Research Article

Improvement of BBRv2 Congestion Control Algorithm Based on Flow-aware ECN

Wansu Pan,¹,² Haibo Tan,¹ Xiaofeng Li,¹,² Jinlin Xu,¹ and Xiru Li¹

¹Hefei Institute of Physical Science, Chinese Academy of Sciences, Hefei 230031, China
²University of Science and Technology of China, Hefei 230026, China

Correspondence should be addressed to Xiaofeng Li; xfli@hfcas.ac.cn

Received 8 August 2021; Revised 13 April 2022; Accepted 18 April 2022; Published 7 May 2022

Academic Editor: Shah Nazir

Copyright © 2022 Wansu Pan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Google proposed a new congestion control algorithm (CCA) based on bottleneck bandwidth and round-trip propagation time (BBR), which is considered to open a new era of congestion control. BBR creates a network path model by measuring the available bottleneck bandwidth and the minimum round-trip time (RTT) to maximize delivery rate and minimize latency. The BBR v2 algorithm is a recently updated version by Google, which aims to improve some of the problems in the original BBR (BBRv1) algorithm, such as interprotocol fairness issues, RTT fairness issues, and excessive retransmissions. The BBRv2 evaluation results show that it can improve the coexistence with the loss_based algorithm and alleviate some of the shortcomings in BBRv1. However, when multiple BBRv2 flows enter the same link at different times, fair convergence cannot be achieved, and RTT fairness still exists. Based on these problems, we analyze the root cause and proposed an improved algorithm BBRv2+, which uses flow-aware explicit congestion notification (ECN) to quantify queue information and feedback on the accurate congestion degree. BBRv2+ algorithm can avoid blind window constraints and selectively mark packets so that different flows can converge to fairness. In the simulation experiment of Network Simulator 3 (NS3), the results show that the BBRv2+ algorithm can improve intraprotocol fairness and RTT fairness and ensure bandwidth utilization and interprotocol fairness.

1. Introduction

With the intensification of the contradiction between user demand and network resources, the importance of congestion control in transmission control protocol (TCP) becomes more and more important. TCP congestion control can effectively improve the management of network resources and the quality of network service. Traditional TCP congestion control algorithms (CCA) are mainly divided into loss_based CCAs and delay_based CCAs [1, 2], such as Reno [3], CUBIC [4], and Vegas [5], but these two kinds of CCAs have some design defects [6–8].

In 2016, Google proposed a hybrid CCA based on bottleneck bandwidth and round-trip propagation time (BBR) [9–11]. BBR calculates the bandwidth delay product (BDP) by measuring the maximum delivery rate (Btlbw) and minimum round-trip propagation time (RTprop) to achieve high throughput and low latency. The BBR implementation released by Google shows that BBR can significantly improve throughput and bandwidth utilization compared to CUBIC. Due to its excellent performance, BBR has gained much attention ever since its release. Mathis et al. [7] claimed that BBR opens a new era in congestion control and obsolesces Jacobson88. However, some evaluations [12–18] found that there are some problems in BBR, such as the unfairness when BBR and Reno/CUBIC share bottlenecks, the intraprotocol fairness, and RTT fairness. For the problem of the BBR algorithm, researchers are also making continuous efforts to obtain better performance by modifying some control parameters of BBR [19–24]. Google is also constantly optimizing the BBR algorithm and launched the second version of BBR in 2018, called BBRv2 [25]. BBRv2 introduces packet loss rate and explicit congestion notification (ECN) [26] marking as congestion signals to adjust congestion window (CWND) and pacing rate. Some evaluations [27–29] show that BBRv2 can coexist better with Reno/
CUBIC and further reduce retransmission. However, the RTT fairness in BBRv2 still exists, and the flow with different start times cannot converge fairly.

Obviously, more experiments are needed to ensure that the BBR algorithm has no potential pitfalls and further improve its network practicability. We further analyzed intraprotocol fairness and RTT fairness in the BBRv2 algorithm proposed in the literature and propose an improved method based on flow-aware ECN, called BBRv2+. BBRv2+ perceives accurate congestion degree and queue information through ECN, selectively marks ECN, avoids blind CWND adjustment, and can effectively improve the fairness of the algorithm. With the advent of programmable switches [30–33], network owners can choose to carry new (standard or proprietary) information in packet headers. Queue information can be easily obtained from the switch, and there is no technical difficulty in placing queue occupancy information in the data packet header [34]. It is feasible to combine the queue information in the switch with the congestion control algorithm. The main contributions of this paper are as follows:

1. Based on the analysis results of BBRv2, a flow-aware ECN for quantifying queue information and congestion degree is proposed. BBRv2+ algorithm can selectively judge packet loss and ECN marking, avoid blind CWND restriction, and balance the sending rate between flows.

2. We comprehensively evaluate the BBRv2+ algorithm from the aspects of CWND, link utilization, intraprotocol fairness, RTT fairness, and interprotocol fairness. The results show that the optimization algorithm can effectively alleviate the fairness problem of BBRv2 without sacrificing the advantages of BBRv2.

3. The evaluation results between algorithms can enable engineers and technicians to select appropriate parameter configurations in the application of the BBR algorithm and provide a reference for the development of the BBR version.

The rest of this article is arranged as follows. Section 2 briefly introduces the related research on BBRv2, and Section 3 elaborates on the specific information of the BBR algorithm and deeply analyzes the causes of the fairness problem of the BBR algorithm. The theoretical model and derivation of the optimization algorithm are in Section 4. Section 5 is simulation results and evaluation. Conclusions will be held in Section 6.

2. Related Work

After the BBRv1 algorithm was released by Google, there have been a lot of evaluation studies and improvement work on it. Hock et al. [12] first evaluated the performance of BBR on high-speed bottleneck links, including RTT fairness, delay, packet loss rate, and coexistence fairness with CUBIC. Scholz et al. [14] analyzed the behavior and performance of BBR, analyzed the advantages of the BBR algorithm, and proposed a publicly available framework for repeatable TCP measurement based on network simulation. Moreover, some researchers have further analyzed the defects of BBR and put forward some optimization algorithms.

Google is also constantly optimizing the BBR algorithm and updating the BBRv2 version to improve problems reported in BBRv1’s studies. At IETF-102 [25] and IETF-104 [35], BBRv2 changed the method of adjusting the bandwidth probe cycles to improve the interprotocol fairness. Google revealed the preview version (called BBRv2 alpha) [36] of the open-source code in IETF 105 [37], encouraging researchers to dig deeper to help evaluate and improve BBRv2. This version improves the coexistence with loss-based CCAs and reduces the loss rate of bottleneck buffer to less than 1.5BDP. At IETF-109 [38], Google proposed a new BBR.Swift algorithm based on the method of using delay as a congestion signal [39]. BBR.Swift can achieve higher fairness and lower retransmission rate when flows using the same CCA.

Since the updated BBRv2 algorithm is not the final version, the performance of BBRv2 is only evaluated on the simulation platform. Zhang [27] discussed BBRv1, BBRv1 variant, and BBRv2 and pointed out that BBRv2 has improved RTT fairness compared with BBRv1 and has better coexistence with CUBIC and Reno. However, the channel utilization of BBRv2 is low when the random loss rate is 5%. Gomez [40] et al. used Mininet simulation to conduct an experimental evaluation on BBRv2, and the results showed that BBRv2 improved coexistence with CUBIC and alleviated RTT fairness in BBRv1. However, some studies show that there are still some problems to be optimized in BBRv2.

