# BBRv2+: Towards Balancing Aggressiveness and Fairness with Delay-based Bandwidth Probing

Furong Yang<sup>a,b,c</sup>, Qinghua Wu<sup>a</sup>, Zhenyu Li<sup>a,\*</sup>, Yanmei Liu<sup>d</sup>, Giovanni Pau<sup>e,f</sup>, Gaogang Xie<sup>g</sup>

<sup>a</sup>Institute of Computing Technology, Chinese Academy of Sciences, China <sup>b</sup>University of Chinese Academy of Sciences, China <sup>c</sup>Sorbonne University, France <sup>d</sup>Alibaba Group, China <sup>e</sup>University of Bologna, Italy <sup>f</sup>University of California, Los Angeles, USA <sup>g</sup>Computer Network Information Center, Chinese Academy of Sciences, China

#### Abstract

BBRv2, proposed by Google, aims at addressing BBR's shortcomings of unfairness against loss-based congestion control algorithms (CCAs) and excessive retransmissions in shallow-buffered networks. In this paper, we first comprehensively study BBRv2's performance under various network conditions and show that BBRv2 mitigates the shortcomings of BBR. Nevertheless, BBRv2's benefits come with several costs, including the slow responsiveness to bandwidth dynamics as well as the low resilience to random losses. We then propose BBRv2+ to address BBRv2's performance issues without sacrificing its advantages over BBR. To this end, BBRv2+ incorporates delay information into its path model, which cautiously guides the aggressiveness of its bandwidth probing to not reduce its fairness against loss-based CCAs. BBRv2+ also integrates mechanisms for improved resilience to random losses as well as network jitters. Extensive experiments demonstrate the effectiveness of BBRv2+. Especially, it achieves 25% higher throughput and comparable queuing delay in comparison with BBRv2 in high-mobility network scenarios.

Keywords: Congestion Control, BBR, BBRv2

#### 1. Introduction

Congestion control has been one of the active research topics in computer networks since it was introduced in the 1980s [1]. More than three decades of research on congestion control have brought us a plethora of congestion control algorithms (CCAs) and TCP variants, aiming at efficient utilization of available bandwidth while fairly sharing the bottleneck bandwidth among multiple flows. For instance, Linux kernel alone has more than 15 different CCAs [2]. While we see many recent proposals on learning-based CCAs (e.g. Remy [3], Aurora [4], PCC-Vivace [5], Indigo [6], Orca [2]), the wildly deployed CCAs today are still classic ones (e.g. Cubic [7], BBR [8]).

In this paper, we focus on BBR and its upgrade, BBRv2, as BBR has been used by 22% of the Alexa Top 20K websites [9] and BBRv2 will likely replace BBR in the near future<sup>1</sup>. BBR is a rate-based CCA that sets its sending rate based on the measured bottleneck bandwidth (*BtlBW*) and round trip propagation time (*RTprop*). That said, instead of reacting to congestion signals such as losses or delay dynamics, BBR tries to actively operate at Kleinrock's optimal operating point [11] to maximize throughput without incurring standing queues at a bottleneck link. The previous empirical studies [12–17] have disclosed several shortcomings

<sup>\*</sup>Corresponding author

*Email addresses:* yangfurong@ict.ac.cn (Furong Yang), wuqinghua@ict.ac.cn (Qinghua Wu),

zyli@ict.ac.cn (Zhenyu Li), miaoji.lym@alibaba-inc.com (Yanmei Liu), giovanni.pau@unibo.it (Giovanni Pau), xie@cnic.cn (Gaogang Xie)

Preprint submitted to Computer Networks

<sup>&</sup>lt;sup>1</sup>As of March 2021, Google has finished roll-out of BBRv2 for internal TCP traffic, and tuning performance to enable roll-out for external traffic [10].

of BBR: (1) it causes excessive retransmissions in shallow-buffered networks; (2) it is not fair when competing with flows using loss-based CCAs (e.g. Cubic) and BBR flows with different Round Trip Times (RTTs); (3) its performance degrades when network jitters are high.

To address these issues in BBR, Google proposed BBRv2 [18] that inherits most of the design principles from BBR, while reacts to losses and Explicit Congestion Notification (ECN) marks for better coexistence with loss-based CCAs and being less aggressive in shallow-buffered networks. Given that BBRv2 may eventually replace BBR, understanding how BBRv2 actually performs is of great importance for improved performance and fairness. We have also seen several studies [19–23] on measuring BBRv2, which have shown that, in comparison with BBR, BBRv2 improves the inter-protocol fairness against loss-based CCAs and reduce retransmissions in shallow-buffered networks. Nevertheless, we found in this paper, these improvements come with several costs, including the low resilience to random packet losses and the slow responsiveness to bandwidth dynamics.

In this paper, we first evaluate BBRv2 in various network conditions with an emphasis on the reasons behind the observed performance issues. Our key observations from the empirical study of BBRv2 are as follows:

- Due to its conservative strategies in bandwidth probing and inflight cap estimation, BBRv2 achieves better inter-protocol fairness against loss-based CCAs in shallow-buffered networks than BBR. On the other hand, these strategies also make BBRv2 slightly less competitive than BBR in terms of throughput under moderate buffers. BBRv2 also improves RTT fairness among flows, compared with BBR.
- In shallow-buffered networks, retransmissions of BBRv2 are significantly reduced compared with that of BBR. However, the throughput of BBRv2 is 13%~16% lower than that of BBR under shallow buffers, as BBRv2 limits its inflight size to about  $0.85 \times$  BDP for most of the time in these networks.
- BBRv2 is less resilient to random losses than BBR. Interestingly, we find that carefully tuning the loss threshold parameter in BBRv2 according to bottleneck buffer sizes can enhance BBRv2's loss resilience without sacrificing its advantages in retransmission and fairness.

- BBRv2 is less responsive to bandwidth dynamics than BBR, which leads to low bandwidth utilization and high queuing delay in networks with bandwidth dynamics. The long bandwidth probing interval and the long expiry time of bottleneck bandwidth estimation are the two major contributors.
- Like BBR, BBRv2's performance still suffers from congestion window (*cwnd*) exhaustion in high-jitter networks that are not rare in wireless scenarios [16, 24–27].

The results of our empirical study of BBRv2 raise one question to us: are we able to improve BBRv2's performance while keeping its advantages in retransmission and fairness? If so, how do we achieve this goal?

Compared with BBR, BBRv2's shortcomings lie in the lower loss resilience and slower responsiveness to bandwidth dynamics. On one hand, the issue regarding loss resilience can be mitigated by carefully tuning the loss threshold parameter in BBRv2. On the other hand, we can increase the aggressiveness of BBRv2 in bandwidth probing to improve its responsiveness to bandwidth dynamics. But, this aggressiveness needs to be cautiously guided, as blindly behaving aggressively may cause unfairness against loss-based CCAs like BBR. Currently, the aggressiveness of bandwidth probing in BBR and BBRv2 is either hard-coded or pre-configured according to designers' experience without the perception of the network environment. This kind of unguided aggressiveness may make the bandwidth probing be either over-aggressive (like BBR) or over-conservative (like BBRv2) in certain environments. Thus, the feedback from the network environment needs to be considered in BBRv2's bandwidth probing strategy to guide the aggressiveness of bandwidth probing.

To address the above gap, we propose BBRv2+. Firstly, BBRv2+ integrates delay information into its path model, which serves as the feedback to guide its aggressiveness in bandwidth probing. Secondly, to utilize the delay information to guide the aggressiveness of BBRv2+'s bandwidth probing, the state-machine of BBRv2 is partially redesigned. In doing so, BBRv2+ balances between the aggressiveness in probing for more bandwidth and the fairness against loss-based CCAs. Thirdly, to avoid being suppressed when co-existing with loss-based CCAs in deep-buffered networks because of using the delay information, BBRv2+ incorporates a dual-mode mechanism, where it switches to use BBRv2's state-machine if no RTT sample approaching RTprop is observed for a long time period and returns back to use the redesigned statemachine if it constantly observes RTT samples approaching RTprop. Finally, as an optimization, BBRv2+ addresses the *cwnd* exhaustion problem in high-jitter networks by compensating its estimated Bandwidth Delay Product (BDP) according to observed jitters; the compensation mechanism allows the estimated BDP to be close to the actual BDP, and can also be applied to BBR and BBRv2.

Extensive experiments based on both Mininet and Mahimahi with real-world traces show that compared with BBRv2, BBRv2+ succeeds to balance the aggressiveness of bandwidth probing and the fairness against loss-based CCAs, improves the resilience to network jitters, and, particularly, achieves 25% higher throughput and comparable queuing delay in high-mobility scenarios where the bandwidth is very dynamic.

To summarize, the contributions of this paper are three-fold: (1) a deep dive into BBRv2 that reveals its pros and cons, compared with BBR; (2) BBRv2+ that addresses the shortcomings of BBRv2 while barely sacrificing BBRv2's advantages; (3) extensive experiments demonstrating that BBRv2+ meets its design goals. We opensource BBRv2+ to the research community for further test and improvement [28].

