A New Congestion-Based Congestion Control for Low Latency and Low Packet Drop Rate

Satoshi Utsumi
Cluster of Science and Technology
Fukushima University
Fukushima, Japan
u-satoshi@sss.fukushima-u.ac.jp

Go Hasegawa
Research Institute of Electrical Communication
Tohoku University
Sendai, Japan
hasegawa@riec.tohoku.ac.jp

***Abstract*—In 2016, Google published BBR, a congestion-based congestion control algorithm that focuses on achieving the optimal operating point with the minimum delay and maximum throughput. When a single BBR flow occupies a bottleneck link, the packet sending rate and number of inflight packets of the flow achieve the optimal operating point. However, multiple BBR flows sharing a bottleneck link lead to a large latency and high packet drop rate. This is due to the maximum throughput being used for the packet sending rate, which makes the operating point far from optimal. In this paper, we propose a new congestion-based congestion control algorithm, called Low Latency and Low Drop Rate (L3DR), that keeps the operating point nearly optimal, even when multiple flows share a bottleneck link, by measuring the available bandwidth to be used for the packet sending rate. We conducted performance evaluation experiments in emulation network environments. The results indicate that L3DR can reduce the buffering delay to 38.5% and packet drop rate to 8.0% compared with BBR while achieving high throughput and high per-flow fairness.**

***Index Terms*—BBR, Available Bandwidth, Congestion-Based Congestion Control**

I. Introduction

In 2016, Google published the congestion-based congestion control algorithm Bottleneck Bandwidth and Round-trip propagation time (BBR) [1]. It focuses on the operating point, which is defined as the conditions in a network, including the throughput, latency, and packet drop rate at the bottleneck link. The design purpose of BBR is to configure the packet sending rate of a flow so that the operating point of the network is optimal, where the network has the maximum total throughput, minimum latency, and minimum packet drop rate.

BBR determines its packet sending rate on the basis of the observed maximum throughput. When a single BBR flow occupies a bottleneck link, the operating point converges to be optimal because the maximum throughput observed by a single BBR flow is equal to the bottleneck link bandwidth [2]. However, when multiple BBR flows share a bottleneck link, the round trip times (RTTs) of the flows increase due to the buffering delay at the bottleneck link. The reason is that the sum of the packet sending rates of the multiple BBR flows exceeds the available bandwidth [2]–[4]. Hock et al. [2] clarified this problem through analytical modeling and emulation experiments. Other studies [3] and [4] also confirmed this problem through emulation experiments. However, these studies do not offer any methods of solving this problem.

We propose a congestion-based congestion control algorithm for keeping the operating point nearly optimal even when multiple flows share a bottleneck link. We first explain why the sum of the maximum throughputs observed by multiple BBR flows exceeds the available bandwidth in the above situation. We then propose our congestion-based congestion control algorithm, called *Low Latency and Low Drop Rate* (*L3DR*). L3DR measures the moving average of the observed throughput to be used for the packet sending rate instead of the observed maximum throughput.

Performance evaluation experiments with the Linux implementation of L3DR are conducted in emulation network environments. The performance metrics are throughput, latency, packet drop rate, and per-flow fairness. We compare the performance of L3DR with other congestion control algorithms, including BBR variants [1], [5]–[8]. We also evaluate its transient behavior against network environmental changes.

The remainder of this paper is organized as follows. In Section II, we describe related works. In Section III, we give an overview of BBR and the problem with it. In Section IV, we provide an overview and details on the L3DR congestion control algorithm. We compare L3DR with other congestion control algorithms in emulation network environments and discuss the results in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

Hock et al. [2] analyzed the behavior of BBR and conducted experimental evaluations for multiple BBR flows sharing a bottleneck link. The results indicated that BBR does not inherently achieve the optimal operating point in a network when multiple BBR flows share a bottleneck link, resulting in an excessive increase in the number of inflight packets, queues at the bottleneck buffer, and packet drops. Other studies [3] and [4] reproduced these results, that is, persistent overloading at the bottleneck link, stretching of queues in bottleneck buffers, and increased queuing delays and packet drops when multiple BBR flows share a bottleneck link, using the network emulator Mininet [9]. However, none of these studies [2]–[4] offers any methods for solving this problem.

