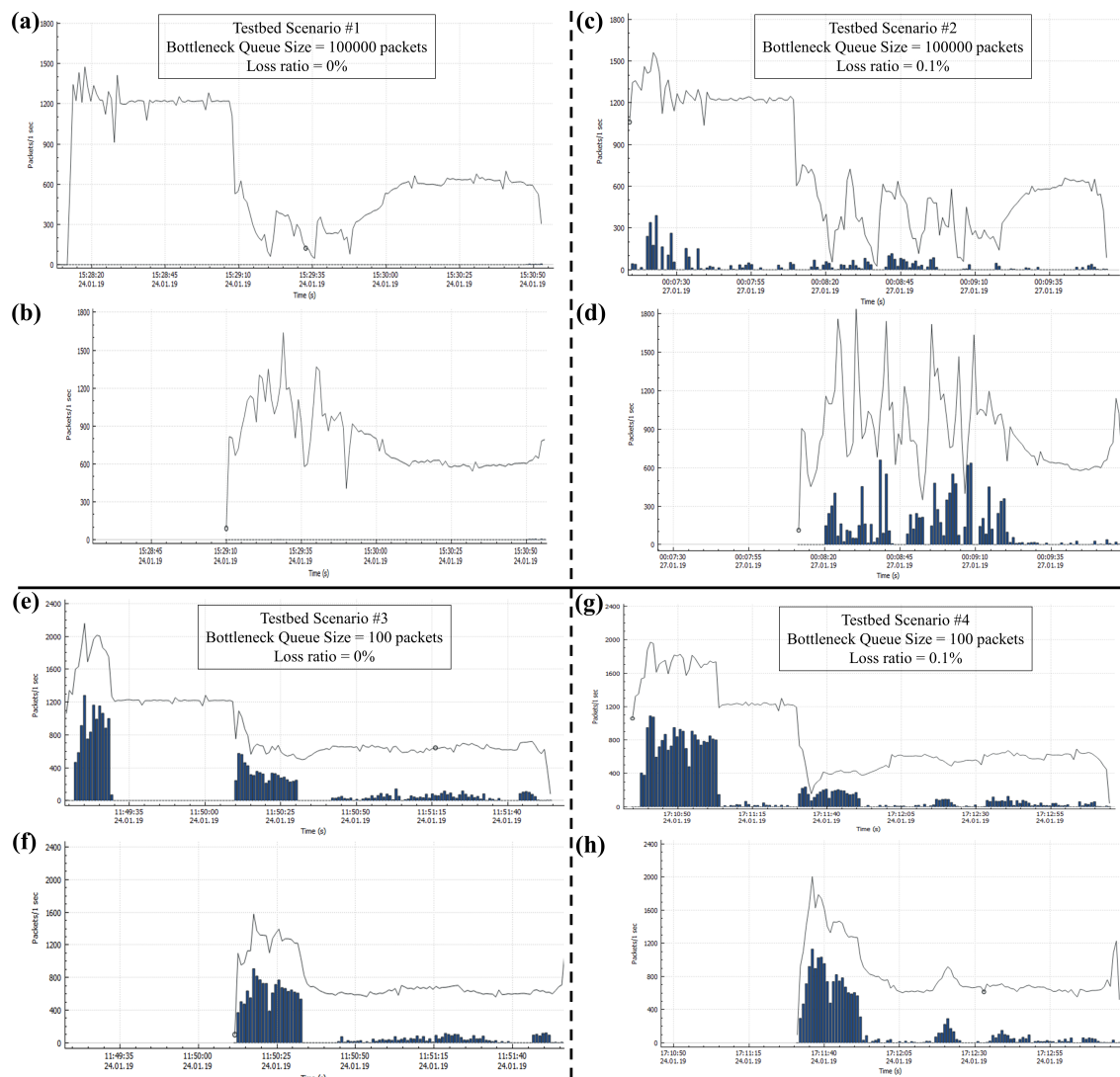

Figure 3. Comparison of input/output packets per second per sender for different testbed scenarios — (**a,b**) for Testbed Scenario #1, (**c,d**) for Testbed Scenario #2, (**e,f**) for Testbed Scenario #3, and (**g,h**) for Testbed Scenario #4. The bars indicate the lost number of packets.

For Testbed Scenario #1, Figure 3a,b shows the performance of BBR. Because the buffer size is sufficiently large, no packet loss is observed. The flows of Sender #1 start normally but start suffering after the start of Sender #2's flows. Then, after about 50 s, all flows cruise normally with almost equal shares. This is the regular behavior of BBR. As Sender #2's flows start, they try to measure the BtlBW by overfilling the pipe. The extra data creates bufferbloat and causes the existing flows to slow down. Gradually, they merge to an almost equal share state. From now on, we refer to this state as "equilibrium-state" and the time required to reach this state as the "equilibrium-time", unless stated otherwise.

Figure 3c,d shows the performance of BBR for Testbed Scenario #2, where we can clearly observe the effect of packet losses due to network failures. BBR responds to packet loss events by slightly reducing the CWND as `cwnd = inflight + acked` [20]; the input/output rate drops for such events. While there is no congestion and the packet losses are only due to network failures, by responding to these events BBR only degrades its overall performance. The drop in the input/output rate is simply of no advantage as it cannot help to get rid of the network failures. Rather, if BBR could maintain its input/output data rate by maintaining normal CWND growth, it would have resulted in much higher throughput. Moreover, it takes more than 70 s as equilibrium-time. This is because of the unstable network condition that falsely triggered BBR's packet conservation mechanism and hindered it from quickly reaching an equilibrium-state.

Figure 3e,f demonstrates the outcome of BBR for Testbed Scenario #3. At the startup phase of BBR, a large number of packets are lost for both the senders. As there is no packet loss due to network failures, it is evident that the packet losses are the outcome of actual network congestion. During the startup phase, BBR sends extra data to estimate the BtlBW of the network. This creates a large queue in the bottleneck buffer resulting in congestion. Moreover, when Sender #2 starts its flows, Sender #1's CWND drops due to packet losses caused by the buffer overflow during the startup phase of Sender #2. In this case, it takes almost 30 s equilibrium-time. Moreover, a continuous packet loss is observed at an average rate throughout the rest of the experiment time. The root cause of such behavior needs to be investigated further. Therefore, we performed simulation experiments which will be discussed in the subsequent sections.

