Detailed Analysis of TCP BBR Implementation in Ns-3 Network Simulator under Heavy Traffic Loads

Toshihiko Kato†, Takahiko Kato‡, Ryo Yamamoto†, and Satoshi Ohzahata†

† University of Electro-Communications
Tokyo, Japan

‡ University of Fukui
Fukui, Japan

e-mail: kato@net.lab.uec.ac.jp, t-kato@u-fukui.ac.jp, ryo-yamamoto@uec.ac.jp, ohzahata@uec.ac.jp

***Abstract*— TCP Bottleneck Bandwidth and Round-trip propagation time (BBR) is a congestion control algorithm that was introduced by Google relatively recently, and it is widely used currently. BBR is based on the congestion-based congestion control which is different from the conventional algorithms. So, a lot of researches have been reported on the performance evaluation. In the early stage, it is done using the BBR code in the Linux operating system over a physical network or a network emulator. Nowadays, a network simulator, such as ns-3 is used for the evaluation. Actually, BBR was introduced in ns-3 version 3.34 released in 2021. However, the BBR code is different from that in Linux, and so it may be possible it behaves differently. This paper reports some results of BBR performance evaluation where sixteen flows share one 1 Gbps bottleneck link with a limited router buffer. The results were different when we used ns-3 version 3.40 and 3.44. We analyzed the BBR source codes in those versions and executed 3.44 BBR code over 3.40 ns-3. This paper also discusses this analysis.**

Keywords- TCP; Congestion Control; BBR; Ns-3 Network Simulator.

I. INTRODUCTION

Along with the proliferation of various types of networks and applications, a lot of TCP variants with different congestion control algorithms are designed, implemented, and widely spread [1]. The congestion control manages the congestion window size, called cwnd according to a network congestion situation. The conventional algorithms are categorized into the loss-based, the delay-based and the hybrid approaches. The loss-based algorithm detects a network congestion by packet losses, and this category includes TCP NewReno [2], HighSpeed TCP [3], CUBIC TCP [4], and Hamilton TCP [5]. The delay-based algorithm detects a congestion by an increase of the Round-Trip Time (RTT), and an example of this category is TCP Vegas [6]. The hybrid algorithm combines the loss-based and delay-based approaches. In TCP Veno [7], the congestion is detected by packet losses and the change of cwnd is controlled according to RTT. In Compound TCP [8], cwnd is the sum of a window size following HighSpeed TCP and one following TCP Vegas, and it behaves in a way such that the former is effective when RTT is small and vice versa.

TCP BBR (Bottleneck Bandwidth and Round-trip propagation time) [9] was recently proposed by Google. It is a new type of congestion control algorithm called congestion-based congestion control. A data sender tries to send data segments according to the Bandwidth Delay Product (BDP) of the TCP connection. A sender measures the bottleneck bandwidth and the minimum RTT independently and estimates the BDP. It does not take care of retransmissions for the congestion control. Another feature of TCP BBR is that it follows the rate-based data transmission. Most of TCP variants adopt the window-based data transmission, where a sender sends out data segments as a burst within the limitation of window size (congestion window and advertised window). Instead, TCP BBR sends data segments in the pace determined by the estimated BDP and the segment size. As a result, data segments are transmitted sparsely and it is highly possible to avoid congestions in multiple TCP flows.

TCP BBR came to be used widely since its proposal. Sawada et al reported that in 2020 TCP BBR was used by 30% of the major web servers. There are a lot of study results on the performance of TCP BBR [10-17]. In early stages, the performance evaluation was done using the BBR code implemented in the Linux kernel over a physical network or a network emulator, such as Mininet [18]. Recently, TCP BBR was introduced in the network simulator, such as ns-3 [19]. Actually, the TCP BBR code was implemented in ns-3 at version 3.34 released in 2021. Some studies performed experiments using ns-3 [14][16][17]. However, TCP BBR in ns-3 is implemented by its original code, and so it may be possible that the code behaves differently from that in Linux.

