Introduction to BBRv2 Congestion Control

Joji Mathew, Benedikt Jaeger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: joji.mathew@tum.de, jaeger@net.in.tum.de

***Abstract*—Congestion happens when an overflow occurs due to more data being sent than the bandwidth and this leads to loss. Congestion control algorithms try to prevent this and the ones currently used widely are TCP algorithms which are loss-based and delay-based starting only once a loss has occurred leading to retransmissions due to which further congestion occurs. Google proposed BBR in 2016 using another approach called congestion-based congestion control which acts at an earlier stage compared to TCP algorithms promising maximum throughput and minimum queuing delay. But it still comes with drawbacks that Google is trying to solve with the second version BBRv2 which is still in the testing stage. In this paper, we give an overview on congestion control overall, BBR, and introduce BBRv2.**

***Index Terms*—congestion control, TCP, BBR, CUBIC, bottleneck bandwidth**

1. Introduction

TCP (Transmission Control Protocol) is a protocol in the transport layer that ensures to establish end to end connection between two hosts and is also responsible for the safe transmission of data between the two. Whenever a data packet is lost it makes sure to re-transmit it. When multiple senders, unaware of the bandwidth of the connection send lots of data, it leads to congestion in the network which causes buffer overflow and packet loss. TCP uses congestion control algorithms to avoid this situation [1]. TCP has several implementations for the same starting with RENO to the present-day CUBIC.

Excessive buffering and delays were reported on the Internet in 2011 [2]. This was mainly due to the fact that TCP operates only having loss as a parameter and starts operating too late once congestion has occurred. By this time senders re-transmit lost data which leads to further congestion. BBR which was developed by Google tries to operate at Kleinrock's optimal point [3] way before a loss occurs.

The paper is structured as follows. Section 2 talks about the different approaches used in congestion control. An introduction to BBR, it's working, and performance analysis is discussed in Section 3. Section 4 talks about some drawbacks of BBR. BBRv2 is introduced briefly and an elaboration of its algorithm and performance analysis is done in Section 5. Finally, the conclusion and future work is given in Section 6.

2. Different approaches in congestion control

These are some terms needed for later and their explanations according to Scholz et al. in [3].

- ***BtlBW***- Bottleneck is a link in the network path with the lowest bandwidth and this bandwidth is called Bottleneck bandwidth(BtlBW).
- ***RTT***- Round Trip Time(RTT) is the time required for a data packet to reach the receiver plus the time for the acknowledgment to reach the sender.
- ***RTprop***- It is the round trip propagation delay when there is no queuing delay and very less data in the buffer.
- ***Inflight data***- Amount of data in the buffer.
- ***BDP***- RTprop · BtlBW is called a pipe's Bandwidth-Delay Product(BDP).
- ***cwnd***- Amount of data being sent in each RTT

2.1. TCP congestion control algorithms

TCP has primarily two congestion control approaches. Even though it has a lot of algorithms in development. The following are the ones used widely:

Loss-based congestion control. This approach identifies loss as congestion and is used in implementations like RENO, CUBIC. The basic idea of the implementation is that it starts with a small cwnd, then increases exponentially after each RTT, and whenever a loss occurs, it interprets it as congestion and reduces the cwnd by a certain amount and now increases it unlike earlier only in small amounts. This process of reducing cwnd and slow incrementation keeps on repeating whenever a loss occurs [4].

Delay-based congestion control. This other approach identifies delay as a parameter for congestion instead of loss and starts working when it sees an increase in RTT.

2.2. Congestion based Congestion control

The problem with loss-based congestion is that it only starts acting when a loss occurs. When the Amount Inflight > BDP queue starts slowly building up in the buffer, increasing RTT, and when the Amount inflight= BDP + Bottleneck buffer size, the buffer becomes full, leading to congestion and packet loss [5].

In bigger buffers loss-based congestion control algorithm sends data at full bandwidth causing buffer bloat,

 doi: 10.2313/NET-2023-11-1_16

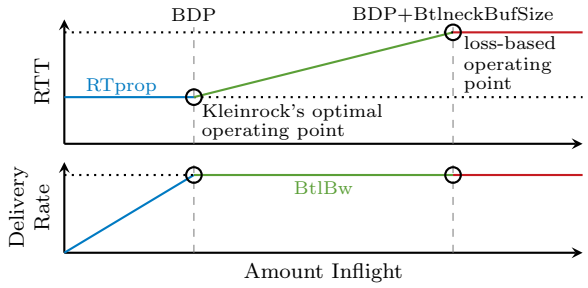


Figure 1: Amount Inflight vs Delivery Rate and RTT [3]

and only starts acting at BDP + Buffer size, where it is already too late since the loss of packets causes retransmission resulting in further delays [6]. In the older days, it used to make sense since memory was expensive and buffers were smaller so it minimized the delay of reaching BDP + Buffer size. But nowadays, with lesser prices and more memory available, it causes a delay of seconds. In the case of small buffers, this algorithm is also overly sensitive and bring about low throughput because it misinterprets loss as congestion, which could have occurred due to other reasons like flows entering the pipe.

