Current State of BBR Congestion Control

Miyu Kitamura, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: ge23cag@mytum.de, jaeger@net.in.tum.de

Abstract—**Widely deployed, loss-based, congestion control and avoidance mechanisms, such as CUBIC and RENO, have served well for many years. Until recently, packet loss may have been a good indicator for congestion, but with increasing wireless networks in public venues, as well as ever-growing buffer sizes leading to bigger delays, packet loss has become an unreliable indicator of congestion. In this paper, we present the Bottleneck-Bandwidth and Round-Trip (BBR) algorithm and its evolution from version 1 to version 3. Originally developed by Google in 2017 [1], BBR uses a model-based approach to regulate congestion, making it better suited to modern network conditions.**

Index Terms—**BBR, Network Path Modeling, Congestion Control, TCP**

1. Introduction

When Jacobson et al. set out to analyze the reason for the "first of [...] a series of congestion collapses" [2] they developed a new way to prevent such an event from happening again. It was then that packet loss was decided to be used as the main indicator for congestion in a network [1]. They proposed to use a mechanism called the congestion window (**cwnd**) in order to start sending packets at a slow rate, increasing it on every successful transmission indicated by a received acknowledgement (**ACK**). This algorithm is now known as TCP Tahoe [3]. Many other congestion control algorithms, such as TCP Reno and TCP CUBIC, the latter of which has become the default algorithm in Linux as stated by Sangtae Ha et al [4], were developed over the years. Lately however, the rise of wireless network in public venues and ever-growing buffer sizes have led to packet loss not being a reliable indicator of congestion anymore [1]. This is where the BBR algorithm developed by Google comes into play. The main contribution of this paper is to give an overview of the current state of BBR congestion control and compare the different versions that have been developed since. In the following sections, we first provide an overview of conventional congestion control methods and discuss the limitations of relying solely on packet loss. BBR's fundamental concepts and goals are then introduced. We explore how each successive version refines the algorithm's design, model parameters, and fairness. Finally we discuss the current challenges, achievements and future directions for BBR.

2. TCP Congestion Control

Traditional congestion control algorithms, dating back to TCP Tahoe and Reno, rely on packet loss as a primary indicator of congestion and use the cwnd as primary mechanism to regulate the sending rate. This cwnd describes the maximum not yet acknowledged amount of data in bytes that can be in flight at any given time as described by Rasool Al-Saadi et al [5]. The sender continually increases its cwnd until loss is detected, then reduces it aggressively. The window is then steadily increased again, starting the process anew. Loss itself is detected via duplicate ACKs triggered by out-of-order arrivals. Instead of reacting to the first duplicate ACK the algorithm waits for three or more duplicates (a process called *fast retransmit*) [5] to confirm that out-of-order arrival is a result of congestion. By how much the cwnd is reduced and in what manner it is restored thereafter differs between algorithms. Loss-based algorithms therefore require a low loss environment to ever efficiently utilize the connection for a prolonged time. Even if the cwnd ever reaches its maximum, loss-based algorithms will continue growing it, leading to an unavoidable loss event and cwnd reduction. Gomez et al. state that "using Reno in a 10 Gbps link with a 100 ms propagation delay needs more than an hour to fully utilize the bandwidth. Avoiding this issue demands a loss rate lower than 0.00000002%" [6]. One important aspect to notice here is that by detecting loss on duplicate acknowledgments it takes at least one round-trip time (**RTT**) to detect the congestion and make adjustments which is why longer buffers make it difficult to adapt quickly due to the delay they inherently introduce by filling the network bottleneck's buffer.

2.1. CUBIC

One of the most widely used algorithms is TCP CUBIC, which was developed by Sangtae Ha, Injong Rhee, and Lisong Xu in 2008 [4]. Instead of growing the cwnd linearly after loss like TCP Reno does, Sangtae Ha et al. propose to instead use a cubic function to restore the cwnd, making the recovery of the cwnd much faster. The cwnd growth after loss can be split into three phases: slowing growth that grows the cwnd very quickly at first, plateau and increasing growth. Figure 1 displays the various phases that CUBIC goes through. Additionally, the cwnd is only reset by 30% after a loss as compared to Reno which reduces it by 50%. This process makes CUBIC and similar loss-based algorithms such as TCP Reno predictable in terms of their sending behaviour.

