Improving Google's BBR for Reduced Latency and Increased Fairness

Menglei Zhang
NYU WIRELESS
New York University
NY, USA
menglei@nyu.edu

Marco Mezzavilla
NYU WIRELESS
New York University
NY, USA
mezzavilla@nyu.edu

Sundeep Rangan
NYU WIRELESS
New York University
NY, USA
srangan@nyu.edu

Shivendra Panwar
NYU WIRELESS
New York University
NY, USA
panwar@nyu.edu

***Abstract*—BBR is a low latency congestion control algorithm developed by Google as an alternative to the traditional loss-based TCP (Transmission Control Protocol) algorithms, such as CUBIC. However, recent work shows that BBR suffers from 1) additional queuing delay due to competing flows and 2) poor fairness among flows with different Round Trip Times (RTTs). In this paper, we derive an analytic model for BBR's Multiplicative Increase/Multiplicative Decrease (MIMD) sending rate control that reveals the origin of these issues. Inspired by Google's recent patch for reducing queuing delay, we further improve BBR for lower latency. Moreover, our analysis suggests that applying a RTT-based adaptive pacing gain to probe bandwidth helps mitigating the fairness issue. Finally, we validate these improvements through simulations using the ns-3 network simulator.**

***Index Terms*—Congestion control, TCP, BBR, Latency, Fairness, ns-3**

I. Introduction

Loss-based versions of TCP, such as CUBIC [1], are today's most used congestion control protocols. As the name suggests, loss-based TCP treats packet loss as an indication of network congestion. As a consequence, the sender keeps transmitting with a high rate and the latency is inevitably increased when congestion occurs, which in turn leads to "bufferbloat" [2]. Nowadays, due to growing Internet bandwidth, the throughput delivered by loss-based TCP protocols is usually high enough to satisfy most applications. However, applications that require low latency, such as real-time video streaming, suffer from loss-based algorithms because of bufferbloat.

Data delivery is even more challenging over wireless links due to time-varying channel conditions. Moreover, as the wireless channel keeps migrating to higher frequencies to harvest more bandwidth, signal propagation becomes more dynamic. For example, the bandwidth in the millimeter wave (mmWave) spectrum may vary between Mbps and Gbps within 1 second, and some recent studies [3], [4] showed that the traditional loss-based TCP fails to achieve a reasonable performance over intermittent links.

Because of the urgent demand for a new congestion control mechanism, Google recently developed BBR [5] to favor latency sensitive applications. Unlike traditional TCP that employs the Additive Increase/Multiplicative Decrease (AIMD) Congestion Window (Cwnd) control algorithm [6], BBR deploys a new MIMD sending rate control to better match the sending rate to the bottleneck bandwidth. A recent work [7] has shown the benefits of having BBR over the 5G deployment scenario compared to other loss-based TCP. However, it also introduces some problems, namely, 1) additional queuing delay due to competing flows and 2) poor fairness among flows with different RTTs in deep buffers, as illustrated in [8].

BBR's mechanism is fundamentally different from the traditional TCP. Consequently, without a thorough understanding of its algorithm, fixing one issue may potentially generate more problems in the future. Realizing that no analytical studies have been conducted to explain the behaviors of this new congestion control, we introduce the first analytic framework that can be used to improve the performance of BBR using theoretical approaches, in addition to experimental methods. The key contributions of this paper are as follows:

- **Analytic model:** We derived a two phase model for BBR as the foundation of our improvement.
- **Latency reduction:** We also modified BBR's algorithm and were able to achieve lower latency by refining the pacing phase and speeding up the probe RTT state.
- **Fairness improvement:** At the same time, we showed that BBR can reach higher fairness if the pacing gain associated with each flow is based on its RTT rather than being based on a fixed value.

This rest of this paper is organized as follows. A brief overview of BBR is presented in Section II; An analytic model is introduced and studied in Section III; our improved BBR algorithm is described in Section IV; the simulation results are reported in Section V; we conclude this paper in Section VI.