Kfoury et al. [28] pointed out that BBRv2 could not quickly detect the available bandwidth in the network environment with unstable bandwidth, resulting in low link utilization. Song et al. [41] pointed out that two BBRv2 flows entering the same bottleneck link at different times cannot achieve fair bandwidth sharing in deep buffers. The first start flow will occupy more bandwidth, which will lead to the subsequent start flow being limited. Nandagiri et al. [42] found that the unfairness between BBRv2 flows with different RTTs is alleviated, and the packet retransmission times are greatly reduced. However, when the buffer size is large enough, long RTT flows still consume more bandwidth than short RTT flows.

With the continuous updating of the BBR algorithm, BBRv2 can solve some fairness problems and limitations of BBRv1. But, BBRv2 needs more research to improve the RTT fairness and intraprotocol fairness, so as to provide more possibilities for the final version of BBRv2 and even TCP congestion control.

3. Detail on Algorithms

3.1. BBRv2 Behavior Analysis. BBR measures the maximum delivery rate and minimum transmission delay alternately to find Kleinrock’s optimal operating point [43]. BBR controls congestion by limiting the pacing rate of packets, limiting inflight to one BDP, as calculated in
BBR adjusts the speed of output packets at the latest estimated delivery rate. At the same time, BBR maintains a CWND in order to maintain consistent throughput in delayed or aggregated ack networks. BBR sets the maximum bandwidth (delivery rate) of the last 10 round trips as Btlbw and regards the minimum delay measured in the past 10 seconds as RTprop through the scaling factor cwnd_gain and pacing_gain to adjust CWND and pacing rate. The BBRv1 algorithm has four phases, as shown in Figure 1. BBRv2 is optimized on the basis of the original BBR v1, and its ProbeBW phase is further divided into four substages.

In the StartUP phase, pacing rate and CWND will increase by setting cwnd_gain and pacing_gain as 2/ln2 (about 2.89). The exponential growth of pacing rate and CWND will lead to queue accumulation on routers. If the newly estimated bandwidth of three consecutive RTTs does not increase by at least 25%, the BBR enters the Drain phase. On the basis of BBRv1, BBRv2 adds two additional conditions: packet loss or ECN mark rate. If the packet loss rate or ECN mark rate exceeds their respective thresholds, inflight_hi is set to an estimate of the maximum inflight. ECN mark rate is calculated by equation (2). When the continuous bandwidth detector reaches a stable value or when inflight_hi is set, the flow exits the StartUP phase.

\[
\alpha_{ecn} = (1 - g) \times \alpha_{ecn} + g \times F, \tag{2}
\]

where \( g \) is a weight factor and is 1/16 in DCTCP and \( F \) is the fraction of packets that are marked in the last window of data.

In the Drain phase, BBRv1 and BBRv2 are the same. BBR passes the pacing_gain which is reduced to ln2/2 (about 0.35) to clear the remaining queue in the previous stage, and cwnd_gain remains unchanged (2/ln2). At the end of this phase, inflight data < the estimated BDP.

In the ProbeBW phase, BBRv1 has 8 cycles in the detection bandwidth (pacing_gain [] = [1.25; 0.75; 1; 1; 1; 1; 1; 1]), and the duration of each pacing_gain was RTprop. In BBRv2, unlike the 8 cycles in BBRv1, it is divided into four cycles, as shown in Figure 2. The pacing_gain in the Up and Down phases is 1.25 and 0.75, respectively, and that in the Cruise and Refill phase is 1. The bandwidth detection time in BBRv2 is adaptive, which improves the fairness when coexisting with Reno and CUBIC.

In the Refill phase, the flow probes for additional bandwidth and inflight capacity to prepare for the Up phase. In the Up phase, if inflight_hi is fully utilized, it will increase the number of packets per round (1, 2, 4, 8, \ldots Packets) exponentially. If the loss rate or ECN mark rate is too high, inflight_hi will be reduced to the length of inflight packets. When inflight_hi or the estimated queue is large enough (inflight data greater than 1.25 times the estimated BDP), the flow process exits the Up phase. And then the flow process enters the Down phase to clear the recently created queue and leave unused headroom. CWND is set to the minimum between inflight_lo and inflight_hi, as shown in equation (3). When inflight data is lower than inflight_hi or inflight data is equal to or lower than the estimated BDP, the flow exits this phase. In the cruise phase, bw_lo, inflight_lo is updated at each RTT, and packet loss and ECN signals are used, so that the sending rate constantly adapts to the control queue level.

\[
inflight\text{ headroom} = \text{inflight}\_hi \times (1 - k\text{HeadRoom}), \tag{3}
\]

where \( k\text{HeadRoom} \) is a constant and the default value is 0.15.

For the ProbeRTT phase, if a new RTprop is not sampled again within 10 seconds, BBR v1 enters ProbeRTT, while that in BBRv2 is 5 seconds. In this phase, the CWND of BBRv1 is set to 4MSS and lasts for 200 ms. BBRv2 sets CWND to 50% of BDP to avoid throughput loss.

3.2. Causes Fairness. The evaluation results [40–42] of BBRv2 show that BBRv2 works well on bottleneck links with small buffers. Compared with BBRv1, BBRv2 not only improves the fairness with other TCP flows but also reduces packet loss. However, BBRv2 still faces some problems of fair convergence. On the one hand, there is an RTT fairness problem between different RTT flows in the protocol. On the other hand, when BBRv2 flows with the same RTT enter bottleneck links with large buffers at different times, there is also a fairness problem.

BBRv2 introduces packet loss rate and ECN marking rate as congestion signals. In the StartUP phase, if the packet loss rate or ECN marking rate exceeds the predefined threshold, the current estimated BDP is set to the upper limit inflight_hi. The flows with different start times enter the same bottleneck link, the RTprop measurement value will increase and the packet loss feedback (threshold 2%) will be triggered when the later start flows enter the link. Then, the flow sets the current estimated BDP as inflight_hi and moves on to the next phase. In the ProbeBW: Up phase, inflight_hi limits the increase of packets inflight due to the presence of the ECN marking rate and will exit the phase early to avoid congestion. Therefore, the second start flow cannot measure the higher bandwidth and always remains at the level of the

![Figure 1: The phases of BBRv1 and BBRv2.](image)
initial probe. If the buffer size is large enough to ensure that the second flow does not experience packet loss, inflight_hi will not be set. This mechanism makes the BBRv2 bandwidth probe sensitive to buffer size, resulting in flows starting at different times not being able to share bandwidth fairly [41].

Moreover, RTT fairness persists in BBRv2 because the transmission rate is periodically increased to detect available bandwidth. The bandwidth sample measured by each BBRv2 flow is larger than the actual available capacity, so the BBRv2 flow produces a standing queue length similar to BBRv1. The estimated BDP value of the long RTT flow is larger than that of the short RTT flow. The short RTT flow is first limited by CWND. Although the ECN feedback is added in BBRv2 to adjust the size of CWND through inflight_hi and inflight_lo, the queue information is not aware. The large proportion of persistent queues for long RTT flows causes long RTT flows to squeeze the bandwidth of short RTT flows.

From the above analysis, it can be seen that the fundamental reason why different flows cannot converge fairly is that the BBRv2 sender restricts CWND based on packet loss and ECN marking rate. The packet loss threshold can only reflect whether the network is congested (the packet loss caused by link errors is ignored here), but it cannot provide the specific degree of congestion of the network. The ECN marking uses a simple marking scheme. When the instantaneous queue length is greater than the preconfigured threshold value $K$, each arriving packet will be marked by ECN, each queue has its own threshold, and the ECN marking will be performed independently for other queues. This ECN marking approach is flow agnostic because it marks packets based on queue length, regardless of the state of the flows. Such feedback regulation behavior cannot cope with the complex and changeable network situation, so it is necessary to further improve the feedback mechanism in BBRv2.

