oBBR: Optimize Retransmissions of BBR Flows on the Internet

Pengqiang Bi, Mengbai Xiao, Dongxiao Yu, and Guanghui Zhang, *Shandong University;* Jian Tong, Jingchao Liu, and Yijun Li, *BaishanCloud*

https://www.usenix.org/conference/atc23/presentation/bi

This paper is included in the Proceedings of the 2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

Open access to the Proceedings of the 2023 USENIX Annual Technical Conference is sponsored by

oBBR: Optimize Retransmissions of BBR Flows on the Internet

Pengqiang Bi
Shandong University

Mengbai Xiao *
Shandong University

Dongxiao Yu
Shandong University

Guanghui Zhang
Shandong University

Jian Tong
BaishanCloud

Jingchao Liu
BaishanCloud

Yijun Li
BaishanCloud

Abstract

BBR is a model-based congestion control algorithm that has been widely adopted on the Internet. Different from loss-based algorithms, BBR features high throughput since it characterizes the underlying link and sends data accordingly. However, BBR suffers from high retransmission rates in deployment, leading to extra bandwidth costs. In this work, we carefully analyze and validate the reasons for high retransmissions in BBR flows. In a shallow-buffered link, the packet losses are deeply correlated to both the bottleneck buffer size and the in-flight data cap. Additionally, bandwidth drops also cause unwanted retransmissions. Based on the analysis, we design and implement oBBR, which aims at optimizing the retransmissions in BBR flows. In oBBR, we adaptively scale the in-flight data cap and update the bandwidth estimate timely so that few excessive data are injected into the network, avoiding packet losses. Our Internet experiments show that oBBR achieves 1.54× higher goodput than BBRv2 and 39.48% fewer retransmissions than BBR-S, which are both BBR variants with improved transmission performance. When deploying BBR in Internet streaming sessions, oBBR obtains greater QoE than BBRv2 and BBR-S without incurring more retransmissions. To summarize, oBBR is designed to help a transmission session reach high goodput and low retransmissions simultaneously, while other CCAs only achieve one of them.

1 Introduction

Congestion control algorithms (CCAs) are essential for data transmission on the Internet. The loss-based CCAs like CUBIC [18] treat the packet loss as a signal of network congestion and thus throttle their sending rate. However, this significantly underutilizes the underlying link capacity since packet losses do not necessarily indicate congestion in the Internet nowadays. To effectively exploit the bandwidth resources, Google developed a model-based CCA based on measuring bottleneck bandwidth and round-trip propagation time, or BBR [8]. BBR ignores packet losses but adjusts its behaviors according to the estimated bandwidth and round-trip time (RTT). After switching to BBR from CUBIC, the throughput in Google's B4 network is consistently improved by 2-25× [9]. Since its release, BBR has been deployed on 22% of websites and accounts for over 40% of Internet traffic [36]. This attracts content providers, like YouTube [7] and Spotify [12], to adopt BBR.

While BBR exploits the bandwidth more effectively, this algorithm also leads to high retransmission rates. BBR is expected to send in-flight data at a volume of the bandwidth-delay product (BDP) only, i.e., operating at Kleinrock's optimal point [29], but it is still observed that BBR injects excessive data to the transmission channel because of bandwidth overestimation [22, 50]. To avoid the excessive data accumulated at the bottleneck, BBR imposes an upper bound to the volume of in-flight data at 2×BDP. But this results in a high packet loss ratio if the bottleneck buffer is not large enough to hold the excessive in-flight data. As a result, a large amount of data needs to be retransmitted, bringing extra bandwidth costs to content providers.

Directly reducing the upper bound of in-flight data in BBR is not feasible. By capping the in-flight data at 2×BDP, a BBR flow could "fairly" share the bandwidth with a loss-based flow (∼40%) when the bottleneck buffer is deep [14, 43, 52, 53]. If we limit the in-flight data in BBR further, it can no longer compete with the loss-based flows, thus achieving a degrading throughput.

Additionally, the high retransmission rates in BBR also result from bandwidth drop. BBR estimates the bottleneck bandwidth with the maximum delivery rate samples collected in an RTT-based time window. If the bandwidth drops, BBR will not update the bandwidth estimate until the outdated (and overestimated) samples leave the time window. While BBR keeps sending data and caps the in-flight data according to the overestimated bandwidth, the bottleneck buffer is quickly crammed and starts to discard packets. Moreover, the congestion at the bottleneck buffer enlarges the latest RTT samples

*Corresponding author

and thus the window size, making the old bandwidth samples expire even later.

In this paper, we propose oBBR, optimizing the retransmission rate and throughput in a BBR session. By extending the analysis model proposed in [52], we notice that the behaviors of a BBR flow are deeply correlated to the bottleneck buffer size and the in-flight data cap. Capping the in-flight data to a smaller value than 2×BDP mitigates the retransmissions in shallow buffers, but also weakens BBR's competitiveness in deep buffers. Furthermore, BBR can no longer compete with loss-based flows if the in-flight data is exactly 1×BDP. Following the analysis, oBBR detects if the bottleneck buffer is shallow once a packet loss event occurs. Then, oBBR reduces the in-flight cap accordingly so that the retransmissions are avoided. The in-flight cap is gradually recovered in case the packet loss is not caused by congestion. Furthermore, identifying a bandwidth drop event is not straightforward in BBR. Bandwidth samples with smaller values than the estimate are common because only the maximum is selected. In oBBR, we use consecutive samples of decreasing bandwidth or increasing latency to identify a bandwidth drop event. However, it is still possible that we falsely detect a bandwidth drop. We compare the delivery performance before and after the bandwidth estimate update. If the delivery performance degrades, we revert it to the old value.

We implement oBBR within a userspace implementation of Quick UDP Internet Connection (QUIC) [30], which is an appealing solution to network applications like video streaming due to its improved performance, high flexibility, and ease of deployment. We evaluate oBBR in both a lab environment and the Internet. In a stable network environment, oBBR reduces the packet loss ratio by up to 30× when compared to BBR. With 2% packet losses, oBBR achieves 14.65× higher goodput than BBRv2 which reacts to packet losses for alleviating retransmissions. In a network with fluctuating bandwidth between 40 Mbps and 10 Mbps, oBBR reduces the retransmissions by 52%. The realistic network emulation and the experiments on the Internet both show that oBBR achieves low retransmission ratios as loss-based algorithms, like BBRv2 and CUBIC, and high goodput as model-based ones, like BBR. Especially when continuously delivering data in a long Internet link, oBBR has the retransmission at ∼6.5% while gaining 1.54× more goodput compared to BBRv2, which retransmits 6.71% data. Compared to BBR-S, another BBR variant that also has reduced retransmissions and high goodput in our experiments, oBBR reduces retransmissions by 39.48%. We also implement oBBR in a video streaming system and measure the user's quality of experience (QoE) of streaming sessions on the Internet. Our experiments show that oBBR guarantees the QoE of a video session in terms of average quality, quality switches, and rebuffering ratio better than other CCAs including BBRv2 and BBR-S while suppressing the retransmission ratio as low as ∼4%. In summary, oBBR achieves both high goodput and low retransmissions in a transmission session,

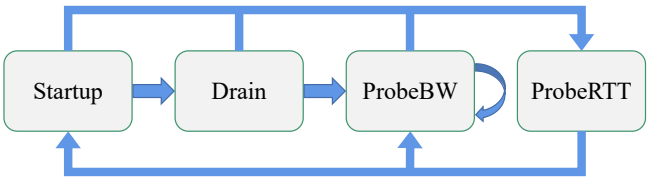

Figure 1: Overview of the BBR algorithm

while other CCAs only achieve one. The contributions of this paper are as follows:

- We carefully analyze why retransmissions are common in BBR flows. The high retransmissions in shallow-buffered links are deeply correlated to both the in-flight data cap and the buffer size. Moreover, the mechanism of BBR modeling the network becomes inaccurate if the bandwidth drops.
- We design and implement oBBR, optimizing the retransmissions of the BBR algorithm. oBBR adjusts the in-flight data cap according to the bottleneck buffer size and updates the bandwidth estimate promptly. Both reduce the excessive data sent by oBBR, avoiding packet losses.
- We carry out extensive experiments in both a lab environment and the Internet, justifying the design of oBBR. We also implement oBBR in a video streaming system on the Internet, and the results show that oBBR reaches higher QoE than BBRv2 while both have low retransmission ratios.

We briefly introduce how BBR works in Section 2 and analyze why the transmission rate of BBR flows is high in Section 3. The design of oBBR is presented in Section 4. In Section 5, extensive experiments are carried out to evaluate our design. Section 6 discusses related work and Section 7 concludes our work.

2 Background

BBR. BBR is a model-based congestion control algorithm released by Google in 2016 [8]. In a transmission session, BBR estimates the bottleneck bandwidth and the round-trip propagation time of the underlying link. To get unbiased estimates, BBR extracts the maximum from recent bandwidth samples and the minimum from recent RTT samples.