II. BBR Congestion Control Overview

BBR detects congestion by measuring the bottleneck bandwidth and RTT, and use them to infer specific network conditions. These measurements are described in Section II-B and Section II-C. Additionally, BBR utilizes pacing to delay packets to match the sending rate to the estimated bottleneck bandwidth. Packets in flight are limited by pacing if the bandwidth estimation is accurate. Further, if the bandwidth is overestimated, and Acknowledgments (ACKs) are consequently delayed, the packets in flight are limited by the Cwnd, which is bounded to 2 $\times$ Bandwidth Delay Product (BDP).

A. Initial State

BBR starts with an initial startup state and quickly increases sending rate. Once a large latency is observed, BBR assumes the maximum bandwidth is achieved and enters a drain state to empty the queue at the bottleneck link.

B. Probe Bandwidth State

After the queuing packets are drained, BBR spends most of the time in probe bandwidth state, where the current bottleneck bandwidth is measured. BBR constantly adjusts its sending rate by multiplying the current bandwidth estimate with a pacing rate that is changed every RTT following the pattern [1.25, 0.75, 1, 1, 1, 1, 1, 1], where the first two phases are designed to probe bandwidth and drain the queue, respectively.

C. Probe RTT State

When entering the probe bandwidth state, BBR also starts a 10 second timer and keeps recording the minimum RTT. The timer resets if a new minimum RTT is detected. Otherwise, after the timer expires, BBR enters a probe RTT state to actively measure minimum RTT. BBR stores the current Cwnd value and changes it to 4 packets to drain the queue. After 200 ms, BBR restores the Cwnd value and switches back to probe bandwidth state again.

III. Latency and Fairness Study

In this section, we derive a simple two phase model for BBR's MIMD sending rate control, and study the latency and fairness issues.

A. Two Phase Model

In this preliminary paper, we present a simple analysis of our approach in a deterministic setting. Given the promising results we present in this paper, we intend to do an analysis in a more general stochastic setting to further validate this modification of BBR.

When multiple BBR flows share the same bottleneck link, the approximate throughput of flow i at time t can be computed by applying Little's law:

$$\lambda_i(t) = \frac{B_i(t)}{T_i + D_i(t)} \tag{1}$$

where $B_i(t)$ is the packets in fight from flow i at time t, T_i is the RTT_{min} of flow i, $D_i(t)$ is the queuing delay of flow i at time t. In steady state, the parameter $B_i(t)$ may be limited by pacing or the Cwnd depending on the accuracy of bandwidth share estimation, as described in Section II.

1) Pacing controlled phase: When the queuing delay $D_i(t)$ is small, either because of accurate bandwidth estimation or under utilization, the packets in flight are limited by pacing. Therefore, $B_i(t)$ has an average value of $C_i(t-\Delta_i)T_i$, where $C_i(t-\Delta_i)$ is the previous bandwidth estimate, and achieves its maximum $1.25 \times C_i(t-\Delta_i)T_i$ in the first pacing phase. From Equation (1) and the above analysis, the maximum throughput at time t can be obtained as,

$$\lambda_i(t)^{MAX} = \frac{B_i(t)^{MAX}}{T_i + D_i(t)} = \frac{1.25T_i}{T_i + D_i(t)} \times C_i(t-\Delta_i) \tag{2}$$

Since all flows share the same buffer and the packets from any flow are spread out evenly in the queue due to pacing, it is a valid assumption that at any time t, the queuing delay $D_i(t)$ observed by each flow is the same and we denote it as $D(t)$. Ideally, a higher throughput should be detected every $\Delta_i = 8 \times T_i$, which is the size of pacing gain pattern. Moreover, a higher throughput updates the bandwidth estimation at time t as following,

$$C_i(t) = \lambda_i(t)^{MAX} = \frac{1.25T_i}{T_i + D(t)} \times C_i(t - 8T_i) \tag{3}$$

Equation (3) suggests that the estimated bandwidth increases if $D(t) < 0.25T_i$; otherwise, it will decrease after 10 RTT.[1] The MIMD factor equals $1.25T_i/(T_i + D(t))$. Moreover, even though the Cwnd is updated in this phase, pacing is the factor that limits sending rate.

