Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 7222030, 16 pages
https://doi.org/10.1155/2022/7222030

Research Article

A QoS-Based Fairness-Aware BBR Congestion Control Algorithm Using QUIC

Yi Han,[1] Mengjie Zuo,[1] Huijun Yuan,[1] Yi Zhong,[1] Zhenhui Yuan,[2] and Ting Bi[3]

[1]*School of Information Engineering, Wuhan University of Technology, Wuhan 430070, China*
[2]*Department of Computer and Information Science, Northumbria University, Newcastle Upon Tyne, UK*
[3]*Department of Computer Science, Maynooth University, Co. Kildare, Ireland*

Correspondence should be addressed to Yi Zhong; zhongyi@whut.edu.cn

Received 29 October 2021; Accepted 30 March 2022; Published 29 April 2022

Academic Editor: Basem M. Elhalawany

Congestion control is a fundamental technology to balance the traffic load and the network. The Internet Engineering Task Force (IETF) Quick UDP Internet Connection (QUIC) protocol has flexible congestion control and at the same time possesses the advantages of high efficiency, low latency, and easy deployment at the application layer. Bottleneck bandwidth and round-trip propagation time (BBR) is an optional congestion control algorithm adopted by QUIC. BBR can significantly increase throughput and reduce latency, in particular over long-haul paths. However, BBR results in high packet loss in low bandwidth and low fairness in multi-stream scenarios. In this article, we propose the enhanced BBR congestion control (eBCC) algorithm, which improves the BBR algorithm in two aspects: (1) 10.87% higher throughput and 74.58% lower packet loss rate in the low-bandwidth scenario and (2) 8.39% higher fairness in the multi-stream scenario. This improvement makes eBCC very suitable for IoT communications to provide better QoS services.

1. Introduction

The goal of congestion control is to maximize the use of the bandwidth of the network link, avoid performance loss caused by network congestion, and achieve higher transmission efficiency. Since the 1980s, the transmission control protocol (TCP) has been proposed in order to avoid congestion while improving bandwidth utilization so that the Internet is no longer affected by continuous overload. Delay-based congestion control algorithms represented by Vegas [1] will be completely overwhelmed when competing with loss-based algorithms. Loss-based algorithms such as NewReno [2] and CUBIC [3] have been widely used. However, driven by the advancement of transmission technology and the development of network equipment, the increase in network transmission capacity has caused existing loss-based algorithms to experience buffer bloat [4] and unexpected performance degradation. With the expansion of the queue depth of network intermediate devices, the loss-based algorithm is no longer suitable for today's network environment. The congestion avoidance phase of such algorithms will gradually increase the sending window until the bottleneck queue is filled. Even if the link starts to get congested, the sending window will not decrease. The algorithm can only adopt a speed reduction strategy to relieve congestion when a packet loss event occurs. But at this time, it has already caused great fluctuations in network delay and affected the overall network throughput. Different from loss-based and delay-based algorithms, BBR [5] tries to achieve high link utilization by estimating available Bottleneck link bandwidth (BtlBw) and round-trip propagation delay (RTprop) to calculate bandwidth-delay product (BDP) and also avoid creating queues in bottleneck buffers. BBR is also widely deployed on Google's own production platforms such as the B4 WAN and YouTube. BBR is also open source and integrated into the Linux kernel.

In today's network transmission, most Internet traffic uses TCP as the control protocol of the transport layer to achieve reliable transmission. But TCP is difficult to meet the user scenarios that require higher and higher transmission performance with the development of the Internet. The QUIC [6] launched by Google in 2012 has the characteristics of multiplexing, mandatory encryption, avoiding head-of-line blocking, and can

establish connections with less round-trip time (RTT), which solves the most urgent problem of TCP and helps network services to further improve the user experience. Although most of the research related to transport protocols in the past dozen years has focused on TCP, this exploration is tedious and slow to develop. The emergence of the QUIC protocol has broken this rigid situation. It uses UDP as the basic protocol and has the characteristics of flexible deployment. It does not require the support of the operating system and the kernel and realizes the reliable transmission, congestion control, and flow control of TCP at the application level. QUIC can maintain optimal network conditions when the number of users, network traffic, transmission content size, and network interactive services increase, providing different but more accurate and effective congestion control for each user, and meeting the high real-time, high bandwidth utilization, and low latency transmission requirements of Internet applications.

Both QUIC and BBR have been reported by Google as significant performance improvements and have attracted widespread interest. QUIC and BBR have been studied and evaluated in different network scenarios, respectively [7, 8]. They are both developed by Google and currently used together in some commercial products, such as Chrome browser and YouTube. According to [6, 9], as of 2018, 7% of total Internet traffic and 30% of Google egress traffic are generated by QUIC connections. At present, CUBIC is still the default congestion control in QUIC implementations, but BBR is also optional congestion control in QUIC. BBR uses a feedback-driven autonomous adjustment mechanism to keep the initial value of the congestion window consistent with the capacity of the network so that the network maintains a state of high throughput and low latency. BBR does not use packet loss events as a signal of congestion. It controls the sending rate, further reducing the risk of congestion. Combined with QUIC's flexible congestion control mechanism, the algorithm can be optimized at the application layer and have lower transmission delay. BBR with QUIC is relatively accurate for bandwidth estimation, which helps to shorten transmission delay. It can cope with various types of network packet loss, maintain a high transmission speed when the packet loss rate increases, and cope with network changes well.

However, BBR also has some deficiencies and defects. In transmission links with high latency and bandwidth, the characteristics related to the BBR pacing rate and RTT will cause excessive data transmission rate during the transmission process, resulting in packet loss that seriously affects the quality and efficiency of the transmission. In a multiuser scenario, the network delay is variable due to multiple concurrent data senders in the same network. When the network encounters congestion, the congestion control algorithm must control the transmission rate of the data flow while ensuring the bandwidth utilization, to reduce the delay as much as possible and maintain a high throughput. In this situation where multiple streams compete in the network link, the streams will interact with each other, and the network transmission performance of the multiple streams will be affected. Moreover, BBR determines the pacing rate based on the bandwidth-delay product (BDP), and the transmission rate is maintained at high throughput. When there is a burst of traffic in the network, it will inevitably cause network congestion, and it may restart from the initial state of BBR with a low recovery speed and the cache queue is also cleared.

Combining the characteristics of loss-based and delay-based algorithms, this paper proposes an enhanced BBR congestion control (eBCC) algorithm, which reduces the packet loss rate and packet retransmission chance, while also trying to improve the fairness of the algorithm in multi-stream transmission. At the same time, when the link suffers from a higher delay, eBCC has proved to be able to increase the transmission throughput by reducing the sending rate and packet loss.

The rest of this article is organized as follows. Section 2 introduces the related research of QUIC protocol and BBR algorithm. Section 3 analyzes the original BBR algorithm and elaborates the proposed eBCC algorithm in this paper. In Section 4, we evaluate the proposed algorithm in different network scenarios and perform the analysis of the experimental results. Finally, the algorithm and experimental results are concluded in section 5.

2. Related Work

2.1. Research on Congestion Control Handling. In order to meet the increasing demand for Internet services and solve the problem of rapid growth of network traffic, multiple servers are usually used to improve network performance [10]. However, the problems of network congestion and overloaded servers still arise. Therefore, a load balancer is needed to overcome these problems, by distributing the load of requests and traffic among multiple resources such as servers and network links, in order to improve the overall network performance. Software-defined networking (SDN) is considered to solve the problems of traditional load balancers and plays an important role in network optimization and performance improvement. Hamed et al. [11] implement the "CPU-based" and "CPU-Memory-based" load balancing techniques and evaluate their performance compared to the static round-robin and random-based load balancing techniques using Ryu OpenFlow controller. The results show that the proposed schemes achieve more reliability and higher resource utilization than the round-robin and random-based load balancing strategies. In addition, they have a lower response time and higher transaction rate and throughput. The proposed schemes exhibit more scalability and low-cost characteristics.

In addition to SDN for congestion control and load balancing, there are also congestion control algorithms that can be used to improve network transmission performance. Congestion control algorithms protect the Internet from continuous traffic by adjusting the sending rate, reducing congested sending, and improving bandwidth utilization. The classic congestion control algorithms include delay-based and loss-based algorithms.

Vegas [1] was the first implementation of congestion control algorithm using delay as a congestion signal. The Vegas algorithm defines BaseRTT and sets it to the minimum of all measured RTT. When receiving a duplicate ACK, Vegas checks whether the difference between the current time and the recorded timestamp is greater than BaseRTT. If it is, the packet is retransmitted immediately without waiting for a

third duplicate ACK. When acknowledging a retransmitted packet, Vegas reduces the congestion window by a factor of 3/4. When a non-duplicated ACK is received and it is the first or second acknowledgment after data retransmission, Vegas will check again whether the data transmission interval exceeds BaseRTT. If Vegas exceeds BaseRTT, it will also perform data retransmission operation.