Previous studies improved BBR to determine its packet sending rate [5], [10]–[12]. BBRx [5] is an algorithm that modifies the overly aggressive packet sending rate. It is based on online learning to maximize an objective function calculated from the throughput and packet drop rate. However, BBRx does not consider per-flow fairness. Su et al. [10] proposed a modification to BBR that prevents the seding rate from being too large on the basis of throughput observation by acknowledgment (ACK) compression [13]. However, the study did not consider the case in which multiple BBR flows share a bottleneck link in its design and evaluation.

BBR-S [11] maintains low latency in fast-varying wireless channels by replacing the maximum filter on observed throughput for determining the BBR packet sending rate with an adaptive Kalman filter [14], [15]. The authors evaluated the buffering delay for multiple flows in environments with varying bottleneck link bandwidth and per-flow fairness when multiple flows share a bottleneck link. However, they did not evaluate the packet drop rate when multiple flows share a bottleneck link. They evaluated BBR-S using the network simulator ns-3 [16] but did not evaluate it in an actual environment. Utsumi and Hasegawa [12] used the moving average of observed throughput for determining the packet sending rate with their algorithm. However, their algorithm does not consider per-flow fairness. The authors evaluated their algorithm using only the network simulator ns-2 [17].

Other previous studies [18] and [19] introduced methods for estimating the appropriate packet sending rate for Transmission Control Protocol (TCP). Tay [18] proposed a method for estimating the minimum delay and maximum throughput with statistical regression but did not evaluate the method. Fridovich et al. [19] proposed a method for calculating the optimal TCP packet sending rate using model predictive control. They applied their method to TCP Reno [20] and H-TCP [21] and evaluated its performance. However, they did not compare it quantitatively with other congestion control algorithms.

III. BBR Congestion Control Algorithm and its Problem

A. BBR congestion control algorithm

Basically, a BBR flow repeats two phases, ProbeBW and ProbeRTT. In the ProbeBW phase, the duration of which is around 10 sec, the flow sets the packet sending rate to the maximum throughput observed in the recent past. Additionally, to search for an increase in the available bandwidth, BBR temporarily increases the packet sending rate to larger than the observed maximum throughput. Specifically, the BBR sender determines the packet sending rate by multiplying the observed maximum throughput by using a parameter called *pacing gain*, denoted as g_{p}. The g_{p} changes cyclically every observed minimum RTT. The default configuration of g_{p} is $\gamma, (2-\gamma), 1, 1, 1, 1, 1, 1$ for every eight cycles, where γ is fixedly set to 1.25 in the Linux implementation. In the ProbeRTT phase, the duration of which is the sum of one RTT and 200 msec, the flow uses a small congestion window to search for the minimum RTT. It also records the last time when the minimum RTT was observed as the reference time so that competing BBR flows enter the ProbeRTT phase simultaneously.

B. Problem with BBR congestion control algorithm

The *optimal operating point* of a network, defined in BBR, is the condition in which the bottleneck link bandwidth is fully used, and no queuing delay and no packet drop occurs. As shown in previous studies [2]–[4], BBR's operating point is far from optimal when multiple flows share a bottleneck link. This is because the sum of the packet sending rates of the BBR flows significantly exceeds the available bandwidth. The reasons are as follows. As mentioned above, a BBR flow increases the packet sending rate to larger than the maximum observed throughput temporarily. When multiple BBR flows share a bottleneck link, their timings at which the packet sending rates increase are generally not synchronized. The flows with an increased packet sending rate would then lead to a larger throughput than the other flows. Therefore, when the available bandwidth remains unchanged, the sum of the packet sending rates of the BBR flows significantly exceeds the available bandwidth. The available bandwidth is defined as the maximum packet arrival rate that does not incur buffering delays and packet drops at the bottleneck link.

In 2019, Google published BBRv2 [8]. BBRv2 has also not solved the above problem. In our performance evaluation experiments presented in Section V, we use BBRv1 [1] as the original BBR and BBRv2 for performance comparison.

IV. L3DR: New Congestion-Based Congestion Control Algorithm

In this section, we present the solution to the problem of the original BBR discussed in Section III and explain the proposed congestion control algorithm, L3DR.