Figure 3g,h shows the experiment results of Testbed Scenario #4. In this scenario, the packet losses due to both the actual congestions and network failures are present, an increase in packet loss is observed in comparison with Testbed Scenario #3. Moreover, equilibrium-time also increases to almost 40 s. The reasons are similar to the reasons explained for Testbed Scenario #3. Moreover, because BBR is slowly responding to packet losses for both actual congestion and network failures, the performance degrades further.

3.3. Simulation Environment

As stated earlier, we performed the simulation analysis of BBR in a simulation environment of NS-3 v3.28 [18]. Recently, Jain et al. [19] implemented BBR in NS-3.28 where they followed the near actual implementation of the Linux implementation of BBR v1.0 [20]. However, we found a small mismatch in the recovery mechanism which we fixed and uploaded the modified code in the GitHub repository available at [32]. For the simulation experiment, we considered a simulation scenario shown in Figure 4. Here, seven sources send data to seven different receivers. There are two intermediate routers (Routers I and II). The sources are connected to Router I by ethernet connections each with 40 Mbps bandwidth and 8 ms delay. The receivers are connected to Router II by ethernet connections each with 40 Mbps bandwidth and 2 ms delay. Routers I and II are connected by an ethernet connection that has 1 Mbps bandwidth and 50 ms delay and that acts as the bottleneck link. Therefore, Router I acts as the bottleneck router in the simulation scenario. Moreover, we set the size of the bottleneck buffer as one BDP. Each source starts a constant bit rate (CBR) data flow at the start of the simulation experiment, i.e., a total of seven flows start together and continue till the end. According to Hock et al. [14], BBR overestimates the BtlBW when the flows do not perform the probeBW at the same time. Therefore, much worse performance and severe congestions are expected for a situation when the flows start at different times. Because our main focus is to observe the impact of a congested network situation, we let the BBR perform in its optimum state and choose to start all the flows at the same time. Table 1 summarizes the key simulation parameters.
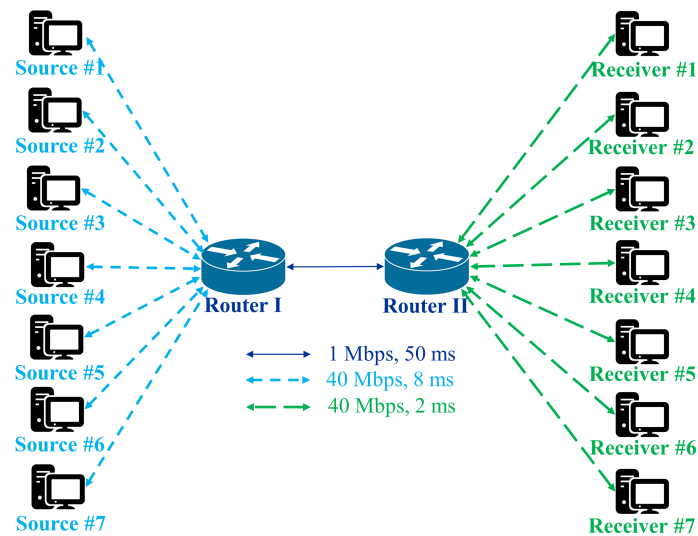
Figure 4. The simulation setup.

Table 1. Simulation parameters.

Parameter	Considerations
Topology	Point-to-point-dumbell
Number of left leafs	7
Number of right leafs	7
Access bandwidth left	40 Mbps
Access delay left	8 ms
Access bandwidth right	40 Mbps
Access delay right	2 ms
Point-to-point bandwidth	1 Mbps
Point-to-point delay	1 ms
Queue length	1 BDP
Application type	Bulk send application

3.4. Evaluation by Simulation

In order to investigate the behavior and performance of BBR in the simulation experiments, we measured the CWND of different flows, occupied buffer size at the bottleneck router, RTT of different flows, and total throughput and total goodput of all the flows. We performed 30 separate simulation experiments in the same scenario with a different set of random variables. All the experiments showed a similar trend of results. However, in order to investigate and understand the results properly, we need to arrange the results in the same timeline so that we can perceive the exact sequence of occurrences of the events. Showing the average results of the experiments would hinder a proper understanding of the results. Therefore, in this work, we show the result of an experiment which was randomly chosen among the 30 simulation experiments. Moreover, interested readers can download the simulation code from the GitHub repository [32] and experiment further.

With the aim of comparing the equal distribution of bandwidth between different flows, we measured the CWND of different flows as shown in Figure 5a. In order to investigate what is going on in the bottleneck buffer, for each addition or deletion of a packet from the bottleneck buffer, we measured how much of the bottleneck buffer is occupied by the currently existing packets in the bottleneck buffer. Figure 5b shows the size of the occupied bottleneck buffer in comparison with the total usable bottleneck buffer size during the simulation experiment. To verify the claim that BBR operates while maintaining a low RTT, we measured the RTT of different flows as can be observed in Figure 5c. Finally, to evaluate the proper resource utilization of the entire network, we measured the total throughput and goodput of the network as shown in Figure 5d. Here, the throughput for

each flow was measured as the size of the received data packets per second. Hence, total throughput represents the size of the total received data packets per second by all the receivers. The goodput for each flow was measured as the size of actually received data packets—disregarding the duplicate data packets—per second per receiver. So that the total goodput represents the total received actual data packets per second by all the receivers.
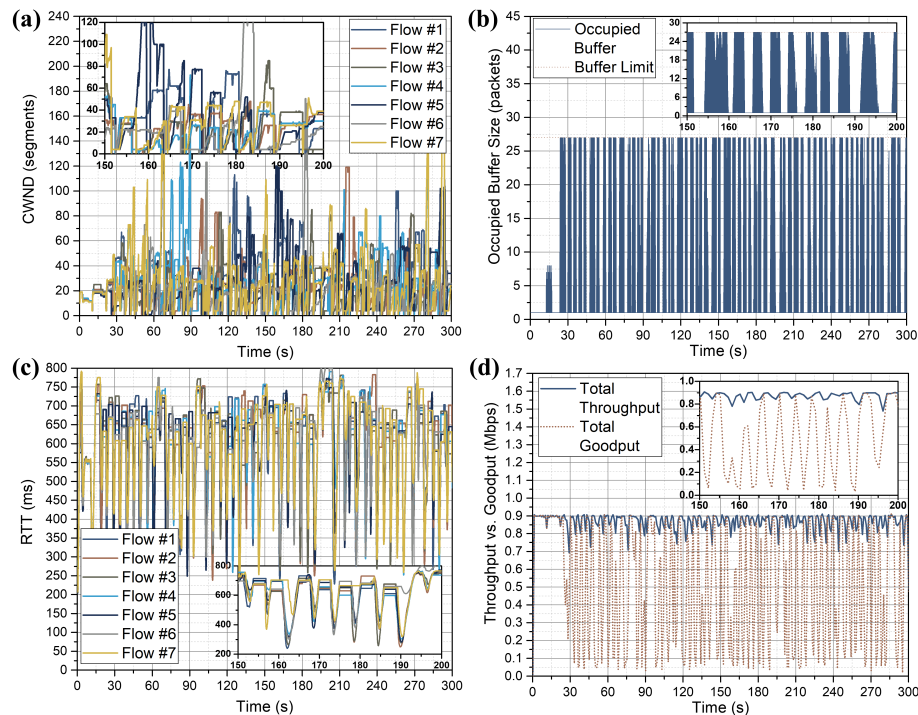