To regulate the traffic, BBR matches its average sending rate to the bandwidth estimate and caps the in-flight data by 2×BDP. Two key parameters, `pacing_gain` and `cwnd_gain`, are used in controlling the sending behaviors. `pacing_gain` inflates or deflates the sending rate while setting it to 1 means the sender transmits data at the rate of estimated bandwidth. This parameter is dynamically scaled in operation. `cwnd_gain` caps the in-flight data and is set to 2 in practice,

which means at most 2×BDP data are sent without acks. A BBR session always switches among four phases: *Startup*, *Drain*, *ProbeBW*, and *ProbeRTT*. An overview of the BBR algorithm is shown in Figure 1

1) Startup: BBR detects the available bandwidth by inflating its sending rate, where `pacing_gain` and `cwnd_gain` are both $2/ln2$. It switches to the *Drain* phase if the measured bandwidth stops growing.

2) Drain: BBR drains the excessive data queued at the bottleneck buffer because of the exponential growth of sending rate in the *Startup* phase. `pacing_gain` is set to $ln2/2$ until the in-flight data is less than 1×BDP. BBR then enters the *ProbeBW* phase.

3) ProbeBW: BBR spends most time in the *ProbeBW* phase, where `pacing_gain` is periodically set to {1.25, 0.75, 1, 1, 1, 1, 1, 1} and `cwnd_gain` is fixed to 2. Periodically inflating the sending rate helps BBR detect if the available bandwidth has increased, and deflating the rate could drain the queued data if the bandwidth is unchanged.

4) ProbeRTT: The *ProbeRTT* phase is independent of the other phases. BBR enters *ProbeRTT* once the RTT estimate has not been updated for 10 seconds. In this phase, BBR sends only 4 packets in-flight and observes their RTTs.

BBRv2. Though BBR gains high throughput in transmission, its mechanism has raised a few concerns: BBR does not react to the packet losses caused by network congestion and thus has high retransmission rates in links with a shallow bottleneck buffer; BBR does not share bandwidth with loss-based algorithms fairly; The throughput drops drastically in the *ProbeRTT* phase. To address these shortcomings, BBRv2 [11] is proposed and has been deployed in Google's internal network [10].

In BBRv2, packet loss events are considered signals of network congestion again. BBRv2 reduces the in-flight data cap multiplicatively if the packet loss ratio exceeds a threshold (2% in practice). But this also impairs the flow throughput as the loss-based CCAs do in a network environment with a high packet loss ratio (> 2%). In this work, we also compare oBBR with BBRv2, and the results are reported in Section 5.

3 Retransmissions in BBR

In this section, we analyze two major reasons that cause high retransmission rates of BBR flows in depth: 1) the underlying link is shallow-buffered, and 2) the available bandwidth drops in the transmission session.

3.1 Shallow-Buffered Link

Ideally, BBR operates at the optimal point of the transmission channel, i.e., no packets are queued in the bottleneck buffer and the latency approximates the physical delay. To achieve this, the in-flight data should equal 1×BDP of the channel. In practice, the bandwidth is usually overestimated [22, 50, 52], and thus the data would be gradually accumulated at the bottleneck buffer until packet losses. To avoid this, the sender employing BBR caps the in-flight data as

$$inflight = \texttt{cwnd\_gain} \cdot \widehat{RTprop} \cdot \widehat{BtlBw},$$

where $\widehat{RTprop}$ and $\widehat{BtlBw}$ are the estimates of round-trip propagation time and bottleneck bandwidth, respectively. `cwnd_gain` bounds the in-flight data to a small multiple of the BDP and is commonly set to 2. We can safely assume the volume of in-flight data reaches the upper bound most of the time because BBR overestimates its bandwidth share when competing with other flows [22].

However, $\widehat{RTprop}$ and $\widehat{BtlBw}$ are substantially affected by the competing flows, especially the ones with loss-based congestion control algorithms (CCAs). If the data from competing flows are also queued in the bottleneck buffer, $\widehat{RTprop}$ is then composed of the physical delay and the queue length, and $\widehat{BtlBw}$ is the link capacity proportional to BBR's buffer occupancy ratio [52]. The volume of in-flight data is calculated as

$$inflight = g \cdot (\frac{pq}{c} + l) \cdot (1-p)c,$$

where g represents `cwnd_gain`, q is the queue capacity of bottleneck buffer, p is the buffer ratio occupied by competing loss-based CCA flows, l is the RTT without congestion, and c is the link capacity. Then, the queued data of a BBR flow is

$$\begin{aligned} queued &= g \cdot (\frac{pq}{c} + l) \cdot (1-p)c - (1-p)cl \\ &= gq \cdot p(1-p) + (g-1) \cdot cl \cdot (1-p). \end{aligned} \tag{1}$$

By defining the occupied ratio of BBR flow at the bottleneck as $Q_r = \frac{queued}{q}$ and the relative size of bottleneck buffer to the BDP as $R = \frac{q}{cl}$, we have

$$Q_r(p; g, R) = gp(1-p) + (g-1)R^{-1}(1-p).$$

This helps us understand the behaviors of BBR with different configuration sets of g and R. We plot Q_R *vs.* p in Figure 2.

The left side of Figure 2 shows how a BBR flow behaves if $g = 2$, the current setup in BBR implementations. When the bottleneck buffer is as deep as 16×BDP, the BBR flow occupies almost half the buffer no matter how many loss-based flows coexist, which has been confirmed in prior studies [52]. If we reduce R, the shallower buffer will be more occupied by the BBR flow. When the buffer is as shallow as 1×BDP, the loss-based CCAs can not compete with BBR anymore. When the buffer size is lower than 1×BDP, we do not have enough space to hold the in-flight data and the packet loss ratio drastically increases.

If we set g to 1, expecting that no packet is queued when there is no competing flow, this also leads to another severe problem. As shown in the right of Figure 2, no matter how large the bottleneck buffer is, the BBR flow can not compete

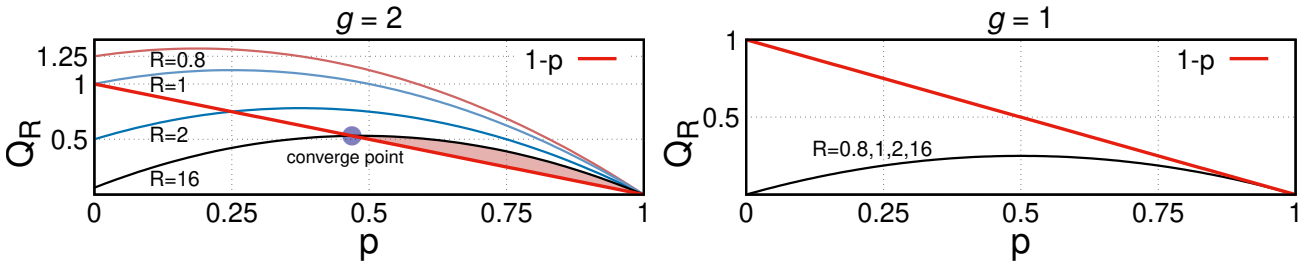

Figure 2: Q_R *vs.* p when setting g to 2 and 1. $1-p$ indicates the remaining buffer space for the BBR flow (from loss-based flows). We take $g = 2$ and $R = 16$ as an example. When $1-p$ is above the Q_R curve (at the left side of the figure), the buffer has enough space to hold the in-flight data of the BBR flow. As the loss-based CCA flows keep ramping up their in-flight data since no packets are discarded, p is increasing, which lets the BBR flow also accumulate more data in the buffer. The intersection of $1-p$ and the Q_R curve means that the buffer is fully occupied and loss happens. The loss-based CCA flows start to back off and then the BBR flow also backs off. As a result, the BBR flow operates around the converge point.

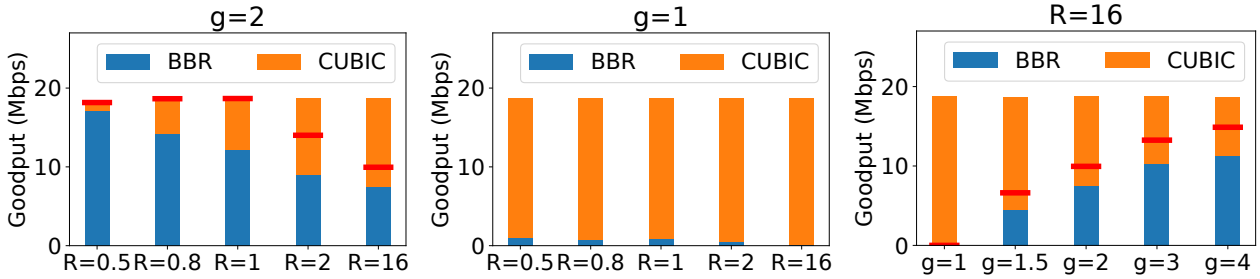

Figure 3: The average goodput of a BBR flow and a CUBIC flow with various configuration sets g and R

with the loss-based CCAs anymore, leading to poor performance.

Experimental verification: We set up the testbed in our lab environment based on nginx-quic [3], where we implement the BBR and CUBIC algorithms in the QUIC protocol. The details of the experimental setup are presented in Section 5.1. In the experiments, we simultaneously launch a BBR flow and a CUBIC flow. Both of them keep sending data without limitation from the application layer. The bandwidth is set to 20 Mbps and the RTT is set to 40 ms. By skipping the first 60 seconds, we measure the average goodput of two flows for 7 minutes when setting g and R to different values.