2) Cwnd controlled phase: However, when the bandwidth share is overestimated by BBR, the queue builds up and $B_i(t)$ is not limited by pacing any more. Instead, it equals the Cwnd $W_i(t-\Delta_i)$ and the throughput at time t becomes,

$$\lambda_i(t) = \frac{B_i(t)}{T_i + D_i(t)} = \frac{W_i(t-\Delta_i)}{T_i + D_i(t)} \tag{4}$$

In this phase, $B_i(t)$ is almost constant, but $D_i(t)$ varies over time. Same as the previous case, all flows perceive the same minimum value $D(t)^{MIN}$ at the same time interval Δ. Therefore, we get the bandwidth estimation at time t,

$$C_i(t) = \lambda_i(t)^{MAX} = \frac{W_i(t-\Delta)}{T_i + D(t)^{MIN}} \tag{5}$$

Since the Cwnd at time t equals $W_i(t) = 2 \times C_i(t)T_i$, from Equation (5), the Cwnd for flow i at time t is derived,

$$W_i(t) = 2C_i(t)T_i = \frac{2T_i}{T_i + D(t)^{MIN}} \times W_i(t-\Delta) \tag{6}$$

and the MIMD factor equals $2T_i/(T_i + D(t)^{MIN})$. According to Equation (6), the Cwnd increases if $D(t)^{MIN} < T_i$; otherwise, it reduces after 10 RTTs.

3) Summary: BBR updates sending rate by multiplying the MIMD factor periodically, the value for flow i in both phases is given below,

$$MIMD_i = \begin{cases} 1.25T_i/(T_i + D(t)) \text{ in pacing controlled phase} \\ 2T_i/(T_i + D(t)^{MIN}) \text{ in Cwnd controlled phase} \end{cases} \tag{7}$$

Equation (7) indicates that a long RTT flow has bigger MIMD factor in both phases, and will eventually obtain a higher sending rate. Moreover, the stability of BBR is achieved only if all flows stop increasing their sending rate, i.e., $MIMD_i = 1$ for any i. Apply this condition to Equation (7), we can solve the equilibrium point as following,

$$D(t)^* = \begin{cases} 0.25T_i \text{ in pacing controlled phase} \\ T_i \text{ in Cwnd controlled phase} \end{cases} \tag{8}$$

[1]The reduction is delayed because the bandwidth estimation selects the the maximum throughput of the past 10 RTTs.

We already know that all flows share the same bottleneck buffer observe the same queuing delay. Therefore, in either phase, stability is achievable only if all flows have the same RTT.

B. Latency of competing flows

When multiple BBR flows share the same bottleneck link, the bandwidth share of each flow is usually overestimated and BBR stays in Cwnd controlled phase most of the time. According to Equation (8), in Cwnd controlled phase, the queuing delay must equal the minimum RTT of every flow to obtain the equilibrium point. That is, BBR flows converge to a constant sending rate requires two conditions: all flows have the same minimum RTT and the queuing delay equals this minimum RTT. This explains the cause of additional queuing delay with competing flows.

C. Fairness of competing flows

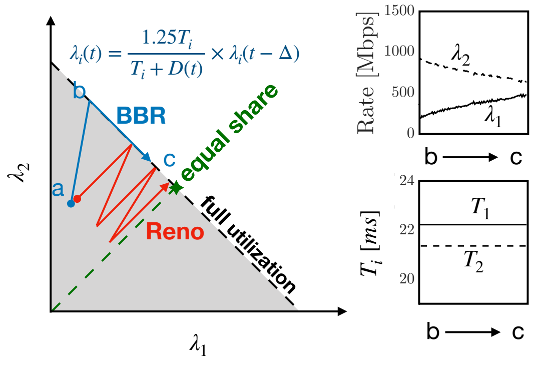
We compare the fairness of BBR with Reno using an example of two flows sharing the bottleneck. The results are given in Figures 1a and 1b, where λ_1 and λ_2 represents the throughput of the two flows, respectively. The intersection of full bandwidth utilization and equal bandwidth share line represents the optimal point.