Figure 5. In the simulation experiment, the performance of the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm in terms of (**a**) congestion window (CWND), (**b**) occupied buffer size, (**c**) round trip time (RTT), and (**d**) throughput vs. goodput.

We can observe from Figure 5a that at the startup phase of BBR, it identifies BtlBW and enters into the drain phase. However, during this startup phase, the CWND does not rise as rapidly as expected and the bottleneck queue (Figure 5b) never becomes full. By close observation of the RTT curve (Figure 5c), we can easily perceive that the RTT increases rapidly to 800 ms at the very beginning during the startup phase. We believe that, because there was no packet at the bottleneck queue at the initial state, when all the flows measured the minRTT, they got a very small value of minRTT, around 200 ms. However, because all the flows started at the same time, the network had to handle a lot of packets at once. This caused the RTT to increase rapidly to about four times the minRTT. This hinders the growth in delivery rate. Because BBR measures that the delivery rate is not increasing, it exits the startup phase. Therefore, we believe that the exiting startup phase in this simulation scenario is mainly caused by the rapid increase in RTT, rather than the actual congestion.

Moreover, from Figure 5a, during the steady-state of BBR, the CWND is not equally distributed between different flows. As a consequence, BBR fails to equally share the available bandwidth among several BBR flows. If we observe the insets closely, whenever a BBR flow enters into the probeBW state (for example, Flow #7 at around 170 s), a bottleneck queue forms (inset of Figure 5b), ultimately the bottleneck buffer becomes full. The drain time is simply insufficient to release this queue. As a result, this queue is sustained, and the RTT increases as can be observed in the inset of Figure 5c. Moreover, the RTT remains the same for several seconds, which implies that the operating point B (Figure 1) is crossed and the packets are dropped. This phenomenon is also supported by Figure 5d, where the goodput starts dropping at 170 s (inset of Figure 5d), and drops until it becomes almost zero. At this moment, retransmission timeout (RTO) occurs, and the CWND resets to one MSS and continues with

the recovery process. Therefore, although a high throughput is achieved by BBR (Figure 5d), the actual goodput is significantly low. To be specific, during the 300 s simulation time, all the flows sent a total of 68,420 packets, among which only 39,645 packets reached the destination, other 28,775 packets were lost, which is almost 42% loss of packets. By default, TCP informs BBR about the packet losses via 3-dupACK, but BBR does not respond properly to such a severe congestion situation. This clearly indicates that BBR could not handle a congested network scenario properly.

Based on the observation of BBR's performance in the congested network scenario for both the testbed and simulation experiments, we can summarize the following key findings:

1. At the startup phase, both congestion and network failures can impact the performance of BBR. In particular, the congestion can cause a significant amount of packet losses.
2. BBR significantly takes longer equilibrium-time after the start of a new flow.
3. During the probeBW phase, BBR inserts a significant number of extra packets into the network and creates a persistent queue. This queue sustains and causes packet losses.
4. Although there is a drain time, it is not enough for the network to get rid of the excess queue in a congested network scenario.
5. TCP notifies BBR about the packet losses, but BBR does not comply properly, leading to RTO and significantly low performance in terms of goodput.

4. BBR-ACD—A Solution to Reduce the Excessive Retransmissions

Based on the findings of the previous section, it becomes inevitable that BBR needs a proper scheme to get rid of the excess queue. There can be two solutions: First, we can increase the drain time so that BBR gets enough time to drain the excess queue; second, we can try to recognize excess queue formation and take proper actions to eliminate it. For the first solution, increasing the drain time might solve the excess queue problem in the case of a congested network scenario, but would result in significant performance degradation in a large bottleneck buffer situation because this will unnecessarily slow down the flow of data for a long time. Therefore, the second option, i.e., to properly detect the excess queue formation and take a proper recovery mechanism so that it can release the excess queue and solve the congested network situation is a better option. However, for that purpose, BBR needs to identify the actual congestion events, i.e., differentiate between packet losses due to network congestion and network failures. Considering these aspects, we propose a derivative of BBR—BBR-ACD—that successfully detects the actual congestion events and take proper actions without losing any merit of BBR. We design BBR-ACD in a very simple way so that it can be easily integrated with the current BBR implementation. In BBR-ACD, firstly, upon the reception of each 3-dupACK, it checks whether the packet loss is due to actual congestion or network failures. Secondly, in the case of an actual congestion event, BBR-ACD halves the CWND so that the network can get rid of the excess queue. By this two-step process, we not only solve the high retransmission problem in BBR but also help BBR to reach the equilibrium-state in a comparatively shorter equilibrium-time. Moreover, we ensure better throughput and goodput by BBR-ACD. The details of BBR-ACD are discussed in the following sections.

4.1. Detection of Congestion

Up until now, we have discussed the consequences of packet losses due to actual congestion and network failures. Now, how to differentiate between packet losses due to actual congestions and network failures? To answer that question, first, we need to identify the preconditions for a packet loss event to be considered as the result of actual congestion. To do so, we have identified two preconditions based on delay-gradient of RTT that surely differentiate between the two.