In Figure 3, the leftmost reports the average goodput of two flows when setting $g = 2$ and varying the bottleneck buffer size. We can see that the goodput proportion of the BBR flow decreases as R is greater, but it is still lower than the converge point $\frac{R-g+1}{gR}$ found in the model, which are marked as red lines on the bars. The reason is that BBR can not adjust its behavior as fast as the CUBIC does at the converging point: When two flows fully occupy the bottleneck buffer and packets are discarded, the CUBIC flow reduces its congestion window and the queue size shrinks, leading to that smaller RTTs are detected by the BBR flow. This immediately updates $\widehat{RTprop}$ and BBR calculates a smaller BDP. However, in the recovery phase, the CUBIC flow ramps up its congestion window quickly while BBR still uses the small BDP because the new RTT samples are greater than $\widehat{RTprop}$. The inaccurate RTT estimate expires in 10 seconds but before that, the CUBIC flow has occupied all free space in the buffer and backs off because of the packet loss again. Furthermore, BBR periodically enters *ProbeRTT* which reduces the in-flight data to 4 packets, also yielding bandwidth to the competing flows. But since the BBR ignores packet loss, this results in the BBR flow evicting the CUBIC flow and starting to experience apparent packet losses in a shallower buffer. When R is reduced to 0.5, BBR dominates the bandwidth and the packet loss ratio increases to 13.36%. The middle of Figure 3 confirms that when g is set to 1 and no matter how large the bottleneck buffer is, BBR can hardly compete with CUBIC and almost all the bandwidth is used by the CUBIC flow. The rightmost of Figure 3 shows the goodput of two flows when setting the bottleneck buffer to 16×BDP and varying g, where BBR gains more goodput with increasing g.

Based on the models and the experiments, we can derive that the packet losses of a BBR flow in a shallow-buffered link are correlated to g and R. If we reduce g, a BBR flow only evicts the coexisting loss-based flows and experiences packet losses in a shallower buffer. But this weakens the

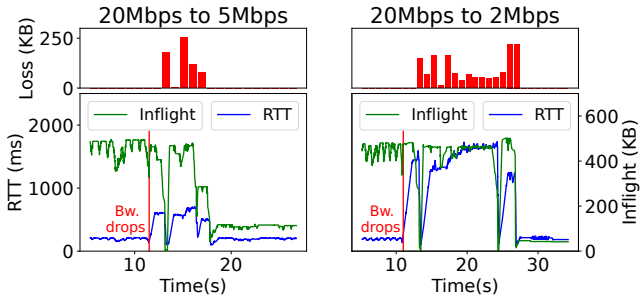

Figure 4: The network characteristics of a BBR flow when the bandwidth drops from 20 Mbps to 5 Mbps or 2 Mbps.

competitiveness of BBR in deep buffers. When g decreases to 1, BBR can hardly compete with loss-based flows no matter how large the bottleneck buffer size is. If set it to a high value, BBR flows have to sustain a high packet loss ratio when the bottleneck buffer is shallow.

3.2 Bandwidth Drop

BBR collects the delivery rate samples in a time window (typically 10 RTTs) and chooses the maximum as the bandwidth estimate. The rationale behind this is that acks can only be delayed. However, such a mechanism makes BBR react to the bandwidth drop sluggishly. When the bandwidth decreases, BBR keeps sending data at a high rate until the overestimated samples expire after 10 RTTs. This ramps up the queue size at the bottleneck buffer and leads to packet loss once the queue is crammed. More severely, as the packet delivery delay is extended due to the congestion at the bottleneck, the overestimated bandwidth samples expire in a longer period.

We conduct experiments to observe how a BBR flow reacts to the bandwidth drop in our testbed. In the experiments, the bandwidth is initially 20 Mbps, the RTT is 100 ms, and the bottleneck buffer is 200 KB. A BBR flow is launched to constantly send data. After ∼10 seconds since the flow starts, we throttle the bandwidth to a lower value (5 Mbps or 2 Mbps).

The experimental results are reported in Figure 4. The x-axis is the time elapsed since the flow starts in seconds and we plot the RTT samples in milliseconds, in-flight data volume in KB, and lost data volume in KB. We can observe that even the bandwidth drops, the in-flight data volume stays almost the same because of the unchanged $\widehat{BtlBw}$ and $\widehat{RTprop}$. While the actual bandwidth can not match the sending rate of BBR, the RTTs of packets rise sharply due to the increasing queue size at the bottleneck. The lost data volume also increases during this period. Only after 10 RTTs, i.e., ∼7 seconds on the left of Figure 4 (the RTT samples rise to ∼700 ms) and ∼15 seconds on the right of Figure 4 (the RTT samples rise to ∼1500 ms), the bandwidth estimate is corrected and the in-flight data match the BDP again. During the overestimation stage, the sudden drops of RTTs are caused by the *probeRTT* phase, but this does not help the BBR client realize the actual BDP.

4 Design

As discussed in Section 3, the main reasons for high retransmission rates in BBR are: 1) In a shallow buffer, the data in-flight could not be fully contained when g is fixed to a constant; 2) The bandwidth estimate is not timely updated when the bandwidth drops. To solve these problems, we propose to

- *Adaptive g*: We estimate the bottleneck buffer size with the RTT samples, and when a packet is discarded, the value of g is adjusted accordingly.
- *Timely bandwidth updates*: We identify the bandwidth drop based on both the RTT and delivery rate samples, updating the bandwidth estimate in time. This is reversible if the transmission throughput decreases.

4.1 Adaptive g

In practical BBR sessions, g is set to a fixed value of 2. This leads to the packet loss ratio drastically rising if the bottleneck buffer is shallower than $(g-1)\times$BDP. In this case, BBR packets are dropped because the in-flight data volume is larger than the buffer size q. By setting $p=0$ in Equation 1, the excessive data volume of a BBR flow is $(g-1)cl$. To avoid the packet loss events, we could adjust g to match the excessive data volume to the shallow buffer size q, i.e.,

$$g = \frac{q+cl}{cl}.$$

Whenever a packet loss event occurs, we can assume this results from a shallow bottleneck buffer. To scale g, we need to evaluate the buffer size by

$$q = c(l' - l),$$

where l' is the delivery latency of a packet that is queued at the end of the bottleneck buffer. Since the current buffer is full, l' can be reasonably estimated using the latest RTT sample RTT_{latest}. Then, we know that if

$$\frac{l'}{l} = \frac{RTT_{latest}}{\widehat{RTprop}} < g,$$

the bottleneck buffer is shallow.

Once we discover the packet loss is caused by a shallow buffer, we could scale g to

$$\min\left(1+\mu\cdot\frac{RTT_{latest}-\widehat{RTprop}}{\widehat{RTprop}}, g_{max}\right),$$

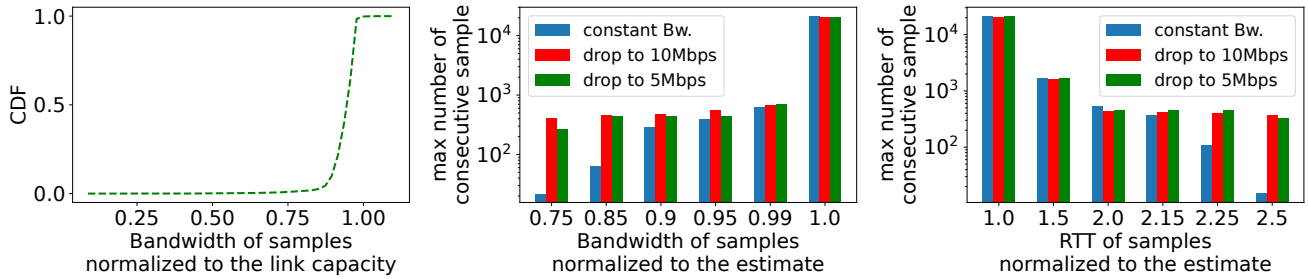

Figure 5: The network statistics of a BBR flow in various network environments. The network bandwidth is consistently set to 20 Mbps in the transmission session or decreases to 5/2 Mbps after 10 seconds. The leftmost figure presents the cumulative distribution function (CDF) of bandwidth samples normalized to the actual bandwidth when the bandwidth is constant. The figure in the middle shows the max number of consecutive bandwidth samples that are below the proportions of the estimated bandwidth. The rightmost figure shows the max number of consecutive RTT samples above thresholds of multiplicative RTTs.

where $0 < \mu < 1$ indicates the proportion of the shallow buffer attributed to the BBR flow and g_{max} is referred to as the fixed value used in deep buffer cases.

When the bottleneck buffer is non-shallow and there are competing flows, RTT_{latest} is expected to be greater than $g_{max} \times \widehat{RTprop}$ no matter if the packet loss event is caused by random loss or congestion. The $\min(\cdot)$ filter helps oBBR behave like the vanilla BBR in these cases if $g_{max} = 2$. If there are no competing flows, a smaller g but greater than 1 still guarantees the bandwidth is fully utilized. As a result, oBBR can adapt to the shallow buffer cases without affecting other scenarios.