Among the congestion control algorithms based on packet loss, NewReno and CUBIC are typical algorithms that are often used for analysis and examples. NewReno [2] adopts the well-known Additive Increase Multiplicative Decrease (AIMD) mechanism. During slow start, its congestion window (cwnd) increases exponentially with each RTT until the slow start threshold ssthresh is reached. After that, cwnd enters the congestion avoidance phase, adding one packet per RTT cycle. If three duplicate ACKs are received, the fast recovery phase is entered, and the new ssthresh is set to cwnd/2. Then, the cwnd is halved to enter the congestion avoidance phase. However, if a timeout occurs, it will go into slow start and start over. NewReno can distinguish one-time congestion from multiple-time congestion, improving the robustness and throughput after packet loss.

BIC [12] calculates the maximum link capacity and maximizes the congestion window. It can reduce the calculation of large link capacity by binary search and quickly find the optimal congestion window size. CUBIC [3] is an enhanced version of BIC that improves TCP friendliness and RTT fairness and simplifies BIC window control. After a packet loss event occurs, the window growth function of CUBIC is controlled by the cubic function of time and provides good stability and scalability. Compared with AIMD based on cubic function, CUBIC adopts a different mechanism. After cwnd is decreased, cwnd rises in a concave shape until it reaches the value of cwnd before the decrease. After that, CUBIC increases its growth rate and rises in a convex shape. CUBIC is currently the default congestion control algorithm in Linux.

2.2. Research on QUIC Protocol. QUIC is developed based on UDP, and its design philosophy enables it compatible with the safety and reliability of TCP and the fast speed of UDP. Kharat et al. [13] explored the QUIC function using an experiment implemented with a local test bench connected to a wireless access point in the campus network environment. The experimental results show that QUIC maintains excellent performance in the form of throughput and TCP/IP acceleration. The fairness of QUIC under competing traffic conditions was also checked, and it was found that it performed well in long-lived traffic.

Transmission protocols continue to evolve to meet the emerging needs of users and new services. Corbel et al. [14] specifically analyzed the protocol fairness when TCP and QUIC streams coexist on the wireless link of the mobile network. The results show that when the loss rate of the mobile network is low, the emulation of multiple TCP connections, the limitation of the congestion window size, and the use of hybrid start (hystart) have a great impact on fairness. The incorrect setting of the default parameters of these mechanisms or the activation of the hystart option will affect the performance of the transmission protocol and therefore also affect fairness.

With the increasing wide-ranging interest in the flexibility and rich features of QUIC, Biasio et al. [15] demonstrated the native implementation of QUIC in ns-3 and illustrated the implemented functions, the main assumptions, differences related to QUIC Internet-Drafts, and a set of examples. Our paper uses this implementation of QUIC in ns-3 to conduct our experiments.

Soni et al. [7] conducted an in-depth study of the QUIC protocol from the perspective of its implementation and application. The authors used Amazon AWS services for test bench implementation to evaluate the performance of the QUIC protocol against TCP and UDP protocols. They found that QUIC performed better than TCP in terms of throughput and data retrieval time. However, its performance is between TCP and UDP. In addition, it provides the fast data transmission of UDP and the reliability of TCP. At the same time, various problems in the QUIC protocol were discovered, such as forward secrecy, replay attacks, and denial of service attacks.

2.3. Related Research on BBR Congestion Control Algorithm. Scholz et al. [8] proposed a publicly available framework for repeatable TCP measurement based on network simulation, designed to analyze the TCP BBR algorithm. They reproduced and confirmed the weaknesses of the current BBR implementation and provided further insights. The two main problems were discovered: One is that bandwidth may be shared unfairly and the second problem is that it takes too long before the bandwidth balance is restored. The behavior on the synchronization between BBR streams was analyzed, showing that it reached a fair balance of long-lived streams.

Zhang et al. [16] revealed that the aggressiveness of BBR would reduce the performance of CUBIC and the entire Internet transmission. They proposed a simple and effective solution based on BBR, Modest BBR. The core idea of Modest BBR is to reduce retransmission and certain aggression by sacrificing a small amount of bandwidth, to obtain better fairness through loss-sensitive methods. Through a large number of test bench experiments and Mininet simulations, it is found that Modest BBR can maintain high throughput and short convergence time when coexisting with Cubic while improving overall performance and achieving better fairness for loss-based solutions.

Kim and Cho [17] proposed a delay-aware BBR (DA-BBR) congestion control algorithm to alleviate the RTT fairness problem among streams using BBR. A network simulation was carried out using Mininet; the results showed that DA-BBR increased the fairness index by 1.6 times of the original BBR, and the number of packet retransmissions was greatly reduced. Even in competition with RTT five times higher, DA-BBR streams with short RTT exhibit fair throughput.

To improve the fairness between the streams using TCP-BBR congestion control algorithm and the ones using delay-based congestion control algorithm, Jia et al. [18] proposed a TCP-BBR-based congestion control algorithm, which has moderate fairness and is called Modest Fairness BBR (MFBBR). The simulation results on Mininet showed that the algorithm can improve the fairness of BBR when coexisting with Westwood

(a congestion control algorithm based on time delay) [19] and it has better fairness than the delay-based congestion control algorithm.

Song et al. [20] proposed a BBR congestion window scaling (BBR-CWS) scheme, which uses a loss-based congestion control algorithm to improve the interprotocol fairness of BBR. The results of Mininet experiments can confirm that the fairness between BBR-CWS and CUBIC is improved by 73% and has a value of 0.9 or higher in most bottleneck buffer scenarios. In addition, compared with the original BBR, the number of packet retransmissions is reduced by 96%.

2.4. Research on BBR Based on QUIC. In the mobile network environment, due to the common but uncertain fluctuations in round-trip time and random loss events in the air, the congestion window growth is unexpectedly hindered. The single-connection strategy still leads to degraded and highly variable completion time interfaces. To maintain a flexible congestion window for networks with such fluctuations, Qian et al. [21] proposed an intelligent connection management scheme based on QUIC. According to the performance evaluation results obtained from the LTE-A/Wi-Fi test network, the proposed multi-QUIC scheme can effectively overcome the existing limitations in congestion control algorithms such as NewReno, CUBIC, and BBR. The median completion time of a piece of web content can be improved up to 59.1%, and the 95th percentile completion time is improved by up to 72.3%. The significance of this work is to achieve highly robust short-term content download performance in response to the uncertainty of various network conditions and different congestion control schemes.

Wang et al. [22] conducted a preliminary evaluation of the performance of QUIC and BBR congestion control algorithm through GEO satellite Internet access on a private network simulation testbed. The obtained results and analysis confirm that compared with the classic CUBIC, the performance of the new satellite Internet using QUIC with BBR is improved.

Due to the strong interest of the ns-3 community in the QUIC module, Paro et al. [23] proposed some extensions of the ns-3 QUIC module to make it more flexible. Integrate BBR into the QUIC module, and implement the necessary pacing and rate sampling mechanisms, as well as a new scheduling interface with three different scheduling styles. Use the network traffic model in the literature to test the new features to verify whether their performance meets expectations. Our paper uses the code of the extension module BBR when conducting the BBR experiment based on QUIC.

Haile et al. [24] used the scalability of QUIC to enhance BBR; instead of using the ACK rate observed at the sender, it applied a more desirable transfer rate calculated at the receiver. Simulation experiments based on 5G tracking in CloudLab proved that the modified QUIC can significantly reduce latency without any significant impact on throughput. In particular, a 39% reduction in round-trip time (RTT) can be observed in some cases, and the throughput is also reduced by 2.7% in the worst case.

The improvement of the BBR algorithm proposed above is an improvement in the fairness of BBR. The eBCC algorithm proposed in this study can achieve higher throughput and lower packet loss rate than the BBR algorithm in the case of low bandwidth in both single and multiple streaming.

3. Proposed Congestion Control Algorithm

3.1. Calculation of Fairness Index. Since multiple links in the network share the network bandwidth, when congestion occurs, the fairness between various congestion control algorithms and the fairness within the same congestion control algorithm is very important. At present, the standard for judging whether the network is fair has not been uniformly stipulated. There are two mainstream standards for judging whether the network is fair. The first standard is that for links with different round-trip delays or using different congestion control algorithms, each link should occupy the same throughput so that it is fair among the links. Another standard is that each link should have the same network throughput when competing, and sources of the same level should get the same amount of network resources, such as response time, throughput, bandwidth, and cache. This standard uses the Jain's fairness index [25] as an evaluation matrix, which is adopted by this paper as a fairness index. The formula is as follows:

$$Jain(x) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{\left(n \times \sum_{i=1}^{n} x_i^2\right)}, \tag{1}$$

where n represents n senders and x_i is the throughput of the i-th link. When multiple data streams compete for bandwidth, the result ranges from 1/n to 1. When all connections are competing for network bandwidth, $Jain(x) = 1$ indicates the fairest situation, while $Jain(x) = 1/n$ reaches the minimum, that is, the most unfair situation. And when all users share the same allocated bandwidth, it is the maximum value.