First, as we have observed in Section 1, when the operating point B (Figure 1) is crossed, the RTT does not change anymore, it remains almost identical. When this phenomenon happens, the buffer becomes full and the network starts discarding the extra packets; therefore, the packets are lost due to network congestion. We use the term "almost identical" because, in practical networks, there always

remains some network glitches that cause the RTT to always change a little. We consider that this network glitch can cause a change of α ms to the RTT. For three consecutive ACK/dupACK events, if the RTT remains in the range RTT $\pm$ α, we consider it as an almost identical RTT and name this state as an identical-RTT state. This identical-RTT state indicates that if a packet is lost during this state, it is lost due to actual congestion. We consider specifically three consecutive ACK/dupACK events because the first two ACK/dupACK with similar RTT create an event of identical-RTT for the first time, and the last two ACK/dupACK with similar RTT create the second event of identical-RTT occurrences and confirm the first event.

Second, Hock et al. [14] showed via testbed experiments that for a small buffer or congested network scenario, in fact, BBR does not operate at the optimal operating point as shown in Figure 1. Rather, multiple BBR flows typically create a queueing delay of about 1.5 times the RTT. We also found the same trend of results and observed that the RTT increases even more, typically beyond 1.5 times. Therefore, we consider that when the current RTT becomes two times greater than the minRTT, if a packet loss occurs at that time, the packet loss is the result of network congestion rather than network failures.

To implement it into Linux Kernel, we introduce a function named bbr_prob_congestion() that is called upon the reception of each ACK/dupACK and returns true if any of the two preconditions for network congestion is true, false otherwise. The Linux Kernel code of the bbr_probe_congestion() function can be found in Figure 6.

```c
static bool bbr_probe_congestion
(struct sock *sk, const struct rate_sample *rs) {
    struct bbr *bbr = inet_csk_ca(sk);
    if(rs->rtt_us > bbr->min_rtt_us) {
        if(rs->rtt_us >= bbr->saved_last_rtt-1000
           && rs->rtt_us <= bbr->saved_last_rtt+1000){
            bbr->count_if_steady_rtt++;
        }
        else {
            bbr->count_if_steady_rtt = 0;
        }
    }
    else {
        bbr->count_if_steady_rtt = 0;
    }
    bbr->saved_last_rtt = rs->rtt_us;
    if (bbr->count_if_steady_rtt >= 2
        || rs->rtt_us > 2*bbr->min_rtt_us)
        return true;
    return false;
}
```

Figure 6. Linux Kernel code of the proposed bbr_probe_congestion() function. Here, α = 1000 μs.

4.2. Action Taken for Congestion

Till now, we have defined a method to evaluate a packet loss event whether it is an outcome of network congestion or network failures. Now, we need to define a mechanism to take proper actions in the case of packet losses due to actual congestion events to get rid of the excess queue in the bottleneck. As we have mentioned earlier, we propose to halve the CWND in such events complying with the behavior of traditional TCP congestion control algorithms [5,6]. According to Ware et al., when BBR flows compete with other flows, BBR becomes window-limited and ACK-clocked, and sends packets at a rate completely determined by its inflight-cap, which is the maximum allowed CWND [22]. Therefore, instead of slowing down the sending rate, we halve the CWND so that the bottleneck gets enough time for releasing the excess queue.

In the Linux Kernel implementation of BBR v1.0 [20], the bbr_set_cwnd_to_recover_or_restore() function is called upon the reception of each ACK/dupACK to check and take necessary actions in the case of a packet loss event. Therefore, we implemented the CWND halving mechanism inside this function. The Linux Kernel code of the modified bbr_set_cwnd_to_recover_or_restore() function can be found in Figure 7. Here, whenever a packet loss event occurs in the existence of any of the actual congestion preconditions, packet_conservation becomes true. For such a case, we update the

inflight-cap to one BDP and set the new CWND as a minimum between halved CWND and inflight-cap. At the end of the recovery process, we set the new CWND as the maximum between the current CWND and the CWND prior to the recovery process.

```c
static bool bbr_set_cwnd_to_recover_or_restore(
    struct sock *sk, const struct rate_sample *rs,
    u32 acked, u32 *new_cwnd, u32 bw) {
    struct tcp_sock *tp = tcp_sk(sk);
    struct bbr *bbr = inet_csk_ca(sk);
    u8 prev_state = bbr->prev_ca_state;
    u8 state = inet_csk(sk)->icsk_ca_state;
    u32 cwnd = tp->snd_cwnd;
    if (rs->losses > 0)
        cwnd = max_t(s32, cwnd - rs->losses, 1);
    if (bbr_probe_congestion (sk, rs)
        && state == TCP_CA_Recovery
            && prev_state != TCP_CA_Recovery) {
        bbr->packet_conservation = 1;
        bbr->next_rtt_delivered = tp->delivered;
    } else if (prev_state >= TCP_CA_Recovery
            && state < TCP_CA_Recovery) {
        cwnd = max(cwnd, bbr->prior_cwnd);
        bbr->packet_conservation = 0;
    }
    bbr->prev_ca_state = state;
    if (bbr->packet_conservation) {
        u32 target_cwnd =
        bbr_target_cwnd(sk, bw, BBR_UNIT);
        cwnd = min(cwnd/2, target_cwnd);
        *new_cwnd = max(cwnd, bbr_cwnd_min_target);
        return true;
    }
    *new_cwnd = cwnd;
    return false;
}
```

Figure 7. Linux Kernel code of the modified bbr_set_cwnd_for_recover_or_restore() function.

Flow chart of Figure 8 summarizes the key concept of actual congestion detection and the CWND halving mechanism of BBR-ACD. The added complexity of BBR-ACD is only O(1). No extra signaling overhead is required to implement BBR-ACD as the RTT information is already available in the TCP/IP stack. The Linux Kernel code of BBR-ACD can be found in [33].
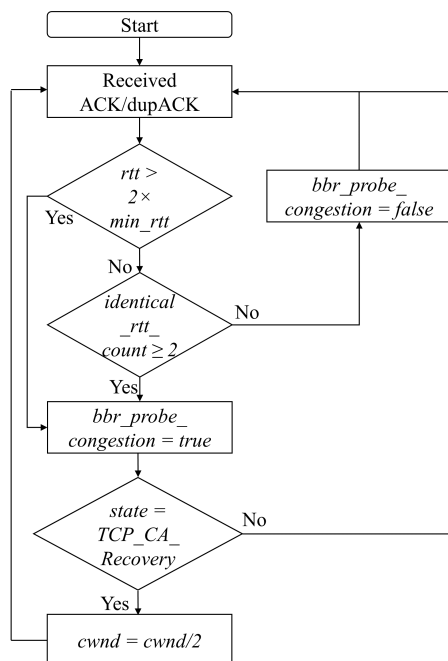

Figure 8. Flow chart of the key advanced congestion detection and CWND halving mechanism of BBR with advanced congestion detection (BBR-ACD).