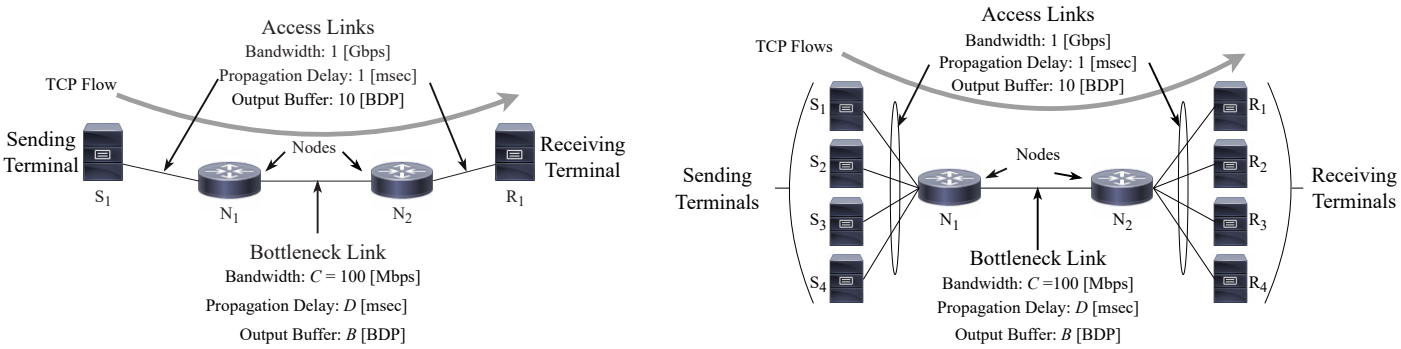
A. Available bandwidth-based packet sending rate

The original BBR uses the maximum throughput observed in the recent past as the *base value* of the packet sending rate. As discussed in Section III, this method overloads a bottleneck link when multiple BBR flows share that link. Therefore, the L3DR congestion control algorithm uses the available bandwidth, defined as $AvailBw$, as the base value of the packet sending rate to maintain the optimal operating point of the network. We discuss the analytical justification for this modification in subsection IV.D and confirm its effectiveness in the performance evaluation experiments in Section V.

B. Calculating $AvailBw$

L3DR uses the moving average of the throughput observed in the ProbeBW phase as $AvailBw$. It does not include some of the observed throughput values in calculating $AvailBw$. This is because the throughput observed when the packet sending rate is larger or smaller than the actual available bandwidth should be excluded for calculating $AvailBw$. Specifically, L3DR calculates $AvailBw$ as follows.

As in the original BBR, L3DR uses the cyclical changes of $g_{\mathrm{p}} = \gamma, (2-\gamma), 1, 1, 1, 1, 1, 1$ for detecting an increase in



(a) For single flow

(b) For multiple flows

Fig. 1: Network topologies for performance evaluation.

the available bandwidth. However, we change the value of γ adaptively as described in the next subsection. Also, for maintaining fairness among competing connections, the initial value of $g_{\rm p}$ is randomly chosen among seven candidates for the pacing gain, excluding $g_{\rm p} = (2-\gamma)$, at every beginning of the eight cycles, as the same as the original BBR. L3DR measures RTTs as well as throughput in the cycle of $g_{\rm p} = \gamma$. When it does not detect an increase in RTT, it does not use only the throughput observed in the following cycle of $g_{\rm p} = (2-\gamma)$ for calculating $AvailBw$ because it is highly expected to be smaller than the available bandwidth. Otherwise, when it detects an increase in RTT, it does not use the throughput observed in the cycles of $g_{\rm p} = \gamma$ and $(2-\gamma)$ because the packet sending rates in these cycles cannot match the available bandwidth.

L3DR does not use the throughput observed in the ProbeRTT and loss recovery phases, where the congestion window is relatively small.