Only scaling down g is not enough: 1) RTT_{latest} might be less than l'. For example, a packet loss happens before the shallow buffer is crammed. 2) The bottleneck might migrate to a deep buffer during the transmission session. Thus, we design a recovery stage to increase g back to g_{max}. Whenever an ack is received, we have

$$g = \min\left(g + \alpha \cdot \frac{S_{ack}}{BDP}, g_{max}\right),$$

where S_{ack} is the acked data size and α is a parameter preventing the congestion window growing too fast. In this way, after the congestion window is shrunk, it keeps increasing the value back until the packet loss happens again.

4.2 Timely Bandwidth Updates

BBR selects the maximum of bandwidth samples during the latest 10 RTTs as $\widehat{BtlBW}$. In the *ProbeBW* stage, BBR periodically inflates the sending rate to $1.25 \times \widehat{BtlBW}$ to verify if the actual bandwidth has increased. As long as a sample of higher bandwidth is observed, $\widehat{BtlBW}$ is updated so that BBR reacts to bandwidth increase timely. However, BBR fails to adapt to the bandwidth drop in time. Though the latest acks could show that fewer data has been received per unit time, these signals are ignored in estimating $\widehat{BtlBW}$. As a result, BBR keeps sending the data at a high rate until all samples of high bandwidth expire.

However, observing a sample of low bandwidth does not necessarily indicate the bandwidth has decreased. Acks might be delayed in the transmission and this is the reason of selecting maximum as the estimate. We collect bandwidth samples by running a BBR flow in the lab with 20 Mbps bandwidth and 100 ms RTT. The bandwidth samples are normalized to the realistic bandwidth and the results are shown in the leftmost of Figure 5. We can find that even for an experiment in the lab, 59% of samples are lower than $0.95 \times BtlBw$, and 24% of samples are lower than $0.85 \times BtlBw$.

Though the bandwidth samples could be inaccurate (we observe a sample of $0.09 \times BtlBw$ in the experiment), the samples that are highly deviated from the realistic bandwidth rarely appear consecutively. The middle of Figure 5 shows the max length of consecutive samples that are below varying thresholds proportional to the bandwidth estimate. We can see that for the constant bandwidth, at most 22 consecutive samples are observed below $0.75 \times \widehat{BtlBw}$. We also manually throttle the bandwidth to 10/5 Mbps in the transmission, and in both cases, the length of consecutive bandwidth samples could clearly signal the bandwidth drop. To effectively detect the decrease of bandwidth, we check if **there are k consecutive bandwidth samples less than $0.75 \times \widehat{BtlBW}$**. With a longer k, we are more confident to believe that the bandwidth has decreased rather than acks are occasionally delayed.

On the other hand, the bandwidth samples could be overestimated in realistic network environments. For example, routers could aggregate acks or have the capacity of handling burst traffic, both leading to calculating a greater bandwidth at the BBR sender. To detect the bandwidth drop even if the bandwidth samples are not reliable, we also use RTT samples. Since at most $g_{max} \times$BDP data are sent in-flight, the RTT samples should be about $g_{max} \times \widehat{RTprop}$ and samples with obviously greater values indicate the bandwidth estimate is not accurate anymore. We plot the max number of consecu-

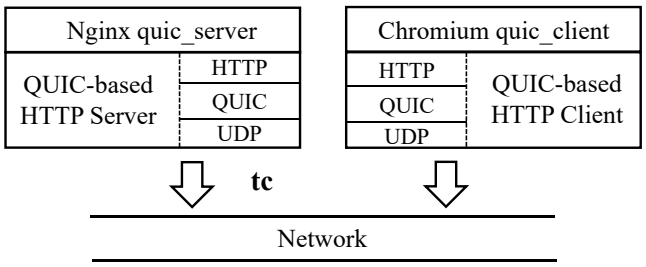

Figure 6: Overview of the testbed

tive RTT samples that are above thresholds of multiplicative $\widehat{RTprop}$ in the rightmost of Figure 5. When the bandwidth is constant, at most 15 samples are observed consecutively higher than $2.5\times\widehat{RTprop}$. This number increases to 373 or 328 if the bandwidth drops to 10/5 Mbps during the transmission, respectively. Thus, we also detect if **there are k consecutive RTT samples greater than** $(g_{max}+0.5)\times\widehat{RTprop}$. To the end, if **any of the two cases happen**, we think the bandwidth has decreased, and we use the average of k most recent bandwidth samples to update $\widehat{BtlBW}$.

Though we update the bandwidth estimate cautiously, it is still possible that an inaccurate value is used. So we monitor the delivery performance before and after the bandwidth update, reverting to the old value if the delivery performance decreases. Specifically, we calculate a delivery score of a fixed time interval T following

$$U = delivered - 10\times unacked,$$

where *delivered* is the amount of data acked and *unacked* is that not acked in T. We calculate the average score for $2T$ before and $2T$ after the bandwidth update. $\widehat{BtlBW}$ is reverted if the score drops.

4.3 Competitiveness Analysis

As discussed in Section 3.1, the competitiveness of BBR is determined by g and R. oBBR scales g to a smaller value once discovering the current link is shallow-buffered, while using a fixed g_{max} in deep buffers. By setting g_{max} to 2, oBBR behaves just like the vanilla BBR in deep-buffered links. A smaller or greater g_{max} weakens or strengthens its competitiveness.

In shallow-buffered links, the competitiveness of an oBBR flow is determined by μ, i.e., the excessive in-flight data proportional to the shallow buffer size. With a greater μ, oBBR is more competitive, but also has a higher risk of packet losses. Since the vanilla BBR selects a fixed g of 2, oBBR can hardly compete with it. But we can still expect that oBBR is less aggressive than BBR when competing with loss-based algorithms.

5 Evaluation

In this section, we evaluate oBBR and other peer methods in both a lab environment and the realistic Internet. We also build a video streaming system with oBBR that verifies our design also benefits network applications.

5.1 Experimental Setup

Implementation: We implement oBBR[1] in *nginx-quic* [3], which only has an RFC-defined congestion control algorithm.[2] In all experiments, we set α that controls the speed of recovery from the packet loss to 0.01. For detecting the bandwidth drop, we set k to 30 and T to 200 ms. We vary μ in 0.5, 0.75, and 1 to have oBBRs with different competitiveness, which is referred to as oBBR-0.5, oBBR-0.75, and oBBR-1 in the following discussion. In oBBR, we also set a lower bound 1.25 to `cwnd_gain` when the *ProbeBW* phase is probing for more bandwidth.

Additionally, we implement a couple of CCAs for comparison: **CUBIC** [18] is the default CCA in the Linux kernel, which considers packet loss as congestion and uses a cubic function to grow the congestion window. The vanilla version of **BBR** [8] manipulates its sending behaviors only depending on the estimates of bottleneck bandwidth and propagation round-trip time. **BBRv2** [11] is the successor of BBR proposed by Google, adding reactions to packet loss and greatly reducing retransmission rates. **BBR-S** [50] handles the bandwidth overestimation caused by the burst capacity of routers. It estimates the bandwidth at the 85th percentile of collected samples. **B3R** [44] is a BBR variant, which adjusts `pacing_gain` to regulate the sending rate in the *ProbeBW* phase. It aims at reducing the packet loss ratio when the bottleneck buffer is shallow. To compare all of these CCAs with oBBR on an equal footing, we implement these CCAs within nginx-quic. Implementing all of the CCAs within the same system architecture allows us to eliminate irrelevant factors that would otherwise complicate our comparison of CCAs.

It is worth noting that the *pacing* mechanism is not limited to BBR and its variants. The pacing mechanism sends data in a controlled manner so that the sending behavior becomes smoother. It has been proven effective in TCP connections with shallow bottleneck buffers [4]. So we also implement the pacing mechanism in CUBIC. Specifically, CUBIC calculates `pacing_rate` by `cwnd` / `srtt` × `pacing_ratio`, where `cwnd` is the congestion window and `srtt` captures the statistics of RTTs in both short-term and long-term. `pacing_ratio` is 2 in slow start and is 1.2 otherwise.

Testbed: We evaluate various CCAs in the client-server mode. nginx-quic is deployed as an HTTP server so that the transmission sessions in our experiments are HTTP sessions. In our lab environment, the server resides on a Linux machine

[1] Our prototype is available at https://github.com/bpq233/oBBR
[2] RFC 9002

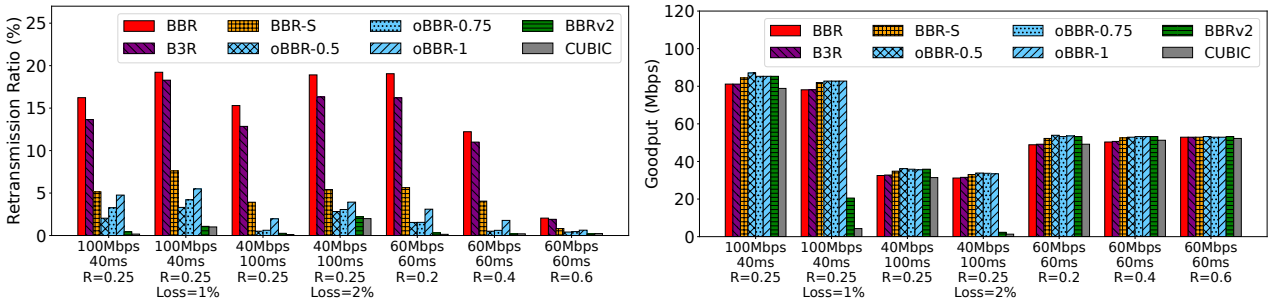

Figure 7: Performance of various CCAs in stable network environments

running Ubuntu 20.04 LTS with an Intel i5-7300HQ CPU @ 2.5 GHz (4 cores) and 16 GB RAM. The HTTP client is a simple implementation in the Chromium project [1] that also supports QUIC. For controlling the network conditions in our lab environment, *tc* [21] is used. In addition to changing the bandwidth, the latency, and the packet loss ratio, we adjust two parameters, `Limit` and `Burst`, to emulate the buffer size and the capacity of handling traffic exceeding the bandwidth at the bottleneck router. In experiments that use *tc*, `Burst` is set to 100 KB. An overview of the testbed is shown in Figure 6.

