Predicting Network Buffer Capacity for BBR Fairness

Ibrahim Umit Akgun, Santiago Vargas, Michael Arkhangelskiy, Andrew Burford, Michael McNeill, Aruna Balasubramanian, Anshul Gandhi, and Erez Zadok
Department of Computer Science
Stony Brook University

Abstract

BBR is a newer TCP congestion control algorithm with promising features, but it can often be unfair to existing loss-based congestion-control algorithms. This is because BBR's sending rate is dictated by static parameters that do not adapt well to dynamic and diverse network conditions. In this work, we introduce BBR-ML, an in-kernel ML-based tuning system for BBR, designed to improve fairness when in competition with loss-based congestion control. To build BBR-ML, we discretized the network condition search space and trained a model on 2,500 different network conditions. We then modified BBR to run an in-kernel model to predict network buffer sizes, and then use this prediction for optimal parameter settings. Our preliminary evaluation results show that BBR-ML can improve fairness when in competition with Cubic by up to 30% in some cases.

1 Introduction

Optimizing I/O subsystems can not only improve the overall performance and stability of the operating system but can also help reduce latency and improve QoS. For cloud providers, it is crucial to maintain low performance variability [12, 19]. One of the dominant contributors to operating systems' performance variability is I/O subsystems [5, 7, 8] (*e.g.*, networking, storage). For example, improving fairness for competing network flows can help cloud operators meet SLO requirements.

BBR [9] (Bottleneck Buffer and Round-trip propagation time) is a new TCP congestion control; released in 2016, BBR has been widely used on the Internet [17]. BBR is a rate-based algorithm that aims to maintain low network-buffer utilization while maximizing throughput. BBR does this by estimating the *Bandwidth-Delay Product* (BDP) of the network's bottleneck link, and then setting the TCP congestion window and pacing rate proportional to its BDP estimate. BBR ignores packet losses, unlike loss-based congestion control algorithms (*e.g.*, Linux's default Cubic [13]), which attempt to maximize sending rates until losses occur—when network buffers are already congested. Thus, BBR is designed to achieve good performance without causing *bufferbloat*, a condition where network buffers are saturated, leading to severe packet loses.

However, BBR is known for its unfairness when used with other loss-based TCP congestion control algorithms [23, 20, 5]. Studies have shown that BBR can take disproportionate amounts of router buffer and hence bottlenecks bandwidth, because it does not react to network losses [20, 5]. In Figure 1, one can observe that BBR is unfair to Cubic under smaller bottleneck router buffer sizes (in the range of 0.01 - 0.07 BDP). On the other hand, when we increase the bottleneck router buffer size, Cubic becomes unfair to BBR. In the optimal case, BBR and Cubic should share the network equally (*e.g.*, 50%). TCP fairness is a major concern since it is important to understand how BBR will interact with other congestion-control algorithms especially as BBR continues to grow in popularity [18]. In fact, recent studies [25, 21] and newer versions of BBR (*i.e.*, BBRv2 [10]) have attempted to correct BBR's unfairness by using loss or delay signals. However, this departure fundamentally changes BBR's behavior, and yet BBRv2 *still* exhibits unfairness [22, 20].

Instead, we approach BBR's unfairness as a tuning and optimization problem. Tuning system parameters has been a well studied research area [8, 6, 15, 1]; because of complex system dynamics and exponential search spaces, it is considered as hard problem suitable for meta-heuritics and

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on ML for Systems.

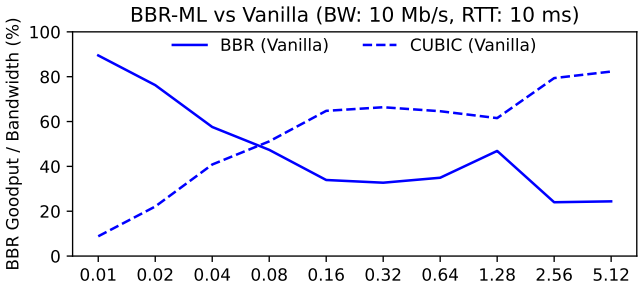


Figure 1: Bandwidth sharing percentages when Cubic and BBR flows run concurrently under different bottleneck router buffer size configurations in BDP.

machine learning. Researchers also have tried to deal with BBR's unfairness with heuristics and new algorithms [25, 21, 9]. Unfortunately, BBR's unfairness is a complex optimization problem which requires techniques that can identify intricate dynamics.