However, CUBIC still does not solve the inherent problem of loss-based algorithms. These algorithms remain stuck in a loop: they probe for the maximum speed, then reduce it again after a loss. This often shows as a sawtooth-like pattern, although in CUBIC it is less pronounced thanks to the cubic function. Since packet loss in itself is only a binary indicator of congestion, it does not provide any information about how strongly the network is congested, which means that the algorithms have only the way of reducing the cwnd by a fixed factor.

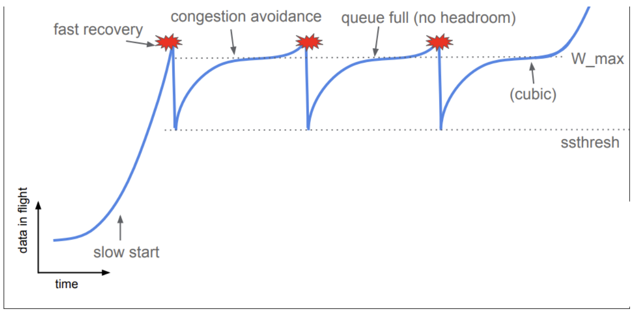

Figure 1: Function of data in flight over time in CUBIC [7].

The question why TCP CUBIC—or, more broadly, loss-based congestion control approaches—may no longer be sufficient, and what led to the development of BBR, arises. Loss-based algorithms and their way of growing the cwnd makes them inherently suffer from the following problems:

- With wireless networks becoming more and more common, packet loss can also be caused by interference or other factors unrelated to congestion.
- Network nodes with deep buffers can store a lot of data before they start dropping packet. Long queueing of packet leading to high RTT make it hard to adapt to congestion swiftly and can lead to a lot of data being sent at an excessive rate.
- Network nodes with shallow buffers might simply be overwhelmed by a short-lived burst of data and drop packets despite there being no congestion. Furthermore even just a slight overshoot of the maximum bandwidth can lead to packet loss and an adjustment of the cwnd by much more than would actually be required.

To solve the stated problems a new approach is needed where the BBR algorithm comes into play.

3. BBR

BBR was originally developed in 2017 and has since evolved over three versions. This section gives an overview of what BBR aims to solve as well as its implementation details and the differences between the versions.

3.0.1. Kleinrock's Optimal Operating Point. BBR tries to operate at a point that is close to the optimal operating point as described by Kleinrock [8]. Figure 2 shows the optimal operating point as a function of the bottleneck bandwidth and the RTT.

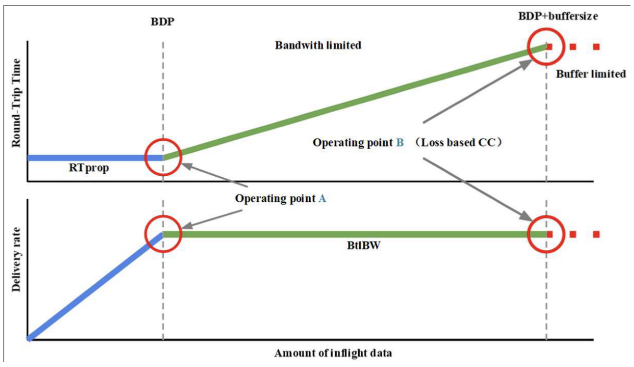

Figure 2: Impact of inflight data volume on RTT and delivery rate [9].

The two graphs display how the RTT and the delivery rate change in correlation to the amount of data in flight. It can be seen that, as the amount of data in flight increases, there is a limit to the increase in delivery rate. Furthermore, the round-trip time continues to rise even long after the delivery rate stopped increasing. This can be explained by the fact that, while the data volume in flight is below a certain threshold, there are no queues yet because the network is under-utilized and delivery-rate can still be increased by sending more packets. As the threshold is exceeded, however, the data in flight will exceed the bottleneck's capacity, which causes it to be queued in the nodes buffer resulting in a RTT increase. Since the bottleneck is already sending as fast as it can, the delivery rate will not increase anymore as newly arriving packets are either queued or dropped entirely. While loss-based algorithms tend to operate at point B as shown in Figure 2, BBR tries to operate at the point where the delivery rate is maximized and the RTT is minimized. This is where the bottleneck is fully utilized excessive forming of queues, Kleinrock's optimal operating point [1].