The fundamental reason for the need to pay attention to fairness is that congestion will inevitably lead to data packet loss and thus cause competition among data streams for limited network resources, and data streams with weaker competitiveness will suffer further degradation in QoS. For example, if a congestion control algorithm is based on packet loss when the sender detects that one data packet is lost, it considers that the network is congested and immediately adopts a congestion avoidance strategy to halve the sending window, actively reducing the sending rate of data packets to avoid network congestion. When there are multiple senders and receivers on the transmission link using traditional congestion control algorithms, multiple data streams with different round-trip times compete for the limited bandwidth of the bottleneck link. The duration of the streams with long round-trip time occupying the bottleneck bandwidth is significantly lower than the streams with short round-trip time. This will result in unfair use of network resources. The BBR congestion control algorithm is the opposite of the traditional congestion control algorithm. The calculation of the sending window of the BBR algorithm is related to RTT. Different RTT streams will lead to different sending windows. Long RTT streams are more aggressive than short RTT streams, and they can send more data in a longer period than short RTT streams,

occupying a lot of bandwidth during transmission. This feature enables it to steal bandwidth from other streams, seize more link bandwidth, impair the transmission performance of short RTT streams simultaneously transmitted on the same path, and in the end, result in a decrease in fairness within the protocol.

3.2. BBR Congestion Control Algorithm. Congestion often occurs in the smallest bandwidth section of a link. This smallest bandwidth is the bottleneck bandwidth of the link, and its size determines the maximum transmission throughput. Packet loss and RTT are not considered congestion control factors in BBR. In BBR, the network link is regarded as a "pipe." The diameter and length of this "pipe" are represented by BtlBw and RTprop, respectively, to estimate the maximum bandwidth and minimum delay of the current network. The volume is BDP (the product of BtlBw and RTprop). When the total amount of data in the link equals BDP, that is, when the sending window is equal to BDP, the throughput is maximized. When the sending window is smaller than BDP, the RTT is a fixed value because the buffer is not occupied at this time. So there is no queuing delay, and the throughput will increase as the sending window increases. When the sending window exceeds BDP, it starts to fill the buffer. At this time, the RTT will increase when the sending window increases, and the throughput reaches the maximum and stop increasing. When the buffer is full, the excess data packets will be discarded. The BBR algorithm always maintains the sending window size near the BDP value, which can guarantee higher throughput and lower delay. However, when the buffer is full, it is already congested before the buffer overflows, which will cause greater delay and introduce a large number of packet retransmissions [5, 26]. Such a large number of retransmissions will also occupy link bandwidth and cause waste of bandwidth resources. It will not only reduce its transmission efficiency but also bring great damage to the flow of loss-based congestion control algorithms.

Multiple QUIC connections of a single user may be initiated from a host, and when they send data at the same time, this may squeeze the bandwidth of the same shared link. Multi-streaming methods designed to improve throughput usually increase queuing delay and packet loss rate. The increased queuing delay leads to a higher BDP value and thus gives more advantage in its bandwidth competition. The retransmission caused by packet loss is a waste of bandwidth. Both of the above will damage other streams on the bottleneck of the shared link. What's more, in the multiuser scenarios, the streams from different users and multi-QUIC connections from the same user are overlapped and compete in the link, which will cause the loss of fairness not only among streams but also among users [27].

3.3. Enhanced BBR Congestion Control Algorithm. The essence of a loss-based congestion control algorithm is still to attribute packet loss to network congestion. To achieve the maximum throughput, these algorithms continuously increase cwnd until the link buffer is filled, causing packet loss, and then reduce the cwnd to perform congestion control. The delay-based congestion control algorithms act based on queues. When the queue exceeds the set threshold, cwnd is processed to reduce the growth rate of cwnd. If packets are still lost after adaptation, the size of cwnd is reduced.

The compound TCP (CTCP) congestion control algorithm [28] combines loss-based and delay-based congestion control methods, allowing the algorithm to quickly increase the sending rate while obtaining high bandwidth scalability and improved TCP fairness. CTCP is designed and usually used in high bandwidth environment, and it is disabled by default [29]. eBCC adjusts the algorithm transition state and congestion control window using loss and queuing status as decision factors. Instead, CTCP runs the legacy TCP's AIMD algorithm and a delay-based high speed congestion control algorithm. The idea of the eBCC algorithm is to use both packet loss and delay as rate control signals. It is different from the CTCP algorithm for the operation after packet loss and the way of judging the delay as a congestion signal. Packet loss is used as the primary factor for adaptation decisions. The gain of eBCC's cwnd is no longer a fixed value of 2 but is adaptively changed according to whether the packet is lost and whether the queue exceeds the threshold. When there are other streams in the link, eBCC will not always fill the bottleneck link and the buffer. It will reduce the sending rate promptly to reduce the packet loss rate. The role of the queuing status is to judge the buffer. eBCC judges the queuing status by calculating the sending window and RTprop. This is to avoid inaccurate measurement of RTprop when the buffer is being occupied by other streams, preventing the depletion of the buffer. In multiuser/multi-QUIC streaming cases, eBCC makes full use of bandwidth when it is radical. When competing with other streams or having congestions, eBCC will respond in time to reduce the sending rate to relieve congestion, reduce packet loss, and sacrifice a small amount of throughput to improve fairness. Enhanced BBR congestion control algorithm proposed by this article is described in the Algorithm 1. The parameters and symbols used in the algorithm are explained in Table 1.

When there is no packet loss and the queue is empty, the growth of the congestion window refers to the idea of the binary search congestion control algorithm [12]. Because all link connections wish to obtain bandwidth resources quickly and fairly, a faster convergence speed is required. The binary search method can converge faster when it approaches the optimal adaptation point after experiencing packet loss. The size of cwnd can be calculated as follows:

$$cwnd = (cwnd + targetCwnd)/2, \tag{2}$$

where $targetCwnd$ indicates the maximum limit of the congestion window.

The CTCP algorithm is deduced by Vegas, using baseRTT as the estimated value of the packet transmission delay on the network path, and baseRTT will be updated according to the minimum RTT value measured during the transmission process. The following algorithm can be used to estimate the number of data that are waiting to be sent in the queue:

Input: t, *Time, targetCwnd*.
Output: *Throughput, delay*.
Time: the total time of transmission;
diff: the extra packets in the queue;
isLost: the packet loss signal;
1: Initialize $t \longleftarrow 0$
2: **while** $t <= Time$ **do**
3: **if** $diff < \gamma$ **and** $isLost$ = false **and** in ProbeBW state **then**
4: keep in ProbeBW state
5: $cwnd = (cwnd + targetCwnd)/2$
6: **elseif** $diff >= \gamma$ **and** $isLost$ = false **and** in ProbeBW state **then**
7: keep in ProbeBW state
8: $cwnd = (pfCwnd + cwnd)/2$
9: **elseif** $diff >= \gamma$**and**$isLost$ = true **and** in ProbeBW state **then**
10: change to ProbeRTT state
11: **elseif** $diff >= \gamma$**and**$isLost$ = true **and** in ProbeRTT state **then**
12: $cwnd = cwnd(1 - \beta)$
13: **end if**
14: **end while**
15: **return***Throughput, delay*.

Algorithm 1: Enhanced BBR Congestion Control Algorithm.

Table 1: Description of parameters and symbols.

Parameter/symbol	Description
n	Number of senders
x_i	Throughput of the i-th link (mbps)
$diff$	The extra packets in the queue
γ	The compromise number of packets in the queue
$baseRTT$	The estimated value of the packet transmission delay (ms)
$targetCwnd$	The maximum limit of the congestion window
$pfCwnd$	The size of the last congestion window
β	The proportion of cwnd size decrease
access_bandwidth	The bandwidth of the transmission link
bottleneck_bandwidth	The bandwidth of the bottleneck link
access_delay	The delay of the transmission link
bottleneck_delay	The delay of the bottleneck link

$$Expected = \frac{cwnd}{baseRTT}, \tag{3}$$

$$Actual = \frac{cwnd}{RTT}, \tag{4}$$

$$diff = (Expected - Actual) * baseRTT. \tag{5}$$

where RTT is the currently measured RTT value and $diff$ is the size of the data packet in the queue. *Expected* refers to the throughput estimate obtained without exceeding the network path, and *Actual* represents the real throughput.

γ is a compromise between fairness and throughput with a value of 30 according to [26+2]. If $diff < \gamma$, the network link is considered to be underutilized. If $diff >= \gamma$, eBCC perceives that the network is fully utilized and the number of data packets in the queue exceeds the specified threshold, and the data transmission rate needs to be reduced accordingly.

$$pfCwnd = 2cwnd - targetCwnd, \tag{6}$$

$$cwnd = (pfCwnd + cwnd)/2. \tag{7}$$

Equations (6) and (7) change the size of the current congestion window to the average value of current *cwnd* and $pfCwnd$. $pfCwnd$ is the size of the last congestion window, that is, the size of *cwnd* when the queue does not exceed the specified threshold. This enables the *cwnd* to increase smoothly when data packets are accumulated in the queue without packet loss.