1) Flows with same RTTs: As shown in [6], Reno flows adapt the AIMD algorithm and grow Cwnd with the same speed from the starting point. When the bandwidth is fully utilized, packet losses cause Cwnd and throughput to halve and start growing again from a more fair point. After several drops, Reno flows converge to the optimal point, as showed in Figure 1a.

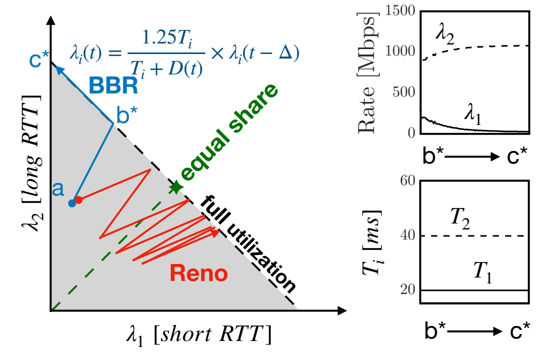
For BBR flows, the MIMD factor that controls sending rate is the same. Therefore, the ratio of bandwidth share at point a is the same as the ratio at point b, and these two flows should converge to point b, as discussed in [6]. However, in practice, even though the true RTT_{min} is the same for both flows, the BBR flow with lower throughput observes a higher RTT_{min} during probe RTT state and therefore has a bigger MIMD factor. This makes sense because when the low rate flow enters probe RTT state, the queue is still occupied by packets from the high rate flow. This prevents the low rate flow from observing the true RTT_{min}. On the other hand, when the high rate flow enters probe RTT state, the queue is almost drained and allows it to detect a lower RTT_{min}. As a result, these two flows eventually converge to a fair bandwidth share as observed from the simulation results provided in Figure 1a. After they have the same throughput, they also perceive the same RTT_{min}.

2) Flows with different RTTs: In this scenario, flow 1 has shorter RTT, i.e., $T_1 < T_2$. From Figure 1b, the Reno flow with shorter RTT grows its Cwnd faster and ends up taking more bandwidth. The bandwidth shared is inversely proportional to each flow's RTT and can be viewed as fair since both flows will utilize roughly equal network resources.

However, the BBR flow with long RTT takes over the entire bandwidth as Figure 1b shows. This can be proved using the MIMD factor in Equation (7). That is, even though the flow with long RTT already has a higher throughput, its MIMD factor is still bigger than the flow with short RTT. Therefore, its throughput will keep increasing until the entire bandwidth is taken, as indicated by point c^* in Figure. 1b.



(a) Flows with the same RTTs



(b) Flows with different RTTs

Fig. 1: TCP Fairness of two competing flows

IV. Improving Latency and Fairness of BBR

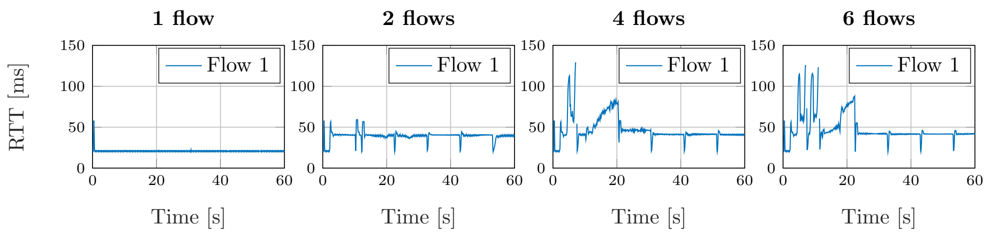
A. Google's Patch for Lower Latency

Google recently released a patch [9] to remove this additional queuing delay. In the probe bandwidth state, the updated BBR allows the second pacing phase, where the gain equals 0.75, to last for more than one RTT until the packets in flight are lower than the BDP, in case the original fixed interval is not long enough to drain the queue.