First, BBR estimates BDP and maintains a maximum of $2BDP$ packets in-flight during steady-state. $2BDP$ is computed from `CWND_GAIN`, a parameter used to scale the number of allowed in-flight packets. The problem is that the `CWND_GAIN` parameter is static and does not adapt to changing network conditions. Under small network buffers, $2BDP$ worth of data overwhelms the bottleneck buffer, BBR unfairness as BBR consumes a significant share of the bandwidth. Conversely, under large buffers, $2BDP$ worth of data is not enough to compete with loss-based congestion control algorithms which fill up these deeper buffers and take the majority of bandwidth share. In fact, Figure 3a confirms the above by showing BBR dominate Cubic in smaller buffers and Cubic overwhelm BBR in larger buffers.

By selectively tuning BBR's static parameters, such as `CWND_GAIN`, we will keep BBR's fundamental model intact while adapting BBR flows to network conditions. A natural approach for solving this type of tuning and optimization problem is using Machine Learning (ML). There has been a growing interest among network researchers in solving congestion control problems by using ML [1, 15, 24]. However, researchers face three significant challenges when employing ML in networks: (i) lack of ample search space for complex network problems, (ii) high computational overheads due to high-frequency data collection/movement (user-kernel space) and inference, and (iii) unstable ML models that behave erratically.

In this paper, we present *BBR-ML*, a lightweight ML model to improve BBR fairness by tuning BBR parameters. We address the aforementioned three challenges with the following contributions: (i) To enable high-frequency inference and data collection, we build BBR-ML using the KML [2, 3] framework, an ML and data-collection framework integrated directly into the Linux kernel. (ii) To ensure we have a large enough and suitable training data set, we trained BBR-ML with 12 million kernel data points (25GB of data) corresponding to experiments for 2,500 unique network conditions. (iii) Lastly, we maintain model stability by designing BBR-ML to retain the core BBR model while just tuning BBR's `CWND_GAIN` parameter. Overall we found that BBR-ML improves fairness with Cubic up to 30% in terms of Jain's fairness index [14] when BBR is especially unfair. We describe how we address the three challenges in BBR-ML's design in Section 2 and discuss preliminary evaluation results in Section 3.

2 Design

In this section, we detail our approach to the BBR-fairness problem using in-kernel ML models. We describe our design according to the challenges mentioned above: large search space, high overheads in using ML, and stability of ML.

2.1 Large Search Space Problem

In general, there are numerous combinations of network conditions that can affect TCP performance. However, we approach BBR tuning primarily in terms of the bottleneck buffer size. We use the term *bottleneck buffer* to refer to the buffer size of the network router that bottlenecks throughput. Our reasoning is based on the fact that *buffer size* for bottleneck router has a crucial impact on BBR's

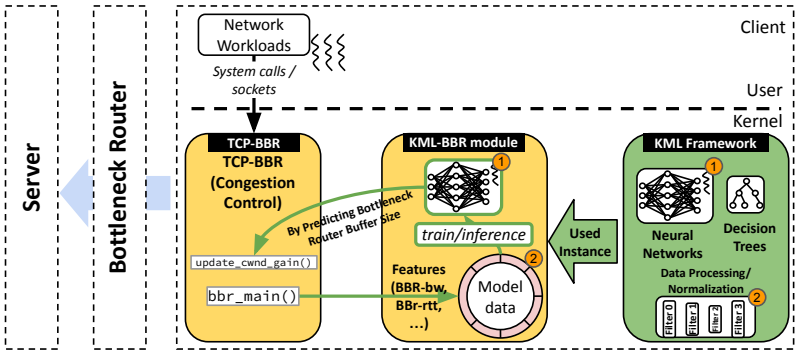


Figure 2: KML BBR Architecture

fairness. This is in line with previous work that shows that BBR's share of bandwidth (when in competition with Cubic) largely changes based on buffer size [5], and this result is also confirmed in our experiments (see Figure 3a).

The main problem, however, is that while the buffer size largely dictates BBR's fairness, the client does *not* inherently know the buffer size of the bottleneck router. Such knowledge is invaluable in helping control BBR's sending behavior. Further analysis of experiment results showed that BBR's behavior does not change for similar buffer sizes.