3.1. Goals

BBR departs from the loss-based approach and instead keeps an updated model of the network path to smoothly and continuously adapt its sending rate to the current network conditions. By doing so, it aims to be proactive in its approach rather than reactive, as loss-based algorithms are. Keeping continuous track of the network path solves the issue of only having packet loss as binary indicator of congestion. The goals of BBR can be summarized as follows:

- **Maximizing Throughput at Minimal Delay:** BBR aims to fully utilize the bottleneck link by sending at a rate close to the measured bottleneck bandwidth, while keeping the in-flight data near the bandwidth-delay product (**BDP**). It avoids persistent queue buildup (and the resultant bufferbloat) by doing so and thus maintains low latency for network traffic [1].
- **Proactive, Model-Based Control:** Instead of waiting for packet loss, BBR continuously monitors both bottleneck bandwidth and minimum RTT to build a near real-time model of the path. It adjusts its sending rate according to that model so

that it operates near Kleinrock's optimal operating point.

- **Improved Robustness to random loss:** By decoupling congestion control from packet loss, BBR is inherently more tolerant of random losses and does not over-correct when just a slight congestion, which could be solved by just a small reduction of the sending rate, occurs.

3.2. BBR State Machine Phases

BBR operates by creating a network path model using the RTT and the bandwidth with which it can calculate the BDP.

$$\text{BDP} = \text{Max. Bandwidth} \cdot \text{Min. RTT}$$

In order to find the minimal RTT one has to send at a rate that avoids the formation of queues so that no delay is introduced by the network. On the other hand, in order to find the maximum bandwidth one has to send at a rate that makes queues form as those are an indicator of congestion and therefore a nodes limit. These two measurements are crucial for BBR to operate at Kleinrock's optimal operating point and can only be measured individually as they are exclusive to each other. BBR does this by utilizing an internal state machine, see Figure 3, that switches between probing for exactly these two measurements [7]. During each of these phases BBR maintains a set of parameters that influence the sending rate [10]. Some of the most important parameters are:

- **pacing_rate**: Main parameter that controls the spacing between packets. It is updated on each received ACK and aims to match the bottlenecks rate instead of sending packets in bursts. In a perfect scenario, the pacing rate would be equal to the maximum bandwidth, but the distinction between pacing rate and sending rate is deliberate. The sending rate, i.e. actual throughput, can however fall below the pacing rate, for example if the sender is application-limited or the cwnd limit has been reached.
- **pacing_gain**: Multiplier for the pacing rate. Used to increase or decrease the rate for probing in various phases.
- **cwnd**: Limits the maximum inflight data volume. Bound by inflight_hi and inflight_lo.
- **cwnd_gain**: Multiplier for the cwnd. Used to increase or decrease the cwnd for probing in various phases.
- **inflight_hi**: New since BBRv2 inflight_hi is a long-term upper bound on inflight data based on past loss events. The congestion window is limited to this value.
- **inflight_lo**: New since BBRv2 inflight_lo is a short-term upper bound on inflight data in the probing current cycle.

3.2.1. Startup Phase. When establishing a new connection, BBR will start in the startup phase. This phase aims to estimate the bottleneck bandwidth very quickly by setting the pacing_gain and cwnd_gain to high values and therefore allowing the cwnd and pacing_rate to effectively