The remainder of this paper is organized as follows. We first give an overview of BBR and BBRv2 in §2. Then, a deep dive into BBRv2, which motivates the design of BBRv2+, is presented in §3. Next, the design and implementation of BBRv2+ is described in §4, and the evaluation of BBRv2+ is shown in §5. After that, we present the related work in §6. Finally, the paper is concluded in §7.

# 2. Background: an overview of BBR and BBRv2

BBR aims at maximizing throughput while keeping the lowest latency; it requires accurate measurements of both BtlBW and RTprop. Since these two variables cannot be measured simultaneously, BBR introduces a state-machine-based method that alternatively estimates BtlBW and RTprop.

As illustrated in Fig. 1, there are four states in the BBR life-cycle. BBR uses *pacing\_gain* to control the sending behavior—to probe for more bandwidth, to drain the queue at the bottleneck link, or



Fig. 1: Illustration of BBR life-cycle



Fig. 2: Illustration of BBRv2 life-cycle

to cruise at the speed of *BtlBW*. Firstly, BBR starts the Startup state which exponentially increases the inflight size and sending rate by setting *pacing\_gain* to  $2/\ln(2)$ . BBR transits into the Drain state when it is in the plateau of BtlBW for three RTTs. In the Drain state, BBR reduces *pacing\_gain* to  $\ln(2)/2$  to drain the standing queue at the bottleneck link induced during Startup. After the above two stages, BBR has successfully built the path model, with RTprop measured at the beginning of Startup and BtlBW measured at the end of Startup. After that, BBR switches to a steady phase where BBR alternatively runs in the ProbeBW and ProbeRTT state. During the ProbeBW state, BBR sets pacing\_gain to 1.0 to cruise at Kleinrock's optimal point for 6 cycles, then sets *pacing\_gain* to 1.25 to explore more bandwidth for 1 cycle and thereafter sets pacing\_gain to 0.75 to drain the possible standing queue for 1 cycle. In the ProbeRTT state, BBR reduces its inflight size to  $4 \times MSS$  (Max Segment Size) and waits for  $max\{RTT, 200ms\}$  to measure an updated value of *RTprop*.

As observed in numerous previous studies [12–16], BBR has two key issues: the unfairness of bandwidth share with loss-based CCAs and the high retransmission rate in shallow-buffered networks.

The reasons behind these issues are that BBR is congestion signal agnostic and is over-aggressive when probing for more bandwidth. To mitigate the problems above, Google proposed BBRv2, which inherits most of BBR's design (e.g. the core principle, the overall building blocks, etc.) yet redesigns the ProbeBW state, as illustrated in Fig. 2.

BBRv2 adds measurements of packet loss and DCTCP-style ECN marks [29] for estimating the capacity of a bottleneck link. Specifically, it introduces *inflight\_lo* and *bw\_lo* as the short-term lower bounds of inflight size and sending rate respectively, in order to capture the temporary status of the network path (e.g. cross-traffic takes a share of capacity); it uses *inflight\_hi* as the long-term upper bound of inflight size to reduce the likelihood of packet loss. To avoid recklessly probing for more bandwidth, BBRv2 decomposes BBR's ProbeBW state into four sub-states: ProbeCruise, ProbeRefill, ProbeUp, and ProbeDown.

**ProbeCruise:** In ProbeCruise, BBRv2 sets pacing\_gain to 1. If any loss or ECN mark occurs, BBRv2 updates inflight\_lo and bw\_lo to max{ $(1 - \beta) \times inflight_lo, BtlBW_{curr}$ } and max{ $(1 - \beta) \times bw\_lo, inflight_{curr}$ } respectively, where  $BtlBW_{curr}$ and  $inflight_{curr}$  are the current measurements of bandwidth and inflight size.

**ProbeRefill:** When BBRv2 has been in Probe-Cruise for a period of T (T is determined in ProbeDown), BBRv2 transits to ProbeRefill, by setting *inflight\_lo* and  $bw_lo$  to  $+\infty$  to refill the "pipe" with BDP-sized inflight data, which lasts for one RTT. The goal of this state is to avoid early losses before the capacity is fully utilized in shallowbuffered networks since BBRv2 will accelerate in the following ProbeUp state, which may lead the bottleneck buffer to overflow.

**ProbeUp:** During ProbeUp, BBRv2 sets *pac-ing\_gain* to 1.25 to probe for more available bandwidth. This state ends either when the current loss rate exceeds a pre-defined explicit loss threshold (2%), (or the ECN mark rate exceeds an ECN threshold), or when the inflight size reaches  $1.25 \times$  BDP and at least one *RTprop* has passed. In the former case, BBRv2 sets *inflight\_hi* to the current inflight size.

**ProbeDown:** During ProbeDown, BBRv2 drains the potential queue at the bottleneck link by setting *pacing\_gain* to 0.75. BBRv2 also sets the duration (*T*) for the next ProbeCruise state to  $min\{rand(2,3), \frac{BDP}{MSS} \times RTT\}$  seconds, where rand(2,3) means a number between two and three. The intention of T is to match the interval between loss recovery epochs of Reno for TCP fairness. The ProbeDown state ends when BBRv2 cuts its inflight size below the minimum value between  $1 \times BDP$  and  $0.85 \times inflight_hi$ . Thereafter, BBRv2 transits to the next ProbeCruise state.

As the bandwidth probing behaviors of BBRv2 are different from BBR, BBRv2 no longer uses a 10-RTT-windowed max\_filter to track the estimation of BtlBW, and it rather takes the maximum bandwidth measured in the recent two ProbeBW stages as the estimation of BtlBW, which ensures that the bandwidth samples from ProbeUp states are considered.

Summary of BBRv2: The inflight bound mechanism, driven by losses or ECN marks, and the less aggressive bandwidth probing strategy make BBRv2 more conservative than BBR in shallowbuffered networks, which thus mitigates the problems regarding excessive retransmissions and unfairness. However, the two changes can potentially reduce BBRv2's performance under random losses and bandwidth dynamics, because, compared with BBR, BBRv2 probes for bandwidth less frequently, takes more time to expire *BtlBW* estimations, and slows down constantly if the random loss rate exceeds 2%.

# 3. A Deep Dive into BBRv2

In this section, we conduct extensive measurements to investigate the improvements and overheads of BBRv2, in comparison with BBR. Our key observations include: (1) BBRv2 improves the inter-protocol fairness and RTT fairness, and also reduces retransmissions in shallow-buffered networks; this observation reaffirms those in the previous studies [19-23]; (2) the improvements of BBRv2 come with the cost of the low resilience to random loss and the slow responsiveness to bandwidth dynamics. That said, it fails to achieve a balance between the aggressiveness in probing for more bandwidth and the fairness against lossbased CCAs; (3) like BBR, BBRv2 experiences low throughput in high-jitter networks because of underestimation of BDP.

#### 3.1. Methodology

We utilize Mininet [30] to build an emulationbased testbed. The testbed, whose topology is shown in Fig. 3, was run on a server with 8 Intel Xeon Platinum cores and 32GB of memory. The operating system is Ubuntu 18.04.5 with BBR and BBRv2 [31] installed. Linux *tc-netem* [32] is used to emulate different network conditions (e.g. router buffer size, link speed, RTT, random loss rate, jitter). Iperf3 [33] generates TCP traffic between senders and receivers. During the transmission, various performance metrics (e.g. RTT, throughput, retransmissions, inflight bytes) are measured by tcpdump [34] and tcptrace [35]. Moreover, a set of internal variables (e.g. cwnd, pacing\_rate, RT*prop*, BtlBW) in BBR and BBRv2 are reported by Linux kernel module and the backlog information of the standing queue in bottleneck routers (R2 and R3) is reported by tc [36]. Each set of experiments is repeated five times and the average results are reported.



Fig. 3: Mininet testbed

#### 3.2. Fairness

We first evaluate the fairness of BBRv2. Two types of fairness are investigated: the inter-protocol fairness against loss-based CCAs and RTT fairness. In this set of experiments, two flows start simultaneously, one from H1 to H3 and the other from H2 to H4, and last for three minutes for the convergence of throughput. The bottleneck bandwidth is fixed at 40Mbps without network jitters or random losses.

We use Jain's fairness index [37] ( $\mathcal{F}$ ) of the two flows as the metrics of fairness, calculated according to Eq. 1, where  $T_i$  is the average throughput of the *i*-th flow.

$$\mathcal{F} = \frac{(T_1 + T_2)^2}{2 * (T_1^2 + T_2^2)} \tag{1}$$

 $\mathcal{F} = 1$  indicates the maximum fairness where two flows have the same average throughput, and  $\mathcal{F} =$ 0.5 represents that one flow's throughput is zero and the fairness is minimized. As the size of the bottleneck buffer also impacts fairness results, we varied the buffer size to study their relationship.



**Fig. 4:** Inter-protocol fairness of BBR/BBRv2 under different buffer sizes.

#### 3.2.1. Inter-protocol fairness

In the experiment, the flow from H1 to H3 uses either BBR or BBRv2, and that from H2 to H4 uses Cubic, which is the default CCA in Linux and MacOS. The RTTs of the two paths of the two flows are set to 40ms.