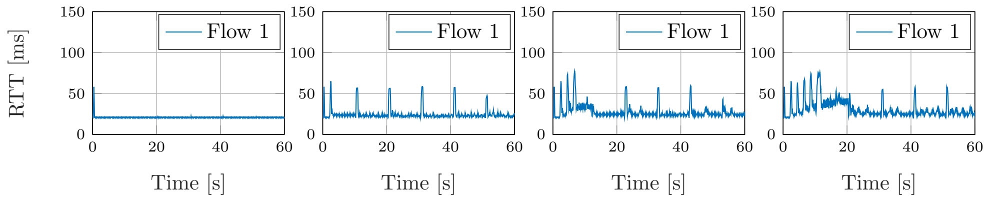
B. Our Improvement for Latency

In the first pacing phase, BBR may inject more packets into the network than the 1.25 $\times$ BDP target if the bandwidth is overestimated. Therefore, our first improvement is to let BBR exit this phase quicker if the number of packets in flight already meets the target.

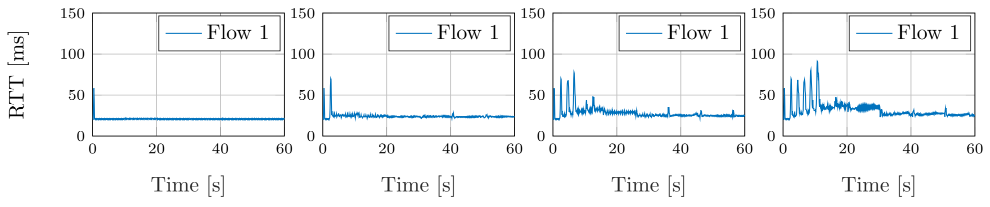
The second one is a faster probe RTT state. Originally, BBR stays in the probe RTT state for 200 ms. However, sometimes the RTT_{min} is observed much before the 200 ms interval is over. Our improved BBR allows the probe RTT state to be terminated whenever a lower RTT is detected, otherwise, it waits for 200 ms.



(a) BBR



(b) BBR with patch



(c) BBR with patch and improvement for latency

Fig. 2: RTT of different number of flows

These improvements do not alter the fundamentals of BBR's algorithm, but significantly reduces the RTT jitter caused by RTT probing. In our simulations, the original algorithm causes the communication to freeze for 200 ms, but the improved one usually terminates around 2 $\times$ RTT, which is more desirable for the connections requiring low latency.

C. *Our Improvement for Fairness*

In Section III, we showed that the BBR flow with long RTT captures almost the entire bandwidth. The previous latency improvement reduces queuing delay and keeps BBR operating in the pacing controlled phase. This mitigates the fairness issue, but the bandwidth is still not evenly distributed among flows with different RTTs due to the differing MIMD factors obtained from Equation (3). To fix this issue, BBR flows with different RTTs should have the same MIMD factor to regulate sending rate to achieve the equilibrium point. We replace the fixed 1.25 and 0.75 gain with (1+α_i) and (1-α_i) in the first two pacing phases. So Equation (3) is updated with:

$$C_i(t) = \frac{(1+\alpha_i)T_i}{T_i + D(t)} \times C_i(t - 8T_i) \tag{9}$$

Consequently, in Equation (8), the equilibrium point in pacing controlled phase is updated with,

$$D(t)^* = \alpha_i T_i \tag{10}$$

Dividing T_i at both sides of Equation (10) implies that as long as all flows configure their α_i to be $D(t)^*/T_i$, they will converge to a constant sending rate and achieve equilibrium, as represented by the point b^* in Figure. 1b. Even though this method prevents the flow with longer RTT takes the entire bandwidth, as indicated by point c^* in Figure. 1b, it does not guarantee a fair bandwidth share among these flows. In the future work, we plan to replace the eight pacing patterns with a continuous pacing function, such as a line or cubic curve, to provide even better fairness. Since the queuing delay may be zero or very high, we limit the range of α to within [0.05, 0.95]. The queuing delay $D(t)$ is measured as the RTT - RTT_{min}, and these parameters are updated every eight pacing phases to align with the size of the pacing gain pattern.