C. Adaptive setting of parameter γ

The original BBR uses a parameter γ to detect an increase in the available bandwidth. The fixed γ in the original BBR cannot follow the temporal changes in the available bandwidth of the bottleneck link. Therefore, L3DR introduces a mechanism to adaptively change the value of γ in accordance with the degree of load on the bottleneck link. Specifically, when no increase in RTT is observed in the cycle of $g_{\rm p} = \gamma$, L3DR increases the value of γ; otherwise, it reduces the value of γ. When increasing and decreasing the value of γ, we apply additive-increase multiplicative-decrease (AIMD) [22] with a limitation of $1 < \gamma \leq 2$. L3DR also resets the value of γ at the end of the ProbeRTT phase to maintain per-flow fairness.

D. Operating point analysis

We mathematically explain the behavior of the operating point with the L3DR congestion control algorithm. Assume that N L3DR flows share a bottleneck link with the available bandwidth A, which remains unchanged. For a flow j, let $s(j)$ be the base value of the packet sending rate in a certain cycle, $g_{\rm p}(j)$ be the pacing gain at the cycle, $\gamma(j)$ be a value of γ, and $b(j)$ be the throughput observed by the packets transmitted in the cycle. Then, $b(j)$ satisfies the following equation.

$$b(j) = g_{\rm p}(j)s(j)\min(A/\Sigma_{j=1}^{N}s(j), 1) \quad (1)$$

Here, we define $s'(j)$ as the base value of the packet sending rate after observing the throughput $b(j)$.

When $\Sigma_{j=1}^{N}s(j) \leq A$, that is, when the utilization of the bottleneck link is equal to, or less than 100%, $b(j) = g_{\rm p}(j)s(j)$ holds from Eq. (1). It is also expected that the observed RTT will not increase. L3DR does not use the $b(j)$ in cycles of $g_{\rm p}(j) = (2-\gamma(j))$ to determine $s'(j)$ by calculating the moving average of $b(j)$. Therefore, $s'(j) \geq s(j)$ holds.

When $\Sigma_{j=1}^{N}s(j) > A$, that is, when the sum of the packet sending rate is larger than the available bandwidth, $b(j) < g_{\rm p}(j)s(j)$ holds from Eq. (1). In this case, the flow would observe the increased RTT. L3DR does not use the $b(j)$ in cycles of $g_{\rm p}(j) = \gamma(j)$ and $g_{\rm p}(j) = (2-\gamma(j))$ to determine $s'(j)$. Therefore, $s'(j) < s(j)$ holds.

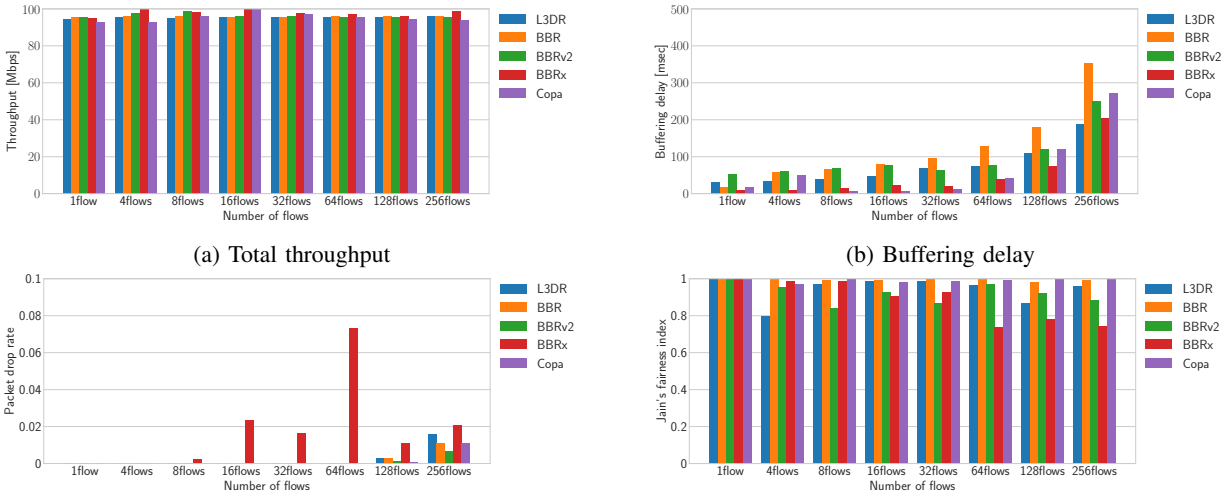
The above discussions on the difference between $s'(j)$ and $s(j)$ indicate that the base value of the packet sending rate approaches A in the following cycle. This means that L3DR can keep the operating point nearly optimal.