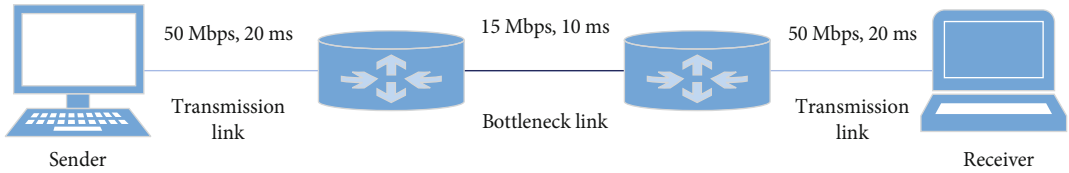

Figure 1: Network simulation for the single-stream scenario.

When a packet loss is detected, the transmission enters the congestion avoidance phase. When the BBR is still in the stable state of ProbeBW, the ProbeBW state is transferred to ProbeRTT. If the BBR is already in the ProbeRTT state, adjust the congestion window size:

$$cwnd = cwnd(1 - \beta), \tag{8}$$

where the value of β is 0.5 according to TCP Reno and NewReno algorithms so that the congestion window can converge to the optimal point faster while the transmission rate is reduced.

4. Experimental Results and Analysis

4.1. General Experimental Setup. The general experimental setup of this paper uses a virtual machine of the Ubuntu operating system with a Linux kernel version of 20.04. We mainly use the extension modules of QUIC [15] and BBR [23] in the network simulator version 3 (ns-3) to simulate the network environment [30]. Two network topologies are created to study the performance of the proposed eBCC algorithm in single-stream and multi-stream scenarios, respectively.

Associated with the Ethernet protocol, the maximum transmission unit in TCP is 1500 bytes, which determines the maximum size of each packet in any transmission. The packet size of the data stream is set to 1400 Bytes, and the transmission interval is set to 100 ms. The total transmission duration is 20 seconds. The transmission starts at end of the first second, which means that the entire process of data streaming lasts for 19 seconds.

4.2. Experiment 1: Varying CWND, RTT, and Throughput of a Single Stream. We set the transmission bandwidth and delay for transmission link and bottleneck link to 50 Mbps and 15 Mbps and 20 ms and 10 ms, respectively. The simulated network scenario of the experiment is shown in Figure 1.

Experiment 1 measured the cwnd, RTT, and transmission throughput of several different congestion control algorithms over time. The measurement results are shown in Figure 2, Figure 3, and Figure 4.

It can be seen from Figure 2 that CUBIC and NewReno regard packet loss events as serious matters. Once the packet loss occurs, an adaption decision to reduce the congestion window will be made. The congestion control window of the BBR algorithm maintains the maximum detected congestion window value most of the time. BBR adjusts the congestion window in the retransmission state when it restores the original maximum detection window value and then exits the retransmission state. In addition to adopting the aggressive transmission strategy of BBR, eBCC considers packet loss when making decisions to reduce the congestion control window.

In Figure 3, NewReno's RTT trend is most stable, followed by CUBIC with two sawtooth changes. And RTT of the above two algorithms is low for a relatively long duration. RTT of BBR and eBCC fluctuates greatly. BBR fluctuates around larger RTT for a long duration, and eBCC fluctuates in a sawtooth shape. The sawtooth peak value of eBCC is close to the larger RTT value maintained by BBR for a long time.

In Figure 4, the throughput variations of the four algorithms can be mapped with cwnd changes in Figure 2. NewReno reduces the congestion control window because of packet loss in its congestion control mechanism. The throughput curve encounters twists and turns around at the 3rd second, and there is a certain drop. The CUBIC algorithm is also for the same reason, because it encounters packet loss and reduces the congestion window to reduce the rate, which causes the throughput curve to twist or the curve has a downward trend at 8th second. The reason for the drop in throughput of BBR and eBCC is because it detects serious congestion and restarts directly from the initial state. BBR is more aggressive in the early stages, but eBCC recovers much faster than BBR when dealing with congestion and has higher average throughput.

4.3. Experiment 2: Varying Link Bandwidth and Delay of a Single Stream. Before conducting multi-stream experiments, two preliminary studies are performed to understand the impact of varying network bandwidth and delay of transmission links and the bottleneck link on the streaming performance when using different congestion control algorithms.

First, in the case of varying bandwidth, we have managed to carry out the experiment using a low bandwidth link. According to the related research on BBR in [8], the difference in bandwidth does not affect the validity of the results. The experiments with high bandwidth can use physical hardware devices in the future so that high-definition video or virtual reality video transmission can be performed. This is considered to be the focus of our following work. Using the same topology illustrated in Figure 1, the transmission link and the bottleneck link were tested with four sets of parameters: 75 Mbps and 17.5 Mbps, 50 Mbps and 15 Mbps, 30 Mbps and 10 Mbps, and 15 Mbps and 5 Mbps, respectively. The delay of the two links is set to 20 ms and 10 ms, respectively.

Second, in the case of varying delay, 50 Mbps and 15 Mbps are selected as the bandwidth of the transmission link bandwidth and the bottleneck link, respectively. The delay values of the transmission link and the bottleneck link are divided into three groups: 20 ms and 10 ms, 15 ms and

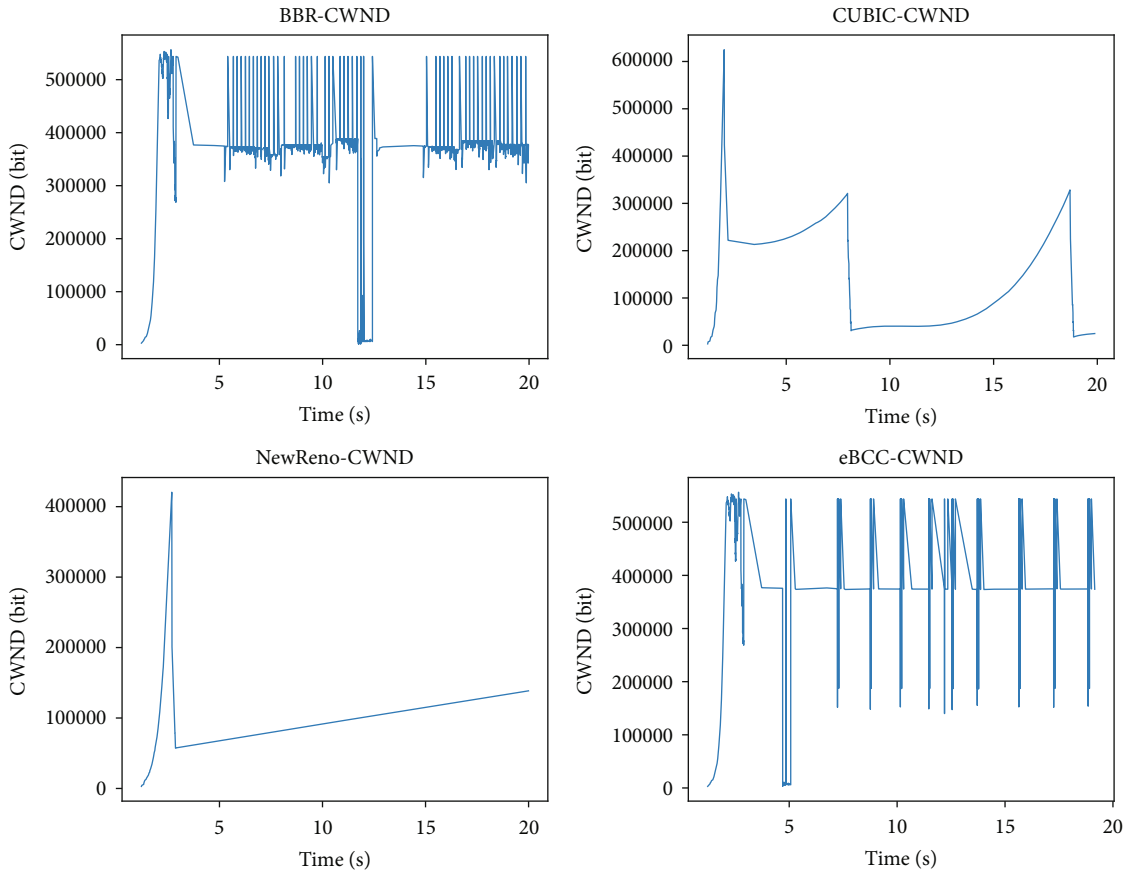

Figure 2: Varying congestion window size during transmission.