Fig. 4 shows the inter-protocol fairness results for both BBR and BBRv2. We can observe that compared with BBR, BBRv2 significantly improves Jain's fairness index when the buffer is shallow (i.e., less than  $2 \times$  BDP). This is due to the fact that BBRv2 reacts to losses caused by buffer overflow and bounds the inflight size by using *inflight\_hi* and *inflight\_lo.* When the bottleneck buffer becomes deeper, Cubic obtains more bandwidth than BBR or BBRv2. This is because that the inflight size of both BBR and BBRv2 is limited by about  $2 \times$  BDP, while Cubic's inflight size can go beyond this value under deep buffers. As the two flows experience similar RTTs, a larger inflight size means higher throughput. We also note that BBRv2 is less competitive than BBR under moderate buffers. The reason lies in that BBRv2 is more conservative in bandwidth probing and inflight cap estimation.

# 3.2.2. RTT fairness

In this experiment, both flows use the same CCA, either BBR or BBRv2. The path between H1 and H3 has an RTT of 40ms and that between H2 and H4 has an RTT of 150ms. Fig. 5 shows the RTT fairness results of BBR and BBRv2, where we set the buffer size to x times of the BDP of the path between H2 and H4.

When the buffer is quite shallow (i.e.,  $0.2 \times BDP$ ), BBR has good fairness. When the buffer size becomes larger, the BBR flow with longer RTT gradually occupies all the bandwidth and starves the flow with shorter RTT. The reason for the poor RTT fairness of BBR is well documented by the



**Fig. 5:** RTT fairness of BBR/BBRv2 under different buffer sizes.

previous studies [14, 17]. The bandwidth probing of BBR leads the aggregated sending rate of two flows to exceed bottleneck bandwidth, thus, forming a persistent queue at the bottleneck link. As the inflight cap of BBR is proportional to RTprop, the flow with longer RTT pours more data into the bottleneck buffer, thus, leading to a larger share of the bottleneck link's capacity. This problem is not severe under shallow buffers, as excess packets are mostly dropped instead of forming a persistent queue.

Compared with BBR, BBRv2 has better RTT fairness, especially under deep buffers. The likely reason is three-fold. First, when the buffer size is moderate, losses are triggered due to buffer overflow, and then both flows reduce their inflight size proportional to BDP. As the flow with longer RTT has a larger BDP, it reduces its inflight size more than the flow with shorter RTT. Second, when BBRv2 flows are cruising at the speed of *BtlBW*, they always try to leave headroom<sup>2</sup> for other flows to explore the bandwidth. Third, BBRv2 enters the ProbeRTT state more often than BBR, thus, leading BBRv2 flows to yield occupied capacity more frequently.

# 3.3. Retransmission and throughput

As one of BBRv2's design goals is to reduce unnecessary retransmissions in shallow-buffered networks, next we investigate whether BBRv2 achieves this design goal. The experimental setup is similar to that in the previous work [12], where the bottleneck bandwidth varies in  $10\sim750$  Mbps and the path RTT varies in  $5\sim150$  ms as these values are commonly employed in modern networks [12,



Fig. 6: The heatmap of the retransmission rate of BBR/BBRv2 under various network conditions. The numbers in squares are retransmission rates in percentage.



**Fig. 7:** The heatmap of *Tput\_Gain* (in percentage) in (a) shallow and (b) deep buffered networks.

14, 38]. The buffer size at the bottleneck link is set to 100KB to emulate a shallow-buffered network because 100KB is less than the BDP of most bandwidth-RTT combinations in our setup. One TCP flow from H1 to H3 runs for 30 seconds and the retransmission rate is recorded for each setup.

The heatmaps in Fig. 6 show the retransmission rates of BBR and BBRv2 under various network conditions. We can observe that the retransmission rate of BBRv2 is significantly reduced compared with that of BBR, especially when the BDP is larger than 400KB (i.e. the buffer size  $\leq 0.25 \times$  BDP).

The lower retransmission rate of BBRv2 in shallow-buffered networks stems from the fact that BBRv2 reacts to packet losses, while BBR does not. In the Startup and ProbeUp state, BBRv2 tries to send at a rate higher than the bottleneck bandwidth, which leads to excessive losses (i.e. loss rate  $\geq 2\%$ ). The excessive losses trigger *inflight\_hi* to be set to the current inflight size that is likely close to BDP in shallow-buffered networks. As a result, BBRv2's inflight size is bounded beblow  $0.85 \times inflight_hi$  in ProbeCruise because it tries to leave headroom for other flows to explore bandwidth. Since BBRv2 flows spend most of their lifecycle in

<sup>&</sup>lt;sup>2</sup>BBRv2 always limits its inflight size below  $0.85 \times in-flight_hi$  to leave headroom for faster throughput convergence with other flows if there is any.

Probe Cruise, the average throughput of BBRv2 is expected to be 15% lower than the available bandwidth.

That said, BBRv2 trades off throughput against retransmission in shallow-buffered networks. To verify this, we compute the throughput gain of BBR over BBRv2 ( $Tput\_Gain$ ), which is defined in Eq. 2, where  $Tput_{BBR}$  (resp.  $Tput_{BBRv2}$ ) is the average throughput of a BBR (resp. BBRv2) flow over 30 seconds.

$$Tput\_Gain = \frac{Tput_{BBR} - Tput_{BBRv2}}{Tput_{BBRv2}} \qquad (2)$$

Fig. 7a plots the  $Tput\_Gain$  under various network conditions. We can observe that in the network conditions where BBRv2 reduces the retransmission rate (when the BDP exceeds 400KB), BBRv2 achieves lower throughput than BBR. Specifically, the throughput of BBRv2 is  $13\% \sim 16\%$ lower than that of BBR in these cases, which coincides with our analysis.

In deep-buffered networks, however, the packet losses are much less often. It is thus expected that the throughput of BBR and BBRv2 are comparable. This is confirmed by the results in Fig. 7b, where the buffer size is configured at 10MB, larger than the BDP of most of the bandwidth-RTT combinations in our setup. The throughput differences between BBR and BBRv2 are indeed marginal in these networks.

#### 3.4. Resilience to random losses

Several early tests [20, 22, 23] have shown that BBRv2 is less resilient to random losses than BBR, since BBRv2 limits its inflight size by the *inflight\_lo* and *inflight\_hi*, which both react to all types of losses. In BBRv2, there are two parameters that decide how the *inflight\_lo* and *inflight\_hi* react to losses. One is the explicit loss threshold ( $\alpha$ ) and the other one is the *inflight\_lo* reduction factor ( $\beta$ ). In our experiments, we investigate BBRv2 variants with different  $\alpha$  and  $\beta$  under random loss, where each specific BBRv2 variant is referred as BBRv2( $\alpha$ ,  $\beta$ ). For  $\alpha$ , we cap it at 20% to match the maximum loss rate that BBR can tolerate; for  $\beta$ , we only evaluate the difference between the case with (i.e.  $\beta = 0.3$ ) and without it i.e. ( $\beta = 0$ )<sup>3</sup>.



Fig. 8: Avg. throughput against different random loss rates (buffer size =  $32 \times$  BDP).

In the experiment, the bottleneck bandwidth is set to 40Mbps, and the path RTT is 40ms. The buffer size is set to  $32 \times$  BDP to avoid packet loss due to buffer overflow. The random loss rate ranges from 0% to 30%.

Fig. 8 reports the average throughput of each CCA against random loss rates. We observe that the throughput of BBRv2 drops significantly after the random loss rate reaches 2%. There is a clear sign that the  $\alpha$  impacts the loss resilience of BBRv2: as the  $\alpha$  increases, we can observe the improvement of loss resilience of BBRv2. For instance, with a 10% random loss rate, BBRv2(20%, 0.3) reaches around half of the maximum bandwidth while BBRv2's throughput nearly drops to zero. The impact of  $\beta$  is also remarkable: the loss resilience of BBRv2(20%, 0.3) is lower than that of BBRv2(20%, 0) that performs similar to BBR.

The above results indicate that BBRv2's loss resilience can be improved via raising the  $\alpha$ . Yet, there is a concern — how does the  $\alpha$  impact the retransmission rate in shallow-buffered networks as we already saw that BBRv2 alleviates the retransmission issue by setting *inflight\_hi* upon the loss rate exceeding  $\alpha$  to lower down its inflight size (see §3.3).

To investigate the aforementioned concern, we further extended the experiments by considering more BBRv2 variants ( $\alpha \in [2\%, 100\%]$ ,  $\beta = 0.3$ ) and more configurations on buffer size (buffer size  $\in \{0.2, 0.5, 1.0, 1.5, 2.0\} \times BDP$ ). Fig. 9 plots the retransmission rates of all those BBRv2 variants under 0% random loss rate (to eliminate the impact of random losses on retransmission rate), which shows the impact of buffer size. Two observations are notable. Firstly, we observe that the retransmission rate increases when the  $\alpha$  exceeds a certain point, which depends on the bottleneck buffer size. The  $\alpha$ values beyond the turning points are too high to be reached by the temporary loss rate, thus, limiting

 $<sup>^{3}\</sup>mathrm{The}$  default value 0.3 is necessary for BBRv2 to co-exist with Cubic [31]



Fig. 9: Retransmission rates versus loss thresholds ( $\alpha$ ) under networks with various buffer sizes. The errorbars in the figure represent the standard deviations of retransmission rate. Note that the  $\beta$  was fixed at 0.3 for all experiments; the default  $\alpha$  in BBRv2 is 2% (the first data point of every line).