5.2 Retransmissions and Throughput

In the following experiments, we configure the client to fetch a data file of 1 GB via the HTTP protocol in a transmission session. The performance of various CCAs is measured in stable networks, variable networks, real network traces, and the Internet.

5.2.1 Stable Network Environment

Single flow: We first run various CCAs in stable network environments, where the bandwidth and the latency are set to {100 Mbps, 40 ms} or {40 Mbps, 100 ms}. We measure the retransmission ratio and the average goodput in a transmission session. The retransmission ratio is the volume of retransmitted data divided by the overall delivered data. R is 0.25 and the random packet loss ratio at 1% or 2% is also added. The results are shown in Figure 7. We can see that CUBIC and BBRv2 have the lowest retransmission ratios since they aggressively reduce the in-flight data when detecting packet losses. But both of them are vulnerable to random packet losses. Introducing a 1% random packet loss ratio reduces their goodput by 76% (BBRv2) and 94% (CUBIC), respectively. And the numbers increase to 94% (BBRv2) and 96% (CUBIC) with a 2% random packet loss ratio. On the other hand, BBR maintains its goodput across all cases but its retransmission ratios are also the highest, reaching 17.42% on average. This is because BBR does not throttle its sending rate until the packet loss ratio, whether caused by congestion or random losses, reaches a relatively high threshold of ~20%, which is not the case in our experiments. Across all scenarios, oBBR schemes consistently have high goodput, which is at least 82.76% of the link capacity. oBBR also suppresses its retransmission ratio of a session to a low value. For example, in the network with 40 Mbps bandwidth and 100 ms latency, oBBR keeps its retransmission ratios at 0.51% (μ=0.5), 0.62% (μ=0.75), and 1.97% (μ=1), respectively. By introducing 2% random packet losses, oBBR has the retransmission ratios at 2.82% (μ=0.5), 3.06% (μ=0.75), and 3.93% (μ=1). B3R lowers its sending rate to avoid excessive in-flight data, but this does not work in our experiments because of the bandwidth estimation. As a result, the retransmission ratios of B3R are similar to those of BBR. BBR-S more accurately estimates the bandwidth in our scenarios where the bottleneck router could handle burst traffic. Benefiting from this, BBR-S retransmits fewer data than BBR and B3R, but is still worse than oBBR schemes.

Then, we vary R to 0.2, 0.4, and 0.6, and fix the bandwidth and latency to {60 Mbps, 60 ms}. The results are also reported in Figure 7. With increasing bottleneck buffer size, BBR, B3R, and BBR-S reach a lower retransmission ratio accordingly because more in-flight data could be held in the buffer. The fewer retransmissions also improve their goodput. oBBR schemes have relatively high retransmission ratios when R=0.2, which are 1.54% (μ=0.5), 1.55% (μ=0.75), 3.10% (μ=1). This is due to we set a lower bound of 1.25 to g to match the 1.25×`pacing_gain` when probing for more bandwidth. When the R increases, the retransmission ratios of oBBR decrease again. For instance, when R=0.4, the retransmission ratios of oBBR are 0.51% (μ=0.5), 0.61% (μ=0.75), 1.78% (μ=1). BBRv2 and CUBIC still have the lowest transmission ratios while BBRv2 flows have goodput (similar to oBBR schemes) higher than CUBIC flows.

Multiple flows: We simultaneously launch at most 5 flows with the same CCA and measure the retransmission ratio and goodput per flow. In the experiments, the bandwidth and the latency are set to {100Mbps, 40ms}, and R is 0.5. No random packet loss is introduced. The results are shown in Figure 8. BBR still has the highest retransmission ratios, ranging from 8.41% (1-flow) to 15.25% (3-flows). As loss-based CCAs,

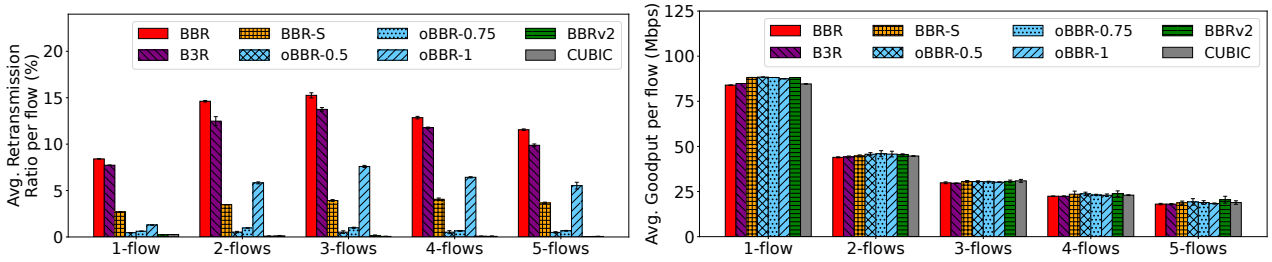

Figure 8: Performance of various CCAs in multi-flow scenarios

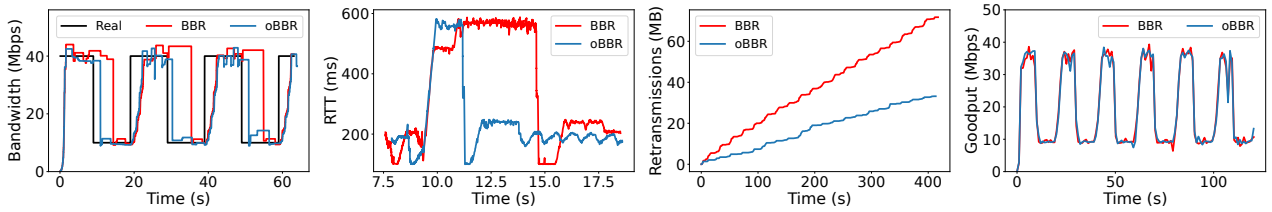

Figure 9: Performance of BBR and oBBR when bandwidth varies in a transmission session

CUBIC and BBRv2 retransmit few packets. When launching multiple BBR-like flows, i.e., BBR, B3R, BBR-S, or oBBR, higher retransmission ratios per flow are observed. The reason is that for each flow, the bandwidth overestimation is more severe: while some of the flows periodically switch to the drain cycle (`pacing_gain=0.75`) of *ProbeBW* or to *ProbeRTT* and yield bandwidth, the other flows obtain bandwidth estimate samples higher than the current ones. These overestimated samples are continuously used even though the co-existing flows scale back their sending rates. Such overestimations are alleviated when the number of flows increases (4-flows and 5-flows) because less bandwidth is allocated to each flow. For oBBR, if we set μ to 1.0 which intends to send excessive data exactly matching the bottleneck buffer, the overestimation easily results in the in-flight data being more than that. As a result, the retransmission ratio per flow of oBBR-1 grows to 7.59% at most (3-flows). When setting μ to 0.5 and 0.75, the bandwidth overestimation can hardly affect the transmission since oBBR leaves a margin in the bottleneck buffer on purpose. The highest retransmission ratios of oBBR-0.5 and oBBR-0.75 are 0.52% (2-flows) and 0.99% (3-flows), respectively. We also report goodput per flow in Figure 8, and it shows that all CCAs could effectively exploit and fairly share the link capacity.

5.2.2 Variable Bandwidth

We also set dynamic bandwidth to observe if oBBR reacts to the bandwidth drop timely as expected. In the experiments, the latency is 100 ms and the bottleneck buffer is 300 KB. We periodically change the bandwidth between 40 Mbps and 10 Mbps every 10 seconds. We evaluate oBBR and BBR, and the results are shown in Figure 9. The left two figures plot

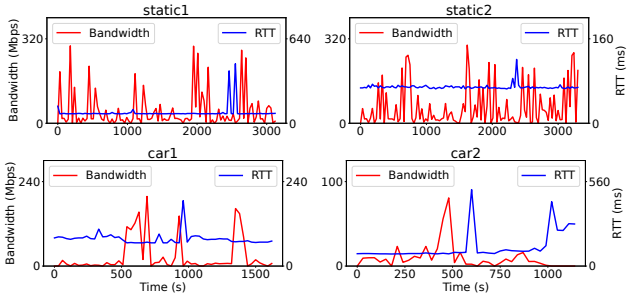

Figure 10: Characteristics of the realistic network traces

the bandwidth samples and the latency samples collected at the sender, respectively. We can see that oBBR reacts to the bandwidth drop more promptly as the bandwidth samples follow the real bandwidth change (the red line) in ~2 seconds. BBR only summarizes a more accurate bandwidth estimate after ~6 seconds because the high bandwidth estimate not only expires after 10 RTTs but also prolongs the latest RTT samples to ~600 milliseconds. While BBR sends in-flight data exceeding the bottleneck buffer, packet losses happen. Figure 9 also shows that BBR spends 2.09× more bandwidth than oBBR on retransmitting lost data while both of them achieve almost the same goodput in the transmission session.