4. The Proposed Algorithm: BBRv2+

4.1. Flow-Aware ECN. In order to solve the problem of intraprotocol fairness and RTT fairness of BBRv2 congestion control, the sender must fine-grained adjust the CWND and pacing rate according to the level of network congestion. This brings about two problems: (1) how the sender accurately knows the degree of network congestion; (2) how the sender reasonably chooses the magnitude of the multiplicative decrease (MD) according to the level of congestion.

Ideally, we want to quantify the size of the queue and sense congestion with each ACK signal. The sending rate can be adjusted reasonably by the queue length. When the queue is too long, the large MD (MD coefficient close to 0) can be used to greatly reduce the sending rate and accelerate the queue draining. When the queue is short, the small MD (MD coefficient close to 1) can be used to slightly reduce the sending rate and maintain high bandwidth utilization. Based on the above issues, on the basis of the BBRv2 algorithm, FAECN (flow-aware ECN) is introduced to replace the original ECN mechanism to give feedback on the congestion degree and queue size of the bottleneck link. Figure 3 shows the proposed architecture diagram for BBRv2+.

The level of congestion is quantified by the change of RTT [44, 45], and $\Delta$ represents the load factor of the bottleneck link, which is calculated by the ratio of the current delay of flow, over a period (5 s) to the maximum delay in the link, as shown in

$$\Delta_i = \frac{T_i}{T_{\max}} \quad (\Delta_i \in (0, 1]),$$

where $T_i$ is the current RTT obtained by flow, from the last ACK, and $T_{\max}$ is the maximum RTT in the entire link from each period. $\Delta = 1$ only if the bandwidth and buffer of the bottleneck link are fully utilized; otherwise, it is $< 1$. Algorithm 1 describes the statistical process of the load factor. The sender uses the feedback RTT values to retrieve the maximum and minimum RTT values and calculate the load factor.

The congestion degree of the link is divided into low load and full load states by load factor $\Delta$. In the StartUP phase, when the link load is low, even if the packet loss threshold exceeds 2%, the upper limit inflight_hi of CWND will not be set and continue to perform bandwidth detection. When the link is fully loaded, perform the operation in the original BBRv2.

In addition, a queue length threshold $K$ needs to be set, and when the queue length exceeds a lower threshold $K_{\text{min}}$, packets from relatively high sending rate flows are selectively marked. According to queuing theory, flows with higher sending rates occupy most of the queue buffers. These high-speed flows are slowed down to effectively maintain the
queue length below the given threshold, while the other unmarked flows keep increasing their sending rates to prevent the buffer underflows. In this way, the bandwidth gap between different flows is reduced. In addition, long RTT flows often have a higher sending rate than short RTT flows, when long and short RTT flows share the same queue, long RTT flows are more likely to be marked and slowed down to alleviate congestion, while short RTT flows can speed up transmission, thus alleviating RTT unfairness in the algorithm.

FAECN uses two ECN bits to notify the sender, who determines the level of congestion based on the ECN marker rate and load factor. In order to distinguish high-speed flows, the sending rate is stored in the Options of the IP packet (size 16 bits). The definitions of ECN [46] in the IP header are shown in Figure 4.

The switch reads the sending rate and compares the size of the average rate to which packet belongs to high-speed flow. When the queue length exceeds the given minimum threshold \( K_{\text{min}} \) and the rate value carried in the packet header is greater than the moving average, the switch marks the packets arriving with ECN. When a packet enters the queue, the switch updates the mean \( \text{ave}S \) with an estimate of the average sending rate of all the flows in the queue, as shown in

\[
\text{ave}S = y \times S + (1 - y) \times S.
\]  

\( y \) is the weight when updating the new value of aveS, and the value range is \( 0 < y < 1 \). In this paper, the value of \( y \) is 1/8. If \( S > \text{ave}S \), the packet will be marked by ECN as probability \( P \in [0, 1] \).

When the queue length changes from \( K_{\text{min}} \) to \( K_{\text{max}} \), the marking probability \( P \) changes linearly with the queue length.

The switch compares the changes in the current queue length. If the instantaneous queue length \( q \) is smaller than \( K_{\text{min}} \), ECN is not marked. If \( q > K_{\text{min}} \) and smaller than \( K_{\text{max}} \), ECN markings are selected according to the sending rate of packets. Algorithm 2 represents the process of packet ECN marking. Firstly, the queue threshold in each ACK is judged, and then the selective ECN marking is carried out in combination with the sending rate in the message.

The FAECN method can effectively quantify the network queue length. Selectively tagging packets based on queue length allows the network to precisely slow down high-speed flows without killing slow-speed flows. Thus, the problem that different flows cannot converge fairly in the BBRv2 algorithm is alleviated. In addition, FAECN simply uses large fields in the data header without adding too much overhead to the switch, making it easy to implement and deploy.

In the BBRv2+ algorithm, when congestion exceeds the threshold \( \Delta \), the inflight_hi is set based on the packet loss rate and ECN marking rate. The threshold of ECN marking rate is 50%, which is consistent with the threshold used in the original BBRv2. On the one hand, the congestion degree threshold avoids blind judgment of the packet loss threshold.

**Algorithm 1:** The statistical process of load factor.

(i) **Input:** rtt_us, \( S \) //Update each ACK in BBR
(ii) **Initialization:** \( T_{\text{max}} \rightarrow 0, T_{\text{min}} \rightarrow 0X7FFFFFFF 
(1) **For** every ACK **do**
(2) **if** BBR in StartUp phase **then**
(3) \( \text{rtt}_\text{us} \rightarrow \text{(now sending time)} \)
(4) **if** \( \text{rtt}_\text{us} < T_{\text{min}} \) **then**
(5) \( T_{\text{min}} \rightarrow \text{rtt}_\text{us} \)
(6) **end if**
(7) **if** \( \text{rtt}_\text{us} > T_{\text{max}} \) **then**
(8) \( T_{\text{max}} \rightarrow \text{rtt}_\text{us} \)
(9) **end if**
(10) \( \Delta = \text{rtt}_\text{us}/T_{\text{max}} //\text{Calculate load factor} \)
(11) **end if**
(12) **end for**

\[
\text{Figure 3: The architecture diagram of BBRv2+}.
\]
and alleviates the convergence problem among different flows. On the other hand, the algorithm selectively carries out ECN marking through the queue information to achieve the staggered arrangement of each flow rate. When high-speed flows and short flows share the same queues, the high-speed flows are more likely to be marked and slowed down to relieve congestion, and the low-speed flows can speed up and finish quickly. Therefore, compared with the original BBRv2, BBRv2+ can make each flow converge to fairness.