Fig. 10: Inter-protocol fairness of BBRv2(20%, 0.3).

the efficacy of *inflight\_hi*. Secondly, if the buffer size is large enough (i.e.  $2 \times$  BDP in our experiments), the retransmissions are eliminated, thus, the value of  $\alpha$  becomes irrelevant.

Another concern about lifting  $\alpha$  is the impact on the inter-protocol fairness because the larger  $\alpha$  is, the slower reaction of BBRv2 to losses is, which makes BBRv2 more aggressive to loss-based CCAs. To investigate this concern, we test the inter-protocol fairness of BBRv2(20%, 0.3) using the same setup in  $\S3.2.1$ , and plot the results in Fig. 10. In comparison with Fig. 4b, we can see that the inter-protocol fairness of BBRv2 is indeed worsened in the case of extremely shallow buffer  $(0.2 \times BDP)$  due to the increased aggressiveness caused by a larger  $\alpha$ . Nevertheless, we also observe that the fairness index is improved under moderate buffers because the increased aggressiveness also makes BBRv2 less vulnerable to Cubic when the bottleneck buffer becomes larger.

Summary of random loss resilience: The loss resilience of BBRv2 can be improved by raising the loss threshold  $\alpha$ . Nevertheless, the threshold  $\alpha$  should be carefully tuned according to the bot-tleneck buffer size to avoid increasing retransmis-

sions and being too aggressive to loss-based CCAs in extremely shallow-buffered networks.

#### 3.5. Responsiveness to bandwidth dynamics

In networks with highly dynamic available bandwidth [27, 39, 40], BBRv2's bandwidth probing may fail to quickly adapt to bandwidth changes. Next, we investigate BBRv2's responsiveness to bandwidth changes.

The experiments are designed as follows. The bandwidth of the bottleneck link is configured to increase or decrease 5Mbps every 2 seconds, the path delay is set to 40ms, and the buffer size is set to  $32 \times BDP$ . The internal variables during flow transmission (including *pacing\_rate* and *BtlBW*, instantaneous throughput, and the queue length at the bottleneck link) are sampled at an interval of 100ms.

Fig. 11 shows how BBR and BBRv2 adapt to bandwidth increases or decreases respectively. In this figure, the upward and downward spikes of queue length correspond to the actions of BBR/BBRv2 in probing for more bandwidth or draining the bottleneck buffer. We can observe that BBRv2 is less effective than BBR in terms of responsiveness to bandwidth dynamics, resulting in low utilization of bandwidth and long queuing delay.

As we discussed in §2, to match the interval between Reno loss recovery epochs for better inter-protocol fairness, BBRv2 uses  $min\{rand(2,3), \frac{BDP}{MSS} \times RTT\}$  seconds as its probing interval. This interval can be tens of RTTs, which is too conservative in such a dynamic environment. That said, BBRv2 improves the inter-protocol fairness, at the cost of poorer responsiveness to bandwidth dynamics.

#### 3.6. Resilience to network jitters

Several works [16, 27] have shown that throughput collapse occurs when BBR operates in highjitter networks that are widely deployed, e.g. WiFi and 5G networks operating in mmWave band [16, 24, 25], and cellular networks [16, 26] especially when high-mobility involves such as high-speed rails [27]. It is interesting to investigate whether BBRv2 operates well in networks with high jitters.

In this experiment, the bottleneck bandwidth is 40Mbps and the path RTT is 40ms. The bottleneck buffer size is set to  $32 \times$  BDP to avoid buffer overflow. To emulate jitters, *tc* is used to add jitters following Gaussian distribution at R3's interface that



Fig. 11: Responsiveness to bandwidth increases (a, c) and decreases (b, d). The red line represents the BtlBW estimation of BBR/BBRv2, and the green line indicates the real bandwidth of the bottleneck. The dark line shows the dynamics of the bottleneck link's queue length. Note that the spikes of queue length in (a) and (c) are caused by the periodical bandwidth probing of BBR/BBRv2, and the sudden drop of queue\_len around 10s in (b) and (d) is because BBR/BBRv2 enters the ProbeRTT state.



Fig. 12: Avg. throughput against different levels of jitters. x = 0 is equivalent to no jitters.

connects to H3. The mean value of the Gaussian distribution varies from  $0 \sim 120$  ms to emulate different degrees of jitters.

Fig. 12 shows the average value and the standard deviation of throughput of Cubic, BBR, and BBRv2 under various levels of jitter. Compared with Cubic, both BBR and BBRv2 experience low throughput under high jitters. As documented by Kumar et al. [16], BBR underestimates *RTprop* in such networks because it uses a recent 10s minimum RTT to approximate *RTprop*, leading to *cwnd* exhaustion. This problem still exists in BBRv2, even if BBRv2 updates *RTprop*  $2\times$  frequently than BBR (i.e. BBRv2 uses the minimum RTT in recent 5s to estimate RTprop). We also note that the significant throughput degradation starts when the average jitter reaches the path RTT without jitter (i.e. 40ms).

# 3.7. Summary and Implication

We observe that BBRv2 improves the interprotocol fairness and RTT fairness, and reduces retransmission rates under shallow buffers, at the cost of slow responsiveness to bandwidth dynamics and low resilience to random loss.

First, the root cause for the slow responsiveness is that BBRv2 is over-conservative regarding bandwidth probing. That said, it fails to achieve a good balance between the aggressiveness in probing for more bandwidth and the fairness against loss-based CCAs. Note that, however, recklessly increasing BBRv2's aggressiveness in bandwidth probing may lead BBRv2 to generate overwhelming retransmissions and unfairly share bandwidth with loss-based CCAs. In the next section, we propose BBRv2+, which incorporates delay information to cautiously guide the aggressiveness of bandwidth probing to avoid reducing the fairness against lossbased CCAs. The challenge is how to effectively



**Fig. 13:** BBRv2+ architecture. The parts that differs from BBRv2 are highlighted in red color.

use this signal and how to avoid being suppressed by other loss-based CCAs in deep-buffered networks as other delay-based CCAs.

Second, the resilience to random loss can be improved by raising the loss threshold  $\alpha$ , where the value of  $\alpha$  needs to be set according to the bottleneck buffer size.

Last, the throughput degradation of BBR and BBRv2 in high-jitter networks is own to the underestimation of RTprop, which in turn leads to a smaller estimation of BDP. We propose a compensation mechanism of BDP that enables the estimated BDP to be close to the real BDP.

# 4. Design and implementation of BBRv2+

Motivated by our measurement results, we design and implement BBRv2+, in order to address the pitfalls of BBRv2 while maintaining its advantages over BBR (i.e. improved fairness and reduced retransmissions in shallow-buffered networks). The basic idea is to incorporate delay information in BBRv2+'s path model to balance between the aggressiveness in probing for more bandwidth and the fairness against loss-based CCAs ( $\S4.2$ ). That said, BBRv2+ tries to be more aggressive than BBRv2. where the aggressiveness is guided by the delay information. As the use of the delay information may lead BBRv2+ to perform poorly when it co-exists with loss-based CCAs, a dual-mode mechanism is introduced in BBRv2+, where BBRv2+ switches to use BBRv2's state-machine (i.e. invalidating the effect of the delay information) or returns back to use the redesigned state-machine depending on whether loss-based competitors co-exist (§4.3). Moreover, BBRv2+ compensates the estimated BDP when detecting high jitters in order to get an accurate estimation of BDP. (§4.4).

#### 4.1. Overview

The architecture of BBRv2+ is shown in Fig. 13. BBRv2+ incorporates delay information in its path model. Specifically, the delay information consists of three state variables (the first three variables listed in Table. 1) of minimum RTTs, which reflect the change of queuing delay over time. The delay information facilitates quick responsiveness to bandwidth dynamics. Particularly, a new substate, ProbeTry, is added into the ProbeBW state. In ProbeTry, BBRv2+ slightly speeds up to examine if this acceleration will lead to increased RTTs. In the case of increased RTTs, BBRv2+ quits this probing and moves to the ProbeDown state to drain the queue at the bottleneck link; otherwise, it moves to the ProbeUp state to further explore available bandwidth. BBRv2+ also uses the delay information to quickly adapt to bandwidth decreases—it quickly updates its bottleneck bandwidth estimation to the current bandwidth measurement if an obvious increase of RTT is observed when BBRv2+ is not probing for bandwidth.

Like other CCAs that use delay-based signals, BBRv2+ will be suppressed when co-existing with loss-based CCAs under deep buffers [41], as the loss-based CCAs constantly fill the buffer, leading BBRv2+ to falsely yield up obtained bandwidth. BBRv2+ uses a dual-mode mechanism that forces BBRv2+ to use BBRv2's state-machine when lossbased CCAs co-exist.

Finally, BBRv2+ uses a BDP compensation mechanism to address the *cwnd* exhaustion problem caused by network jitters. Our key observation is that in high-jitter networks, the BDP will be underestimated because of the underestimation of RT*prop*. The mechanism compensates BDP by taking the recent RTT variations into consideration; this compensation mitigates the underestimation issue significantly.