V. Performance Evaluation in Emulation Network Environments

We evaluated the performance of the L3DR congestion control algorithm in emulation network environments with a Linux implementation.

A. Experimental setup

We used the single-flow and multi-flow network topologies generated by Mininet in Figs. 1a and 1b, respectively. In the single-flow network topology in Fig. 1a, the sending terminal was equipped with one TCP sender. In the multi-flow network topology in Fig. 1b, each sending terminal was equipped with the same number of TCP senders, which was 1, 2, 4, 8, 16, 32, or 64, resulting in the total number of flows in the network being 4, 8, 16, 32, 64, 128, or 256, respectively. Each TCP sender had either L3DR, BBR, BBRv2 [8], BBRx [5] or Copa [6]. We used Copa without competitive mode because



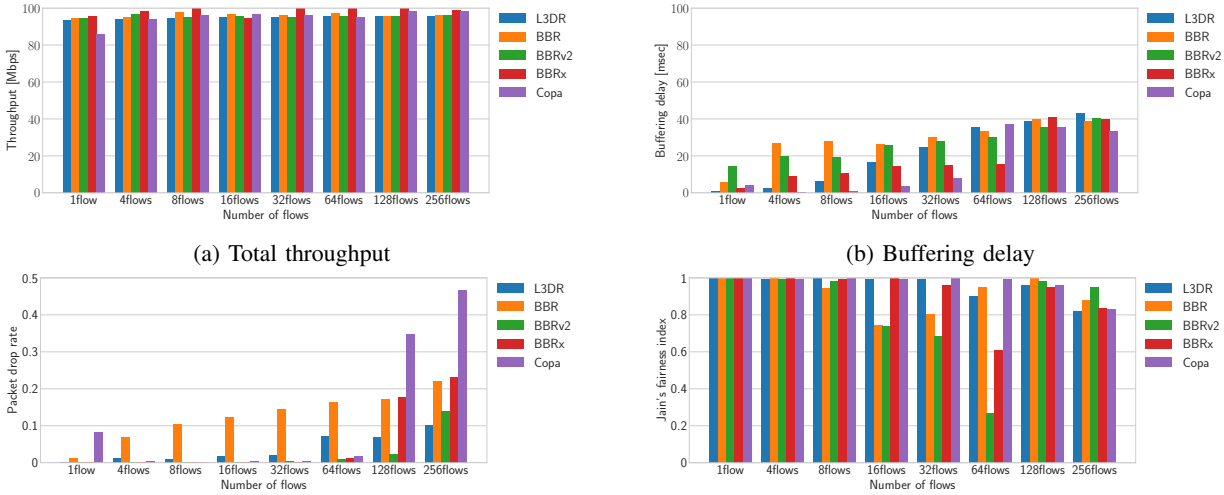
(a) Total throughput

(b) Buffering delay

(c) Packet drop rate

(d) Per-flow fairness

Fig. 2: Error-free network environment with $C = 100$ [Mbps], $D = 18$ [msec], and $B = 10$ [BDP].