We performed the TCP BBR throughput test under the condition that sixteen BBR flows share a 1Gbps bottleneck link. When the output buffer of the router connected to the bottleneck link is limited, the throughput of some BBR flows becomes extremely low. Although Hock et al reported the intra-protocol unfairness of TCP BBR due to small bottleneck buffer [10], the results of our experiment are much worse than that. We conducted this experiment using ns-3 whose version is 3.40. We performed the same experiment using ns-3 version 3.44, which is the newest as of writing this paper. The experimental results were different from that of using ns-3 version 3.40, and all of the sixteen BBR flows provided similar throughput. This paper describes the details on these experiments and discusses the difference between ns-3 version 3.40 and 3.44.

The rest of this paper is organized as follows. Section 2 gives some background information including the overview of TCP BBR and the performance evaluation studies reported so far. Section 3 describes the details of the performance evaluation experiments. Section 4 compares the BBR codes implemented in ns-3 version 3.40 and 3.44 in detail. In the end, Section 5 concludes this paper.

II. BACKGROUNDS

A. Overview of TCP BBR

TCP BBR uses estimates for the available bottleneck bandwidth, *BtlBw*, and the minimal round-trip propagation time, *RTprop*, to calculate a path's available BDP. A BBR sender data segments with the pacing rate given by pacing_gain × BltBw × RTprop with the help of pacing. The parameter pacing_gain, which is set to 1 most of the time, is used to control the actual pacing rate.

The BBR algorithm has four different phases: Startup, Drain, Probe Bandwidth, and Probe RTT.

The first phase adapts the exponential Startup behavior where the pacing_gain is set to 2 / ln2 (2.885). Once the measured bandwidth does not increase further, BBR assumes to have reached the bottleneck bandwidth, and shifts to the Drain phase. Here, BBR reduces the pacing_gain to ln2 / 2. Afterwards, BBR enters the Probe Bandwidth phase in which it probes for more available bandwidth. This is performed in eight cycles, each of which takes RTprop. The pacing_gain of each cycle is 5/4, 3/4, 1, 1, 1, 1, 1,1 and 1. This phase BBR continuously samples the bandwidth and uses the maximum as the BtlBw estimator, whereby values are valid for the timespan of ten RTprop. After not measuring a new RTprop value for ten seconds, BBR stops probing for bandwidth and enters the Probe RTT phase. During this phase the bandwidth is reduced to four packets to drain any possible queue and get a real estimation of the RTT. This phase is kept for 200 ms plus one RTT. If a new minimum value is measured, RTprop is updated and valid for time window W_R (typically, ten seconds).

B. Studies on Performance Evaluation of TCP BBR

Table I summarizes the performance evaluation studies of TCP BBR. It shows the BBR codes and the network environment used by the evaluation, the target TCP congestion control algorithms, and the maximum numbers of TCP flows examined in the experiment. TCP BBR was proposed in 2017, and introduced to the Linux operating system. So the evaluations in the early stage used the BBR code in Linux [10-13][15]. The BBR software was introduced to the ns-3 network simulator by Vivek Jain et al [20] in 2018. This software was implemented over ns-3 version 3.27, which was released in 2017. [14] and [16] used this version of TCP BBR. TCP BBR was introduced into ns-3 officially in the version 3.34 released in 2021. It is considered that [17] used this or later version of ns-3.

TABLE I. PERFORMANCE EVALUATION STUDIES OF TCP BBR.

reference	BBR code	network	evaluation target	max number of flows
[10]	Linux	physical	BBR, BBR + CUBIC	6 BBRs, 1 BBR + 1 CUBIC
[11]	Linux	physical	BBR + CUBIC	10 BBRs + 10 CUBICs
[12]	Linux	Mininet	BBR, BBR + CUBIC	6 BBRs, 10 BBRs + 10 CUBICs
[13]	Linux	physical	BBR, BBR + CUBIC	8 BBRs, 3 BBRs + 3 CUBICs
[14]	ns-3 (Vivek Jain's version)	ns-3	BBR, BBR + BIC, BBR + BIC + NewReno + Vegas	2 BBRs, 2 BBRs + 2 BIC, 1 BBR + 1 BIC + 1 NewReno + 1 Vegas
[15]	Linux	Mininet	BBR, (Delay-aware BBR)	2 BBRs
[16]	ns-3 (Vivek Jain's version)	ns-3	BBR, (other BBR variants including BBRv2)	4 BBRs
[17]	ns-3	ns-3	BBR, BBR + CUBIC	8 BBRs, 2 BBRs + 2 CUBICs

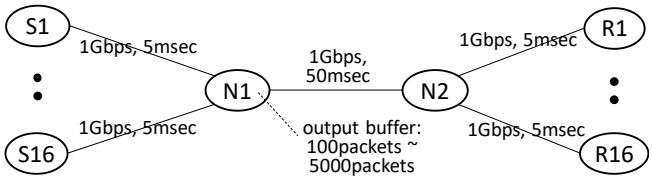


Figure 1. Performance evaluation network.