5.2.3 Realistic Network Traces

We also test various CCAs in our lab by emulating the real network conditions following publicly available traces [41], which are collected from a major mobile operator in Ireland. Four traces are selected: *static1*, *static2*, *car1*, and *car2*. *static** means the trace is collected from a static object, probably a

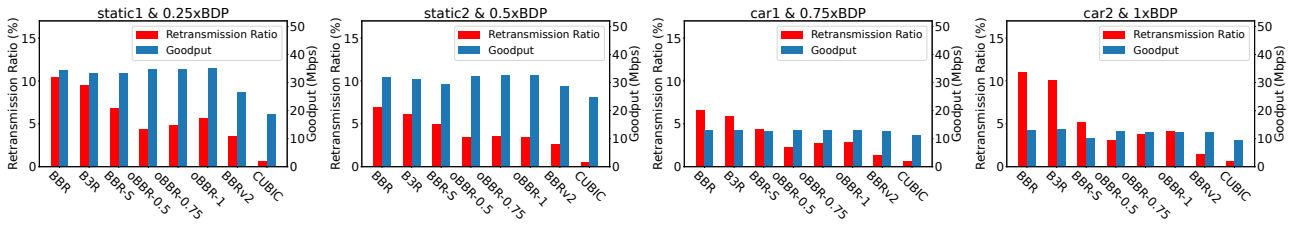

Figure 11: Performance of various CCAs in the testbed emulating the realistic network

Table 1: Characteristics of the realistic network traces

	avg. bandwidth	avg. RTT	duration
static1	45±62 Mbps	80±47 ms	3,137 s
static2	70±80 Mbps	69±6 ms	3,322 s
car1	32±64 Mbps	78±18 ms	1,645 s
car2	10±15 Mbps	118±78 ms	1,155 s

PC, and *car** means the trace is collected from a moving object, probably a car. Table 1 characterizes the traces and Figure 10 shows how the bandwidth and RTT vary over time.

We set R to different values for these traces. It is worth noting that no random packet losses are introduced in the emulations. We record retransmission ratios and average goodput of various CCAs. The results are reported in Figure 11. We can see that BBRv2 and CUBIC still have the lowest retransmission ratios in all cases, and more interestingly, they have lower goodput when compared to the oBBR schemes because oBBR reacts to the network dynamics more timely. Other performance trends are similar to those in stable network environments: BBR and B3R have the highest retransmission ratios. BBR-S has lower retransmission ratios but is still worse than oBBR. oBBR achieves the highest goodput in three out of four traces (35 Mbps of oBBR-1 in *static1*, 32 Mbps of oBBR-1 in *static2*, and 13 Mbps of oBBR-0.75 in *car1*).

5.2.4 Competitiveness

In this section, we measure how oBBR competes with other CCAs. We compare oBBR to BBR, CUBIC, and BBRv2, respectively. We also set μ to {0.5, 0.75, 1} for observing how this parameter affects oBBR's competitiveness. In experiments, the bandwidth is 40 Mbps and the latency is 100 ms. R is set to 0.5 or 1.0. For a transmission session, two flows, where one must be an oBBR flow, are launched simultaneously. We record the goodput of each flow and terminate both flows once the 1 GB data file has been fully delivered.

The results are shown in Figure 12. When competing with BBR, oBBR can hardly gain bandwidth when the buffer is as shallow as 0.5×BDP because BBR sends more in-flight data. This can be alleviated by increasing μ. Setting μ to 1 helps oBBR share 33% bandwidth. When the buffer size increases to 1×BDP, BBR performs less aggressively and even setting μ to 0.5 lets oBBR take 40.5% bandwidth. When competing with CUBIC, oBBR is aggressive in shallower buffers. But oBBR yields more bandwidth to CUBIC than BBR which takes almost all bandwidth in a 0.5×BDP buffer. In a buffer of 1×BDP, CUBIC takes up to 46.6% bandwidth share. When competing with BBRv2 in a 0.5×BDP buffer, the trend of bandwidth shares is similar to CUBIC because both of them react to packet losses. But since BBRv2 probes RTT more frequently, it takes more bandwidth in a 1×BDP buffer.

5.2.5 Internet

We further carry out the data delivery experiments in the Internet environment. We deploy the server in a data center in Virginia, USA, and the client in Shandong, China. The server is equipped with a 2-core CPU @ 2.5 GHz and 4 GB RAM, and its maximum egress bandwidth is 80 Mbps. The operating system is Ubuntu 20.04 LTS. The average link latency is about 270 ms. We experiment with data transmission in the morning, afternoon, and midnight. The results are shown in Figure 13. From the figure, CUBIC has the lowest retransmission ratio at 0.16% because it is the most sensitive algorithm to packet losses, and it also has the lowest goodput of 6.30 Mbps. BBRv2 has a retransmission ratio of 6.71% which is higher than the oBBR schemes because it fails to adjust its behavior timely in such a complex network environment, and its average goodput is 17.66 Mbps since the unavoidable packet losses in long Internet links. By setting μ to 0.75, oBBR achieves the highest goodput of 27.17 Mbps, which is 1.54× higher than BBRv2. Our oBBR schemes also have low retransmission ratios at 6.30% (μ=0.5), 6.58% (μ=0.75), and 6.85% (μ=1), which means 39.48%, 36.79%, and 36.11% fewer data are retransmitted compared to BBR-S, another BBR variant reaching a high goodput (26.94 Mbps) and a low retransmission ratio (10.41%) in the Internet experiments. These experiments indicate that oBBR could fully exploit the Internet link capacity while suppressing the retransmissions effectively.

5.3 Video Streaming

We also evaluate oBBR in video streaming, which is the dominant data delivery service on the Internet today.

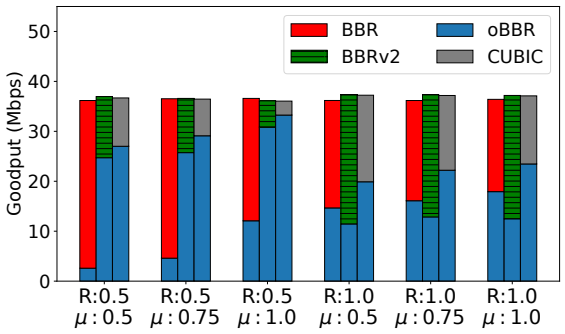

Figure 12: Competitiveness of oBBR against other CCAs

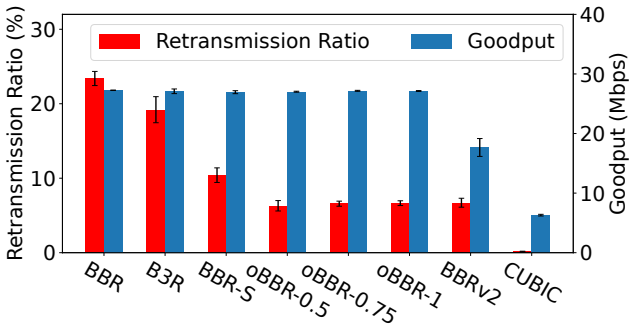

Figure 13: Performance of various CCAs on the Internet

5.3.1 Experimental setup

We build a video streaming system based on dynamic adaptive streaming over HTTP (DASH) [47], which uses HTTP as the vehicle for delivering video data. On the server, the test video is encoded into multiple bitrates and is further separated into segments of fixed length. Each of the segments is identified by a uniform resource locator (URL). We use *dash.js* [2] as the video player that fetches video segments to the client. In the player, the adaptive bitrate (ABR) algorithm dynamically switches between a buffer-based one and a throughput-based one to determine the quality of the next segment to download [46].

We use a 10-min video Big Buck Bunny as the test video, which is encoded into 10 bitrates of {254, 507, 759, 1013, 1254, 1883, 3134, 4952, 9914, 14931} Kbps. [3] For each bitrate level, the video is chunked into segments of 4 seconds. We put the server in Virginia and the client in Shandong, China. The experiments are carried out on the real-world Internet.

5.3.2 Metrics

Three metrics are used in the evaluation: average quality (AQ), quality switches (QS), and rebuffering ratio (RB).

Average quality: the average quality level (from 0 to 9) of all played segments. We expect the viewer to have a better quality of experience (QoE) with a greater AQ.

Quality switches: the number of segments with a lower quality than the previous one. A low QS means the visual quality does not fluctuate in playback, also indicating better QoE.