Life cycle phases	**Property**	**BBRv1**	**BBRv2**	**BBRv3**
Startup	cwnd_gain	$2/\ln 2$ (~ 2.89)	$2/\ln 2$ (~ 2.89)	2.00
	pacing_gain	$2/\ln 2$ (~ 2.89)	$2/\ln 2$ (~ 2.89)	2.77
	Max. cwnd	3xBDP	-	-
	inflight_hi	-	max.cwnd	max(est. BDP, last cwnd)
	Exit send.	rate <25% for 3 consec. RTTs	loss/ECN rate $\geq$ thresh. (8)	loss/ECN rate $\geq$ thresh. (6)
Drain	pacing_gain		0.35	
	Exit		cwnd $\leq$ 1xBDP	
ProbeBW	Phases	8 fixed gain cycles	{Cruise, Refill, Up, Down}	{Cruise, Refill, Up, Down}
	*	Cycle = RTprop	cwnd limits = {inflight_hi, inflight_lo}	-
	cwnd_gain	-	cwnd_gain(Up)=2.0	cwnd_gain(Up)=2.25
	pacing_gain	[1.25, 0.75, 1, 1, 1, 1, 1, 1]	pacing_gain(Down)=0.75	pacing_gain(Down)=0.90
	Exit	cwnd $\geq$ (pacing_gain $\cdot$ BDP) or loss	loss/ECN rate $\geq$ thresh.	loss/ECN rate $\geq$ thresh.
ProbeRTT	Frequency	10s	5s	5s
	cwnd	4	BDP/2	
	Duration	200 ms + RTprop	-	-

TABLE 1: Evolution of BBR parameters over various versions [11].

double every round. Generally the maximum bandwidth is found in around $\mathcal{O}(\log_2(\text{BDP}))$ RTTs in version 3 [10].

This phase is exited as soon as packet loss has reached a certain threshold or the maximum bandwidth has been found as indicated by a plateau. In context of BBR a plateau is reached when the delivery rate has increased by less than 25% over the last three RTTs.

3.2.2. Drain Phase. During the startup, phase a queue of around 1 BDP [10] has been built up at the bottleneck. In order to remove this queue, the drain phase reduces the pacing gain which in turn reduces the sending rate. While any value below 0.5 should be able to drain the queue within ≤ 1 RTTs, BBRv3 uses a value of 0.35 [10]. Once the volume of data in-flight has been reduced to <= 1 BDP the drain phase is exited. If inflight_hi was set during startup, indicating that there was packet loss, the drain phase will empty the queue even further to provide some headroom for better coexistence with other flows.

3.2.3. Probe Bandwidth Phase. The probe bandwidth phase is a long-lived phase in which BBR operates most of the time. While originally this phase was only called ProbeBW it has been split into four sub-phases in BBRv2 between which the algorithm cycles: ProbeBW_DOWN, ProbeBW_CRUISE, ProbeBW_REFILL and ProbeBW_UP [12].

- **ProbeBW_DOWN**: Aims to drain the volume in flight to below 1 BDP with potentially additional headroom that can be used by other flows.
- **ProbeBW_CRUISE**: Sending rate is adjusted to the maximal available bandwidth with some margin **BBRPacingMarginPercent** [10]. If packet loss occurs bw_lo and inflight_lo are adjusted to reduce the sending rate as necessary. The phase is exited after a certain amount of packets depending on the environment to be more fair towards coexisting Reno and CUBIC flows [10].
- **ProbeBW_REFILL**: Aims to "refill the pipe" [10] to prepare for the next phase. Necessary to not underestimate the path by causing packet loss with a sudden burst of data.
- **ProbeBW_UP**: Probes for changes in maximum bandwidth by sending with an increased rate. It is exited after a delivery rate increase plateau has been found or packet loss exceeds a threshold.

3.2.4. Probe RTT Phase. ProbeRTT is entered after at most 5 seconds have passed since the last ProbeRTT phase and can be switched into from any other phase. The goal of this phase is to measure the minimal RTT. Because existing queues would introduce a delay and therefore

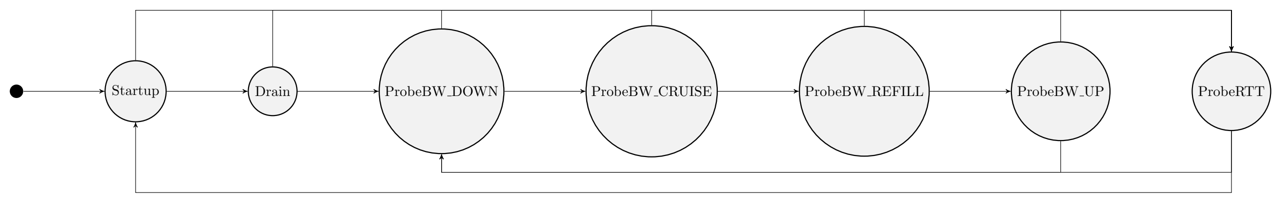

Figure 3: State transitition diagram for BBRv3 [10].

skew the measured minimal RTT they must be drained by sending a volume of data that is lower than the BDP. This is done by halving the cwnd_gain to lower inflight data volume before taking measurements [10].