V. Simulation Results in ns-3

We conducted several experiments in the ns-3 simulator [10] using the topology presented in Figure 3, where 3 servers are connected with the router with different propagation delays, 9 ms, 19 ms and 39 ms. The bottleneck link is between the router and the client, which has a 1.1 Gbps rate and 1 ms delay. So the RTT between the servers and client without buffering is 20 ms, 40 ms and 80 ms. The ns-3 implementation of BBR is taken from [11].

A. *Additional Queuing Delay due to Competing Flows*

In this section, we study the performance of BBR with the applied patch along with our improvements targeting the additional queuing delay problem.

We establish one BBR flow between server 1 and the client. Then, we repeat the same experiment while increasing the flows to 2, 4 and 6. Each flow is started 2 seconds after

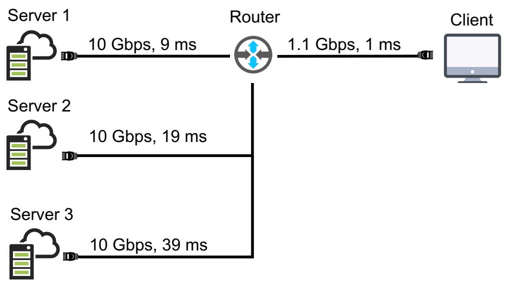


Fig. 3: Simulation setup

TABLE I: Results of flows with different RTTs

		Original	Improved
Total rate [Gbps]	2 FL	1.08	1.09
	3 FL	1.07	1.09
Jain's fairness index	2 FL	0.56	0.99
	3 FL	0.42	0.78
Average bottleneck queue size [MB]	2 FL	4.22	1.74
	3 FL	7.70	3.64

the previous one. The buffer size is 8 × BDP = 20 MB, large enough to prevent packet drops. As Figure 2a shows, the original BBR implementation doubles the RTT in the multiple flow scenarios.

The results of BBR with the patch are provided in Figure 2b. As expected, it removes the additional queuing delay. However, some RTT spikes occurred after every probe RTT state with multiple flows. This is because when one flow enters the probe RTT state and sends the default 4 packets, all the other flows quickly increase sending rates to grab the bandwidth that was vacated by the probing RTT flow. When the flow enters the probe bandwidth state gain, it restores the original sending rate and causes the queue to build up because the bandwidth is already taken by other flows. All flows take some time to react to this sudden RTT increase before draining the queued packets.

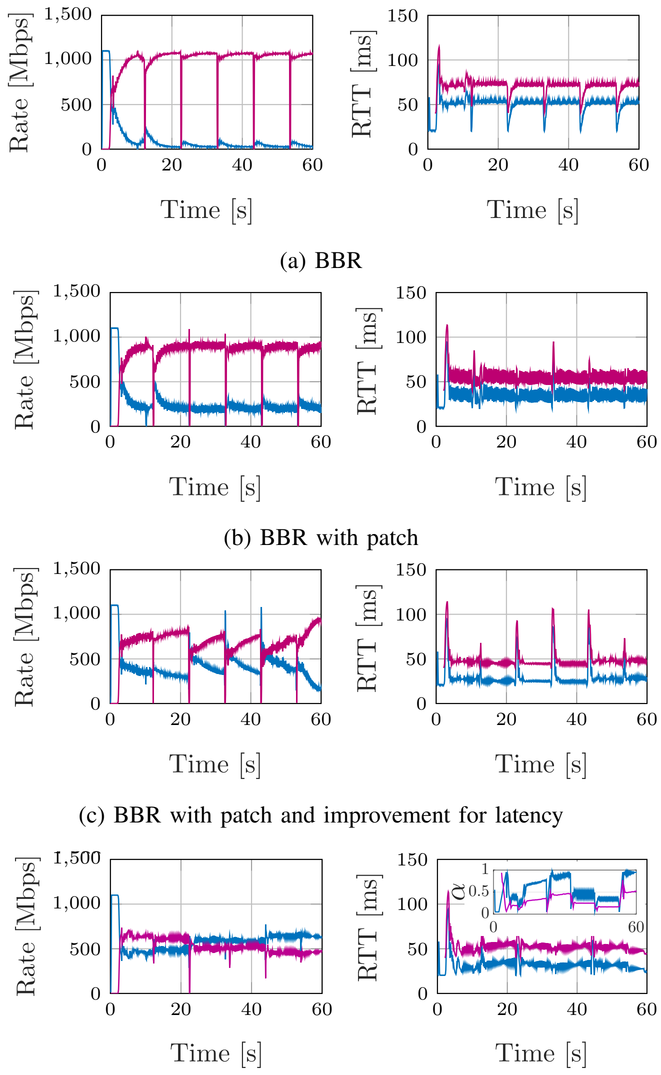
The results of our improvement for latency is shown in Figure 2c. Some of the spikes are removed thanks to the fast probe RTT state. The RTT is also smoother due to the improved first pacing phase. However, there are still some spikes whenever a new flow starts, which cannot be removed because the new flows raise the RTT to probe the bandwidth.