The performance evaluation experiments were done for single BBR, multiple BBRs and mixture of multiple BBRs and other congestion control algorithms. However, the number of flows were limited, ten BBR flows at the maximum. In contrast with them, our performance evaluation used sixteen BBR flows when the bottleneck buffer is limited.

III. PERFORMANCE EVALUATION

A. Experimental Setup

Figure 1 shows the network configuration used in this evaluation, which is a dumbbell configuration where sixteen data senders (S1 through S16) and the corresponding receivers (R1 through R16) are connected through two routers (N1 and N2). The bandwidth and transmission delay are specified in the figure, where the backbone link between N1 and N2 has a longer delay, 50 msec, than the other links. The output buffer in N1 is set to 100 packets through 5,000 packets. The maximum segment size is 1,420 bytes. Since RTT is 120 msec and the bottleneck bandwidth is 1 Gbps, the BDP in this experiment is 15 MB, i.e., around 10,000 packets.

The details of ns-3 parameters are as follows.

- Queue discipline: First-In First-Out queue discipline following the drop tail policy [21].
- TCP loss recovery algorithm: TCP classic recovery [22].
- DelAckCount (Number of packets to wait before sending a TCP Ack): 1.
- Explicit congestion notification functionality: not used.
- SACK: used.

In the evaluation, all data senders start data transfer at the same time and continue the communication until time 50 sec.

We used the ns-3 version 3.40, released on Sep. 27, 2023, and version 3.44 released Mar. 9, 2025.

B. Evaluation Results Using Ns-3 Version 3.40

This subsection describes the results using ns-3 version 3.40. In the configuration shown in Figure 1, we set the output buffer of N1, 100, 500, 1,000, 2,000, and 5,000 packets. As described above, the BDP in this configuration is around 10,000 packets, and so the buffer is not large enough, especially the value of 100 packets is extremely small.

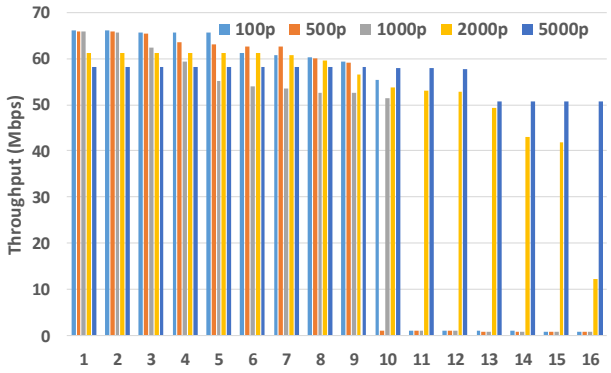


Figure 2. Throughput of individual BBR flows.

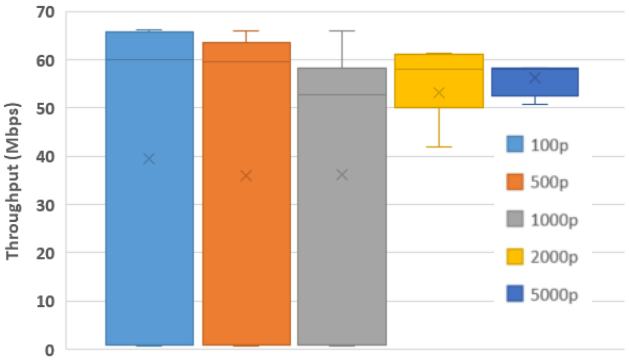


Figure 3. Box-and-whisker plots of BBR flow throughput.