3.3. Evolution of BBR

The most notable changes between versions are the incorporation of packet loss when determining the bottleneck bandwidth [11], [13] and the breakdown of the ProbeBW phase into four sub-phases in BBRv2 [12]. Piotrowska found in a study that BBRv1 caused excessive packet loss in some scenarios [12] which likely led to this decision. To further limit the inflight data volume the parameters inflight_hi and inflight_lo have been introduced. The former is used in BBR's long term path model while the latter is used in the short term model to limit the cwnd [10]. The long term model is supposed to be more stable and keep track of long term safe data rates and volumes, while the short term model is adjusted more often to adapt to current network conditions [10]. BBRv3's changes were much less radical than the ones introduced in v2. BBRv3 is mostly BBRv3 with the inclusion of fixes for bugs that affected fairness towards other flows [12], [14]. Furthermore parameters have been adjusted, pacing_gain during startup has been been lowered to 2.77 and cwnd_gain was drastically reduced to 2.00. A full overview of parameter changes can be found in Table 1.

3.4. Fairness

BBR is not the only congestion control algorithm in use and therefore has to compete with others for bandwidth. While only a couple of congestion control algorithms have been outlined in this paper, there are many more in use for various scenarios. All concurrent internet flows using various algorithms have to compete for their share of available bandwidth and therefore it is important that none takes away most of it while others are left with the bare minimum. This section briefly summarizes research about BBR's fairness towards itself as well as other congestion control algorithms.

3.4.1. Fairness towards BBR. Zeynali et al. have evaluted fairness between multiple BBR flows in various scenarios such as flows starting at the same point in time and flows being started at different times [11]. They have found that while for two same version BBR flows starting at the same time the fairness as described by Jain's Fairness Index [15] has indeed improved a little bit, it has worsened extremely for staggered flows. While two BBRv1 and BBRv2 flows being started with a gap of 15 s of each other converged to a similar bandwidth rather quickly, it took almost 5 minutes for the same to happen among two BBRv3 flows for a buffer with 16*BDP size. In total, while BBRv1 displays many more retransmissions due to its complete ignorance of packet loss as a signal for congestion, it ends up being the fairest when fighting for bandwidth among itself.

3.4.2. Fairness towards other congestion control algorithms. Gomez et al. have evaluated fairness between various versions of BBR among themselves as well as CUBIC [6]. They observed various scenarios such as two flows competing with each other without any loss, 100 flows (split up in 50 of each version) competing with each other and 100 flows (split up in 50 of each version) without loss, as well as the same scenarios with a loss of 0.025%. For 100 flows and a loss of 0.025% they have found that BBRv3 and CUBIC have similar throughput for higher buffer sizes. For smaller buffer sizes BBRv3 reaches a higher throughput than CUBIC. This effect is even worse for BBRv2 and CUBIC, where BBRv2 takes almost all the bandwidth in shallow buffer settings. All in all their results show that better resource sharing and therefore fairness is achieved with bigger buffer sizes the more flows compete against each other. However, even for shallow buffers Piotrowska et al. found that BBRv3 does indeed enhance "fairness by 12%" towards CUBIC [12]. On the other hand Zeynali et al. found that even multiple CUBIC flows cannot compete against a single BBR flow in terms of Jain's Fairness Index and "end up competing between themselves for the bandwidth leftover[...]" [13], [15].