Our experiments show that BBR's fairness behavior is similar across similar BDP ranges. Instead of building a regression model, we designed a multi-class classification model to predict *buffer size* in BDP ranges. We reduce the search and optimization space by discretizing the classes of buffer sizes in terms of connection BDP.

2.2 Computational & Data Collection Overheads

Why KML? KML [3, 2] is a low-overhead, efficient ML framework that is designed to build ML models for optimizing and tuning operating system (OS) components. OSs heavily rely on heuristics/static-configurations to optimize and adapt OS components for ever-changing workloads and hardware. However, neither static configurations nor heuristics can adapt in the face of these problems, which have a large search space and require learning complex patterns. That is why we choose KML to tune BBR's configurations (Figure 2 ①) to improve BBR's fairness to other congestion control algorithms (*e.g.*, Cubic).

Data collection and feature extraction. Data collection is an integral part of any ML solution. However, when dealing with OS components, it is critical to maintain low data collection overhead to avoid noise and variance in data. To that end, we used Re-Animator [4], a versatile, low-overhead data collection framework based on Linux kernel tracepoints [11] (Figure 2 ②). We also implemented our own Linux kernel tracepoint functions for tracing BBR at a fine-grained level. We extracted five features: (i) the time difference between two consecutive transactions, in μs; (ii) RTT (roundtrip time), which is observed from BBR; (iii) number of packets in flight; (iv) BBR's estimated bandwidth; and (v) RTT from `rate_sample`, which is populated by TCP. We have applied common normalization techniques on top of these features:

$$\begin{aligned}
\text{Let } \lambda &= \text{ RTT} \\
\text{Features } &= [\Delta_{Time}, \lambda_{BBR}, \text{Packets In Flight}, \text{BBR Bandwidth}, \lambda_{TCP}] \\
\text{Bucketization } &= \left[\frac{\Delta_{Time}}{1000}, \frac{\lambda_{BBR}}{25000}, \frac{\text{Packets In Flight}}{100}, \frac{\text{BBR Bandwidth}}{20}, \frac{\lambda_{TCP}}{200000}\right] \\
Z &= \frac{x-\mu}{\sigma} \quad \text{We next apply the Z score to each element in this vector.}
\end{aligned}$$

ML model design. We modeled the BBR-fairness problem as a multi-class classification problem and developed a neural network with 5 linear layers (with a hidden layer size of 256 for all of them). We connected the linear layers with `tanh`, `sigmoid` for the last layer, and `cross-entropy` as the loss function. We applied Tune [16] with KML and tuned hyper-parameters. We have applied 10-fold validation: BBR-ML's average prediction accuracy reached 70%. We ensured that class frequencies in the training data were close (in the range of 5%).

2.3 Implementation

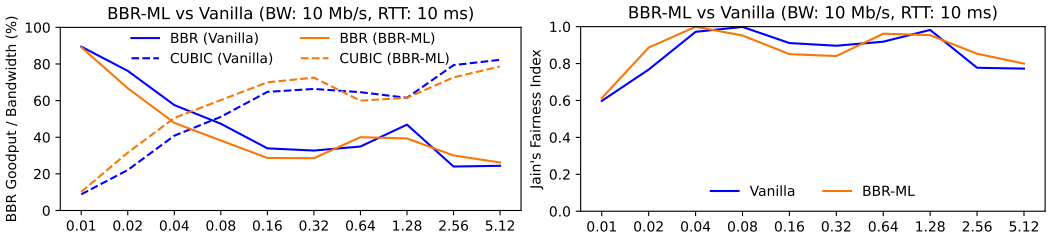
BBR-ML pipeline. Predicting router buffer size is the first step in addressing the BBR fairness problem. Based on the predicted router buffer size, we must tune BBR to improve fairness. To this end, we first investigated how BBR parameters affect BBR's fairness under various bottleneck router buffer sizes to choose the most influential parameter to tune. Our empirical study showed that `CWND_GAIN` is the dominant factor that impacts BBR's fairness; therefore, we choose to tune this parameter. `CWND_GAIN` is important since it is used to scale the number of in-flight packets as follows:

$$\texttt{In-Flight Packets} = \texttt{CWND\_GAIN} \times \texttt{BDP\_Estimate}$$