TABLE II. AVERAGE, STANDARD DEVIATION, AND TOTAL OF THROUGHPUT OF BBR FLOWS (MBPS).

buffer size	average	std. dev.	total
100 pkts.	39.5	31.0	632.0
500 pkts.	35.9	32.0	574.8
1000 pkts.	36.1	28.5	577.9
2000 pkts.	53.1	12.7	850.3
5000 pkts.	56.3	3.3	900.2

Figure 2 shows the throughput of individual TCP BBR flows for five cases of the N1 output buffer. The bars are sorted in the descending order of throughput.

When the N1 output buffer is 100 packets, ten BBR flows provided high throughput more than 50 Mbps, and on the other hand, the other six flows provided very low throughput less than 1 Mbps. When the N1 output buffer is 500 and 1,000 packets, the results were similar. Nine or ten BBR flows out of the sixteen provided high throughput but the throughput of the rest was extremely low. But, for the cases that the N1 output buffer is 2,000 and 5,000 packets, the results were different, and almost all BBR flows provided high performance. Figure 3 shows the box-and-whisker plots where the whiskers show the minimum and maximum values, and the boxes give the first quartile, median and third quartile. The cross marks in the figure give the average. Table II shows the average, standard deviation, and total of sixteen flows' throughput. For the cases that the N1 output buffer is

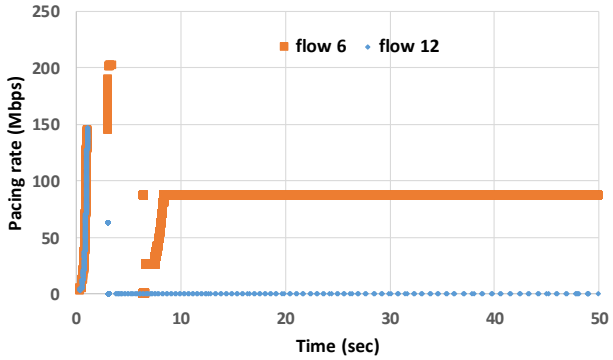


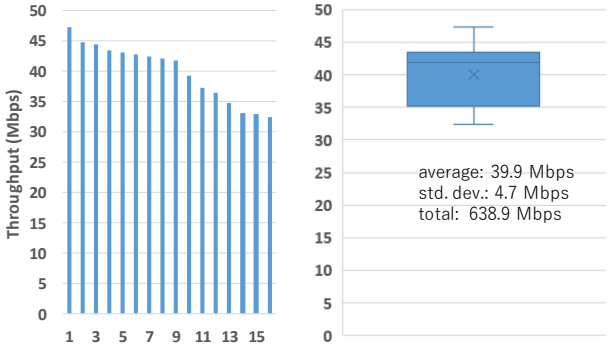
Figure 4. Pacing rate vs. time for flow #6 and #12 when the N1 output buffer is 100 packets.

100, 500, and 1,000 packets, the distribution of the throughput is large, and the total throughput is around 600 Mbps. Since the transmission rate of the bottleneck link is 1 Gbps, the efficiency is around 60 %. In contrast, in the cases that the N1 output buffer is 2,000 and 5,000 packets, the distribution becomes small and the efficiency is as high as 85 % or 90%.

Figure 4 shows the time variant of the pacing rate of 6th BBR flow and 12th BBR flow when the N1 output buffer is 100 packets. In the beginning, the pacing rate increases similarly and some packet losses and timeout retransmissions occur. After that, the pacing rate of 12th flow keeps a very low value, such as 0.1 and 0.01 Mbps. In contrast, that of 6th flow recovers to the value of 87 Mbps. This situation continues to the end of experiment. Other flows showed similar behaviors. This is the reason for the high and low throughput.

C. Evaluation Results Using Ns-3 Version 3.44

Next, we performed the same experiment using ns-3 version 3.44. Actually, we used the same script file as that used for ns-3 version 3.40. Figure 5 shows the results for the case that the N1 output buffer is 100 packets. Figure 5(a) gives the individual throughput of sixteen BBR flows and Figure 5(b) is its box-and-whisker plot. The distribution becomes small and all flows give high performance, between



(a) Individual throughput (b) Box-and-whisker plots

Figure 5. Results by ns-3 3.44 for 100 packet N1 output buffer.