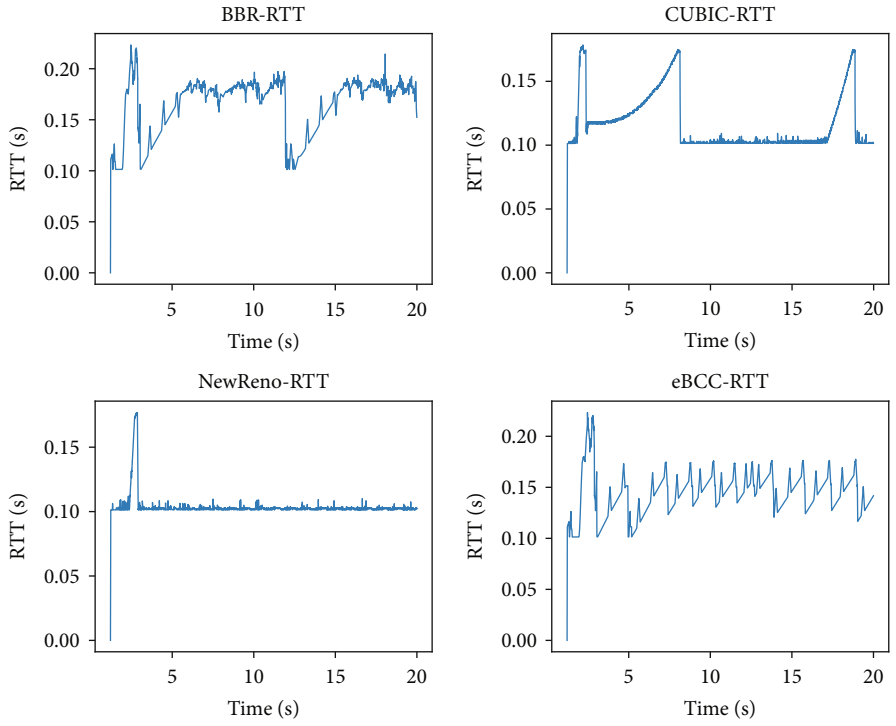

Figure 3: Varying RTT during transmission.

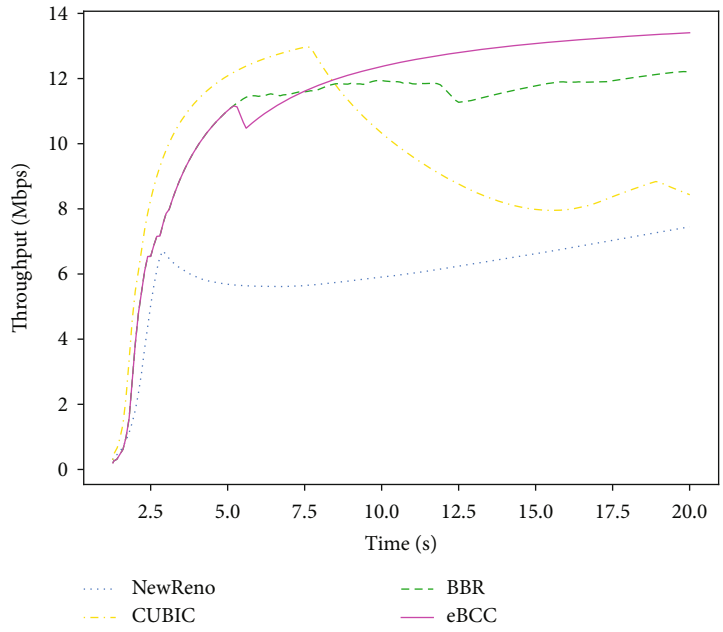

Figure 4: Variation of throughput in real-time transmission.

Table 2: Packet sending/receiving rate (Mbps) for varying bandwidth of different links.

Varied bandwidth(Mbps)	NewReno	CUBIC	BBR	eBCC
access_bandwidth =15; bottleneck_bandwidth =5	4.220/4.188	3.245/3.160	4.776/4.697	4.776/4.697
access_bandwidth =30; bottleneck_bandwidth =10	6.845/6.801	6.509/6.386	9.661/8.870	9.576/9.2662
access_bandwidth =50; bottleneck_bandwidth =15	7.743/7.692	8.840/8.672	14.731/12.465	14.353/13.820
access_bandwidth =75; bottleneck_bandwidth =17.5	8.769/8.708	10.055/9.855	17.413/14.631	16.634/16.087

Table 3: Packet loss rate (%) for varying bandwidth of different links.

Varied bandwidth (Mbps)	NewReno	CUBIC	BBR	eBCC
access_bandwidth =15; bottleneck_bandwidth =5	1.0617	4.8268	2.0855	2.0855
access_bandwidth =30; bottleneck_bandwidth =10	1.008	3.4845	11.1037	4.9245
access_bandwidth =50; bottleneck_bandwidth =15	1.2216	3.4133	19.7338	5.6094
access_bandwidth =75; bottleneck_bandwidth =17.5	1.2343	3.4354	19.9143	5.0625

7.5 ms, and 10 ms and 5 ms, respectively. Both of these experiments were performed under a low link bandwidth and different delay environments, and the experimental results were obtained after repeated measurements. The results are averaged from repeated experiments. The packet sending rate, packet receiving rate, and packet loss rate are shown in Table 2 and Table 3:

Access_bandwidth in Tables 2 and 3 represents the transmission link bandwidth, and bottleneck_bandwidth is the bottleneck link bandwidth. It can be seen from Table 2, Table 3, and Figure 5 that the sending rate of BBR is the highest, but its throughput is not as good as that of eBCC. Table 3 shows that BBR suffers from great packet loss that leads to a large number of packet retransmission and thus its received throughput is lower than the proposed eBCC.

From Figure 5(a) and Table 2, it can be seen that in the four cases, the throughput of the four algorithms all increase as the access_bandwidth increases, and eBCC achieves the highest. The throughput of CUBIC is the lowest when the access_bandwidth is low. When the access_bandwidth increases, the throughput of NewReno becomes the worst among the four algorithms. From Figure 5(b) and Table 3, it can be seen that in terms of packet loss rate performance, NewReno's congestion control is conservative in a rate increase, so packet loss rate is the lowest among the four algorithms, remaining at about 1%. The CUBIC algorithm has the worst packet loss performance

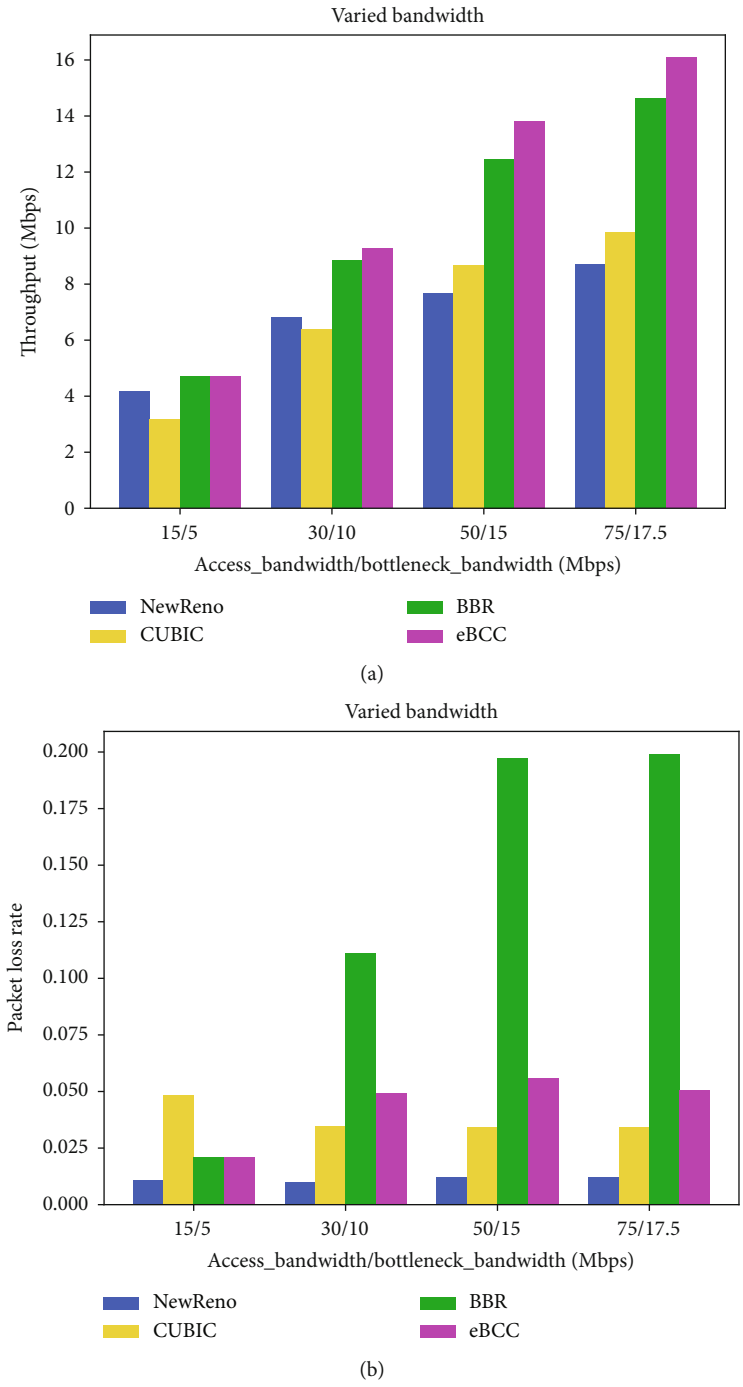

Figure 5: Network performance test results for different link bandwidths and bottleneck link bandwidths. (a) Received throughput. (b) Packet loss rate.