4.2. Algorithm Model Analysis. Establish a fluid model to simplify the BBR operation mechanism. Suppose that there are \( n \) flows with different RTTs passing through a bottleneck link with bandwidth \( C \), where let \( f_i \) (\( i \in [1; n] \)) represents flow \( i \) and \( d_i(t) \) represents the delivery rate (bottleneck bandwidth) at time \( t \). \( S_i(t) \) is the sending rate at time \( t \), calculated by

\[
S_i(t) = \text{pacing gain} \times \max(d_i(t)) \times (T \in [t - 10RTT; t]).
\]

\[
(6)
\]

\( R_i(t) \) A represents the round-trip time of flow \( i \) at the time \( t \), as shown in

\[
R_i(t) = \frac{q_i(t)}{C} + \text{RTprop}_i,
\]

where \( q_i(t)/C \) denotes the queuing delay. Let \( Q_i(t) \) denote the disparity between inflight (\( I_i(t) \)) and delivery capacity (\( D_i(t) \)), as shown in

\[
Q_i(t) = I_i(t) - D_i(t) = d_i(t) \times R_i(t) - d_i(t) \times RTprop_i = d_i(t) \times \frac{q_i(t)}{C}.
\]

\[
(7)
\]

It can be seen that if \( Q_i(t) > 0 \), the transmission capacity of flow \( i \) is exceeded. In order to reduce the burden of the bottleneck, the flow \( i \) should reduce the number of packets injected into the pipeline in the next period through the feedback mechanism. ECN is marked by comparing the send rate to the average send rate of all flows passing through the same exit. The calculation of the average rate \( \text{ave}S(t) \) is shown in

\[
\text{ave} S(t) = \frac{\sum_{i=1}^{N} S_i(t)}{N}.
\]

(9)

The probability of marking according to the queue length is shown in the equations (10) and (11):

\[
P_i(t) = \begin{cases} 0 & \text{if } q(t) < K_{\min} < S_i(t) < K_{\max} \text{ and } \text{ave} S(t), \\ f(q(t))K_{\min} \leq q(t) < K_{\max}, S_i(t) < \text{ave} S(t), \\ 1q(t) \geq K_{\max}, 
\end{cases}
\]

\[
f(q(t)) = P_{\min} + \frac{(q(t) - K_{\min})(P_{\min} - P_{\max})}{K_{\max} - K_{\min}},
\]

where \( q(t) \) is the instantaneous queue length at time \( t \), and \( K \) is the threshold of queue length. \( K_{\min} \) should be set to a low value to maintain low network latency, and \( K_{\max} \) should be set to a high value to maintain low queue oscillation to avoid queue buffer overflow.

Based on the standard ECN marking threshold derived from the single-queue model [45, 46], the only queue of the bottleneck link shared by synchronous flows with the same RTT is considered. In order to make full use of link bandwidth and realize low latency, the ECN marking threshold \( K \) is set as follows: \( K = C \times \text{RTT} \times \lambda \), where \( \lambda \) is a tunable parameter closely related to CCAs. In this configuration, any queue can independently make full use of the link capacity. But the problem is that when \( N \) queues are busy at the same time, the total buffer occupancy can easily reach \( N \) times the standard threshold, resulting in high queue delay and huge buffer pressure. Therefore, the selection of \( K \) should vary with the bandwidth and traffic nature. Generally, \( K_{\min} \) is set to a low value to maintain low network delay, while \( K_{\max} \) is set to a high value to maintain low queue oscillation, so as to avoid buffer idleness [47–50]. On the other hand, to prevent the impact of \( K \) from being underestimated, on the other hand, considering that the weighted fair share rate should not be greater than the link capacity, \( K \) can be constrained by

\[
K_i = \min \left( \frac{\text{quantum}}{T_{\text{round}}}, C \right) \times \text{RTT} \times \lambda,
\]

(12)
where quantum represents the maximum number of bits that can be sent in each round of queue, and \( \lambda \) is set to 0.17. \( T_{\text{round}} \) represents the completion time of each round, which is calculated by the weighted moving average of

\[
T_{\text{round}} = g_t \times T_{\text{round}} + (1 - g_t) \times T_{\text{sample}},
\]

where \( g_t \) is a parameter in \((0, 1)\), which represents the forgetting speed of the historical value by \( T_{\text{round}} \). Here, \( g_t \) is taken as 1/3 to prevent violent fluctuations when the estimated value of \( K \) is too small. We assume that each queue maintains a variable \( T_{\text{pre}} \) to store the timestamp when the queue completes the service in the previous round. Each time the queue completes its service, it records the current timestamp \( T_{\text{now}} \) and calculates a round of time samples \( T_{\text{sample}} \) as follows: \( T_{\text{sample}} = T_{\text{now}} - T_{\text{pre}} \). Then, reset \( T_{\text{pre}} \) with \( T_{\text{now}} \). FAECN maintains the same scale and implementation complexity as ECN, and only one additional registration is required for each port to store \( T_{\text{round}} \).

For marked probability \( P \), \( P_{\text{min}} \), and \( P_{\text{max}} \) should be set to a larger value when concurrent traffic is large; otherwise, \( P_{\text{min}} \) and \( P_{\text{max}} \) should be set to a smaller value. We conducted a simulation experiment, and the distribution between queue length and \( P_{\text{min}} \) and \( P_{\text{max}} \) is shown in Figure 5.

As seen from Figure 5, a larger \( P_{\text{min}} \) can maintain a smaller queue length, which will lead to instability of queue range. The average queue length increases significantly with the increase in the number of concurrent streams. Therefore, we set \( P_{\text{min}} = 0.5 \) and \( P_{\text{max}} = 1 \) in the simulation experiment. However, the actual network load is variable, and the number of concurrent streams may vary with time and space. The fixed threshold \( K \) and \( P \) cannot satisfy all cases, and further verification and optimization are needed after that.

Moreover, regarding the adjustment of CWND, from equation (4), we can know that when \( T_i \) moves to \( T_{\text{min}} \), \( \Delta \) increases to the lowest possible value \( \Delta_{\text{min}} \) which means that the link load is light, as shown in

\[
\Delta_{\text{min}} = \lim_{T_i \to T_{\text{min}}} \frac{T_i}{T_{\text{max}}} = \frac{T_{\text{min}}}{T_{\text{max}}},
\]

Conversely, if \( T_i \) moves in the \( T_{\text{max}} \) direction, it indicates that \( \Delta \) increases to the maximum possible value \( \Delta_{\text{max}} \) which indicates that the network load is busy, as shown in

\[
\Delta_{\text{max}} = \lim_{T_i \to T_{\text{max}}} \frac{T_i}{T_{\text{max}}} = \frac{T_{\text{min}}}{T_{\text{max}}},
\]

The judgment of congestion degree depends on \( \Delta \), considering that the link-state threshold should be the connection point between the idle state and the full state of the network. The threshold of \( \Delta \) should be close to 1 so that the sender can use the idle bandwidth faster, but it also needs to balance the link convergence time. We set up a 100 Mbps bottleneck link and five flows with 80 ms RTT for the simulation experiment. Figure 6 shows the influence of \( \Delta \) on link utilization and convergence time. The fairness convergence time is normalized by the minimum fairness convergence time in the simulation. It can be seen from the simulation results that both link utilization rate and fair convergence time increase with the increase of \( \Delta \). The difference is that the growth rate of link utilization slows down after \( \Delta > 0.7 \), while the growth rate of fairness convergence time increases sharply after \( \Delta > 0.8 \). To balance these two parameters, we set the threshold to 0.8.

5. Results and Discussion

Based on the implementation of the BBR algorithm framework [36, 51, 52], a large number of simulation experiments are carried out on the NS3 platform to compare the performance of BBRv1, BBRv2, and BBRv2+. This section describes the results of running tests under different network conditions, where the condition variables include start time, RTT, and buffer size. As shown in Figure 7, an experimental environment is constructed.