5. Performance Evaluation

Now to evaluate BBR-ACD, first, we start with the testbed experiments, then we investigate the improvements through simulation experiments. We evaluate the proposed BBR-ACD with the exact same testbed scenario. In the case of simulation experiments, we experimented with the same simulation environment with the exact same set of random variables. We assess BBR-ACD in respect to the same parameters used to assess BBR in the previous section.

5.1. Testbed results

We start by evaluating the testbed experiment results for BBR-ACD in terms of input/output packets per sender per second and the number of lost packets per sender per second which is given in Figure 9.
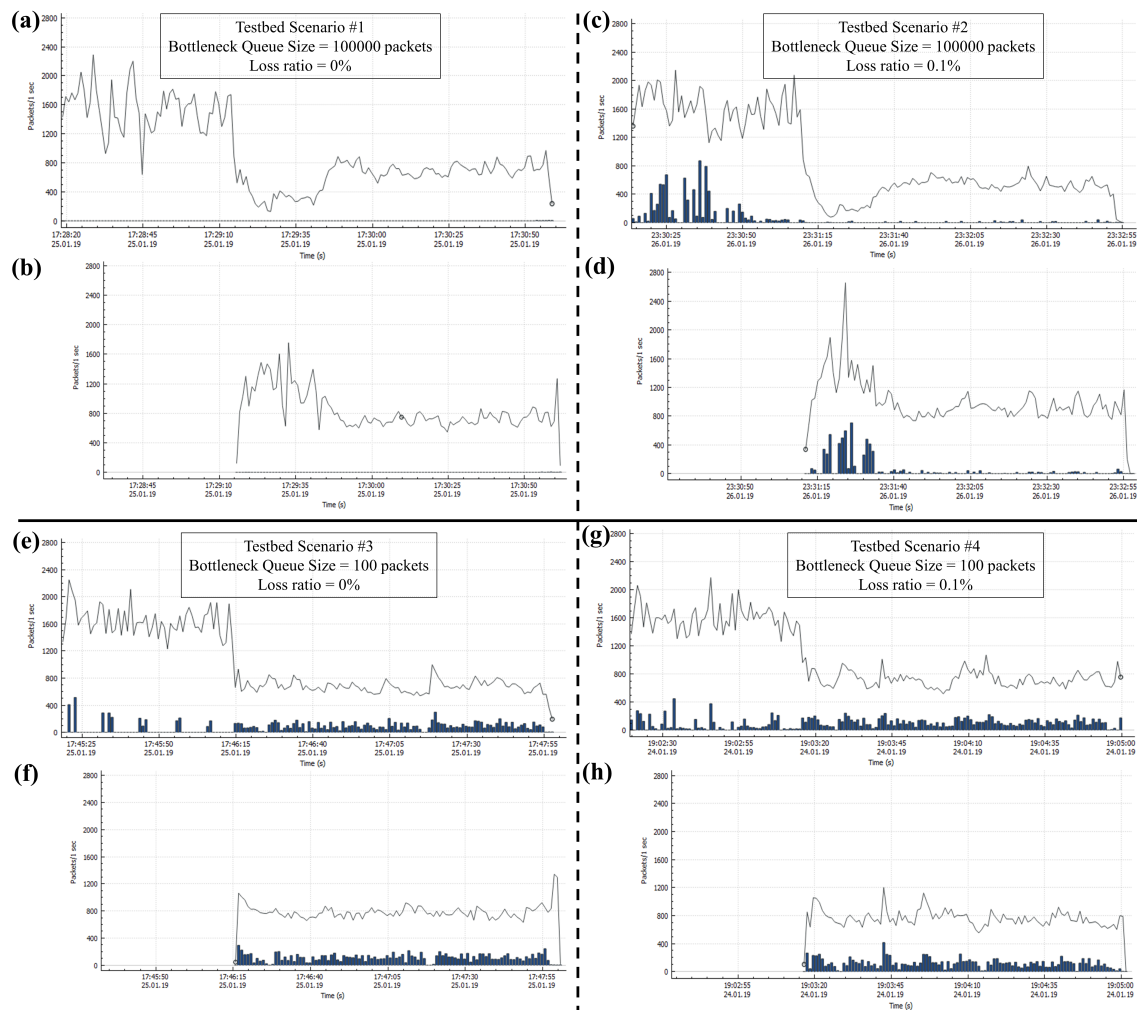

Figure 9. Comparison of input/output packets per second per sender of BBR-ACD for different testbed scenarios — (**a**,**b**) for Testbed Scenario #1, (**c**,**d**) for Testbed Scenario #2, (**e**,**f**) for Testbed Scenario #3, and (**g**,**h**) for Testbed Scenario #4. The bars indicate the lost number of packets.

In the case of Testbed Scenario #1, as shown in Figure 9a,b, because the buffer is sufficiently large, no packet loss is observed similar to BBR. Moreover, BBR-ACD takes about 40 s equilibrium-time, which is almost equal to the normal BBR. We can also perceive that with the implementation of BBR-ACD, the normal performance of BBR is not hampered, rather similar performance is achieved for this scenario.

From Figure 9c,d, we can observe the performance of BBR-ACD for Testbed Scenario #2. Here, as the buffer is sufficiently large, the packets are lost mainly due to network failures. Therefore, BBR-ACD does not respond to every packet loss event. However, the packet loss events cause out-of-order delivery at the receiver, which leads to head-of-line blocking. As a result, a decrease in the input–output curve is observed even in BBR-ACD for a large number of packet losses. Moreover, if we compare the average input/output packets per sender per second, we can perceive that the rate is significantly higher for BBR-ACD in comparison to normal BBR. Moreover, after the start of Sender #2's flow, the flows take a shorter equilibrium-time (30 s) than the BBR (70 s). At the start of each flow, the new flows send extra data that create large queues and increase the RTT. BBR-ACD successfully recognizes it and reacts by halving the CWND whenever packets are lost in such a scenario. As a result, the flows were able to reach the equilibrium-state in a short time. Therefore, BBR-ACD significantly improves the performance of BBR in Testbed Scenario #2.