Rebuffering ratio: the proportion of time consumed in rebuffering. The viewers will stop watching if the rebuffering time is long. RB is calculated as $(\text{playback time} - \text{video length})/\text{video length}$.

5.3.3 Results and analysis

Figure 14 shows the performance of different CCAs in real-world video streaming sessions. For the retransmission ratio, BBR-S, BBRv2, CUBIC, and the oBBR schemes have similar performance, which is much lower than BBR and B3R. However, the low retransmission ratio of BBR-S, BBRv2, and CUBIC comes at the cost of low throughput, i.e., these algorithms fail to support the application layer to deliver high-quality videos. They have the lowest average quality, the highest quality switches, and the highest rebuffering ratio. For CUBIC, as it is too sensitive to packet loss, its throughput is suppressed at the lowest level, and the client almost selects the lowest quality level to fetch. This also decreases the rebuffering ratio of the CUBIC session. The oBBR also has low retransmission ratios (<4%) in the streaming sessions, but the user's QoE is not sacrificed. For example, by setting μ to 0.5, oBBR achieves the average quality of 6.37, the quality switches of 4, and the rebuffering ratio of 0.28%. Thus, the experiments show that oBBR benefits video streaming applications. It improves the QoE without incurring high retransmissions as BBR, which is friendly to content providers.

6 Related Work

BBR. BBR has been well-studied since it was proposed. It has been deployed in various networking scenarios, including mobiles [51], Wireless LANs [16], and clouds [17], and these research point out that the pacing mechanism in BBR may lead to poor performance. BBR is also implemented in MPTCP on Linux and the superior performance over other congestion control algorithms is observed [5]. The BBR-based MPTCP is further improved with respect to fairness and throughput in following studies [20, 35]. The performance of BBR against most existing CCAs has been extensively evaluated [31]. Such a wide range of research uncover that BBR still has problems.

Bandwidth overestimation in BBR. Since BBR selects the maximum of delivery rate samples within the latest 10 RTTs as the estimated bandwidth [56], it is easy to overestimate the bandwidth, which could result in network congestion. Chiariotti et al. [13] adopt the Adaptive Tobit Kalman Filter instead of the maximum filter for estimating bandwidth more accurately. Another study [19] suggests using Kalman Filter at

[3] Available at https://dash.akamaized.net/akamai/bbb_30fps/

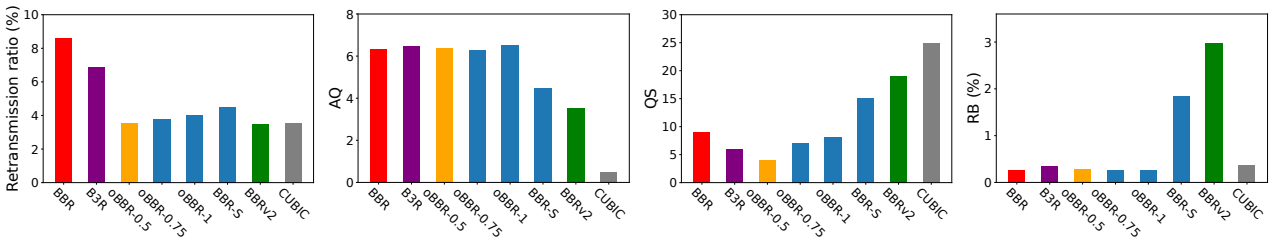

Figure 14: Performance of various CCAs in an Internet video streaming session

the receiver side and to communicate with the sender with the newly defined feedback frame of QUIC. A delayed bandwidth update strategy [48] also helps, which does not update the bandwidth estimate unless consecutive rate samples are received. BBRx [24] adjusts the estimated bandwidth based on the RTT deviation in an online learning manner. Different from these studies, we use a utility function to evaluate the delivery before and after the bandwidth update and revert to the old value if the throughput degrades.

High retransmissions in BBR. The high retransmission rates of BBR in shallow buffers have been recognized in prior studies [6, 14, 22]. To reduce the retransmissions, BBR-ACD [34] halves its congestion window when three consecutive identical RTT samples are received or the latest RTT sample is greater than 2×RTT estimate. BBR-A [33] is an extension to BBR-ACD that additionally decreases its pacing rate while the congestion window is reduced. In another study [45], the accurate propagation latency is detected when competing with CUBIC flows, and it cuts the congestion window to 1/3 of the in-flight data once there is packet loss. The existing optimizations for reducing high retransmissions in BBR flows apply loss-based multiplication to reduce the congestion window. oBBR sets the congestion window as BDP plus a proportion of bottleneck buffer so that the in-flight data will not keep decreasing even in a complex network, effectively guaranteeing the transmission throughput.

Unfairness of BBR. BBR has also been criticized for its unfairness when competing with loss-based CCAs [23, 42, 55]. Models [37, 52] are proposed for dissection. Researchers have found that a BBR flow occupies the same share of bandwidth regardless of how many loss-based flows coexist [52]. Furthermore, if too many BBR flows coexist, their advantage in throughput diminishes [37]. A learning-based model [27] has been proposed for determining the type of competing flows and mitigating the BBR's aggressiveness once the competitors are loss-based. Besides, a BBR flow gains a competitive advantage against another BBR flow if its propagation time is longer [32, 39, 40, 43, 49]. BBQ [32] enforces a cap to the span of the period that BBR pours more data for bandwidth probing. This bounds the advantage that a BBR flow with a long delivery latency can gain. BBR-E [28] reduces the in-flight data cap when the recent RTTs exceed a threshold. In another work [26], a factor γ is designed to compensate for the impacts of different RTTs, which makes the in-flight data from various flows almost the same. The fairness of BBR is out of the scope of this paper, but this is important and worth further exploring in future work.

BBRv2. To address these problems, Google introduced BBRv2 [11], which has been tested in a few studies [15, 25, 38, 54]. BBRv2 improves the fairness between flows with different RTTs and behaves less aggressively against loss-based CCAs [15, 38]. BBRv2 also reduces the high retransmission rates in BBR but at the cost of weakened resistance to packet losses [25]. In a network with bandwidth fluctuations, BBRv2 performs poorly [54]. In this work, we strive to solve the problems, i.e., degrading throughput and low responsiveness towards network dynamics, that still exist in BBRv2.

7 Conclusion

In this work, we carefully analyze the reasons for high retransmissions in BBR flows. We notice that the packet losses in shallow-buffered links are closely correlated to the in-flight data cap and the buffer size. Additionally, the slow reaction to bandwidth drops also makes BBR send excessive data to the transmission channel, thus leading to high retransmissions. To solve the problems, we design oBBR, which intelligently detects the bottleneck buffer size and scales the in-flight data cap accordingly, avoiding packet losses in the shallow-buffered links. oBBR also detects the bandwidth drop in an accurate and timely manner and tweaks its sending rate to avoid congestion. Extensive experiments in both a lab environment and the Internet have shown that oBBR significantly reduces retransmissions while still reaching a high data delivery rate.

Acknowledgments

We thank our shepherd, Eric Eide, and the anonymous reviewers for their constructive comments. This work has been partially supported by the grants from the National Natural Science Foundation of China (No.62102229 and No.62122042), the Natural Science Foundation of Shandong Province, China (No.ZR202206140010), and the Shandong Excellent Young Scientists Fund Program Overseas (No.2023HWYQ-045).

References

[1] Chromium. https://github.com/chromium/chromium/.

[2] Dash-Industry-Forum. https://github.com/Dash-Industry-Forum/dash.js/.

[3] Nginx-Quic. https://quic.nginx.org/.

[4] Amit Aggarwal, Stefan Savage, and Thomas E. Anderson. Understanding the Performance of TCP Pacing. In *Proceedings of the 19th IEEE Conference on Computer Communications (INFOCOM)*, pages 1157–1165, 2000.

[5] Phillipe Austria, Chol Hyun Park, Ju-Yeon Jo, Yoohwan Kim, Rahul Sundaresan, and Khanh D. Pham. BBR Congestion Control Analysis with Multipath TCP (MPTCP) and Asymmetrical Latency Subflow. In *Proceedings of the 12th Computing and Communication Workshop and Conference (CCWC)*, pages 1065–1069, 2022.

[6] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study. In *Proceedings of the 19th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 130–136, 2019.

[7] Neal Cardwell and Yuchung Cheng. TCP BBR Congestion Control Comes to GCP – Your Internet Just Got Faster. https://cloud.google.com/blog/products/networking/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster/, 2017.

[8] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control: Measuring Bottleneck Bandwidth and Round-Trip Propagation Time. *ACM Queue*, 14(5):20–53, 2016.

[9] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control. *Communications of the ACM*, 60(2):58–66, 2017.

[10] Neal Cardwell, Yuchung Cheng, Kevin Yang, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Luke Hsiao, Matt Mathis, Van Jacobson, Ian Swett, Bin Wu, and Victor Vasiliev. BBRv2 Update: Internet Drafts & Deployment Inside Google. https://datatracker.ietf.org/meeting/112/materials/slides-112-iccrg-bbrv2-update-00/, 2021.

[11] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, and Van Jacobson. BBR Congestion Control. https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/, 2022.

[12] Erik Carlsson and Eirini Kakogianni. Smoother Streaming with BBR. https://engineering.atspotify.com/2018/08/smoother-streaming-with-bbr/, 2018.