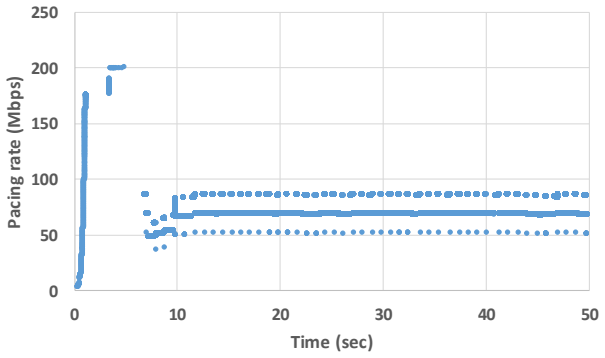


Figure 6. Pacing rate vs. time for flow #5 by ns-3 3.44 when the N1 output buffer is 100 packets.

32 Mbps and 47 Mbps. It should be noted that the total throughput is similar to that in ns-3 version 3.40. This means that the behaviors of flows is fair compared with the former experiment.

Figure 6 shows the time variance of the pacing rate of 5th BBR flow. The behavior in the first 10 seconds is similar to that of ns-3 version 3.40 depicted in Figure 4. But after that, the pacing rate changes among 50 Mbps, 70 Mbps, and 85 Mbps. For other flows, the behavior of pacing rate is similar, and as a result, the throughput of individual flows provided similar value in the experiment using ns-3 version 3.44.

Both results seem to be explainable. As mentioned above, it is possible that TCP BBR shows the intra-protocol unfairness when the bottleneck buffer is limited [10]. So, the results by using ns-3 version 3.40 are not necessarily wrong. The problem is that the behavior changed for ns-3 version 3.40 and 3.44. The next section discusses this issue in more detail.

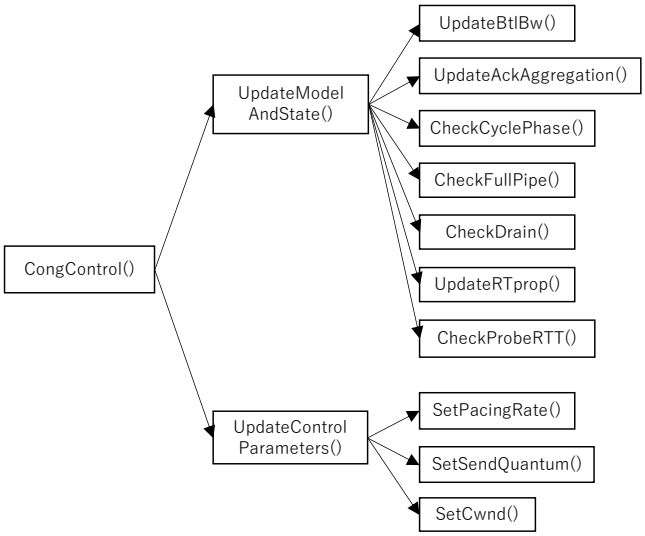


Figure 7. Function calls in TCP BBR Code in ns-3 version 3.40.

IV. Detailed Analysis of Ns-3 BBR Code

A. Program Strucsture of Ns-3 BBR Code

In ns-3, the main procedures of TCP are implemented in the tcp-socket-base.{h, cc} files. They give the functions for handling data send requests from the upper layer and for handling TCP segments received from the remote node. The control variables, such as `m_nextTxSequence` and `m_cWnd` are implemented in tcp-socket-state.h. The congestion control algorithms are implemented in other files independently. TCP BBR is implemented in the tcp-bbr.{h, cc} files.

Figure 7 depicts the function call graph of the TCP BBR congestion control (function `CongControl()`). It calls two main functions, `UpdateModelAndState()` and `UpdateControlParameters()`. The former calls the functions changing the states corresponding to BtlBw and RTprop. The latter calls the functions updating the related parameters, such as `m_pacingRate` and `m_cWnd`. The overviews of those functions are listed in Table III.