Table 4: Packet sending/receiving rate (Mbps) for different link delays and bottleneck link delays.

Varied delay (ms)	NewReno	CUBIC	BBR	eBCC
access_delay = 20; bottleneck_delay = 10	7.743/7.692	8.840/8.672	14.126/12.037	14.353/13.823
access_delay = 15; bottleneck_delay = 7.5	11.737/11.678	10.717/10.593	14.650/13.099	14.469/14.036
access_delay = 10; bottleneck_delay = 5	12.860/12.827	11.154/11.058	14.448/14.290	14.220/14.079

Table 5: Packet loss rate (%) for different link delays and bottleneck link delays.

Varied delay(ms)	NewReno	CUBIC	BBR	eBCC
access_delay =20; bottleneck_delay =10	1.222	3.217	18.928	5.609
access_delay =15; bottleneck_delay =7.5	0.661	2.118	14.023	4.617
access_delay =10; bottleneck_delay =5	0.456	1.573	1.712	1.744

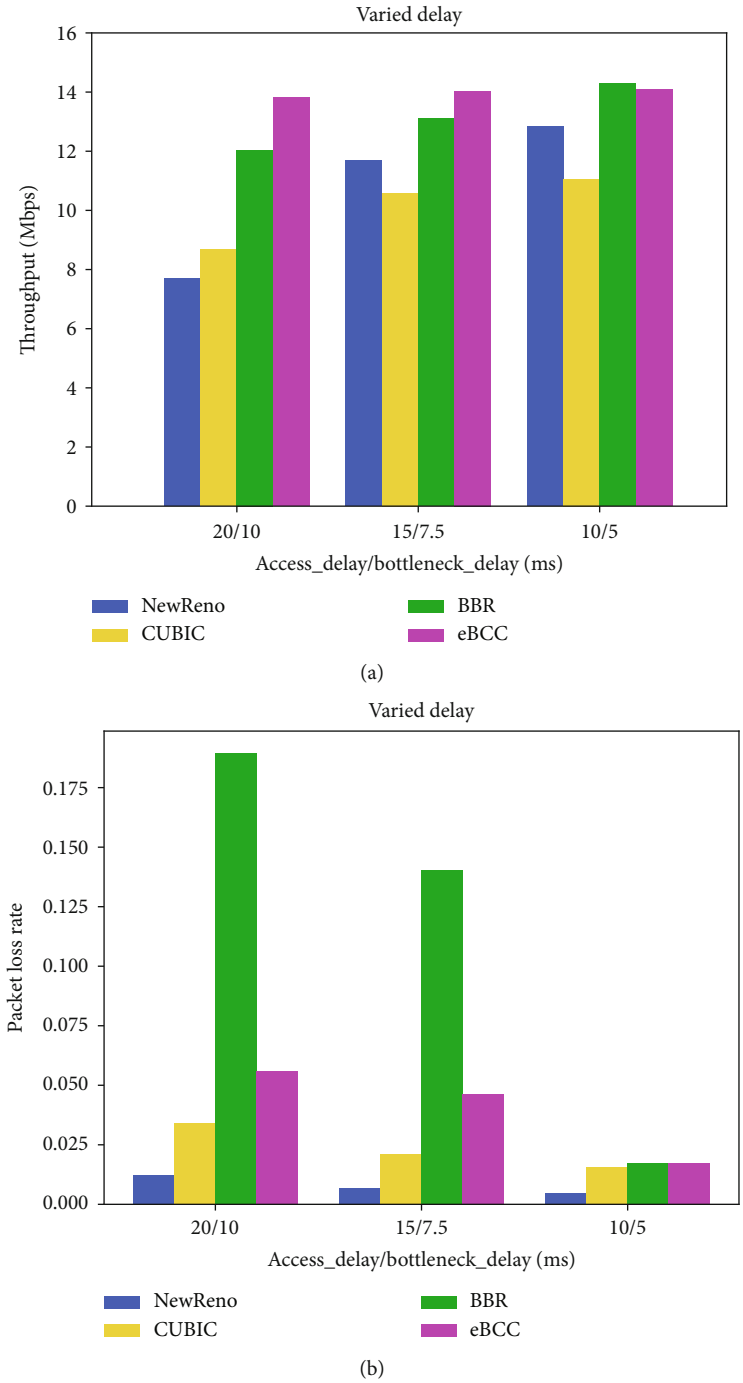

Figure 6: Network performance test results for different link delays and bottleneck link delays. (a) Throughput. (b) Packet loss rate.

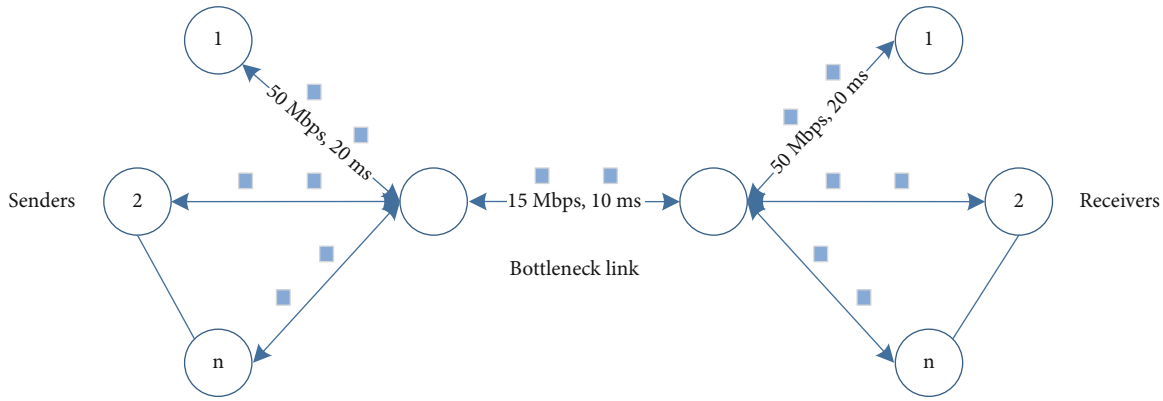

Figure 7: QUIC multi-stream transmission dumbbell network topology.

when the access_bandwidth is low. But when the access_bandwidth increases, the packet loss of CUBIC is stable, maintaining around 3.4%. The packet loss rate of BBR and eBCC is better than CUBIC only when the bandwidth is low, i.e., 15 Mbps and 5 Mbps. In other cases, the packet loss rate of both is relatively high. BBR has the worst performance in terms of packet loss, more than twice that of eBCC.

In Tables 4 and 5, access_delay refers to the transmission link delay, and bottleneck_delay refers to the bottleneck link delay. It can be noted from Tables 4 and 5 and Figure 6 that when the transmission access_delay and bottleneck_delay are 10 ms and 5 ms, BBR has the best throughput performance. eBCC has lower throughput and higher packet loss rate than BBR. When the transmission access_delay and bottleneck_delay are 20 ms and 10 ms and 15 ms and 7.5 ms, the throughput of eBCC is the highest among several algorithms, and the performance of eBCC's packet loss rate is also better than BBR. In these three cases, NewReno maintains the lowest packet loss rate.

Tables 4 and 5 record the packet sending/receiving rate and packet loss rate of NewReno, CUBIC, BBR, and eBCC for different transmission link delays and bottleneck link delays. Inferring from the packet loss rate, in the two sets of experiments with higher delay, the throughput of BBR is lower than eBCC due to retransmission, but in the group with low delay, the performance gap between the throughput and the packet loss rate of the two is not significant.

As shown in Figure 6, Table 4, and Table 5, as the access_delay and bottleneck_delay decrease, the throughput of the four algorithms has different degrees of improvement. The throughput of NewReno and CUBIC is worse than that of BBR and eBCC. Among them, CUBIC performs better than NewReno when the access_delay is 20 ms and the bottleneck_delay is 10 ms, and the throughput of NewReno is higher when the delay is lower. In terms of packet loss, NewReno and CUBIC are lower than BBR and eBCC but also have lower throughput. NewReno performs particularly well in terms of packet loss (around 0.5%) when the delay is low. In the case of higher latency, eBCC reduces cwnd when packet loss is detected to reduce unnecessary data packet retransmissions by slightly slowing the sending rate, which is not implemented by BBR. This not only reduces the packet loss rate but also improves the overall transmission throughput. In the case of lower latency, BBR can maintain high throughput, and the utilization of network links has reached a high peak. In this case, high throughput is a reward for a high packet loss rate risk.

Table 6: Average packet sending/receiving rate (Mbps) per flow in multi-stream transmission.

Number of flows	NewReno	CUBIC	BBR	eBCC
3 flows	4.234/4.209	4.511/4.452	5.245/4.076	5.005/4.527
5 flows	2.796/2.779	2.602/2.554	3.281/2.415	3.105/2.677

Table 7: Average packet loss rate (%) in multi-stream transmission.