4. Conclusion and future work

In this paper, we presented the past and current state of Google's BBR algorithm. BBR, according to Google, shows much promise in terms of operating at near BDP and providing high bandwidth. They state that "Playbacks using BBR show significant improvement in all of YouTube's quality-of-experience metrics" [1] and "BBR reduces median RTT by 53 percent on average globally and by more than 80 percent in the developing world." [1]. BBRv3 has become the default congestion control algorithm for traffic in Google's services [13] and now accounts for at least more than 40% of the internet's total traffic volume according to Mishra et al [16]. It is consistently developed and improved upon and - given its parametrized nature and usage of a state machine - it is easy to imagine that more states could be added in the future to further improve the algorithm or better adapt to specific scenarios. Although BBRv3 brings notable improvements over earlier versions, research has revealed that it can still be highly unfair towards other BBR connections. Therefore it is critical that further refinements are needed to enhance fairness not only when BBRv3 coexists with other congestion control algorithms, but especially when several BBRv3 flows share the same bottleneck, especially if BBRv3 is to become the default congestion control algorithm of the future. As of March 2024 however, BBRv3 was not yet included in Linux's mainline TCP, for which a submission was planned as soon as possible [17]. Looking at the latest meetings of the Congestion Control Working Group (**ccwg**) it becomes clear that development of BBR is in full force with one of the most recent topics that are being discussed being making improvements to BBR for real-time connections [18].

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] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.

[3] J. Sharma and A. K. Garg, "Analysis of tahoe: A tcp variant," *International Journal of Engineering and Advanced Technology (IJEAT) ISSN*, pp. 2249–8958.

[4] 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.

[5] R. Al-Saadi, G. Armitage, J. But, and P. Branch, "A survey of delay-based and hybrid tcp congestion control algorithms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3609–3638, 2019.

[6] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "Evaluating tcp bbrv3 performance in wired broadband networks," *Computer Communications*, vol. 222, pp. 198–208, 2024.

[7] N. Cardwell, Y. Cheng, K. Yang, D. Morley, S. Hassas, P. Jha, and Y. Seung, "Bbr congestion control: Fundamentals and updates," 2023.

[8] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *ICC 1979; International Conference on Communications, Volume 3*, vol. 3, 1979, pp. 43–1.

[9] S. Yang, Y. Tang, W. Pan, H. Wang, D. Rong, and Z. Zhang, "Optimization of bbr congestion control algorithm based on pacing gain model," *Sensors*, vol. 23, no. 9, p. 4431, 2023.

[10] N. Cardwell, I. Swett, and J. Beshay, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-ietf-ccwg-bbr-01, Oct. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-ccwg-bbr/01/

[11] D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, "Promises and potential of bbrv3," in *International Conference on Passive and Active Network Measurement*. Springer, 2024, pp. 249–272.

[12] A. Piotrowska, "Performance evaluation of tcp bbrv3 in networks with multiple round trip times," *Applied Sciences*, vol. 14, no. 12, p. 5053, 2024.

[13] D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, "Bbrv3 in the public internet: a boon or a bane?" in *Proceedings of the 2024 Applied Networking Research Workshop*, 2024, pp. 97–99.

[14] N. Cardwell, Y. Cheng, K. Yang, D. Morley, S. Hassas, P. Jha, Y. Seung, V. Jacobson, I. Swett, B. Wu *et al.*, "Bbrv3: algorithm bug fixes and public internet deployment," *Presentation in CCWG at IETF*, vol. 117, 2023.

[15] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, p. 1, 1984.

[16] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great internet tcp congestion control census," in *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, 2020, pp. 59–60.

[17] N. Cardwell, Y. Cheng, K. Yang, D. Morley, S. Hassas Yeganeh, P. Jha, Y. Seung, and V. Jacobson, "Bbrv3: Algorithm overview and google's public internet deployment," IETF 119 Meeting Materials, March 2024, presented at IETF 119, CCWG meeting, Brisbane. [Online]. Available: https://datatracker.ietf.org/meeting/119/materials/slides-119-ccwg-bbrv3-overview-and-google-deployment-00

[18] C. Huitema, S. Nandakumar, and C. Jennings, "Bbr improvements for real-time connections," IETF 120 Meeting Materials, July 2024, presented at IETF 120, CCWG meeting, Vancouver. [Online]. Available: https://datatracker.ietf.org/meeting/120/materials/slides-120-ccwg-bbr-improvements-for-real-time-connections-00.pdf