Our BBR-ML inference pipeline works as follows: (i) KML-BBR's kernel module collects input data from `bbr_main`, which is part of TCP-BBR congestion control kernel module. (ii) Collected input data is accumulated in efficient lock-free circular buffers (part of KML framework) and then pre-processed and normalized. (iii) KML-BBR's kernel module then runs inference on input data and gets the predicted BDP class for bottleneck router buffer size. (iv) Finally, we use the BDP-class-to-optimal-`CWND_GAIN` mapping, which is populated via our empirical study, to return updated `CWND_GAIN` to TCP-BBR kernel module. At the end of this BBR-ML pipeline, based on the updated `CWND_GAIN` network conditions and BBR's behavior changes, gets the next cycle's input data. Thus BBR-ML's life-cycle creates a *close-circuit* flow.

3 Evaluation

To evaluate BBR-ML, we fixed all other network conditions (*i.e.*, bandwidth, RTT) except for bottleneck router buffer size and used `iperf` to run a BBR flow and a Cubic flow concurrently. We have evaluated BBR-ML from three evaluation perspectives: (i) from the networking perspective, we discussed fairness improvements that BBR-ML can achieve and how these two flows share a network running under different buffer sizes; (ii) from ML perspective, we measured BBR-ML's run-time prediction accuracy and examined how we can improve the BBR-ML model; and (iii) from ML performance and system overheads perspective, we investigated how much performance overheads BBR-ML causes, and how we can reduce them even more.



(a) BBR is unfair to Cubic with smaller buffers, whereas Cubic overwhelms BBR with larger buffers. BBR-ML helps to reduce this unfairness.

(b) Comparison of vanilla BBR and BBR-ML using Jain's Fairness Index. BBR-ML especially helps improve fairness under smaller buffers.

Figure 3: Comparisons between BBR-ML and Vanilla experiments.

Networking perspective. Figure 3a demonstrates how BBR-ML affects the percentage of network sharing for both BBR and Cubic flows. We can observe how BBR-ML helps reduce the network-sharing percentage gap between BBR and Cubic flows, especially in small and large buffer sizes. BBR-ML reduces the `CWND_GAIN` multiplier in shallow buffers just enough to make BBR more fair to Cubic flows. Conversely, in large buffer sizes, BBR-ML increases `CWND_GAIN` just enough to let BBR flows consume more bandwidth. BBR-ML takes these actions dynamically based on BDP predictions. Figure 3b shows the evaluation of how BBR-ML improves the overall fairness in terms of Jain's Fairness Index [14]. BBR-ML can improve the overall fairness by as much as 0.15 points. Jain's fairness index is bounded by $[\frac{1}{n}, 1]$ where n equals to a number of concurrent flows in the systems. In our use case, there are two concurrent flows (BBR and Cubic). As a result, Jain's fairness index range is between 0.5 to 1.0 for our setup. That is why we can translate 0.15 increase in Jain's fairness index to a 30% improvement in our use case.

Router Buffer	0.01	0.02	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12
Prediction Accuracy	68%	88%	93%	46%	54%	85%	100%	100%	100%	100%

Table 1: KML Prediction Accuracy by Class

ML perspective. Table 1 shows the BBR-ML prediction accuracies for each BDP class. We tested running BBR-ML in-kernel inference mode for all these tests. We see that there is still room for improvement for the BBR-ML model (part of our future work).

ML performance and overheads perspective. We measured that BBR-ML inference takes 3ms on average. Since we are running inference asynchronously on a separate kernel thread once every second to predict BDP class, this inference overhead of BBR-ML is considerably low. Data collection and normalization overheads for BBR-ML are less than 1% in total. Overall, these overheads make BBR-ML a realistic approach for running in production systems.

4 Conclusion & Future Work

We believe that unfairness between congestion control algorithms can be one of the most suitable candidate for ML applications in networking and systems. We implemented a prototype system called BBR-ML that can predict bottleneck buffer sizes in terms of BDP ranges with 70% accuracy and improve fairness by up to 30%. We think that static configurations and heuristics in BBR can be enhanced with ML-based approaches.

Future work. BBR-ML is supporting predictions only for fixed BDP ranges. As part of future work, we plan to improve our neural network to classify more fine-grained buffer size ranges and integrate deep reinforcement learning (RL) techniques to make BBR-ML adapt to new network conditions. We are also investigating other static configurations and their effects on fairness (*e.g.*, `PACING_GAIN`).