5.1. CWND Evolution. The evolution of CWND directly influences other performance metrics, such as throughput, bandwidth utilization, and sharing fairness. In the BBRv2+ algorithm, we judge the link state by optimizing the ECN marking mode and adding the load factor, so as to adjust the CWND limit in the original BBRv2 algorithm. Through NS3 simulation experiments, we verify the CWND size of multiple flows with different packet loss rates (buffer size is 1BDP). Figure 8 shows the comparison of three algorithms BBRv1, BBRv2, and BBRv2+ in terms of CWND evolution, which can be divided into the aggregated CWND of each algorithm when the packet loss rate is 0 and 1. It can be seen from Figure 8 that the CWND regulation frequency of BBRv2+ is more frequent, indicating that BBRv2+ adjusts the size of CWND timely through the fine-grained perception of flow status. For Figure 8(a), although BBRv1 reaches the maximum CWND earlier than BBRv2 and
BBRv2+, the fluctuation range of CWND after stabilization is significantly larger than BBRv2 and BBRv2+. When BBRv2+ reaches dynamic stability, its CWND is slightly lower than that of BBRv1 but higher than that of BBRv2. In Figure 8(b), the CWND of the three algorithms is reduced, and BBRv2+ shows low sensitivity to packet loss, which is only reduced by about 5%, and CWND is the most stable.

5.2. Link Utilization. A single-flow scenario was created based on NS3 to evaluate the link utilization of the improved algorithm BBRv2+. The three algorithms were tested for link utilization on links with random packet loss, buffer configurations ranging from 0.1 to 20 BDP, and random packet loss rates of 0% and 1% respectively. The bandwidth utilization of all flows is calculated according to
where $\text{bytes}$ is the length of all received packets for flow $i$, $\text{Cap}$ is the bandwidth of the bottleneck link, and $\text{duration}$ is the continuous simulation running time.

The experimental results of link utilization of BBRv1, BBRv2, and BBRv2+ are shown in Figure 9. In Figure 9(a), the link utilization of BBRv2+ is lower than that of BBRv1 but higher than that of BBRv2, and the difference between BBRv2+ and BBRv1 is only about 0.25%. This is because the detection rate in BBRv2 decreases, sacrificing part of the bandwidth. BBRv2+ algorithm ensures the rate of low-speed flow by selective ECN marking, so the link utilization does not decrease. When the packet loss rate is 1% in Figure 9(b), the link utilization of the three algorithms decreases, and the link utilization of BBRv2 is the lowest. In BBRv2, the upper limit of CWND is set by packet loss feedback, while the BBRv2+ algorithm adjusts the threshold judgment of node packet loss by the congestion degree, which avoids blind CWND limit and maintains the original link utilization. The experimental results show that BBRv2+ has a certain antipacket loss ability without sacrificing link utilization.

5.3. Multiple Flows with Different Start Times and Buffer Sizes. In the previous analysis, we know that the flows with different start times in the BBRv2 algorithm cannot converge fairly. Therefore, we test the throughput of flows starting at 0s or 5s to analyze the effectiveness of the BBRv2+ algorithm. The bottleneck bandwidth of each test is 100 Mbps, and the start time of flow1 and flow2 is 0s and 5s, respectively. Each experiment consists of flows running the BBRv1, BBRv2, or BBRv2+ algorithm.

In a 0.5 BDP buffer, the throughput of the three algorithms is shown in Figures 10(a) and 10(b). For the BBRv1 algorithm, the throughput of flow 1 and flow 2 is shown in Figure 10(a). Flow 1 can quickly obtain bandwidth in the StartUP phase. After flow 2 is added, flow 1 will have a short-term throughput decline and then remain stable. The throughput difference between flow 1 and flow 2 is about 2.2 Mbps. In Figure 10(b), the throughput difference between BBRv2 increases to approximately 6.7 Mbps. For the BBRv2+ algorithm, the throughput trends of flow 1 and flow 2 are similar to the above two algorithms. Although the throughput difference between flow 1 and flow 2 is larger than that in BBRv1, it is smaller than that in BBRv2. The throughput difference between flow 1 and flow 2 is reduced by 32% compared with BBRv2.

In the 5 BDP buffer, the throughput of the three algorithm flows is shown in Figures 10(c) and 10(d). Compared with the results in 0.5 BDP buffer, the throughput difference between flow 1 and flow 2 increases. In Figure 10(c), the throughput difference between flow 1 and flow 2 in the BBR algorithm is larger than that in 0.5 BDP, but the throughput difference is small. The throughput of flow 1 and flow 2 can still maintain good fairness. In Figure 10(d), the throughput difference between BBRv2 is significantly increased compared with Figure 10(b), and there is obvious bandwidth unfairness. The throughput difference is about 1.7 times. Compared with BBRv2, the throughput fluctuation of BBRv2+ is relatively large, but the throughput difference is reduced to 1.3 times. Overall, BBRv2+ improves the unfairness of throughput in BBRv2.

According to the situation considered, a multiflow scenario is established continuously to evaluate the performance of three algorithms on congestion bottleneck to simulate real network scenarios. In order to further verify the effectiveness of BBRv2+, we divided 100 flows into 5 groups and set the start time of each flow as 0 s, 2 s, 5 s, 10 s, and 20 s in turn. Other conditions remain unchanged, repeated experiments are carried out to calculate the average throughput of each flow. The experimental results are shown in Figure 11.

In a 0.5 BDP buffer, the average throughput of the three algorithms is shown in Figures 11(a), 11(b), and 11(c). For the BBRv1 algorithm, the average throughput of five flows is shown in Figure 11(a). The flow starting at 0 s can quickly obtain bandwidth at the start, but with the addition of other flows, the bandwidth of flows starting at 0 s gradually decreases. Each flow starts with a peak in bandwidth and then decreases. Finally, the five flows gradually share bandwidth after 60 s, converging to fairness. In Figure 11(b), the flow of BBRv2 starting at 0 s can obtain the most bandwidth, and then after the flow at different times joins the link, each flow competes for average throughput. The average throughput of flow starting at 0 s is about 38 Mbps, which has a great advantage. The average throughput of flow started in 2 s and 5 s is 20 Mbps, while the bandwidth of flow started in 10 s and 20 s is only about 9 Mbps. In the BBRv2+ algorithm in Figure 11(c), the average throughput trend of the five flows is similar to BBRv1, which can achieve better convergence than BBRv2. After the bandwidth competition is stable, the average throughput of each flow is about 17–21 Mbps, and the throughput difference is reduced by 80% compared with BBRv2.

In the 5 BDP buffer, the average throughput of the two algorithm flows is shown in Figures 11(d), 11(e), and 11(f). In Figure 11(d), compared with Figure 11(a), the bandwidth difference of the five BBRv1 flows is increased, but the bandwidth difference is very small, and it can still converge well. For the BBRv2 algorithm, the throughput difference of five flows becomes larger as shown in Figure 11(e), and there is obvious bandwidth unfairness. The average throughput of flow starting at 0 s is about 40 Mbps, that of flow starting at 2 s and 5 s is 19 Mbps and 15 Mbps, respectively, and that of flow starting at 10 s and 50 s is only 9 Mbps and 7 Mbps. In Figure 11(f), the bandwidth fairness of the five flows in the BBRv2+ algorithm is reduced compared with 0.5 BDP. However, compared to the BBRv2 algorithm, five flows maintained a relatively fair average throughput, eventually stabilizing at 17–22 Mbps. Although the fairness is not as good as BBRv1, it is greatly improved compared with the BBRv2 algorithm.

We find that the fairness convergence of the BBR algorithm varies with the buffer size. In order to evaluate the difference in the fair convergence problem of the BBR algorithm in different buffer sizes, we conducted a lot of tests. In this paper, the
Figure 9: Comparison of link utilization of three algorithms under different buffer sizes. (a) No random packet loss. (b) Random packet loss rate of 1%.