(a) Total throughput

(b) Buffering delay

(c) Packet drop rate

(d) Per-flow fairness

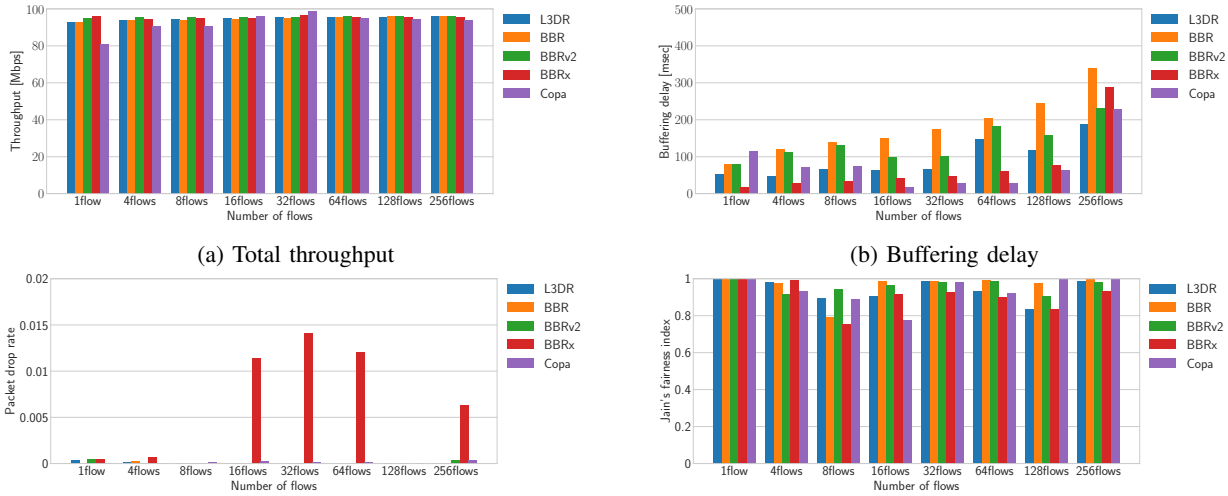
Fig. 3: Error-free network environment with $C = 100$ [Mbps], $D = 18$ [msec], and $B = 1$ [BDP] (small buffer).

we compared the performance of L3DR with a native delay-based approach. All TCP senders had the same congestion control algorithm. The access links had a bandwidth of 1 [Gbps] and propagation delay of 1 [msec]. The bottleneck link had a bandwidth of $C = 100$ [Mbps], propagation delay of D [msec], and output buffer of B [BDP]. Here, BDP refers to bandwidth-delay products, where 1 [BDP] is defined as the product of the bottleneck link bandwidth and the round-trip propagation delay between sending and receiving terminals. Each receiving terminal was equipped with a TCP receiver with selective ACK, timestamp, and delayed ACK options. The sizes of the data and ACK packets were set to 1,500 and 40 [bytes], respectively. Each TCP sender started bulk data transmission to the corresponding TCP receiver by using iPerf [23] at 0 [sec] and terminated it at 120 [sec].

B. Metrics

The following metrics were used for the performance evaluation.

- Total throughput.
- Average buffering delay. Buffering delay is obtained from the difference between RTT and the round-trip propagation delay between sending and receiving terminals.
- Packet drop rate, which is defined as $\frac{L}{T}$, where T [packets] is the number of all packets transmitted from all sending terminals, and L [packets] is the number of dropped packets detected at the terminals, that is, the number of packets retransmitted by TCP senders.
- Per-flow fairness, which is defined by Jain's fairness index [24].



(a) Total throughput

(b) Buffering delay

(c) Packet drop rate

(d) Per-flow fairness

Fig. 4: Error-free network environment with $C = 100$ [Mbps], $D = 48$ [msec], and $B = 10$ [BDP] (large propagation delay).

C. Performance evaluation

We compared the performance of L3DR and the other congestion control algorithms for the case where there is no packet loss due to errors in the network. Fig. 2 shows the evaluation results with $C = 100$ [Mbps], $D = 18$ [msec], and $B = 10$ [BDP]. The total throughput, buffering delay, packet drop rate, and per-flow fairness as a function of the number of concurrent flows are plotted in Figs. 2a, 2b, 2c, and 2d, respectively. Compared with BBR, L3DR reduced the buffering delay to a minimum of 53.3% with high throughput, low packet drop rates, and relatively high per-flow fairness.

Next, we assessed the performance in the network with a small buffer. Fig. 3 shows the evaluation results with $C = 100$ [Mbps], $D = 18$ [msec], and $B = 1$ [BDP]. L3DR reduced the packet drop rate to a minimum of 8% compared with BBR, while maintaining high throughput, low latency, and relatively high per-flow fairness. When the number of flows was larger, the packet drop rates of BBR, BBRx and Copa were also larger. BBRv2 might have a very low per-flow fairness when the buffer size of a bottleneck is small.