Number of flows	NewReno	CUBIC	BBR	eBCC
3 flows	0.857	2.392	28.135	14.488
5 flows	0.929	2.721	35.430	20.443

Table 8: Average fairness index in multi-stream transmission.

Number of flows	NewReno	CUBIC	BBR	eBCC
3 flows	0.999	0.962	0.951	0.996
5 flows	0.994	0.991	0.910	0.986

From Figures 5–6, the delay has a greater impact on packet loss rate, and bandwidth has a greater impact on throughput performance. In the case of higher bandwidth and higher latency, eBCC performs better in throughput, and the performance of packet loss is much better than BBR.

4.4. Experiment 3: Performance Study of Multi-Stream Transmission. The multi-stream transmission experiment sets up multiple nodes in a dumbbell network topology. The experimental network topology is shown in Figure 7. Multiple pairs of clients and servers are defined to simulate different numbers of streams sending and receiving data packets simultaneously. Four congestion control algorithms

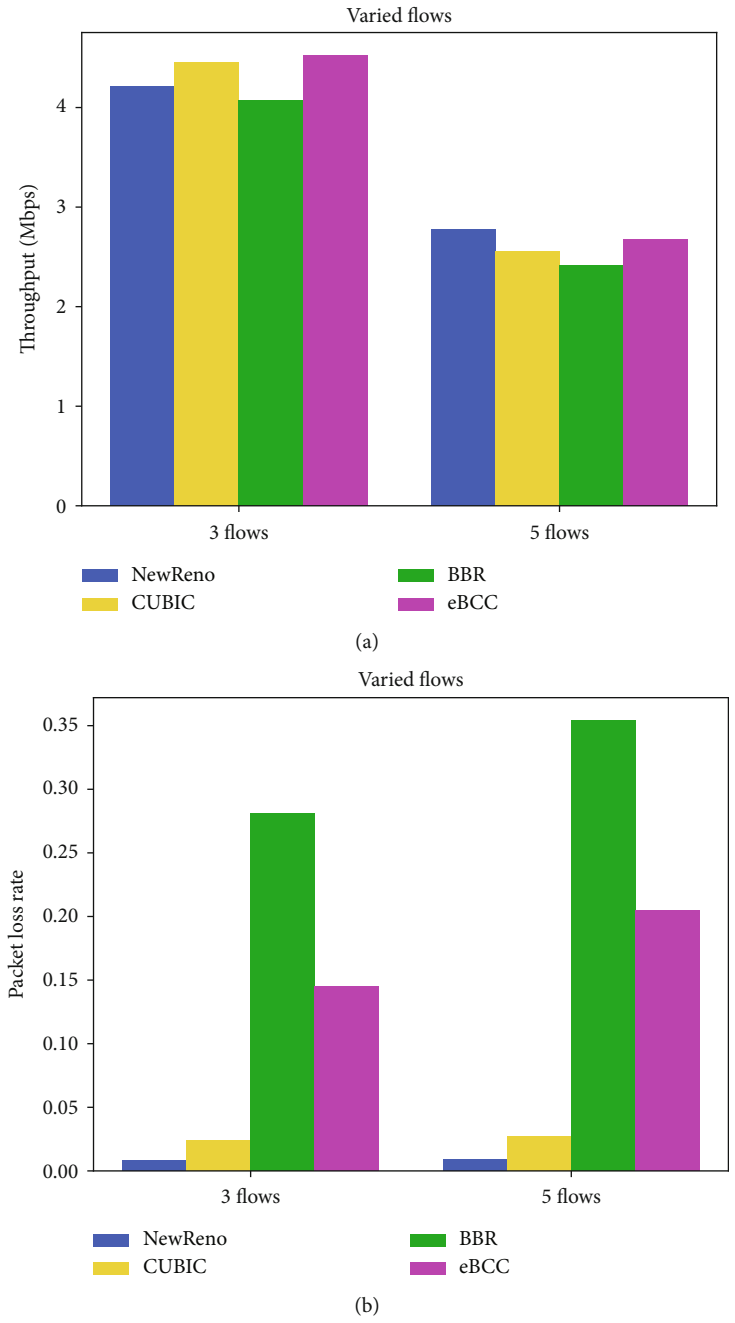

Figure 8: Multi-stream transmission experimental test results. (a) Throughput. (b) Packet loss rate.

are adopted independently to study their performance. The link bandwidth of transmission and bottleneck link is set to 50 Mbps and 15 Mbps, respectively. The experiment is repeated to conduct 3 and 5 different numbers of flows in the network transmission to be compared against a single-stream scenario in Experiments 1 and 2. The experimental results are also the average value obtained through repeated experiments. The fairness index in the multi-stream experiment is calculated by Equation (1).

Table 6, Table 7, and Table 8 show the average sending rate, average packet loss rate, and average fairness index of NewReno, CUBIC, BBR, and eBCC congestion control algorithms. When the packet loss rate is always low, NewReno has good throughput performance in 5 streams. CUBIC has a higher packet transmission rate and throughput in the 3 streams case, but the fairness is the second-lowest. In the case of the 5 streams case, its packet sending rate and throughput are relatively low, but the fairness performance is better than that in the case of 3 streams. The packet sending rate of BBR and eBCC are both very high, but the high packet loss rate damages BBR's throughput significantly. eBCC achieves higher throughput performance in all cases including the single-streaming case compared with BBR by reducing the packet loss rate. While NewReno has the

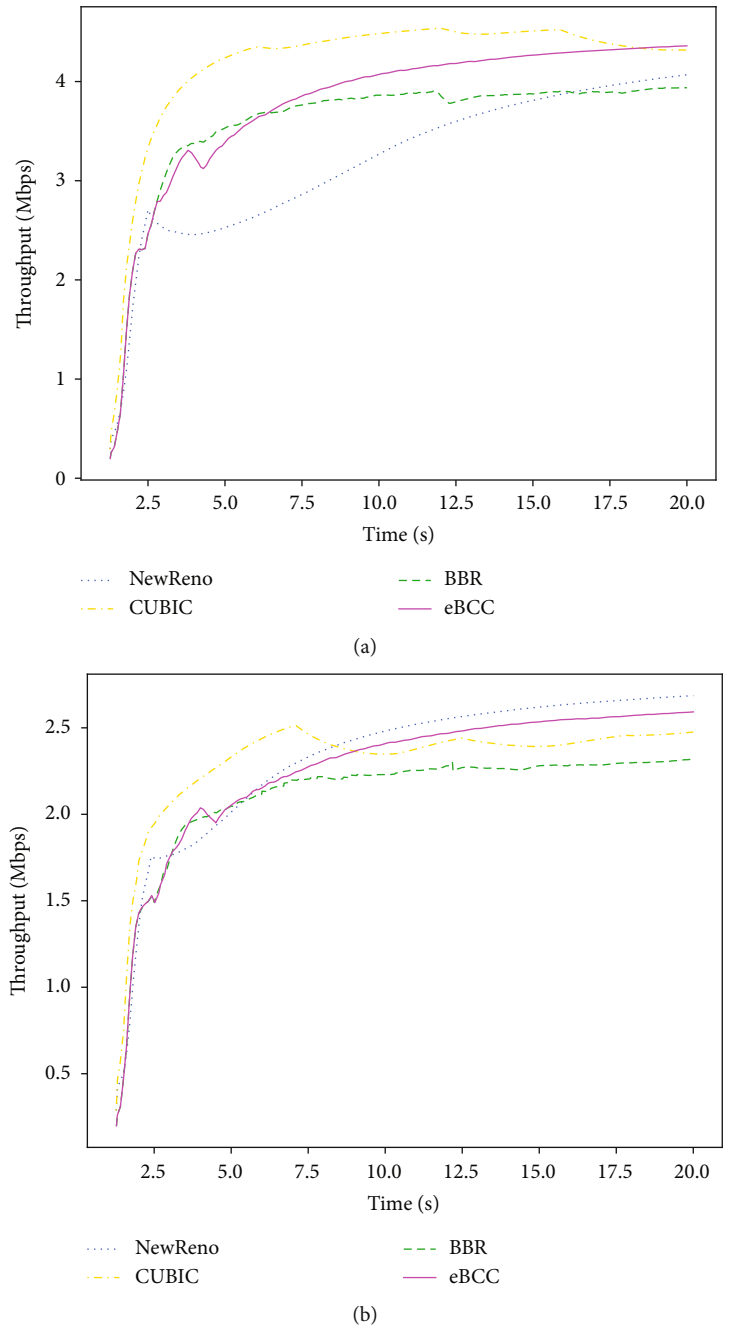

Figure 9: The real-time change of the average throughput in the multi-stream transmission experiment. (a) Three streams case. (b) Five streams case.

highest fairness in both multi-streaming cases, the fairness of BBR is the lowest as a consequence of its radical transmission strategy.