[13] Federico Chiariotti, Andrea Zanella, Stepán Kucera, and Holger Claussen. BBR-S: A Low-Latency BBR Modification for Fast-Varying Connections. *IEEE Access*, 9:76364–76378, 2021.

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-Learning Congestion Control. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 343–356, 2018.

[15] Jose Gomez, Elie Kfoury, Jorge Crichigno, Elias Bou-Harb, and Gautam Srivastava. A Performance Evaluation of TCP BBRv2 Alpha. In *Proceedings of the 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 309–312, 2020.

[16] Carlo Augusto Grazia, Natale Patriciello, Martin Klapez, and Maurizio Casoni. BBR+: Improving TCP BBR Performance over WLAN. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[17] Phuong Ha, Minh Vu, Tuan-Anh Le, and Lisong Xu. TCP BBR in Cloud Networks: Challenges, Analysis, and Solutions. In *Proceedings of the 41th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 943–953, 2021.

[18] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[19] Habtegebreil Haile, Karl-Johan Grinnemo, Per Hurtig, and Anna Brunström. RBBR: A Receiver-Driven BBR in QUIC for Low-Latency in Cellular Networks. *IEEE Access*, 10:18707–18719, 2022.

[20] Jiangping Han, Kaiping Xue, Yitao Xing, Peilin Hong, and David S. L. Wei. Measurement and Redesign of BBR-Based MPTCP. In *Proceedings of the ACM SIGCOMM Conference Posters and Demos*, pages 75–77, 2019.

[21] Stephen Hemminger. Network emulation with NetEm. In *Proceedings of the Linux conf au*, volume 5, 2005.

[22] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental Evaluation of BBR Congestion Control. In *Proceedings of the 25th IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, 2017.

[23] Per Hurtig, Habtegebreil Haile, Karl-Johan Grinnemo, Anna Brunström, Eneko Atxutegi, Fidel Liberal, and Åke Arvidsson. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. In *Proceedings of the 30th International Teletraffic Congress (ITC)*, pages 218–226, 2018.

[24] Jae Won Chung, Feng Li, and Beomjun Kim. BBRx: Extending BBR for Customized TCP Performance. In *Proceedings of the Proc. NetDev 0x12*, pages 262–276, 2018.

[25] Elie F. Kfoury, Jose Gomez, Jorge Crichigno, and Elias Bou-Harb. An Emulation-Based Evaluation of TCP BBRv2 Alpha for Wired Broadband. *Computer Communications*, 161:212–224, 2020.

[26] Geon-Hwan Kim and You Ze Cho. Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement. *IEEE Access*, 8:4099–4109, 2020.

[27] Geon-Hwan Kim, Yeong-Jun Song, and You-Ze Cho. Improvement of Inter-protocol Fairness for BBR Congestion Control Using Machine Learning. In *Proceedings of the International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 501–504, 2020.

[28] Geon-Hwan Kim, Yeong-Jun Song, Imtiaz Mahmud, and You-Ze Cho. Enhanced BBR Congestion Control Algorithm for Improving RTT Fairness. In *Proceedings of the 11th International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 358–360, 2019.

[29] Leonard Kleinrock. Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 43.1.1–43.1.10, 1979.

[30] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan R. Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*, pages 183–196, 2017.

[31] Jinting Lin, Lin Cui, Yuxiang Zhang, Fung Po Tso, and Quanlong Guan. Extensive Evaluation on the Performance and Behaviour of TCP Congestion Control Protocols under Varied Network Scenarios. *Computer Networks*, 163:106872, 2019.

[32] Shiyao Ma, Jingjie Jiang, Wei Wang, and Bo Li. Fairness of congestion-based congestion control: Experimental evaluation and analysis. *arXiv preprint arXiv:1706.09115*, 2017.

[33] Imtiaz Mahmud and You-Ze Cho. BBR Advanced (BBR-A) - Reduced Retransmissions with Improved Fairness. *ICT Express*, 6(4):343–347, 2020.

[34] Imtiaz Mahmud, Geon-Hwan Kim, Tabassum Lubna, and You-Ze Cho. BBR-ACD: BBR with Advanced Congestion Detection. *Electronics*, 9(1):136, 2020.

[35] Imtiaz Mahmud, Tabassum Lubna, Yeong-Jun Song, and You-Ze Cho. Coupled Multipath BBR (C-MPBBR): A Efficient Congestion Control Algorithm for Multipath TCP. *IEEE Access*, 8:165497–165511, 2020.

[36] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The Great Internet TCP Congestion Control Census. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):45:1–45:24, 2019.

[37] Ayush Mishra, Wee Han Tiu, and Ben Leong. Are we heading towards a BBR-dominant internet? In *Proceedings of the 22th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 538–550, 2022.

[38] Aarti Nandagiri, Mohit P. Tahiliani, Vishal Misra, and K. K. Ramakrishnan. BBRvl vs BBRv2: Examining Performance Differences through Experimental Evaluation. In *Proceedings of the 26th IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, 2020.

[39] Wansu Pan, Xiaofeng Li, Haibo Tan, Jinlin Xu, and Xiru Li. Improvement of RTT Fairness Problem in BBR Congestion Control Algorithm by Gamma Correction. *Sensors*, 21(12):4128, 2021.

[40] Wansu Pan, Haibo Tan, Xiru Li, and Xiaofeng Li. Improved RTT Fairness of BBR Congestion Control Algorithm Based on Adaptive Congestion Window. *Electronics*, 10(5):615, 2021.

[41] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. Beyond Throughput, the Next Generation: A 5G Dataset with Channel and Context Metrics. In *Proceedings of the 11th ACM SIGMM Conference on Multimedia Systems (MMSys)*, pages 303–308, 2020.

[42] Kanon Sasaki, Masato Hanai, Kouto Miyazawa, Aki Kobayashi, Naoki Oda, and Saneyasu Yamaguchi. TCP Fairness Among Modern TCP Congestion Control Algorithms Including TCP BBR. In *Proceedings of the 7th IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–4, 2018.

[43] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. Towards a Deeper Understanding of TCP BBR Congestion Control. In *Proceedings of the 17th IFIP International Conference on Networking*, pages 109–117, 2018.

[44] Tarun Singhania, Wasim Arif, and Debarati Sen. B3R: A New Approach to BBR Congestion Control for Shallow Buffers. In *Proceedings of the Advanced Communication Technologies and Signal Processing (ACTS)*, pages 1–6, 2021.

[45] Yeong-Jun Song, Geon-Hwan Kim, and You-Ze Cho. Improvement of Cyclic Performance Variation between TCP BBR and CUBIC. In *Proceedings of the 25th Asia-Pacific Conference on Communications (APCC)*, pages 1–6, 2019.

[46] Kevin Spiteri, Ramesh K. Sitaraman, and Daniel Sparacio. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2s):67:1–67:29, 2019.

[47] Thomas Stockhammer. Dynamic Adaptive Streaming over HTTP: Standards and Design Principles. In *Proceedings of the 2nd ACM SIGMM Conference on Multimedia Systems (MMSys)*, pages 133–144, 2011.

[48] Bo Su, Xianliang Jiang, Guang Jin, and Haiming Chen. Rethinking the Rate Estimation of BBR Congestion Control. *Electronics Letters*, 56(5):239–241, 2020.

[49] Weifeng Sun, Minghan Jia, Guanghao Zhang, and Zun Wang. RFBBR: A RTT Faireness Awared Algorithm Based on BBR. In *Proceedings of the International Conferences on Smart Internet of Things (SmartIoT)*, pages 124–131, 2020.

[50] Santiago Vargas, Rebecca Drucker, Aiswarya Renganathan, Aruna Balasubramanian, and Anshul Gandhi. BBR Bufferbloat in DASH Video. In *Proceedings of the 30th The Web Conference (WWW)*, pages 329–341, 2021.

[51] Santiago Vargas, Gautham Gunapati, Anshul Gandhi, and Aruna Balasubramanian. Are Mobiles Ready for BBR? In *Proceedings of the 22th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 551–559, 2022.

[52] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the 19th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 137–143, 2019.

[53] Ranysha Ware, Matthew K Mukerjee, Justine Sherry, and Srinivasan Seshan. The Battle for Bandwidth: Fairness and Heterogeneous Congestion Control. *USENIX Symposium on Networked Systems Design and Implementation Poster*, 2018.

[54] Furong Yang, Qinghua Wu, Zhenyu Li, Yanmei Liu, Giovanni Pau, and Gaogang Xie. BBRv2+: Towards Balancing Aggressiveness and Fairness with Delay-Based Bandwidth Probing. *Computer Networks*, 206:108789, 2022.

[55] Yuxiang Zhang, Lin Cui, and Fung Po Tso. Modest BBR: Enabling Better Fairness for BBR Congestion Control. In *Proceedings of the 23th International Symposium on Computers and Communications (ISCC)*, pages 646–651, 2018.

[56] Zhenzhe Zhong, Isabelle Hamchaoui, Rida Khatoun, and Ahmed Serhrouchni. Performance Evaluation of CQIC and TCP BBR in Mobile Network. In *Proceedings of the 21th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 1–5, 2018.