With the same RTT, the bandwidth is shared fairly among the BBR flows. Our improved BBR is also fair among the identical RTT flows, therefore the throughput results are not presented in these figures.

B. Fairness among Different RTT Flows

Another issue of BBR is that long RTT flows "steal" more bandwidth from short RTT flows. We conducted experiments of two and three flows over a 8 × BDP buffer, because the lack of fairness is mainly over large buffer configurations, as described in [8]. In the two flow case, we establish a BBR flow from server 1 to the client, and start a flow from server 2 to the client after 2 seconds. In the three flow case, we start another flow from server 3 to the client after 4 seconds.

From the analysis in Section III and Section IV, the fairness issue can be mitigated by reducing the latency, and achieving even better fairness by using the adaptive pacing factor. As Figure 4a shows, the fairness among the two flows with different RTTs is very poor, and the longest RTT flow occupies most of the bandwidth. Figure 4b shows the results of the Google's patch, which reduces the queued packets and provides better fairness. Our improvement for latency produces better fairness and lower latency, as shown in Figure 4c. However, even if these different RTT flows start with the same bandwidth share, they still diverge over time, which happens at 22s, 32s and 42s. Finally, with our improvement for fairness, both flows observe roughly the same queuing delay, but the larger pacing gain factor α (the original fixed gain is 0.25), as shown at the



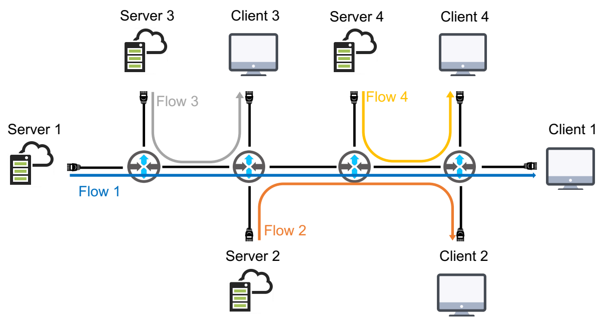
(a) BBR

(b) BBR with patch

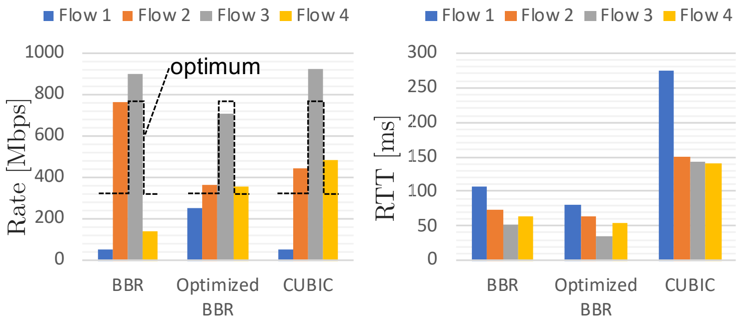
(c) BBR with patch and improvement for latency

(d) BBR with patch and improvement for latency and fairness

Fig. 4: Rate and RTT plot of two flows (20ms, 40ms)



(a) Topology



(b) Simulation results with deep buffers

Fig. 5: A parking lot network scenario

top corner of RTT plot, cause the short RTT flow to probe the bandwidth more aggressively as compared to the original BBR, and provides better fairness.