For Testbed Scenario #3, we can perceive the performance of BBR-ACD from Figure 9e,f. At the beginning of the first flow, the packets are lost because BBR-ACD sends extra data to estimate the BtlBW following BBR. However, BBR-ACD quickly recognizes this phenomenon by observing the increase in RTT and reacts whenever a packet is lost. This very much helps to reach the equilibrium-state at a very short time. In particular, in this scenario, BBR-ACD takes only about 5 s as equilibrium-time. Furthermore, we can observe that the input/output rate moves much more smoothly than normal BBR. In the case of packet losses, unlike normal BBR, an average packet loss is observed almost continuously after the start of Sender #2's flows. This needs further investigation which has been carried out by simulation experiments in the upcoming section.

In the case of the Testbed Scenario #4, a similar trend is observed as observed in the case of Testbed Scenario #3. In comparison with BBR, BBR-ACD produces less packet losses as we can observe from Figure 9g,h. Moreover, the equilibrium-time is shorter, about 10 s. And we can perceive that the input/output rate is higher than the normal BBR, which clearly indicates the efficacy of BBR-ACD over existing BBR.

Finally, Table 2 summarizes the performance difference between BBR and BBR-ACD in terms of the equilibrium-time and the total number of transmitted packets by both the senders during the entire testbed experiment time period as measured by Wireshark, and the total number of retransmitted packets by both senders at the same time period measured by Netstat. Moreover, it includes the total throughput and goodput results during the testbed experiment. In all testbed scenarios except Testbed Scenario #2, BBR-ACD could reduce the number of retransmissions while uprising the number of sent packets. However, in the case of Testbed Scenario #2, an increase in the number of retransmissions is observed. The total throughput and goodput results also showed a similar trend. This is because we designed BBR-ACD in such a way that it would only slow down for the packet losses due to actual network congestion. If the packets are lost due to network failures as was induced in this scenario, BBR-ACD does not slow down, but rather continues ignoring the packet losses. This ensures better utilization of the network resources. Moreover, Testbed Scenario #2 was designed in such a way that one packet will be lost upon transmission of every 1000 packets. Hence, the more the number of transmitted packets, the more the number of lost packets. Therefore, we can observe that, although the number of retransmitted packets increases, the number of total transmitted packets increases more strongly. A similar trend is observed for total throughput and goodput comparison too. In addition, it is certain that because the packets are not lost due to network congestion, reducing the CWND or send rate would not help to mitigate the problem. Rather keeping the same CWND or send rate ensures better network utilization.

Table 2. Performance comparison between BBR and BBR-ACD.

	Parameter	BBR	BBR-ACD
Testbed Scenario #1	Equilibrium-time (seconds)	50	40
	Total transmitted (packets)	199,051	238,971
	Total retransmitted (packets)	1	1
	Total throughput (Mbps)	8.026	9.636
	Total goodput (Mbps)	8.026	9.636
Testbed Scenario #2	Equilibrium-time (seconds)	70	30
	Total transmitted (packets)	210,184	245,978
	Total retransmitted (packets)	636	2852
	Total throughput (Mbps)	8.475	9.918
	Total goodput (Mbps)	8.449	9.803
Testbed Scenario #3	Equilibrium-time (seconds)	30	5
	Total transmitted (packets)	228,826	243,739
	Total retransmitted (packets)	9990	4568
	Total throughput (Mbps)	9.227	9.828
	Total goodput (Mbps)	8.824	9.644
Testbed Scenario #4	Equilibrium-time (seconds)	40	10
	Total transmitted (packets)	227,853	245,893
	Total retransmitted (packets)	9539	5103
	Total throughput (Mbps)	9.188	9.915
	Total goodput (Mbps)	8.803	9.709

5.2. Evaluation by Simulation

In this section, we investigate the performance of BBR-ACD in a congested network scenario in the NS-3 simulation environment. As stated earlier, during the simulation, all the parameters, variables, and the scenario were exactly the same as for the case of evaluating BBR. We also ran 30 separate simulation experiments and found the same trend of results. For the proper comparison with BBR, we present the simulation result of BBR-ACD in Figure 10 which had been carried out in the exact same environment with the same set of random variables that had been used to produce Figure 5 for BBR. We also evaluate the performance of BBR-ACD in terms of the same performance metrics.

In the case of the startup phase of BBR-ACD, we can observe the behavior of CWND from Figure 10a. Similar to BBR, the CWND or the occupied bottleneck buffer does not grow too much as expected. The same as BBR, the RTT grows rapidly to about 600 ms, whereas the measured minRTT was about 125 ms. The reasons explained previously for the same behavior of BBR also applies for BBR-ACD.

However, in the steady-state phase, we can perceive the significant performance improvement by BBR-ACD. As can be observed from Figure 10a, the CWND is distributed almost equally. As a result, a fair share of bandwidth is allocated for each flow. If we observe the inset of Figure 10a,d closely, a queue grows at the bottleneck during the probeBW phase. As stated earlier, by design nature, BBR inserts extra data into the network during the probeBW phase. The buffer becomes fully occupied quickly, the queue is sustained, resulting in bufferbloat, and the network fails to handle this extra burden, causing a huge number of packet losses. However, this design is necessary to correctly measure the BtlBw and ensure fair share among the different flows. Because BBR-ACD also follows the same design principle, it also inserts extra data during the probeBW phase. However, it contains a proper recovery mechanism to handle network congestion as proposed in this work. As the flows enter the probeBW phase, a queue forms in the bottleneck, this causes the RTT to grow longer. Moreover, sometimes the RTT does not change anymore as can be observed at between 170 and 180 s of the simulation experiment. Here, as the queue is sustained longer, the operating point B (Figure 1) is crossed; as a result, the RTT remains at its highest state. BBR-ACD considers both the elongated RTT and identical-RTT as a signal of congestion. At this time, whenever a packet loss is signaled by 3-dupACK, BBR-ACD halves its CWND in order to provide enough time to the bottleneck for processing the extra