# 4.2. Redesign of the ProbeBW state

In the case of bandwidth increments, BBRv2+needs to start probing for more bandwidth quickly instead of spending time on cruising with the current estimated *BtlBW*. Thus, the probing interval needs to be reasonably shortened, which is set to approximately match the probing interval of BBR

Variable	Functionality
MinRTT <sub>prev_rtt</sub>	The minimum RTT measured in the previous RTT round.
$MinRTT_{curr_rtt}$	The minimum RTT measured in the current RTT round.
$MinRTT_{before\_probe}$	Saving the $MinRTT_{curr_{rtt}}$ before entering ProbeUp.
$MinRTT_{curr\_cruise}$	The minimum RTT measured in the current ProbeCruise state.
$\mathrm{Max}_{\mathrm{4RTT}}(\mathrm{jitter})$	The <i>max_filter</i> tracking the maximum jitter in recent four RTT rounds. Here, the jitter is equivalent to the RTT variation maintained by TCP.

Table 1: The new state variables in BBRv2+

(8 rounds of RTT). However, if BBRv2+ is already sending at the speed close to bottleneck bandwidth, the probing interval above may result in more packet losses and thus unfairness against lossbased CCAs in shallow-buffered networks.

Thus, a two-step probing mechanism incorporating the delay information is introduced in BBRv2+, as shown in Fig. 13. A new sub-state ProbeTry, which lasts for two RTTs, is inserted before entering ProbeUp in the state machine. In the first RTT of ProbeTry, BBRv2+ slightly increases its pacing\_rate by increasing pacing\_gain to 1.1. In the second RTT, BBRv2+ reduces pacing\_gain to 1.0 and monitors if MinRTT<sub>curr\_rtt</sub> is larger than  $\gamma \times \text{MinRTT}_{\text{prev_rtt}}$ , where MinRTT is measured on the ACKs for the packets sent in the previous round, thus, reflecting the queuing delay caused by the previous round (see Table 1). The rationale of using  $\gamma > 1$  is to introduce a relaxing factor tolerate noises in RTT measurements, where a small  $\gamma$  may lead BBRv2+ to miss some chances to explore bandwidth while a large  $\gamma$  may make BBRv2+ over-aggressive. In our current implementation, we set  $\gamma = 1.02$  to tolerate noises for 2% of RTT measurements. It is worth noting that  $\gamma$  is a design parameter and can be tuned by designers<sup>4</sup>. If  $MinRTT_{curr_rtt} > \gamma \times MinRTT_{prev_rtt}$ , BBRv2+transits to ProbeDown with *pacing\_gain* as 0.9 to drain the queue accumulated during the first RTT of ProbeTry. Otherwise, it enters ProbeUp to probe for more bandwidth.

To further boost the speed of bandwidth discovery, BBRv2+ also incorporates a continuous probing mechanism based on the delay information . Specifically, at the end of ProbeUp, if the MinRTT<sub>curr\_rtt</sub>  $\leq \gamma \times MinRTT_{before_probe</sub>$ , BBRv2+ re-enters ProbeUp. The rationale behind this is that there is possible more free bandwidth capacity as no significant increment of queuing delay arises in the current ProbeUp sub-state.

In the case of bandwidth decrements, BBRv2+ needs to update its *BtlBW* estimation to new bandwidth measurements as soon as possible. When bandwidth decreases, BBRv2+ sends data faster than the bottleneck bandwidth and packets accumulate in the buffer of the bottleneck link. We thus also leverage the delay information to detect bandwidth decrement. If BBRv2+ is in ProbeCruise or ProbeDown, on the receipt of a new ACK in an RTT round, Algorithm 1 is called<sup>5</sup>. The reason that the algorithm is only applicable in ProbeCruise or ProbeDown is to eliminate the impact on delay variations caused by ProbeTry and ProbeUp substates. In Algorithm 1, BBRv2+ expires its current BtlBW estimation if MinRTT<sub>curr\_rtt</sub> is larger than the recently measured minimum RTT by  $\theta$  times.  $\theta$  is a parameter to balance the speed to converge to new bandwidth and the resistance to noises in bandwidth measurements. A small  $\theta$  may lead to throughput oscillation, while a large  $\theta$  may reduce BBRv2+'s responsiveness to bandwidth dynamics. We recommend  $\theta \in [1.05, 1.15]$  according to our experiences.

Algorithm 1: Advance <i>BtlBW</i> max_filter		
	Input : conn: BBRv2+ TCP connection	
1	$target\_rtt \leftarrow \theta * \texttt{conn}.RTprop$	
2	should_advance $\leftarrow$ (conn.MinRTT <sub>curr_rtt</sub> >	
	target_rtt)	
3	${f if}\ should\_advance\ {f then}$	
4	$expire_the_oldest_value(conn.BtlBW)$	

 $<sup>{}^{5}</sup>$ It is called once at maximum for every RTT round to avoid expiring the BtlBW estimation too frequently.

<sup>&</sup>lt;sup>4</sup>All the design parameters of BBRv2+ in our current implementation are exposed to user-space through the /sys/module interfaces, enabling designers to change the parameters without recompiling the kernel module.

## 4.3. Dual-mode mechanism

Due to the use of the delay information to guide the aggressiveness in bandwidth probing, BBRv2+ will be starved by loss-based CCAs under deep buffers, suffering from the similar problem existing in most delay-based CCAs [41]. The root cause is that loss-based CCAs constantly fill the bottleneck buffer, where BBRv2+ falsely treats the increments of RTT as the signal of bandwidth decrements.

As BBRv2+ periodically drains the bottleneck buffer, during which the measured minimum RTT  $(MinRTT_{curr_cruise})$  listed in Table. 1) is close to RTprop if no loss-based competitor exists. By comparing the  $MinRTT_{curr\_cruise}$  with the recorded RTprop value, BBRv2+ estimates the existence of lossbased competitors. If loss-based competitors coexist, BBRv2+ switches to use BBRv2's ProbeBW state, which enables BBRv2+ to co-exist with lossbased CCAs in the same way as BBRv2 that does not yield up obtained bandwidth due to RTT increments. Further, if the loss-based competitors no longer exist, BBRv2+ returns back to use the redesigned ProbeBW state. We note that the dualmode mechanism does not switch BBRv2+ to use BBRv2's ProbeBW state if the bottleneck buffer is very shallow, because loss-based CCAs can not bloat the bottleneck buffer and BBRv2+ will not be starved.

_				
Algorithm 2: The dual-mode mechanism				
	Input : conn: BBRv2+ TCP connection			
1	1 if $conn.probe_bw_mode = BBRv2+$ then			
<b>2</b>	switch_thld $\leftarrow \lambda_1 * \text{conn.} RT prop$			
3	if conn. $MinRTT_{curr\_cruise} > switch\_thld$			
	then			
4	conn.buffer_filling++			
<b>5</b>	else			
6	$conn.buffer_filling \leftarrow 0$			
7	if conn.buffer_filling $\geq \eta_1$ then			
8	$conn.probe_bw_mode \leftarrow BBRv2$			
9	restart_from_startup(conn)			
10	o else			
11	switch_thld $\leftarrow \lambda_2 * \text{conn.} RT prop$			
<b>12</b>	if conn. $MinRTT_{curr\_cruise} \leq switch\_thld$			
	then			
13	conn.buffer_empty++			
14	else			
15	$conn.buffer\_empty \leftarrow 0$			
16	if conn.buffer_empty $\geq \eta_2$ then			
17	$conn.probe_bw_mode \leftarrow BBRv2+$			

The dual-mode mechanism is detailed in Algorithm 2, which runs at the end of ProbeCruise. If the sender is running in BBRv2+'s ProbeBW state and has not seen RTT samples close to the *RTprop* for a number of  $(\eta_1)$  successive ProbeCruise substates, it switches to use BBRv2's ProbeBW state and restarts itself from Startup (line 1–9). We note that restart\_from\_startup(conn) in line 9 is a heuristic to quickly regain the bandwidth that has been potentially yielded up to loss-based competitors by BBRv2+ recently. If the sender's ProbeBW state is BBRv2 and it has seen low RTTs for  $\eta_2$ successive ProbeCruise sub-states, it returns back to use BBRv2+'s ProbeBW state because the competitors are most likely gone. We note that the four parameters in Algorithm 2,  $\lambda_1$ ,  $\lambda_2$ ,  $\eta_1$ , and  $\eta_2$ , are to control the sensitivity of BBRv2+ to the coexistence of loss-based CCAs. In practice, we used 1.1, 1.05, 2, and 4 for  $\lambda_1$ ,  $\lambda_2$ ,  $\eta_1$ , and  $\eta_2$  respectively. Nevertheless, these parameters can be tuned to fit specific networks in user space in our current implementation.

# 4.4. Compensation for BDP estimation

We have seen in §3.6, when network jitters are high, BBRv2 (also BBR) underestimates RTprop, thus the BDP of the network path, leading to *cwnd* exhaustion and thus performance degradation. To boost BBRv2+'s performance under high network jitters, BBRv2+ takes network jitters into account when estimating the BDP of the network path.