Then, we evaluated the performance in the network with a large propagation delay. Fig. 4 shows the evaluation results with $C = 100$ [Mbps], $D = 48$ [msec], and $B = 10$ [BDP]. Compared with BBR, L3DR reduced the buffering delay to a minimum of 38.5% while maintaining similar throughput and per-flow fairness.

From the above results, L3DR demonstrated robust performance for throughput, buffering delay, packet drop rate, and per-flow fairness when network parameters were changed. L3DR kept the buffering delays and packet drop rate low regardless of the length of the propagation delay. Even in environments where the output buffer was only 1 [BDP], L3DR kept the packet drop rate small. Per-flow fairness was stable at around 0.8 or better in all evaluated environments.

BBR was not able to keep a small buffering delay when multiple flows shared a bottleneck link. Also, the per-flow fairness of BBRv2 was degraded in environments where the output buffer was small. BBRx could not achieve high per-flow fairness for some environments, and a low packet drop rate even in environments where the output buffer was large. Copa performed poorly in buffering delay and packet drop rate in environments where many flows shared a bottleneck link.

D. Transient behaviors

We used the network topology in Fig. 5 to confirm the transient behaviors of the L3DR congestion control algorithm. As shown in Fig. 5, sending terminal S_1 was equipped with a TCP sender. Sending terminal S_2 was equipped with a User Datagram Protocol (UDP) sender, receiving terminal R_1 was equipped with a TCP receiver, and receiving terminal R_2 was equipped with a UDP receiver. The parameters of the bottleneck link are $C = 100$ [Mbps], $D = 18$ [msec], and $B = 100$ [BDP] without packet errors. S_1 started bulk data transmission to R_1 at 0 [sec] and terminated it at 120 [sec]. S_2 started a constant bit rate (CBR) transmission of 50 [Mbps] to R_2 at 30 [sec] and terminated it at 90 [sec].

As mentioned in subsection IV.C, when increasing and decreasing the value of γ, we applied AIMD with a limitation of $1 < \gamma \leq 2$. Specifically, when no increase in RTT is detected in the cycle of $g_\mathrm{p} = \gamma$, L3DR updates γ to $\gamma + \Delta\gamma$, where $\Delta\gamma = 0.125$ in our implementation. Otherwise, it updates γ to $\gamma_{\min} + \frac{\gamma - \gamma_{\min}}{2}$, where $\gamma_{\min} = 1.031$. Furthermore, L3DR resets γ to γ_{reset} at the end of each ProbeRTT phase, where $\gamma_{\mathrm{reset}} = 1.75$. We selected $\Delta\gamma = 0.125$, $\gamma_{\min} = 1.031$ and $\gamma_{\mathrm{reset}} = 1.75$ because these values empirically achieved better per-flow fairness.

As shown in Fig. 6, the packet sending rates of L3DR followed the available bandwidth of 50 [Mbps] from 30 to 90 [sec] with a smaller fluctuation. At the change points of the available bandwidth at 30 and 90 [sec], the packet sending rates of L3DR decreased and increased immediately following

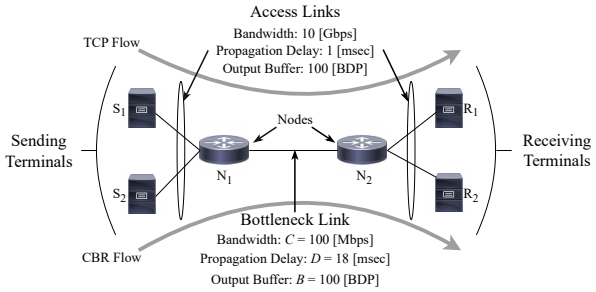


Fig. 5: Network topology for available bandwidth changes.

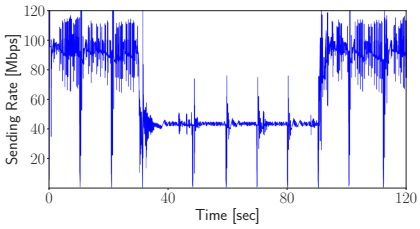


Fig. 6: Time variation of packet sending rates in L3DR flow.

the available bandwidth changes. As shown in Fig. 7, the fluctuation in RTTs for L3DR was small except immediately after 30 [sec]. The large fluctuation in RTTs for L3DR immediately after 30 [sec] was due to the reset of γ_{reset} at the end of the ProbeRTT phase.