packets. As a result, the queue releases quickly, reduces the RTT, and stops unwanted retransmissions. From Figure 10d, we can observe that the throughput and the goodput remain the same almost all the time. The decreases are caused when the buffer becomes fully occupied and results in congestions. Because of proper identification and management of such congestions by BBR-ACD, it recovers quickly and produces not only a higher throughput but also a higher goodput. Specifically, BBR-ACD could send a total of 70,324 packets during the same 300 s simulation time and successfully received 68,013 packets during this time period, i.e., only 2311 packets are lost, which is only 3.3% loss of packets. The interested readers can investigate further with the simulation code uploaded in the GitHub repository given in [32].
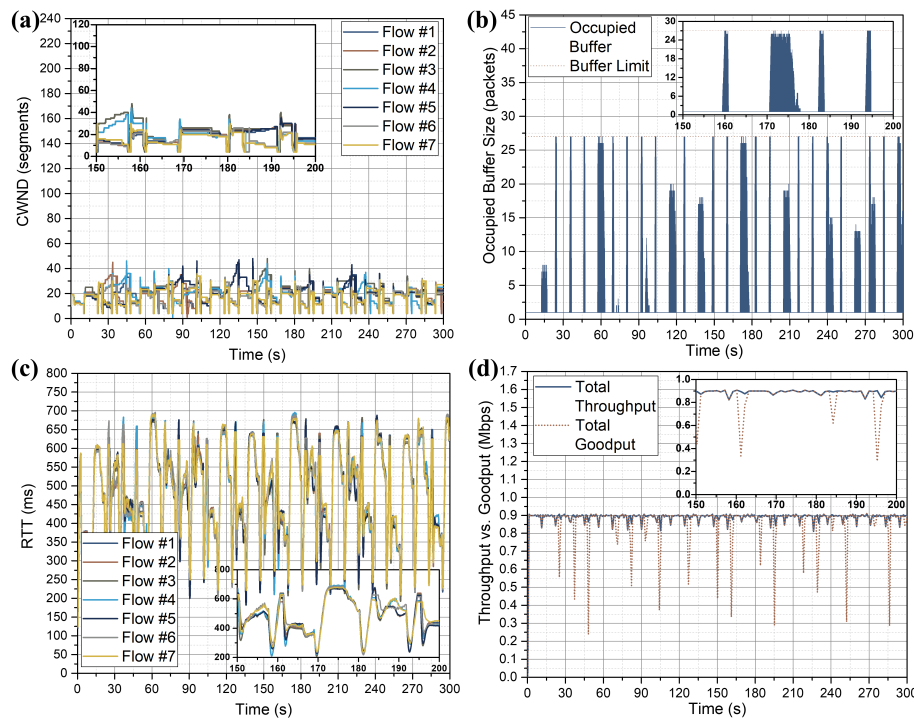

Figure 10. In the simulation experiment, the performance of BBR-ACD in terms of (**a**) congestion window (CWND), (**b**) occupied buffer size, (**c**) round trip time (RTT), and (**d**) throughput vs. goodput.

Finally, Table 3 summarizes the effect of α on the performance of BBR-ACD during the simulation experiment. As can be observed, the smaller value of α makes BBR-ACD less sensitive to losses and results in comparatively higher retransmissions with a smaller number of sent packets. As a result, both the throughput and goodput are comparatively low. On the other hand, BBR-ACD becomes comparatively more sensitive to losses with a higher value of α. While the higher value of alpha reduces the number of retransmissions, the number of sent packets reduces too. By setting the value of α as 1000 μs, we could achieve both high throughput and goodput, i.e., we could send more packets with a smaller number of retransmissions. However, we believe that the proper value of α may vary depending on the scenarios. Therefore, in future work, traffic classification by network visibility techniques [34,35] can be adapted to classify different scenarios in order to properly adjust the value of α.

Table 3. Performance of BBR-ACD for different values of α.

Parameter	α = 500 μs	α = 1000 μs	α = 1500 μs
Total transmitted (packets)	69,236	70,324	69,139
Total retransmitted (packets)	4523	2311	1568
Total throughput (Mbps)	0.881	0.895	0.882
Total goodput (Mbps)	0.821	0.863	0.855

6. Conclusions

In this work, we investigated the excessive retransmission problem of BBR in the case of a congested network scenario. To tackle this issue, we proposed a derivative of BBR as BBR-ACD that consists of a novel advanced congestion detection mechanism which enables it to successfully differentiate between packet losses because of actual congestions and network failures. Upon detection of packet losses due to actual congestion events, BBR-ACD halves the CWND so that the network gets enough time to handle the congested network situation. The proposed BBR-ACD is easily implementable inside the current BBR v1.0, the Linux Kernel code of which has been made available online.

We performed testbed experiments to investigate the performance of BBR and BBR-ACD, and simulation experiments to investigate the root cause of the performance degradation of BBR and the improvements by BBR-ACD. We could reduce about half of the retransmissions in testbed experiments by BBR-ACD. In addition, we improved the goodput in both congested and uncongested network scenarios because BBR-ACD could avoid contraction of the CWND for the packet loss events due to network failures. From these results, we conclude that the addition of BBR-ACD would make BBR more balanced and compatible with the current network and improve the performance further.

In future work, the proposed actual congestion detection approach can be extended to other delay-based congestion control algorithms where the minRTT information is available. Moreover, network visibility techniques can be used to further improve BBR-ACD.

Author Contributions: I.M. proposed the idea, conducted the experiments, and wrote the manuscript. G.-H.K. contributed by setting up the testbed environment. T.L. contributed by processing the experiment results. Y.-Z.C. supervised the entire research. All authors have read and agreed to the published version of the manuscript.

Abbreviations

The following abbreviations are used in this manuscript:

BBR	Bottleneck bandwidth and round-trip time
BBR-ACD	BBR with actual congestion detection
TCP	Transmission control protocol
minRTT	minimum Round-trip time
BtlBW	Bottleneck bandwidth
RTT	Round-trip time
NS-3	Network simulator 3
IETF	Internet engineering tasking force
3-dupACK	3 duplicate acknowledgement
CWND	Congestion window
MSS	Maximum segment size
RTO	Retransmission time-out