BBRv2+ compensates the BDP estimation with a component proportional to RTT variations when network jitters are high, which is detailed in Algorithm. 3. As instantaneous RTT variations could be very dynamic, to ensure that BBRv2+ can tolerate jitters up to the maximum extent, we use the recently measured maximum RTT variation,  $Max_{4RTT}$  (jitter) in Table. 1, as the indicator of recent jitters. When  $Max_{4RTT}$  (jitter) exceeds  $\mu \times RT prop$ , the estimated RT prop is increased to the sum of the original *RTprop* and the delay variation  $(Max_{4RTT}(jitter))$  to mitigate the underestimation of *RTprop*. We recommend setting  $\mu$  around 0.5 because the performance of BBRv2 starts to degrade when jitters approach half of *RTprop* as observed in §3.6.

# 4.5. Implementation

BBRv2+ is implemented as a Linux kernel module ( $\sim$ 2100 LoCs), based on Google's BBRv2 alpha kernel module [31]. Therefore, it is easy to

Algorithm 3: Compensating BDP estima-		
tion		
<b>Input</b> : conn: BBRv2+ TCP connection		
<b>Output:</b> the BDP estimation of BBRv2+		
1 jitter $\leftarrow \text{conn.Max}_{4\text{RTT}}$ (jitter)		
<b>2</b> threshold $\leftarrow \mu * \text{conn.} RT prop$		
<b>3</b> fixed_ $RTprop \leftarrow conn.RTprop$		
4 if $jitter > threshold$ then		
5 fixed_ $RTprop \leftarrow fixed_{RTprop} + jitter$		
6 return conn. $BtlBW$ * fixed_ $RTprop$		

deploy BBRv2+ on the hosts where BBRv2 is already in use. The parameters of BBRv2+ are exposed to user-space through the /sys/module interfaces, which allows users to change the parameters according to their need without recompiling the kernel module. The code of BBRv2+ is opensourced on Github [28] to the research community for further test and improvement.

#### 5. Evaluation of BBRv2+

In this section, we evaluate BBRv2+ based on both Mininet-based emulation and real-world trace driven emulation. First, we describe our experiment setup in §5.1. We then evaluate the benefits of BBRv2+ from the perspectives of the responsiveness to bandwidth changes (§5.2) and the resilience to network jitters (§5.3). Next, we demonstrate that BBRv2+ is able to keep the advantages of BBRv2 in inter-protocol fairness (§5.4), RTT fairness (§5.5), and low retransmissions (§5.6). Finally, we evaluate the performance of BBRv2+ through real-world trace driven emulation in §5.7.

#### 5.1. Evaluation setup

Two testbeds are used for the evaluation of BBRv2+. One is the Mininet-based testbed used in §3, as a controlled environment to evaluate BBRv2+ from various perspectives. The other is based on Mahimahi [42], a trace-driven emulator that can accurately replay real-world packet-level traces, as illustrated in Fig. 14. The physical server running the two testbeds is the same one as that used to evaluate BBRv2 in §3. The toolset for data analysis and traffic generation is also the same as that in §3.

In all experiments in this section, the  $\alpha$  (the loss threshold) of BBRv2+ is set to 20% as the value is suitable for most of the buffer sizes according to the results in §3.4.



Fig. 14: Mahimahi testbed

#### 5.2. Responsiveness to bandwidth dynamics

To evaluate BBRv2+'s responsiveness to bandwidth dynamics, we use the same settings as that in §3.5, in order to run BBRv2+ to have a microscopic view on how it reacts to bandwidth dynamics. Fig. 15 shows the results.

In Fig. 15a, when there is no bandwidth increment, BBRv2+ only enters ProbeTry for a very short duration and finishes bandwidth probing very soon, which leads to an instantaneous short standing queue. However, when the bandwidth is increased, BBRv2+ can timely adapt its' BtlBW estimation to the real bandwidth. Compared with the results of BBR in Fig. 11a and BBRv2 in Fig. 11c, BBRv2+ is capable to utilize newly available bandwidth as quick as BBR, while its guided probing strategy (by the delay information) incurs lower queuing delay than BBR.

In Fig. 15b, when bandwidth decreases, BBRv2+ notices that the queuing delay is obviously rising up via increased RTT (see §4.2). It expires the old BtlBW estimation and adapts its BtlBW estimation to the available bandwidth. Compared with the results of BBR and BBRv2 in Fig. 11, BBRv2+ adapts its sending rate to the decreased bandwidth much faster, which leads to lower queuing delay.

Next, we compare the responsiveness of Cubic, BBR, BBRv2, and BBRv2+ to bandwidth dynamics in our trace-driven emulation testbed Mahimahi, using five synthesized network traces where the bandwidth changes as step functions, as illustrated in Fig. 16. Following the settings in [2], we set the buffer size to 1.5MB, the delay to 20ms, and the loss rate to zero. In each experiment, the flow throughput, as well as the sojourn time of each packet in the buffer of the bottleneck link (denoted as queuing delay), are recorded.

To compare the overall performance of all CCAs on a network trace, we normalized the average queuing delay and the average throughput of all CCAs to the minimum average queuing delay and



Fig. 15: BBRv2+'s responsiveness to bandwidth increases (a) and decreases (b). The red line represents the BtlBW estimation of BBRv2+, and the green line indicates the real bandwidth of the bottleneck. The dark line shows the dynamics of the bottleneck link's queue length.



Fig. 16: An example of traces with bandwidth changing as a step function.



**Fig. 17:** Normalized throughput and queuing delay of different CCAs. (markers: average throughput and queuing delay; left end of the lines: 95%-tile of queuing delay; ellipses: the standard deviations)

the maximum average throughput achieved on that trace, respectively. In addition, we also normalized the 95% tile queuing delay of all CCA on a network trace to the minimum average queuing delay achieved on that trace. Then, we averaged all normalized values over all traces. The results are shown in Fig. 17. We observe that BBRv2+achieved significantly higher throughput and lower queuing delay than BBRv2, and lower queuing delay at the cost of slightly lower throughput than BBR. These observations stem from the facts that: (1) BBRv2+ probes for bandwidth at a frequency similar to BBR's one, thus, achieving high bandwidth utilization as BBR does; (2) BBRv2+ adapts its sending rate to decreased bandwidth faster than BBR as it quickly updates its BtlBW estimation upon increased queuing delay.

#### 5.3. Resilience to network jitters

Next, we evaluate the performance of BBRv2+ under network jitters, using the same settings as in §3.6. The throughput of BBRv2+, Cubic, BBR and BBRv2 under various levels of jitters are shown in Fig. 18a. Different from BBR and BBRv2, the throughput of BBRv2+ does not degrade when the network jitters become larger. Fig. 18b further plots the average inflight bytes of four CCAs; the results confirm that the BDP compensation mechanism of BBRv2+ succeeds to increase the inflight size for higher throughput when the network jitter becomes larger. Nevertheless, the throughput of BBRv2+ is slightly lower than Cubic. The reason is that our compensation to BDP is a bit conservative; in contrast, Cubic's *cwnd* can grow far beyond the real BDP as it is not affected by network jitters.



Fig. 18: Resilience to network jitters

## 5.4. Inter-protocol fairness

The inter-protocol fairness of BBRv2+ is evaluated using the same settings as that in §3.2.1. We considered BBRv2+ with/without the dual-mode mechanism (see §4.3) to study the impact of this mechanism on inter-protocol fairness. The results are shown in Fig. 19. Several observations are notable.

First, the results demonstrate the efficacy of the dual-mode mechanism. In Fig. 19a, we can observe that BBRv2+ without the dual-mode mechanism is starved by Cubic in deep-buffered cases. This is because BBRv2+ falsely treats the RTT increments caused by Cubic as a signal of bandwidth shrinking, thus, constantly yielding up bandwidth to Cubic. The problem is eliminated by the dual-mode mechanism as shown in Fig. 19b.

compared Second. with  $_{\mathrm{the}}$ results of BBRv2(20%, 0.3) in Fig. 10, we can see that: (1) BBRv2+ provides better inter-protocol fairness than BBRv2(20%, 0.3) under an extremely shallow buffer (i.e.  $0.2 \times BDP$ ; (2) BBRv2+'s inter-protocol fairness is similar to that of BBRv2(20%, 0.3) under other buffer sizes. The reason for the better inter-protocol fairness of BBRv2+ under an extremely shallow buffer is that BBRv2+ does not enter ProbeUp, which is more aggressive than ProbeTry, thanks to the two-step probing mechanism (see  $\S4.2$ ) while BBRv2(20%, 0.3) periodically enters ProbeUp.

Third, compared with the results of BBR in Fig. 4a and BBRv2 in Fig. 4b, BBRv2+ performs no worse than the better one among BBR and BBRv2 under different buffer sizes. The reasons are three-fold: (1) under shallow buffers, BBRv2+ achieves similar inter-protocol fairness to that of BBRv2 thanks to its cautiously aggressive bandwidth probing strategy (see §4.2); (2) under moderate buffers, BBRv2+ is close to BBRv2(20%, 0.3) that has better inter-protocol fairness than BBRv2 in these cases as explained in §3.4; (3) under deep buffers, the three CCAs perform closely as they all have an inflight cap around  $2 \times$  BDP, thus, unable to beat loss-based CCAs.