Figure 10: Throughput of two flows which has different start times. (a) BBRv1 in 0.5BDP buffer. (b) BBRv2 and BBRv2+ in 0.5BDP buffer. (c) BBRv1 in 5BDP buffer. (d) BBRv2 and BBRv2+ in 5BDP buffer.
A fairness index is introduced. As described in the literature [53], Jain’s fairness index is used to measure equity. Equation (17) shows a method to calculate Jain’s fairness index.

\[
J = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}, \tag{17}
\]

where \(x\) is the throughput of the flow. For example, in the case of 100 simultaneous flows, \(n = 100\) and \(i = 1, 2, \ldots, 100\). The closer Jain’s fairness index is to 1, the better the fairness of bandwidth allocation. Jain’s fairness index can well reflect the throughput difference.

We repeated the experiment 10 times, calculating the average throughput of each flow in each test duration (200 s). The average throughput of each group is obtained by averaging the throughput of 50 flows corresponding to the same start time. Each group consists of 10 samples, which avoids accidental experimental errors to a certain extent. Figure 12 shows the average throughput and Jain fairness index of the three algorithms running in 0.1~100 BDP buffer with different start time flows.

As shown in Figure 12(a), the fairness index of the BBRv1 algorithm is close to 1, and each flow forms a good bandwidth sharing. For BBRv2 in Figure 12(b), when the buffer size is less than 0.2 BDP, the fairness index decreases with the increase of buffer size, and the flow started first takes up more bandwidth. When the buffer size is greater than 10 BDP, the fairness index is only 0.93. For BBRv2+, the fairness index decreases with the increase of buffer size. But compared with the BBRv2 algorithm, the fairness has been greatly improved, and the minimum fairness index can be maintained at about 0.99. In general, the fairness index of the BBRv2+ algorithm is 6% higher than that of the BBRv2 algorithm, and it has better fairness on the whole.

### 5.4. RTT Fairness

Unlike traditional CCAs, the longer RTT flows in BBRv1 will occupy more bandwidth than the shorter RTT flows. This RTT bias is a trade-off between low latency and high transmission rates, breaking down the notion of finding the optimum operating point with the minimum RTT. Some evaluation results suggest that BBRv2 alleviates RTT unfairness in BBRv1. This section compares the RTT unfairness of BBRv1, BBRv2, and BBRv2+ through simulation experiments and evaluates whether BBRv2+ alleviates this limitation. These tests compete the 10 ms RTT flows.
with the 50 ms RTT flows. In the buffer condition of 0.1 – 100 BDP, the average throughput and Jain fairness index of flow are shown in Figure 13.

Figure 13(a) shows the average throughput change and Jain fairness index when 10 ms RTT flow competes with 50 ms RTT flow in the BBRv1 algorithm. With the increase in buffer size, the throughput difference between 10 ms RTT flows and 50 ms RTT flows becomes larger. When the buffer size is less than 0.5 BDP, different RTT flows can share bandwidth fairly, and the fairness index is about 0.99. When the buffer is larger than 0.5 BDP, the bandwidth difference between the two flows increases with the increase of the buffer. When the buffer is larger than 6 BDP, the fairness index is only about 0.63. In Figure 13(b), the average throughput variation trends of BBRv2 and BBRv2+ algorithms are similar, and the bandwidth difference between 10 ms RTT flows and 50 ms RTT flows decreases compared with BBRv1. With the increase in buffer size, the initial advantage of the 10 ms RTT flows increases and the fairness index drops to about 0.93. However, when the buffer is
increased between 7 BDP and 8 BDP, the bandwidth of 10 ms RTT flows and 50 ms RTT flows is reversed, and 50 ms RTT flows are gradually dominant. When the buffer is larger than 10 BDP, the fairness index of BBRv2 is about 0.9, while the fairness index of BBRv2+ is 0.94. In buffer sizes from 0.1 to 100 BDP, the BBRv2+ fairness index remained above 0.94, with a 31% improvement over BBRv1 and 4% improvement over BBRv2, especially in large buffers.

5.5. Multiple Flows with Different Start Times and RTTs. We further carry out hybrid experiments for different RTT flows and flows with different start times. The intrafairness of BBRv1, BBRv2, and BBRv2+ is verified by experiments. We are divided into four groups, each set as shown in Table 1.

The experimental results are shown in Figure 14. For the BBRv1 algorithm in Figure 14(a), the unfairness of average throughput mainly comes from the RTT unfairness. When the buffer size is less than 0.5 BDP, each flow can compete fairly for bandwidth. With the increase in buffer size, the fairness between different RTT flows (flow 1 and flow 2, flow 3 and flow 4) decreases, but the fairness between flows with different start times (flow 1 and flow 3, flow 2 and flow 4) can still maintain relatively stable fairness. When the buffer is larger than 10 BDP, the fairness index is only about 0.63. Figure 14(b) shows the experimental results of the BBRv2 algorithm. When the buffer is less than 1 BDP, four flows can compete fairly for bandwidth, and 10 ms RTT flows are relatively dominant. With the increase of buffer, the fairness among the four flows decreases. When the buffer size is between 5 BDP and 6 BDP, the bandwidth of 10 ms RTT flows and 50 ms RTT flows is reversed, and 50 ms RTT flow is dominant. Moreover, because of the bandwidth difference caused by the start time, the throughput gap between the four flows becomes larger and larger. When the buffer size increases to 10 BDP, the bandwidth difference remains stable, and the fairness index is approximately 0.9. For the BBRv2+ algorithm, when the buffer is less than 10 BDP, the average throughput trends of the four flows in Figure 14(c) are basically the same as those in Figure 14(b), but the throughput difference between the four flows is smaller. When the buffer is larger than 10 BDP, the BBRv2+ algorithm can better balance the bandwidth occupation of the four flows. Compared with the BBRv2 algorithm, the BBRv2+ algorithm alleviates the bandwidth unfairness, and the fairness index remains above 0.93. In particular, in the deep buffer size, the fairness index is 34% higher than BBRv1 and 7% higher than BBRv2. To sum up, the proposed BBRv2+ algorithm has better fairness than BBRv1 and BBRv2 algorithms; in particular, in deep buffer size, the fairness index is the highest.

5.6. Coexistence and Fairness with CUBIC. In this part, the fairness of BBRv1, BBRv2, and BBRv2+ is evaluated when coexisting with CUBIC in different buffer sizes. The competition between a single flow and multiple flows is considered. The simulation experiments of competition between single BBRv1, single BBRv2, or single BBRv2+ and single CUBIC, 50 BBRv1 flow, 50 BBRv2, or 50 BBRv2+ flows and 50 CUBIC flows are designed. The average throughput and fairness index results of the three algorithms are shown in Figure 15.

In Figure 15(a), a single BBRv1 flow competes with a single CUBIC. When the buffer is less than 0.4 BDP, the bandwidth of BBRv1 is more than 80%, and the fairness index is only about 0.67. When buffer size is between 0.5 BDP and 3 BDP, BBR’s average throughput is declining and CUBIC’s gradually occupies a favorable position, the average throughput between BBRv1 and CUBIC is reversed finally. In this range, the fairness index has also gradually increased, reaching the highest value of 0.99 near 2 BDP. When buffer size is greater than 3 BDP, the throughput difference between BBRv1 and CUBIC increases, and the final result is that the bandwidth occupancy ratio of CUBIC is about 75%, and the fairness index is stable near 0.76. Figure 15(b) shows the trend of average throughput and fairness index changes when a single BBRv1 flow competes with a single CUBIC. Compared with the BBRv1 algorithm, BBRv2 improves the fairness when coexisting with CUBIC. When the buffer size is 10 BDP, BBRv2 can maintain good fairness with CUBIC, and the fairness index is above 0.92. With the increase in buffer size, the average throughput of CUBIC increases gradually and occupies a dominant position. But the fairness index can be maintained at about 0.78, which is significantly better than the BBRv1 algorithm. For the BBRv2+ algorithm, the average throughput trend is very similar to BBRv2. When the buffer is less than 10BDP, the fairness index can remain above 0.92. When the buffer is larger than 10 BDP, the fairness index is greater than 0.82. Overall, the fairness of the BBRv2+ algorithm is improved by 6% compared with the BBRv1 algorithm and 4% compared with the BBRv2 algorithm, indicating that the BBRv2+ algorithm achieves better interprotocol fairness.