Figure 8 depicts the function call graph in tcp-socket-base.cc when an ACK segment is received. First, `ReceivedAck()` is called. As for the ACK processing, it calls `ProcessAck()`, and `CongControl()` in tcp-bbr.cc. `ProcessAck()` manages the ACK related parameters, and if the received ACK is a duplicate ACK, `DupAck()` is called and the retransmission is processed if necessary. After

TABLE III. Overview of BBR Functions in Fig. 7.

function	behavior
UpdateBtlBw()	Updates maxBwFilter if a new delivery rate is larger than prior value.
UpdateAck Aggregation()	Takes account of delayed ACK.
CheckCyclePhase()	If in ProbeBW phase and a RTT elapsed, advance cycle.
CheckFullPipe()	If delivery rate becomes large enough, set m_isPipeFilled = true
CheckDrain()	If in Startup phase and m_isPipeFilled is true, enter Drain phase. If in Drain phase and inflight data decreased, enter ProbeBW phase.
UpdateRTprop()	If RTT didn't increase or W_R (10 sec) elapsed (m_rtPropExpired = true), update m_rtProp.
CheckProbeRTT()	If not in ProbeRTT phase and m_rtPropExpierd, enter ProbeRTT phase and save cwnd. If in ProbeRTT phase and 200 msec elapsed, enter ProbeBW if m_isPipeFilled or Startup otherwise.
SetPacingRate()	Set rate = Max value in maxBwFilter × pacing_gain. If m_isPipeFilled or rate > m_pacingRate, set m_pacingRate = rate.
SetSendQuantum()	Set m_sendQuantum = MSS.
SetCwnd()	Determine cwnd considering targetCwnd and packet losses. If in ProbeRTT phase, set cwnd to 4 MSS.

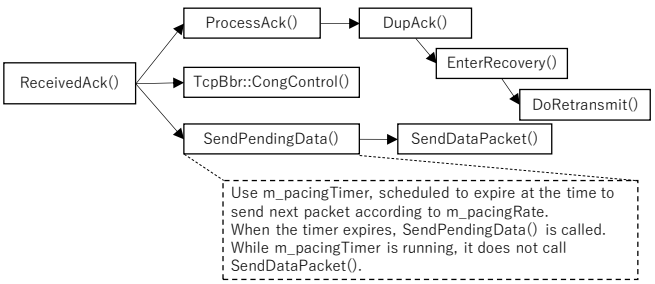


Figure 8. Function calls in TCP when receiving ACK in ns-3 version 3.40.

`CongControl()`, `SendPendingData()` is called to send the following data when the received ACK opens a window. In BBR, the rate-based data transfer is adopted and this is implemented by the TCP Pacing mechanism [23]. A timer called `m_pacingTimer` is used which expires according to a rate to send data segments according to `m_pacingRate`, calculated in `SetPacingRate()`.

These are described based on the program structure of ns-3 version 3.40. The structure itself is the same in version 3.44.

B. Analysis on Difference of Ns-3 3.40 and 3.44

We analyzed the BBR codes in version 3.40 and 3.44 to clarify why the results are different in these two versions. From the homepage of ns-3 [24], the changes related to TCP from version 3.40 to 3.44 are as follows.

1) TCP Cubic now supports TCP-friendliness by default, making the congestion window growth somewhat more aggressive. (from 3.40 to 3.41)
2) TcpCubic and TcpLinuxReno will no longer grow their cwnd when application-limited. (from 3.41 to 3.42)
3) Deprecated EventId::IsRunning(). It has been replaced with EventId::IsPending(). (from 3.41 to 3.42)
4) TCP Proportional Rate Reduction (PRR) recovery has been aligned to the updates in draft-ietf-tcpm-prr-rfc6937bis. (from 3.42 to 3.43)
5) A new trace source TcpSocketBase::LastRtt has been added for tracing the last RTT sample observed. The existing trace source TcpSocketBase::Rtt is still providing the smoothed RTT, although it had been incorrectly documented as providing the last RTT. (from 3.42 to 3.43)

Among them, 1), 2), and 4) do not give any impacts to our experiment, because we do not use TCP Cubic, NewReno, nor PRR recovery.

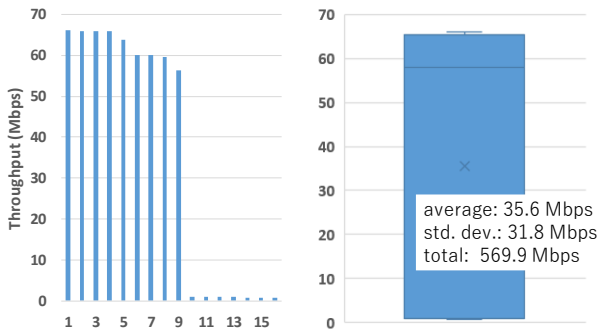
Then we compared the source codes of tcp-bbr.{h, cc} and tcp-socket-base.{h, cc}. The followings are differences that are considered to give some impacts to the BBR behaviors.