The results for the three flow case are similar to the two flow case, and are therefore not presented in these figures. However, we repeated both experiments multiple times and reported the averaged results in Table I.

C. Performance in a Parking Lot Network

We simulate a parking lot network with 4 competing flows, as shown in Figure 5a, to study the performance of BBR in a realistic scenario. The ethernet cable connecting each node has a 1Gbps rate and 5 ms one way delay; therefore, the total round trip propagation delay for these four flows without buffering is 50 ms, 40 ms, 30 ms and 30ms, respectively. The buffer size is 14 MB, which is roughly $2 \times BDP$ of the longest link. We start four flows at the same time and repeat this experiment multiple times. The averaged results are provided in Figure 5b. As expected, BBR produces poor fairness and flows 2 and 3 utilize more bandwidth than other flows. However, the improved BBR performs much better in terms of the max-min fairness [12]. That is, flows 1, 2 and 4 that share the bottleneck have similar throughput. Flow 1 is slightly lower than the other two because it is also competing on a link with flow 3. The max-min fairness throughputs, labeled as optimum, is also shown in the plots as a reference. Moreover, the RTTs are also reduced with the improved BBR. The results from CUBIC is also provided. The fairness of CUBIC is better than BBR since flow 2 and flow 3 have the same throughput, but worse than the improved BBR because the throughput of flow 1 is lower. CUBIC also has the highest RTT in this deep buffer configuration.

VI. Conclusions

By means of ns-3 simulations, we show that Google's patch helps BBR reduce the additional queuing delay problem. We also show how our additional improvement results in reduced latency and increased fairness. Our improved BBR exhibits improvements across flows with varying RTTs in both a simple network and the parking lot network. Moreover, the latency improvements can be safely integrated since they do not fundamentally alter BBR's algorithm. In terms of the fairness improvement, we argue that it is vital to introduce an adaptive pacing gain control in the first two pacing phases. As part of the future work, we plan to extend our analytical framework in a more general stochastic setting and change the eight pacing patterns to a continuous pacing function, such as a line or cubic curve.

Acknowledgment

This research is supported by the Semiconductor Research Corporation (SRC), by the New York State Center for Advanced Technology in Telecommunications (CATT) and the NYU WIRELESS Industrial Affiliates Program.

References

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

[2] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[3] M. Zhang, M. Mezzavilla, R. Ford, S. Rangan, S. Panwar, E. Mellios, D. Kong, A. Nix, and M. Zorzi, "Transport layer performance in 5G mmWave cellular," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE, 2016, pp. 730–735.

[4] M. Zhang, M. Mezzavilla, J. Zhu, S. Rangan, and S. Panwar, "TCP dynamics over mmWave links," in *Signal Processing Advances in Wireless Communications (SPAWC), 2017 IEEE 18th International Workshop on*. IEEE, 2017, pp. 1–6.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[6] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[7] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi, "Will TCP work in mmWave 5G cellular networks?" *arXiv preprint arXiv:1806.05783*, 2018.

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

[9] N. Cardwell. (2018) RFC: Linux TCP BBR patches for higher wifi throughput and lower queuing delays. [Online]. Available: https://groups.google.com/forum/#!topic/bbr-dev/8pgyOyUavvY

[10] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-end simulation of 5G mmWave networks," *IEEE Communications Surveys & Tutorials*, 2018.

[11] V. Jain, *GitHub:bbr-dev*. [Online]. Available: https://github.com/Vivek-anand-jain/ns-3-dev-git/tree/bbr-dev

[12] J.-Y. Le Boudec, "Rate adaptation, congestion control and fairness: A tutorial," *Web page, November*, 2005.