When 50 BBRv1 flows compete with 50 CUBIC, as shown in Figure 15(c), BBRv1 has obvious advantages in different buffers; in particular, when the buffer is less than 1 BDP, BBR occupies about 85% of the bandwidth, and the fairness index is only 0.62. When buffer size is larger than 1 BDP, the average throughput of BBR decreases slightly but still dominates, and the fairness index increases to about 0.94. In Figure 15(d), 50 BBRv2 flows compete with 50 CUBIC flows. Different from the case in Figure 15(b), the average throughput of BBRv2 is always dominant (in Figure 15(b), when the buffer is larger than 0.5 BDP, the average throughput of CUBIC is dominant). When the buffer size is 1 BDP to 2 BDP, the fairness index reaches above 0.98.

When the buffer is larger than 10 BDP, the fairness index decreases to 0.85. The fairness index first increases and then

| Flow | RTT (ms) | Start time (s) |
|------|----------|---------------|
| Flow 1 | 10 | 0 |
| Flow 2 | 50 | 0 |
| Flow 3 | 10 | 5 |
| Flow 4 | 50 | 5 |
| Flow 5 | 100 | 10 |

Table 1: The configuration of flows.
Figure 14: Average throughput and Jain fairness index of four flows with different RTT and different start times: (a) BBRv1, (b) BBRv2, and (c) BBRv2+.

Figure 15: Continued.
decreases with the increase of buffer and finally stabilizes at about 0.84. In general, when competing with CUBIC on the same link, the average throughput and fairness index of BBRv2+ and BBRv2 are basically the same. When single flow competes, the fairness of BBRv2+ is much better than BBRv1 and slightly better than that of BBRv2. When multiple streams compete simultaneously, the fairness index of BBRv2 and BBRv2+ is higher than that of BBRv1 in shallow buffers, but the fairness index of BBRv2+ is 1% lower than that of BBRv2 in deep buffers. The experimental results show that BBRv2+ does not worsen the fairness between protocols.

6. Conclusions

This paper analyzes the causes of intraprotocol fairness and RTT fairness in the BBRv2 algorithm in detail. Based on BBRv2, BBRv2+ judges packet loss selectively according to congestion degree, avoiding blind CWND limitation. In addition, the BBRv2+ algorithm adds a flow-aware ECN (FAECN) marking scheme which marks packets from flows according to queue information. In this scheme, the high-speed flows are more likely to be marked and slowed down, while the low-speed flows can speed up and finish quickly, so as to make different flows converge to fairness. On the NS3 platform, a large number of simulation experiments are carried out on multiple flows with different start times or RTTs under different buffer sizes. The results show that the algorithm can effectively solve the intraprotocol and RTT fairness problems without sacrificing the performance of the BBRv2 algorithm.

As programmable switches become more pervasive, realizing the idea of FAECN in the programmable switch may be feasible, which is our important future work. In addition, there are several factors that can influence the performance of the BBRv2+ algorithm, such as \( \Delta, K_{\text{min}}, \) and \( P. \) We will further optimize these parameters and plan to use machine learning methods to predict network load so that \( K_{\text{min}} \) or \( P \) can adjust adaptively.

Data Availability

No data were used to support this article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61602435 and in part by the National Natural Science Foundation of Anhui under Grant 1708085QF153. The authors sincerely thank Dr. Shuang Yang of the University of Science and Technology of China for their valuable comments and suggestions on this article, which will comprehensively improve the quality of this article.

References

[1] B. Turkovic, F. A. Kuipers, and S. Uhlig, “Fifty shades of congestion control: a performance and interactions evaluation,” 2019, https://arxiv.org/abs/1903.03852.
[2] R. Al-Saadi, G. Armitage, J. But, and P. Branch, “A survey of delay-based and hybrid TCP congestion control algorithms,” IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3609–3638, 2019.
Security and Communication Networks

[3] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," 1999, https://tools.ietf.org/html/rfc2582.

[4] 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.

[5] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in Proceedings of the conference on Communications architectures, protocols and applications, October 1994.

[6] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," IEEE Control Systems Magazine, vol. 22, no. 1, pp. 28–43, 2002.

[7] M. Mathis and J. Mahdavi, "Deprecating the TCP macroscopic model," ACM SIGCOMM - Computer Communication Review, vol. 49, no. 5, pp. 63–68, 2019.

[8] K. Sasaki, M. Hanai, and K. Miyazawa, A. Kobayashi and N. Oda, TCP fairness among modern TCP congestion control algorithms including TCP BBR," in Proceedings of the IEEE 7th international conference on cloud networking (CloudNet), pp. 1–4, Tokyo, Japan, October 2018.

[9] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," Communications of the ACM, vol. 60, no. 2, pp. 58–66, 2017.

[10] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR congestion control," in Proceedings of the IETF 97th Meeting, Seoul, Republic of Korea, November 2016.

[11] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR congestion control: an update," in Proceedings of the IETF 98th Meeting, Chicago, IL, USA, March 2017.

[12] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in Proceedings of the International Conference on Network Protocols (ICNP), Toronto, ON, Canada, October 2017.

[13] D. Scholz, B. Jaeger, L. Schwaighofer, L. Raumer, F. Geyer, and G. Carle, "Toward a deeper understanding of TCP BBR congestion control," in Proceedings of the IFIP Networking, Zurich, Switzerland, May 2018.

[14] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: experimental evaluation and analysis," 2017, https://arxiv.org/abs/1706.09115.

[15] K. Miyazawa K, K. Sasaki, N. Oda, and S. Yamaguchi, "Cyclic performance fluctuation of TCP BBR," in Proceedings of the IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 811–812, Tokyo, Japan, July 2018.

[16] Csdn, "BBRplus," 2020, https://blog.csdn.net/dog250/article/details/80629551.

[17] M. Jia, W. Sun, and Z. Wang, "MFBBR: an optimized fairness-aware TCP-BBR algorithm in wired-cum wireless network," in Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, July 2020.

[18] W. S. Pan, X. F. Li, H. B. Tan, and J. Xu, "Improvement of RTT fairness problem in BBR congestion control algorithm by gamma correction," Sensors, vol. 21, no. 12, 2021.

[19] I. Mahmud, G. H. Kim, T. Lubna, and Y. Z. Cho, "BBR-ACD: BBR with advanced congestion detection," Electronics, vol. 9, no. 1, p. 136, 2020.

[20] F. Chiariotti, A. Zanella, S. Kucera, and H. Clausen, "BBR-S: a low-latency BBR modification for fast-varying connections," IEEE Access, vol. 9, Article ID 76364, 2021.

[21] S. Najmuddin, M. Asim, K. Munir, T. Baker, Z. Guo, and R. Ranjan, "A BBR-based congestion control for delay-sensitive real-time applications," Computing, vol. 102, no. 12, pp. 2541–2563, 2020.