References

1. Musku, M.R.; Chronopoulos, A.T.; Popescu, D.C. Joint rate and power control using game theory. In Proceedings of the 2006 3rd IEEE Consumer Communications and Networking Conference, Glasgow, UK, 8–11 May 2006; Volume 2, pp. 1258–1262.
2. Tsiropoulou, E.E.; Vamvakas, P.; Papavassiliou, S. Joint utility-based uplink power and rate allocation in wireless networks: A non-cooperative game theoretic framework. *Phys. Commun.* **2013**, *9*, 299–307.
3. Samaraweera, N. Non-congestion packet loss detection for TCP error recovery using wireless links. *IEE Proc. Commun.* **1999**, *146*, 222–230.
4. Seddik-Ghaleb, A.; Ghamri-Doudane, Y.; Senouci, S.M. TCP WELCOME TCP variant for Wireless Environment, Link losses, and COngestion packet loss ModEls. In Proceedings of the 2009 First International Communication Systems and Networks and Workshops, Bangalore, India, 5–10 January 2009.
5. Jacobson, V. Congestion avoidance and control. *ACM Sigcomm Comput. Commun. Rev.* **1988**, *18*, 314–329.
6. Nishida, Y. The NewReno Modification to TCP's Fast Recovery Algorithm. Standards Track. 2012; pp. 1–16. Available online: http://ftp://163.22.12.51/rfc/rfc6582.txt.pdf (accessed on 9 January 2020)
7. 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.
8. Ha, S.; Rhee, I.; Xu, L. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 64–74.
9. Mascolo, S.; Casetti, C.; Gerla, M.; Sanadidi, M.Y.; Wang, R. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, Rome, Italy, 16–21 July 2001; pp. 287–297.
10. Kliazovich, D.; Granelli, F.; Miorandi, D. TCP Westwood+ enhancement in high-speed long-distance networks. In Proceedings of the 2006 IEEE International Conference on Communications, Istanbul, Turkey, 11–15 June 2006; Volume 2, pp. 710–715.
11. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *ACM Queue* **2016**, *14*. doi:10.1145/3012426.3022184.
12. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; et al. B4: Experience with a globally-deployed software defined WAN. *ACM Sigcomm Comput. Commun. Rev.* **2013**, *43*, 3–14.
13. Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in computer communications. *Proc. Int. Conf. Commun.* **1979**, *43*, 1–43.
14. Hock, M.; Bless, R.; Zitterbart, M. Experimental evaluation of BBR congestion control. In Proceedings of the 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017.
15. Atxutegi, E.; Liberal, F.; Haile, H.K.; Grinnemo, K.J.; Brunstrom, A.; Arvidsson, A. On the use of TCP BBR in cellular networks. *IEEE Commun. Mag.* **2018**, *56*, 172–179.
16. Ma, S.; Jiang, J.; Wang, W.; Li, B. Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis. *arXiv* **2017**, arXiv:1706.09115.
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, Amsterdam, The Netherlands, 21–23 October 2019; pp. 130–136.
18. Riley, G.F.; Henderson, T.R. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 15–34.
19. Jain, V.; Mittal, V.; Tahiliani, M.P. Design and implementation of TCP BBR in ns-3. In Proceedings of the 10th Workshop on ns-3, Surathkal, India, 13–14 June 2018; pp. 16–22.
20. Jacobson, V.; Cardwell, N.; Cheng, Y.; Hassas, S. Bottleneck Bandwidth and RTT (BBR) Congestion Control. Available online: https://github.com/torvalds/linux/blob/master/net/ipv4/tcp_bbr.c (accessed on 29 November 2019).
21. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control: An Update. Available online: https://www.ietf.org/proceedings/98/slides/slides-98-iccrg-an-update-on-bbr-congestion-control-00.pdf (accessed on 29 November 2019).

22. Ware, R.; Mukerjee, M.K.; Seshan, S.; Sherry, J. Modeling BBR's Interactions with Loss-Based Congestion Control. In Proceedings of the Internet Measurement Conference, Amsterdam, The Netherlands, 21–23 October 2019; pp. 137–143.

23. Yeganeh, S.H.; Swett, I.; Vasiliev, V.; Jha, P.; Seung, Y.; Mathis, M.; Jacobson, V. BBR v2 A Model-Based Congestion Control IETF 104 Update. Available online: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00 (accessed on 29 November 2019).

24. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Jha, P.; Seung, Y.; Swett, I.; Victor, V.; Wu, B.; Mathis, M.; Jacobson, V. BBR v2 A Model-Based Congestion Contro IETF 105 Update. Available online: https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control (accessed on 29 November 2019).

25. Linux v4.17. Available online: https://kernelnewbies.org/Linux_4.17 (accessed on 29 November 2019).

26. Neukirchen, L. fq—Job Queue Log Viewer. Available online: http://manpages.ubuntu.com/manpages/bionic/en/man1/fq.1.html (accessed on 29 November 2019).

27. Hemminger, S. fq—Job Queue Log Viewer. Available online: http://manpages.ubuntu.com/manpages/bionic/man8/tc-netem.8.html (accessed on 29 November 2019).

28. Miller, D. ethtool—Query or Control Network Driver and Hardware Settings. Available online: http://manpages.ubuntu.com/manpages/bionic/man8/ethtool.8.html (accessed on 29 November 2019).

29. Dugan, J.; Elliott, S.; Mah, B.A.; Poskanzer, J.; Prabhu, K. iperf3—Perform Network throughput Tests. Available online: http://manpages.ubuntu.com/manpages/bionic/en/man1/iperf3.1.html (accessed on 29 November 2019).

30. Combs, G. wireshark—Interactively Dump and Analyze Network Traffic. Available online: https://manpages.ubuntu.com/manpages/bionic/man1/wireshark.1.html (accessed on 29 November 2019).

31. Baumgarten, F.; Welsh, M.; Cox, A.; Hoang, T.; Eckenfels, B.; Micek, B. netstat—Print Network Connections, Routing Tables, Interface Statistics, Masquerade Connections, and Multicast Memberships. Available online: http://manpages.ubuntu.com/manpages/bionic/man8/netstat.8.html (accessed on 29 November 2019).

32. Mahmud, I. BBR-ACD: NS3 Implementation Code. Available online: https://github.com/imtiaztee/BBR-ACD-complete-NS3-implementation (accessed on 29 November 2019).

33. Mahmud, I. BBR-ACD: Linux Kernel Implementation Code. Available online: https://github.com/imtiaztee/BBR-ACD-Linux-Kernel-Code (accessed on 29 November 2019).

34. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 445–458.

35. Pescape, A.; Montieri, A.; Aceto, G.; Ciuonzo, D. Anonymity Services Tor, I2P, JonDonym: Classifying in the Dark (Web). *IEEE Trans. Dependable Secur. Comput.* **2018**, doi: 10.1109/TDSC.2018.2804394.