tcp-bbr.cc:

- `m_rtProp` in 3.40 is changed to `m_minRtt` in 3.44. This seems to be just a text-level modification.
- After rate is calculated in `SetPacingRate()` as shown in Table III, the following modification is added.
  `rate *= (1.f – m_pacingMargin),`
  `m_pacingMargin` is set to 0.01
  This needs to be considered.
- One `if` sentence is modified in `UpdateBltBw()`.
  3.40: `if (rs.m_deliveryRate == 0)`
  3.44: `if (rs.m_delivered < 0 || rs/<_interval.IsZero())`
  This needs to be considered.
- `m_rtProp` (`m_minRtt`) is set to `m_lastRtt` in 3.40, but to `m_srtt` in 3.44. This is related to item 5) in the change history. This may need to be considered.

tcp-socket-base.cc

- One else if sentence is modified in `DupAck()`:
  3.40: `else if (m_txBuffer->IsLost(m_high RxAckMark + m_tcb->m_segmentSize)`
  3.44: `else if (m_txBuffer->IsLost(m_high RxAckMark)`
  This needs to be considered.
- Related to item 5) in the change history, `Estimate Rtt()` is largely modified. This needs to be considered.



(a) Individual throughput (b) Box-and-whisker plots

Figure 9. Results of Approach 2 porting 3.44 BBR code to 3.40 for 100 packet N1 output buffer.

Based on these considerations, we took the following two approaches.

Approach 1: Modify tcp-bbr.cc and tcp-socket-base.cc for the points we decided to be considered.

Approach 2: Use tcp-bbr.cc and tcp-socket-base.cc of version 3.44 in ns-3 version 3.44.

Approach 2 required the following modification in the BBR source code,

tcp-socket-base.cc to be ported:

- Change `IsPending()` for `EventId` variables back to `IsRunning()`. This is related to item 3) in the change history.

tcp-socket-state.h in version 3.40:

- Define `TraceValue<Time> m_srtt` and `bool m_isCwndLimited`.

We performed both examination for the case that the bottleneck buffer is 100 packet. The results were similar and did not improve the performance of ns-3 version 3.40. Figure 9 shows the result of Approach 2. Out of sixteen BBR flows, nine provided high throughput, but the throughput of the other seven flows was extremely low. The average and total throughput was even worse than that of the original version 3.40. This result says that the difference of BBR behaviors in ns-3 version 3.40 and 3.44 may come from some part of other than TCP and BBR in the simulator. In the current stage, we do not clarify the details, but it can be said that the performance evaluation using ns-3 needs to be carefully done.

V. Conclusion and Future Work

In this paper, we presented some results of TCP BBR performance evaluation where sixteen BBR flows share a 1 Gbps bottleneck link whose output router has a limited output buffer. We used 100, 500, 1,000, 2,000, and 5000 packets as the output buffer size. For, small buffer size, 100/500/1,000 packets, some flows provided high throughput, but the performance of the others were extremely low. Although the BBR intra-protocol unfairness is mentioned in [10], the

degree of unfairness is much larger in this result. This experiment was done using ns-3 version 3.40. We also performed the same experiment using ns-3 version 3.44, which was the highest as of writing this paper. The result was completely different from that by version 3.40. All of the sixteen BBR flows provided high throughput, and the fairness was high.

We examined the BBR source codes of version 3.40 and 3.44. We analyzed the source codes in detail, and decided that there are not large differences between them. Next, we ported the version 3.44 BBR source code into ns-3 version 3.40, and conducted the same experiment using 100 packet output buffer. The result was similar with that in original version 3.40 case. This means that the difference between version 3.40 and 3.44 does not depend on the TCP related source code.

Although this paper could not point out the reason for the performance difference, we showed the details on TCP BBR behaviors over ns-3 network simulator. When using ns-3 for network performance evaluation, much care needs to be paid. We are planning to examine the ns-3 TCP BBR source code in more detail.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP25K15075.