5 Acknowledgments

This work was made possible in part thanks to Dell-EMC, NetApp, Facebook, and IBM support; a SUNY/IBM Alliance award; and NSF awards CNS-1729939, CNS-1900706, CCF-1918225, CNS-1951880, CNS-2106263, CNS-2106434, CNS-1909356, and CNS-2214980.

References

[1] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan 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*, pages 632–647, 2020.

[2] Ibrahim Umit Akgun, Ali Selman Aydin, Aadil Shaikh, Lukas Velikov, Andrew Burford, Michael McNeill, Michael Arkhangelskiy, and Erez Zadok. Kml: Using machine learning to improve storage systems. *arXiv preprint arXiv:2111.11554*, 2021.

[3] Ibrahim 'Umit' Akgun, Ali Selman Aydin, Aadil Shaikh, Lukas Velikov, and Erez Zadok. A machine learning framework to improve storage system performance. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage (HotStorage '21)*, Virtual, July 2021. ACM.

[4] Ibrahim Umit Akgun, Geoff Kuenning, and Erez Zadok. Re-animator: Versatile high-fidelity storage-system tracing and replaying. In *Proceedings of the 13th ACM International Systems and Storage Conference (SYSTOR '20)*, Haifa, Israel, June 2020. ACM.

[5] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. When to use and when not to use bbr: An empirical analysis and evaluation study. In *Proceedings of the Internet Measurement Conference*, page 130–136, 2019.

[6] Zhen Cao, Geoff Kuenning, and Erez Zadok. Carver: Finding important parameters for storage system tuning. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)*, Santa Clara, CA, February 2020. USENIX Association.

[7] Zhen Cao, Vasily Tarasov, Hari Raman, Dean Hildebrand, and Erez Zadok. On the performance variation in modern storage stacks. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*, pages 329–343, Santa Clara, CA, February-March 2017. USENIX Association.

[8] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *Proceedings of the Annual USENIX Technical Conference*, Boston, MA, July 2018. USENIX Association. Data set at *http://download.filesystems.org/auto-tune/ATC-2018-auto-tune-data.sql.gz*.

[9] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[10] Neal Cardwell, Yuchung Cheng, S Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. Bbrv2: A model-based congestion control. In *Presentation in ICCRG at IETF 104th meeting*, 2019.

[11] Mathieu Desnoyers. Using the Linux kernel tracepoints, 2016. *https://www.kernel.org/doc/Documentation/trace/tracepoints.txt*.

[12] A. Gandhi, P. Dube, A. Karve, A. P. Kochut, and L. Zhang. Providing Performance Guarantees for Cloud-deployed Applications. *IEEE Transactions on Cloud Computing*, 2018.

[13] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[14] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 21, 1984.

[15] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[16] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[17] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The great internet tcp congestion control census. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), 2019.

[18] Ayush Mishra, Wee Han Tiu, and Ben Leong. Are we heading towards a bbr-dominant internet? In *Proceedings of the Internet Measurement Conference*, 2022.

[19] Dejan Novakovic, Nedeljko Vasic, Stanko Novakovic, Dejan Kostic, and Ricardo Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of 2013 USENIX Annual Technical Conference*, ATC '13, pages 219–230, San Jose, CA, USA, 2013.

[20] Simon Scherrer, Markus Legner, Adrian Perrig, and Stefan Schmid. Model-based insights on the performance, fairness, and stability of bbr. In *Proceedings of the Internet Measurement Conference*, 2022.

[21] Yeong-Jun Song, Geon-Hwan Kim, and You-Ze Cho. Bbr-cws: Improving the inter-protocol fairness of bbr. *Electronics*, 9(5), 2020.

[22] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm. *IEEE Access*, 9:37131–37145, 2021.

[23] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of the Internet Measurement Conference*, page 137–143, 2019.

[24] Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen. Liteflow: towards high-performance adaptive neural networks for kernel datapath. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 414–427, 2022.

[25] Yuxiang Zhang, Lin Cui, and Fung Po Tso. Modest bbr: Enabling better fairness for bbr congestion control. In *IEEE Symposium on Computers and Communications*, pages 646–651, 2018.