# 5.5. RTT fairness

Next, we evaluate the RTT fairness of BBRv2+ using the same setting as that in §3.2.2. The results are presented in Fig. 20. Compared with the results of BBR in Fig. 5a and those of BBRv2 in Fig. 5b, BBRv2+ has better RTT fairness than



(a) Without the dual-mode (b) With the dual-mode mechanism mechanism

Fig. 19: BBRv2+: Inter-protocol fairness



Fig. 20: BBRv2+: RTT fairness

BBR and behaves close to BBRv2. The results are expected as the mechanisms in BBRv2 that improves the RTT fairness over BBR (see §3.2.2) remain unchanged in BBRv2+.

## 5.6. Retransmissions in shallow-buffered networks

In the following, we evaluate whether BBRv2+ is as aggressive as BBR to lead to excessive retransmissions in shallow-buffered networks, using the same setting as that in \$3.3. The results are shown in Fig. 21. We observe that when the buffer is extremely shallow (e.g.  $0.02 \times BDP$  when the bandwidth is 500Mbps and the RTT is 75ms), BBRv2+ incurs more retransmissions than BBRv2. This is because the bandwidth probing frequency of BBRv2+ is higher than that of BBRv2. Although BBRv2+ uses a relatively small  $pacing_gain$  (1.1) when it starts to probe for more bandwidth in ProbeTry, it still causes buffer overflow when the network buffer is extremely shallow. However, compared with the results of BBR in Fig. 6a, BBRv2+ reduces retransmissions significantly.

#### 5.7. Real-world trace driven emulation

To evaluate how BBRv2+ performs in real network conditions, we compare BBRv2+ with Cubic, BBR, BBRv2, and Orca [2] in the emulation-based



Fig. 21: BBRv2+: retransmission rate (100KB buffer)

Mahimahi testbed, using traces collected in realworld networks. Orca<sup>6</sup> is used for comparison as a representative of the state-of-the-art learning-based CCAs.

**Trace collection:** We collected traces from WiFi and LTE networks, using *saturatr* [39]. In total, 20 network traces are collected, half of which are collected when the collector is stationary to the base station (LTE) or the Access Point (WiFi), and the other half are collected when the collector is moving at high speeds (i.e. in vehicles or on high-speed rails). In stationary scenarios, the network bandwidth is usually stable, while in high-mobility scenarios, the bandwidth fluctuates greatly. Examples of stationary and high-mobility traces are shown in Fig. 22. The network delay and loss rate are also measured using *ping*.



Fig. 22: Example of traces used in our trace-driven evaluation

**Experimental results:** The network buffer size is set to 1.5MB. The collected traces, including bandwidth dynamics, network delay, and loss rate are the inputs of the emulator. Using the same metrics as that in §5.2, the results for the stationary and high-mobility scenarios are shown in Fig. 23a and Fig. 23b respectively.

In stationary scenarios, BBR, BBRv2, and BBRv2+ perform very close to each other because the bandwidth is usually stable. Cubic shows slightly better throughput for most of the time at the cost of high queuing delays. Orca has the lowest throughput probably because the network scenario where the model was trained is different from our collected traces, which also demonstrates the limitation of learning-based CCAs.

In high-mobility scenarios, BBR and BBRv2+ achieve the highest and the second-highest throughput respectively. Meanwhile, BBRv2 and Cubic fail to achieve consistently high throughput across different high-mobility traces. Compared with BBRv2+, BBR achieves higher throughput at the cost that it incurs higher queuing delays as it is more aggressive. Orca fails to achieve consistently high throughput and low delays in high-mobility scenarios; the results of Orca raise a concern on the generalization ability of learning-based CCAs.

The above results of trace-driven emulation using Mahimahi demonstrate that BBRv2+ performs closely to BBR and BBRv2 in stationary network scenarios, but shows great improvements over BBRv2 in high-mobility scenarios as it has better responsiveness to bandwidth dynamics.

#### 5.8. Summary of experimental results

We can conclude from the above experiments that BBRv2+ succeeds to balance the aggressiveness of bandwidth probing and the fairness against loss-based CCAs. With such a balance, which is neither achieved by BBR nor BBRv2, BBRv2+ achieves higher throughput and lower delay than BBRv2 in scenarios where the bandwidth fluctuates, while keeping the advantages of BBRv2 with regard to inter-protocol fairness and reduced retransmissions under shallow buffers. Moreover, the dual-mode mechanism makes BBRv2+ able to coexist with loss-based CCAs under deep buffers and the compensation mechanism for BDP estimation efficiently enhances the performance of BBRv2+ under high network jitters.

#### 6. Related work

**BBR evaluation:** Since BBR [8] was released by Google in 2016, it has been examined under various network conditions by researchers [12–17]. BBR is unfair when sharing a bottleneck link with Cubic flows or BBR flows with different RTTs [12–14, 17]. In particular, BBR flows are always able to claim at least 35% of the total bandwidth in

 $<sup>^6\</sup>mathrm{We}$  directly used the model trained by the authors in our experiments.



Fig. 23: Normalized throughput and queuing delay of different CCAs. (markers: average throughput and queuing delay, left end of the lines: 95%-tile of queuing delay, ellipses: the standard deviations of the throughput and queuing delay)

deep-buffered networks when competing with Cubic flows [13]. This percentage, however, depends on the link capacity and delay, the bottleneck buffer size, and the number of BBR flows [15]. In shallowbuffered networks, BBR can lead to massive retransmissions [14]. Moreover, the throughput of BBR collapses when network jitters are high, either in experimental emulation [16] or networks in high-speed train scenarios [27].

**BBR enhancement:** The issues of BBR identified by empirical studies motivated optimizations on BBR from various perspectives. For instance, [16, 27] proposed several modifications in the *RTprop* estimation of BBR to counter against network jitters, [17, 43, 44] improved BBR's RTT fairness, [45, 46] improved BBR's inter-protocol fairness with loss-based CCAs, and [47] reduced BBR's aggressiveness in shallow-buffered networks to suppress the unnecessary retransmissions.

**BBRv2 evaluation:** Google proposed BBRv2 [18] to solve the problems identified in BBR. In Google's early tests [18, 20], BBRv2 shows better interprotocol fairness with loss-based CCAs and reduced retransmissions in shallow-buffered networks. There are also several evaluations on BBRv2 [19, 21–23]. Gomez et al. [19] and Nadagiri et al. [21] studied the inter-protocol fairness and RTT fairness of BBRv2 through emulation. Kfoury et al. [23] evaluated BBRv2 in emulated networks and found that BBRv2 eliminates the massive retransmissions in shallow-buffered networks. Song et al. [22] found that BBRv2 cannot quickly utilize the newly available link capacity when the bottleneck bandwidth increases.

Our work differs from the above studies in two perspectives: (1) from the measurement perspective, we not only systematically evaluate the performance of BBRv2 under various network conditions, but also analyze the reasons behind the observed performance issues; (2) from the optimization perspective, we propose BBRv2+ that addresses the shortcomings of BBRv2 while barely sacrificing its advantages in fairness and reduced retransmissions.

# 7. Conclusion

In this paper, we first comprehensively evaluated BBRv2, revealed its pros and cons over BBR, and analyzed the reasons behind BBRv2's performance issues. Motivated by the results of BBRv2's evaluation, we propose BBRv2+ that incorporates delay information in its path model and redesigns the ProbeBW state to achieve a good balance between the aggressiveness of bandwidth probing and the fairness to loss-based CCAs. Extensive experiments demonstrate that BBRv2+ significantly improves the performance over BBRv2, especially in high-mobility scenarios, while barely sacrificing the advantages of BBRv2.

#### References

- V. Jacobson, Congestion avoidance and control, Comput. Commun. Rev. 25 (1995) 157–187.
- [2] S. Abbasloo, C.-Y. Yen, H. J. Chao, Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet, 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, ACM, Virtual Event USA, 2020, pp. 632-647. doi:10.1145/3387514.3405892. URL https://dl.acm.org/doi/10.1145/3387514. 3405892

- [3] K. Winstein, H. Balakrishnan, TCP ex Machina: Computer-Generated Congestion Control 12.
- [4] N. Jay, N. H. Rotman, B. Godfrey, M. Schapira, A. Tamar, A deep reinforcement learning perspective on internet congestion control, in: ICML, 2019.
- [5] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, M. Schapira, Pcc vivace: Online-learning congestion control, in: NSDI, 2018.
- [6] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, K. Winstein, Pantheon: the training ground for internet congestion-control research, in: USENIX Annual Technical Conference, 2018.
- [7] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger, CUBIC for Fast Long-Distance Networks, Tech. Rep. RFC8312, RFC Editor (Feb. 2018). doi:10.17487/RFC8312.