REFERENCES

[1] T. Sawada, R. Yamamoto, S. Ohzahata, and T. Kato, “Congestion Control Algorithm Estimation by Deep Recurrent Neural Network and Its Application to Web Servers on Internet,” International Journal on Advances in Networks and Services, vol. 16, no. 1&2, pp. 27-35, 2023.

[2] S. Floyd, T. Henderson, A. Gurtov, and Y. Nishida, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” IETF RFC 6582, 2012.

[3] S. Floyd, “HghSpeed TCP for Large Congestion Windows,” IETF RFC 3649, 2003.

[4] S. Ha, L. Rhee, and L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64-74, 2008.

[5] D. Leith and R. Shorten, “H-TCP: TCP for high-speed and long distance networks,” Proc. Int. Workshop on PFLDnet, pp. 1-16, 2004.

[6] L. Brakmo and L. Perterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” IEEE J. Selected Areas in Commun., vol. 13, no. 8, pp. 1465-1480, 1995.

[7] C. Fu and S. Liew, “TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks,” IEEE J. Sel. Areas in Commun., vol. 21, no. 2, pp. 216-228, 2003.

[8] K. Tan, J. Song, Q. Zhang, and M. Sridharan, “A Compound TCP Approach for High-Speed and Long Distance Networks," In Proc. IEEE INFOCOM 2006, pp. 1-12, 2006.

[9] N. Cardwell, Y. Cheng, C. S. Gumm, S. H. Yeganeh, and V. Jacobson, V., “BBR: Congestion-Based Congestion Control,” ACM Queue vol. 14 no. 5, pp. 20-53, 2016.

[10] M. Hock, R. Bless, and M. Zitterbart, “Experimental Evaluation of BBR Congestion Control,” In Proc. IEEE 25th ICNP, pp. 1-10, 2017.

[11] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, “Cycle and Divergence of Performance on TCP BBR,” In Proc. IEEE 7th CloudNet, pp. 1-6, 2018.

[12] D. Scholz, et al., “Towards a Deeper Understanding of TCP BBR Congestion Control,” In Proc. 2018 IFIP Networking, pp. 1-9, 2018.

[13] S. Claypool, M. Claypool, J. Chung, and F. Li, “Sharing but not Caring – Performance of TCP BBR and TCP CUBIC at the Network Bottleneck,” In Proc. IARIA AICT 2019, pp. 74-81, 2019.

[14] H. Zhang, et al., “Performance Analysis of BBR Congestion Control Protocol Based on NS3,” In Proc. 7th Int. Conf. on Advanced Cloud and Big Data, pp. 363-368, 2019.

[15] G. Kim and Y. Cho, “Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement,” IEEE Access, vol. 8, pp. 4099-4109, 2020.

[16] K. Lanigan, “An Evaluation of BBRv2 Congestion Control Using NS-3,” Trinity College Dublin, 2020, availabe at https://publications.scss.tcd.ie/theses/diss/2020/TCD-SCSS-DISSERTATION-2020-046.pdf, accessed Jul. 2025.

[17] L. Tang, “BBR Fairness Evaluation Using NS-3,” arXiv:2410.22560v1 [cs.NI], 2024, available at https://arxiv.org/html/2410.22560v1, accessed Jul. 2025.

[18] “Mininet An Instant Virtual Network on your Laptop (or other PC),” https://mininet.org/, accessed Jul. 2025.

[19] “ns-3 Network Simulator,” https://www.nsnam.org/, accessed Jul. 2025.

[20] V. Jain, V. Mittal, ad M. P. Tahiliani, “Design and Implementation of TCP BBR in ns-3,” In Proc. Workshp on ns-3, pp. 16-22, 2018.

[21] “ns-3 document, 32.3. Fifo queue disc,” https://www.nsnam.org/docs/models/html/fifo.html, accessed Jul. 2025.

[22] “ns-3 document, 16.5.2.10. Loss Recovery Algorithm,” https://www.nsnam.org/docs/models/html/tcp.html#loss-recovery-algorithms, accessed Jul. 2025.

[23] A. Aggarwal, S. Savage, and T. Anderson, “Understanding the Performance of TCP Pacing,” In Proc. INFOCOM 2000, pp. 1157-1165, 2000.

[24] “ns-3: API and model change history,” https://gitlab.com/nsnam/ns-3-dev/blob/ns-3.44/CHANGES.md, accessed Jul. 2025.