Figure 8 demonstrates a visual performance comparison of these four congestion control algorithms. In multi-stream transmission, the throughput of BBR has always been the lowest. The packet loss rate is the highest among all algorithms. NewReno has always been the holder of the lowest packet loss rate (around 1%). In Figure 8(a), eBCC has the best throughput in the case of 3 streams, and its Jain fairness index is only smaller than NewReno. In Figure 8(b), in the case of 5 streams, NewReno has the highest throughput and the lowest packet loss rate. The throughput of eBCC is only lower than NewReno and higher than CUBIC and BBR. The reason that eBCC's throughput is not as good as NewReno's is due to the high packet loss rate of eBCC. In this case, eBCC's sending rate has not been greatly improved, which leads to a slight gap in the final throughput of eBCC compared to NewReno.

From Figure 9, although the throughput curves of NewReno, BBR, and eBCC have twists and turns, the overall trend is a gradual increase. After the throughput rises to a

certain value, CUBIC starts to decrease due to its congestion control mechanism. Through the comparison of the average throughput curves of 1, 3, and 5 streams, the average throughput curve of a larger number of streams is more aggregated, and the throughput curve of a single stream is more scattered. The throughput curve of NewReno has a smaller decline and a greater rise as the number of streams increases. The throughput of BBR only fluctuates around the value after it reaches a certain value, which is consistent with its control method for obtaining BDP as the sending rate. The throughput curves of eBCC are similar varying from single stream to 3 and 5 streams. In a 5-stream scenario, 5 streams are competing in the link, and aggressive BBR flows can easily cause packet loss, which greatly increases the number of lost packets in the link and ultimately leads to a decrease in throughput performance. Compared with the BBR algorithm, the eBCC algorithm reduces packet loss and improves the throughput performance. The NewReno algorithm does not have a high utilization rate of the link, so it is easy to lead to waste of bandwidth resources. However, the simultaneous transmission of 5 streams makes up for this shortcoming. Therefore, the NewReno algorithm achieves the best throughput performance. The CUBIC algorithm is more aggressive than the NewReno algorithm, and it is easier to lose packets in the competition, resulting in performance loss. Its throughput degrades in the presence of massive packet loss due to stream contention.

Base on the analysis of the three experiments, we noticed that the eBCC algorithm improves the BBR algorithm in two aspects: (1) 10.87% higher throughput in experiment 1 and 74.58% lower packet loss rate in experiment 2 in the low-bandwidth scenario and (2) 8.39% higher fairness in experiment 3 in the multi-stream scenario. In comparison to NewReno and Cubic algorithms, eBCC outperforms both in throughput in most cases while maintaining a high level of fairness. The eBCC algorithm provides better quality for transmission in low bandwidth scenarios, which is applicable IoT communications.

5. Conclusions

In this paper, we proposed the eBCC algorithm under the QUIC protocol. Several experiments were performed to compare its performance against BBR, CUBIC, and NewReno algorithms in multiple network scenarios. In the low-bandwidth link scenario, through changing the bandwidth and delay on both transmission link and bottleneck link, we measured the transmission throughput and packet loss rate of four algorithms. In the simulation, the transmission throughput, packet loss rate, and fairness of the four algorithms were measured. The simulation results show that in the experiment of changing the bandwidth and the delay of the low-bandwidth network link scenario, eBCC achieves the best transmission throughput among the four algorithms. Although its packet loss rate is slightly higher than CUBIC and NewReno, but significantly lower than that of BBR. In the 3-stream scenario under the multi-stream transmission simulation, eBCC also gains the highest throughput, and second-highest fairness, only slightly lower than that of NewReno. In the 5-stream scenario, the throughput performance of the four algorithms is not much different. eBCC is slightly worse than NewReno in throughput performance and still performs better than BBR in fairness, throughput, and packet loss. In view of the above advantages, eBCC can obtain better QoS quality in the transmission of low-bandwidth links in IoT communications.

The experiment of the high bandwidth link will be carried out in the future. At the same time, our multi-stream experiment in this article is only an experiment between the same congestion control algorithm and the same RTT flow, and it does not compare the network performance between different congestion control algorithm flows and different RTT flows. Moreover, the experiment in this article is carried out in a wired dumbbell network simulated by ns-3 and has not been tested in a wireless network, a wide area network, or a cellular network. It is expected that these parts of the experiment will be completed in our following work.

Data Availability

The simulation data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by a grant from the National Natural Science Foundation of China (Grant No. 61801341). This work was also supported by the Research Project of Wuhan University of Technology Chongqing Research Institute and the Science Foundation Ireland (SFI) Industry Fellowship Programme under Grant Number 19/IFA/7445(T).

References

[1] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[2] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *Technical Report*, 2004.

[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[4] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control: measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[6] A. Langley, A. Riddoch, A. Wilk et al., "The QUIC transport protocol: design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 183–196, CA, Los Angeles, USA, 2017.

[7] S. Mukesh and B. S. Rajput, *Security and Performance Evaluations of QUIC Protocol*, Data Science and Intelligent Applications. Springer, Singapore, 2021.

[8] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1–9, Zurich, Switzerland, 2018.

[9] J. Iyengar and M. Thomson, "QUIC: a UDP-based multiplexed and secure transport," *Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-27*, 2020.

[10] S. Kaur and J. Singh, "Implementation of server load balancing in software defined networking," in *Information Systems Design and Intelligent Applications*, pp. 147–157, Springer, New Delhi, 2016.

[11] M. I. Hamed, B. M. ElHalawany, M. M. Fouda, and A. S. T. Eldien, "A novel approach for resource utilization and management in SDN," in *2017 13th International Computer Engineering Conference (ICENCO)*, pp. 337–342, Cairo, Egypt, 2017.

[12] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," *IEEE Conference on Computer Communications (INFOCOM)*, vol. 4, 2004.

[13] P. K. Kharat, A. Rege, A. Goel, and M. Kulkarni, "QUIC protocol performance in wireless networks," in *IEEE International Conference on Communication and Signal Processing (ICCSP)*, pp. 0472–0476, 2018.

[14] R. Corbel, S. Tuffin, A. Gravey, A. Braud, and X. Marjou, "Impact of QUIC on fairness in mobile networks," in *IEEE 10th International Conference on Networks of the Future (NoF)*, pp. 82–89, 2019.

[15] A. De Biasio, F. Chiariotti, M. Polese, A. Zanella, and M. Zorzi, "a QUIC Implementation for ns-3," *Proceedings of the 2019 Workshop on ns-3*, 2019.

[16] Y. Zhang, L. Cui, and F. P. Tso, "Modest BBR: enabling better fairness for BBR congestion control," in *IEEE Symposium on Computers and Communications (ISCC)*, pp. 00646–00651, Natal, Brazil, 2018.

[17] G. H. Kim and Y. Z. Cho, "Delay-aware BBR congestion control algorithm for RTT fairness improvement," *IEEE Access*, vol. 8, pp. 4099–4109, 2020.

[18] M. Jia, W. Sun, Z. Wang, Y. Yaohua, Q. Hongyu, and M. Kelong, "MFBBR: an optimized fairness-aware TCP-BBR algorithm in wired-cum-wireless network," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 171–176, Toronto, ON, Canada, 2020.

[19] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP Westwood: bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001.

[20] Y. J. Song, G. H. Kim, and Y. Z. Cho, "BBR-CWS: improving the inter-protocol fairness of BBR," *Electronics*, vol. 9, no. 5, p. 862, 2020.

[21] P. Qian, N. Wang, and R. Tafazolli, "Achieving robust mobile web content delivery performance based on multiple coordinated QUIC connections," *IEEE Access*, vol. 6, pp. 11313–11328, 2018.

[22] Y. Wang, K. Zhao, W. Li, J. Fraire, Z. Sun, and Y. Fang, "Performance evaluation of QUIC with BBR in satellite internet," in *6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 195–199, Huntsville, AL, USA, 2018.

[23] U. Paro, F. Chiariotti, A. A. Deshpande, M. Polese, A. Zanella, and M. Zorzi, "Extending the ns-3 QUIC module," in *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 19–26, Alicante, Spain, November 2020.

[24] H. Haile, K. J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "WIP: leveraging QUIC for a receiver-driven BBR for cellular networks," in *22nd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 252–255, Pisa, Italy, 2021.

[25] K. R. Jain, W. M. D. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, 1984.

[26] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10, Toronto, ON, Canada, 2017.

[27] G. Kim, I. Mahmud, and Y. Cho, "Fairness improvement of BBR congestion control algorithm for different RTT flows," in *International Conference on Electronics, Information, and Communication (ICEIC*, pp. 1-2, Auckland, New Zealand, 2019.

[28] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *IEEE Conference on Computer Communications (INFOCOM)*, 2006.

[29] X. Wu, M. C. Chan, A. L. Ananda, and C. Ganjihal, "Sync-TCP: a new approach to high speed congestion control," in *17th IEEE International Conference on Network Protocols*, pp. 181–192, 2009.

[30] T. Bi and G. Muntean, "Location-aware network selection mechanism in heterogeneous wireless networks," in *IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, Atlanta, GA, USA, May 2017.