URL https://www.rfc-editor.org/info/rfc8312

- [8] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, V. Jacobson, Bbr: Congestion-based congestion control, Queue 14 (2016) 20 – 53.
- [9] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, B. Leong, The Great Internet TCP Congestion Control Census, Proceedings of the ACM on Measurement and Analysis of Computing Systems 3 (3) (2019) 1–24. doi:10.1145/3366693.
  - URL https://dl.acm.org/doi/10.1145/3366693
- [10] S. H. Yeganeh, P. Jha, Y. Seung, L. Hsiao, M. Mathis, V. Jacobson, BBR Updates: Internal Deployment, Code, Draft Plans 8.
- [11] L. Kleinrock, Power and deterministic rules of thumb for probabilistic problems in computer communications, 1979.
- Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, A. Gandhi, When to use and when not to use BBR: An empirical analysis and evaluation study, in: Proceedings of the Internet Measurement Conference, ACM, Amsterdam Netherlands, 2019, pp. 130–136. doi:10.1145/3355369.3355579.
  URL http://dl.acm.org/doi/10.1145/3355369. 3355579
- [13] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, G. Carle, Towards a Deeper Understanding of TCP BBR Congestion Control, in: 2018 IFIP Networking Conference (IFIP Networking) and Workshops, 2018, pp. 1–9. doi:10.23919/IFIPNetworking. 2018.8696830.
- [14] M. Hock, R. Bless, M. Zitterbart, Experimental evaluation of BBR congestion control, in: 2017 IEEE 25th International Conference on Network Protocols (ICNP), 2017, pp. 1–10. doi:10.1109/ICNP.2017.8117540.
- R. Ware, M. K. Mukerjee, S. Seshan, J. Sherry, Modeling BBR's Interactions with Loss-Based Congestion Control, in: Proceedings of the Internet Measurement Conference, ACM, Amsterdam Netherlands, 2019, pp. 137–143. doi:10.1145/3355369.3355604.
   URL http://dl.acm.org/doi/10.1145/3355369.3355604
- [16] R. Kumar, A. Koutsaftis, F. Fund, G. Naik, P. Liu, Y. Liu, S. Panwar, TCP BBR for Ultra-Low Latency Networking: Challenges, Analysis, and Solutions,

in: 2019 IFIP Networking Conference (IFIP Networking), 2019, pp. 1–9, iSSN: 1861-2288. doi:10.23919/ IFIPNetworking.2019.8816856.

[17] S. Ma, J. Jiang, W. Wang, B. Li, Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis, arXiv:1706.09115 [cs]ArXiv: 1706.09115.

URL http://arxiv.org/abs/1706.09115

- [18] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, V. Jacobson, BBR v2 A Model-based Congestion Control 36.
- J. Gomez, E. Kfoury, J. Crichigno, E. Bou-Harb, G. Srivastava, A Performance Evaluation of TCP BBRv2 Alpha, in: 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), IEEE, Milan, Italy, 2020, pp. 309-312. doi:10.1109/TSP49548.2020.9163512. URL https://ieeexplore.ieee.org/document/ 9163512/
- [20] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, I. Swett, V. Vasiliev, B. Wu, M. Mathis, V. Jacobson, BBR v2: A Model-based Congestion Control IETF 105 Update 21.
- [21] A. Nandagiri, M. P. Tahiliani, V. Misra, K. K. Ramakrishnan, BBRvl vs BBRv2: Examining Performance Differences through Experimental Evaluation, in: 2020 IEEE International Symposium on Local and Metropolitan Area Networks (LAN-MAN, IEEE, Orlando, FL, USA, 2020, pp. 1–6. doi:10.1109/LANMAN49260.2020.9153268. URL https://ieeexplore.ieee.org/document/

9153268/

- [22] Y.-J. Song, G.-H. Kim, I. Mahmud, W.-K. Seo, Y.-Z. Cho, Understanding of BBRv2: Evaluation and Comparison with BBRv1 Congestion Control Algorithm, IEEE Access (2021) 1-1doi:10.1109/ACCESS.2021.3061696. URL https://ieeexplore.ieee.org/document/ 9361674/
- [23] E. F. Kfoury, J. Gomez, J. Crichigno, E. Bou-Harb, An emulation-based evaluation of TCP BBRv2 Alpha for wired broadband, Computer Communications 161 (2020) 212-224. doi:10.1016/j.comcom.2020.07.018. URL https://linkinghub.elsevier.com/retrieve/ pii/S014036642030092X
- [24] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, M. Zorzi, Will TCP work in mmWave 5G Cellular Networks?, arXiv:1806.05783 [cs]ArXiv: 1806.05783.

 $\mathrm{URL}\ \mathtt{http://arxiv.org/abs/1806.05783}$ 

- [25] D. Chitimalla, K. Kondepu, L. Valcarenghi, M. Tornatore, B. Mukherjee, 5G fronthaul-latency and jitter studies of CPRI over ethernet, IEEE/OSA Journal of Optical Communications and Networking 9 (2) (2017) 172–182, conference Name: IEEE/OSA Journal of Optical Communications and Networking. doi: 10.1364/JOCN.9.000172.
- [26] J. D. Beshay, A. T. Nasrabadi, R. Prakash, A. Francini, Link-Coupled TCP for 5G networks, in: 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS), 2017, pp. 1–6. doi:10.1109/ IWQoS.2017.7969170.
- [27] J. Wang, Y. Zheng, Y. Ni, C. Xu, F. Qian, W. Li, W. Jiang, Y. Cheng, Z. Cheng, Y. Li, X. Xie, Y. Sun, Z. Wang, An Active-Passive Measurement Study of

TCP Performance over LTE on High-speed Rails, in: The 25th Annual International Conference on Mobile Computing and Networking, ACM, Los Cabos Mexico, 2019, pp. 1–16. doi:10.1145/3300061.3300123. URL https://dl.acm.org/doi/10.1145/3300061. 3300123

- [28] Y. Furong, yangfurong/BBRv2plus (Jul. 2021). URL https://github.com/yangfurong/BBRv2plus
- [29] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center tcp (dctcp), in: SIGCOMM '10, 2010.
- [30] Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet. URL http://mininet.org/
- [31] google/bbr.
- URL https://github.com/google/bbr/tree/v2alpha [32] tc-netem(8) - Linux manual page.
- URL https://www.man7.org/linux/man-pages/man8/ tc-netem.8.html
- [33] esnet/iperf. URL https://github.com/esnet/iperf
- [34] TCPDUMP/LIBPCAP public repository. URL https://www.tcpdump.org/
- [35] tcptrace(1): TCP connection analysis tool Linux man page.
- URL https://linux.die.net/man/1/tcptrace [36] tc(8) - Linux manual page.
- URL https://man7.org/linux/man-pages/man8/tc.8. html
- [37] S. Floyd, Metrics for the Evaluation of Congestion Control Mechanisms, RFC 5166 (Mar. 2008). doi: 10.17487/RFC5166.
  - URL https://rfc-editor.org/rfc/rfc5166.txt
- [38] B. Huffaker, M. Fomenkov, D. Plummer, D. Moore, K. Claffy, Distance metrics in the internet, 2002.
- [39] K. Winstein, A. Sivaraman, H. Balakrishnan, Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks 13.
- [40] L. Li, K. Xu, T. Li, K. Zheng, C. Peng, D. Wang, X. Wang, M. Shen, R. Mijumbi, A measurement study on multi-path TCP with multiple cellular carriers on high speed rails, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '18, ACM Press, Budapest, Hungary, 2018, pp. 161–175. doi:10.1145/3230543.3230556. URL http://dl.acm.org/citation.cfm?doid=

3230543.3230556 [41] R. Al-Saadi, G. Armitage, J. But, P. Branch, A Survey

- [41] R. Al-Saadi, G. Armitage, J. But, F. Branch, A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms, IEEE Communications Surveys Tutorials 21 (4) (2019) 3609–3638, conference Name: IEEE Communications Surveys Tutorials. doi:10.1109/COMST. 2019.2904994.
- [42] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, Mahimahi: Accurate Record-and-Replay for HTTP, in: ATC, 2015, p. 14.
- [43] M. Yang, P. Yang, C. Wen, Q. Liu, J. Luo, L. Yu, Adaptive-BBR: Fine-Grained Congestion Control with Improved Fairness and Low Latency, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), 2019, pp. 1–6, iSSN: 1558-2612. doi:10. 1109/WCNC.2019.8885527.
- [44] G.-H. Kim, Y.-Z. Cho, Delay-Aware BBR Con-

gestion Control Algorithm for RTT Fairness Improvement, IEEE Access 8 (2020) 4099-4109. doi:10.1109/ACCESS.2019.2962213. URL https://ieeexplore.ieee.org/document/ 8943219/

- [45] Y. Zhang, L. Cui, F. P. Tso, Modest BBR: Enabling Better Fairness for BBR Congestion Control, in: 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 00646–00651, iSSN: 1530-1346. doi: 10.1109/ISCC.2018.8538521.
- [46] Y.-J. Song, G.-H. Kim, Y.-Z. Cho, BBR-CWS: Improving the Inter-Protocol Fairness of BBR, Electronics 9 (5) (2020) 862. doi:10.3390/electronics9050862. URL https://www.mdpi.com/2079-9292/9/5/862
- [47] I. Mahmud, G.-H. Kim, T. Lubna, Y.-Z. Cho, BBR-ACD: BBR with Advanced Congestion Detection, Electronics 9 (1) (2020) 136. doi:10.3390/ electronics9010136. URL https://www.mdpi.com/2079-9292/9/1/136