[22] H. Do, M. A. Gregory, and S. Li, "SDN-based Wireless access networks utilising BBR TCP congestion control," in Proceedings of the 29th International Telecommunication Networks and Applications Conference (ITNAC), Auckland, New Zealand, November 2019.

[23] W. Wei, K. Xue, J. Han, Y. Xing, D. S. L. Wei, and P. Hong, "BBR-based congestion control and packet scheduling for bottleneck fairness considered multipath TCP in heterogeneous wireless networks," IEEE Transactions on Vehicular Technology, vol. 70, no. 1, pp. 914–927, 2021.

[24] C. A. Grazi, N. Patriciello, M. Klapze, and M. Casoni, "BBR+: improving TCP BBR Performance over WLAN," in Proceedings of the IEEE International Conference on Communications (ICC), Dublin, Ireland, July, 2020.

[25] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, and V. Vasilev, "BBRv2: a model-based congestion control," in Proceedings of the IETF 102th Meeting, Montreal, Canada, July 2018.

[26] M. Alizadaeh, A. Greenberg, D. A. Maltz, J. Padhye, and P. Patel, "Data center TCP (dctcp)," in Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India, August 2010.

[27] S. Zhang, "An Evaluation of BBR and its variants," 2019, https://arxiv.org/abs/1909.03673.

[28] E. F. Klourey, J. Gomez, J. Cricigno, and E. Bou-Harb, "An emulation-based evaluation of TCP BBRv2 alpha for wired broadband," Computer Communications, vol. 161, pp. 212–224, 2020.

[29] Y. J. Song, G. H. Kim, I. Mahmud, W. K. Seo, and Y. Z. Cho, "Understanding of BBRv2: evaluation and comparison with BBRv1 congestion control algorithm," IEEE Access, vol. 9, pp. 37131–37145, 2021.

[30] A. Sivaraman, S. Subramaniam, M. Alizadaeh, S. Chole, and S.-T. Chuang, "Programmable packet scheduling at line rate," in Proceedings of the 2016 ACM SIGCOMM Conference, pp. 44–57, Florianopolis, Brazil, August 2016.

[31] N. K. Sharma, M. Liu, and K. Atrey, "Approximating fair queueing on reconfigurable switches," in Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 1–16, Renton, WA, USA, April 2018.

[32] Barefoot Networks, "Tofino2: second-generation of world’s fastest P4-programmable ethernet switch ASICs. (June 2019)," 2019, https://www.barefootnetworks.com/products/brief-tofino-2/.

[33] Broadcom, "High-capacity StrataXGS trident 4 ethernet switch series," 2019, https://www.broadcom.com/products/ethernetconnectivity/switching/traxtags/bcm56880-series.

[34] L. Tan, W. Su, W. Zhang et al., "In-band network telemetry: a new techniques for congestion detection and avoidance," in Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 44–57, Florianopolis, Brazil, August 2016.

[35] N. K. Sharma, M. Liu, and K. Atrey, "Approximating fair queueing on reconfigurable switches," in Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 1–16, Renton, WA, USA, April 2018.

[36] Barefoot Networks, "Tofino2: second-generation of world’s fastest P4-programmable ethernet switch ASICs. (June 2019)," 2019, https://www.barefootnetworks.com/products/brief-tofino-2/.

[37] Broadcom, "High-capacity StrataXGS trident 4 ethernet switch series," 2019, https://www.broadcom.com/products/ethernetconnectivity/switching/traxtags/bcm56880-series.

[38] L. Tan, W. Su, W. Zhang et al., "In-band network telemetry: a survey," Computer Networks, vol. 186, Article ID 107763, 2021.

[39] N. Cardwell, Y. Cheng, S. H. Yeganeh, and S. H. Yeganeh, "BBRv2: a model-based congestion control," in Proceedings of the ICCRG IETF 104th Meeting, Prague, Czech Republic, March 2019.

[40] Github, "TCP BBR v2 alpha/preview release," 2019, https://github.com/google/bbr/tree/-v2alpha.

[41] N. Cardwell, Y. Cheng, S. H. Yeganeh, and S. H. Yeganeh, "BBRv2: a model-based congestion control," in Proceedings of the IETF 105th Meeting, Montreal, Canada, July 2019.
[38] N. Cardwell, Y. Cheng, and K. Yang, “BBR update: 1: BBR.Swift; 2: scalable loss handling,” in Proceedings of the IETF 109th Meeting, Online, November 2020.

[39] G. Kumar, N. Dukkipati, and K. Jang, “Swift: delay is simple and effective for congestion control in the datacenter,” in Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Online, July 2020.

[40] J. Gomez, E. Kfoury, J. Crichigno, E. B. Harb, and G. Srivastava, “A performance evaluation of TCP BBRv2 alpha,” in Proceedings of the 43rd International Conference on Telecommunications and Signal Processing (TSP), Milan, Italy, July 2020.

[41] Y. J. Song, W. J. Eom, J. K. Kim, C. H. Park, and G. H. Kim, “Intra-protocol convergence problem in BBRv2’s bandwidth probing,” in Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, October 2020.

[42] A. Nandagiri, M. P. Tahiliani, V. Misra, and K. K. Ramakrishnan, “BBRv1 vs BBRv2: examining performance differences through experimental evaluation,” in Proceedings of the IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Orlando, FL, USA, July 2020.

[43] L. Kleinrock, “Power and deterministic rules of thumb for probabilistic problems in computer communications,” in Proceedings of the International Conference on Communications, vol. 43, Boston, MA, USA, June 1979.

[44] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, “Host-to-host congestion control for TCP,” IEEE Communications surveys & tutorials, vol. 12, no. 3, pp. 304–342, 2010.

[45] M. A. Alrshah, M. A. Al-Maqri, and M. Othman, “Elastic-TCP: flexible congestion control algorithm to adapt for high-BDP networks,” IEEE Systems Journal, vol. 13, no. 2, pp. 1336–1346, 2019.

[46] J. Luo, J. Jin, and F. Shan, “Standardization of low-latency TCP with explicit congestion notification: a survey,” IEEE Internet Computing, vol. 21, no. 1, pp. 48–55, 2017.

[47] P. Zhou, H. Yu, G. Sun, L. Luo, S. Luo, and Z. Ye, “Flow-aware explicit congestion notification for datacenter networks,” Cluster Computing, vol. 22, no. 4, pp. 1431–1446, 2019.

[48] M. Alizadeh, A. Javanmard, and B. Prabhakar, “Analysis of DCTCP: stability, convergence, and fairness,” in Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems-SIGMETRICS ’11, vol. 39, no. 1, pp. 73–84, San Jose, CA, USA, June 2011.

[49] W. Bai, L. Chen, K. Chen, and H. Wu, “Enabling ECN in multi-service multi-queue data centers,” in Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’16), Santa Clara, CA, USA, March 2016.

[50] D. Shan and F. Ren, “Improving ECN marking scheme with micro-burst traffic in data center networks,” in Proceedings of the IEEE INFOCOM 2017-IEEE Conference on Computer Communications, Atlanta, GA, USA, May, 2017.

[51] V. Jain, V. Mittal, and M. P. Tahiliani, “Design and implementation of TCP BBR in ns-3,” in Proceedings of the 10th Workshop on NS-3, Surathkal, India, June 2018.

[52] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, “Reproducible measurements of TCP BBR congestion control,” Computer Communications, vol. 144, pp. 31–43, 2019.

[53] R. K. Jain, D. M. W. Chiu, and W. R. Hawe, A Quantitative Measure of Fairness and Discrimination, Eastern Research Laboratory, Digital Equipment Corporation: Hudson, Maynard, MA, USA, 1984.