VI. Conclusion

In this paper, we proposed a new congestion-based congestion control algorithm, L3DR, for keeping the operating point nearly optimal even when multiple flows share a bottleneck link, for which the original BBR performs poorly. L3DR accurately measures the available bandwidth to be used for the packet sending rate. Also, we introduced a new mechanism for tuning a parameter for detecting an increase in the available bandwidth for the L3DR congestion control algorithm. Performance evaluation experiments with a Linux implementation of L3DR were conducted in emulation network environments. L3DR significantly improved the performance relative to the original BBR in most cases.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP20K11786 and JP21KK0202.

References

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

[2] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 2017, pp. 1–10.

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

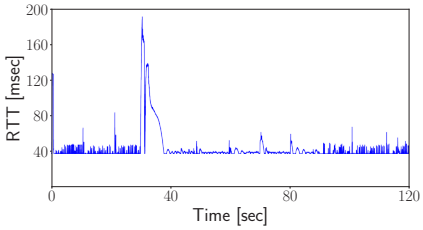


Fig. 7: Time variation of RTT in L3DR flow.

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

[5] J. Chung, F. Li, and B. Kim, "BBRx: Extending BBR for customized TCP performance," in *Proc. NetDev 0x12*, 2018, pp. 262–276.

[6] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the Internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 329–342.

[7] 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.

[8] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control," in *Presentation in ICCRG at IETF 104th meeting*, 2019.

[9] An Instant Virtual Network on your Laptop (or other PC), http://mininet.org/, 2022.

[10] B. Su, X. Jiang, G. Jin, and H. Chen, "Rethinking the rate estimation of BBR congestion control," *Electronics Letters*, vol. 56, no. 5, pp. 239–241, 2020.

[11] F. Chiariotti, A. Zanella, S. Kucera, and H. Claussen, "BBR-S: a low-latency BBR modification for fast-varying connections," *IEEE Access*, vol. 9, pp. 76 364–76 378, 2021.

[12] S. Utsumi and G. Hasegawa, "Refining calculation algorithm for packet pacing rate of BBR," in *2021 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2021)*. IEEE, 2021, pp. 1–6.

[13] J. C. Mogul, "Observing TCP dynamics in real networks," *ACM SIGCOMM Computer Communication Review*, vol. 22, no. 4, pp. 305–317, 1992.

[14] B. Allik, C. Miller, M. J. Piovoso, and R. Zurakowski, "The Tobit Kalman filter: an estimator for censored measurements," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 1, pp. 365–371, 2015.

[15] W. Gao, J. Li, G. Zhou, and Q. Li, "Adaptive Kalman filtering with recursive noise estimator for integrated SINS/DVL systems," *The journal of navigation*, vol. 68, no. 1, pp. 142–161, 2015.

[16] ns-3 Network Simulator, https://www.nsnam.org/, 2022.

[17] The Network Simulator - ns-2, https://www.isi.edu/nsnam/ns/, 2011.

[18] Y. Tay, "A technique to estimate a system's asymptotic delay and throughput," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 3, pp. 156–159, 2018.

[19] D. Fridovich-Keil, N. Hanford, M. P. Chapman, C. J. Tomlin, M. K. Farrens, and D. Ghosal, "A model predictive control approach to flow pacing for TCP," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2017, pp. 988–994.

[20] M. Allman, V. Paxson, and E. Blanton, "RFC5681, TCP congestion control," Tech. Rep., 2009.

[21] G. Armitage, L. Stewart, M. Welzl, and J. Healy, "An independent H-TCP implementation under FreeBSD 7.0: Description and observed behaviour," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 27–38, 2008.

[22] 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.

[23] iPerf - The ultimate speed test tool for TCP, UDP and SCTP, https://iperf.fr/iperf-servers.php, 2022.

[24] R. Jain, D. Arjan, and B. Gojko, "Throughput fairness index: An explanation," *ATM Forum contribution*, vol. 99, no. 45, pp. – – –, 1999.