The optimal point to start congestion control is when Amount Inflight = BDP, as shown in Figure 1, also known as Kleinrock's optimal point [6]. This condition ensures that the bottleneck does not starve, and at the same time, the data does not overfill the pipe. Also the bottleneck could be fully utilized if the sending and arrival rate of packets at the bottleneck are equal. BBR tries to satisfy both of these conditions.

3. BBR

BBR(Bottleneck Bandwidth and Roundtrip) was proposed as an alternative approach by Google in 2016 [3]. It is a congestion-based congestion control approach that mainly tries to achieve high throughput with a small queue and can withstand random losses upto $15\,\%$, according to Cardwell et al. in [7]. BBR operates on the parameter BDP mainly, and to measure BDP, it has to constantly determine BtlBW and RTT. But they cannot be observed simultaneously, since as shown in Figure 1, RTT can be measured accurately when Amount Inflight < BDP so that there is no congestion resulting in RTT = RTprop and BtlBW has to be measured when Amount Inflight > BDP so that we get the maximum bandwidth.

3.1. Algorithm

According to Scholz et al. in [3], BBR algorithm operates in 4 phases.

Phase 1(Startup): *Pacing gain* is a parameter that controls the amount of data sent, that when multiplied with the BtlBW shows the current sending rate. The sending rate is doubled after each roundtrip, and once the bandwidth has reached its maximum value, which BBR assumes as the BtlBW, it continues with the second phase.

Phase 2(Drain): Here, BBR tries to reduce the queue created at the bottleneck due to the first phase by temporarily decreasing the *pacing gain* and starts with next phase.

Phase 3(Probe Bandwidth): In this phase, with a total of 8 cycles BBR tries to estimate the BtlBW. In the first cycle, the *pacing gain* is set to 1.25 to probe for extra bandwidth and then at 0.75 to drain the queues created. Then for the rest of the phase, i.e., six cycles, the *pacing gain* is set to 1. The bandwidth is sampled constantly throughout the time and the maximum is used as the BtlBW. This value holds for a period of 10 RTprop. Then it enters the next phase.

Phase 4(Probe RTT): In this phase, which is about $200\,\mathrm{ms}$ plus one RTT, bandwidth is set to four packets to drain any possible queues created by the third phase to get the current estimation of RTT. RTprop value is updated if a new minimum is measured and is valid for tens seconds.Both Probe_BW and Probe_RTT phase are repeated continuously and updated.

3.2. Performance Analysis of BBR

When we look at the performance analysis of BBR and CUBIC done by Cardwell et al. in [7], it is to be seen that BBR achieves high bandwidth despite losses. At a loss rate of $0.01\,\%$, CUBIC lowers the throughput to around $30\,\mathrm{MB\,s^{-1}}$ and at $0.1\,\%$, further lowers to $12.5\,\mathrm{MB\,s^{-1}}$ while BBR maintains the maximum at around $90\,\mathrm{MB\,s^{-1}}$, which shows that even small losses lead to low throughput in a loss-based congestion control approach. When measuring the buffer size(MB) against latency (s), CUBIC has a linear increase in latency while BBR maintains a low queue delay despite bloated buffers.

4. Problems of BBR

Even though BBR promised to solve a lot, it still comes with problems. According to experiments done by Ma et al. in [8], Song et al. in [5], and Scholz et al. in [3], the following drawbacks were observed:

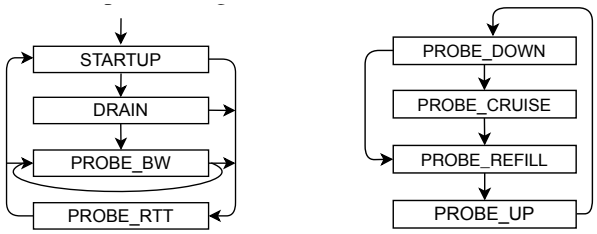
4.1. RTT unfairness

With multiple flows, flows with longer RTT recieve a larger share of bandwidth compared to flows with smaller RTT is the opposite of traditional loss-based and delay-based algorithms, which favor flows with shorter RTT. This leads to the following problems. There is an unpleasant trade-off between low latency and high delivery rates. It no longer makes any sense to find a route that has minimum RTT since flows with longer RTT receive more bandwidth leading to a high delivery rate. Secondly, since BBR is also a sender-based congestion control like TCP, it will be easier to manipulate and increase the RTT from the receiver side to get more share of bandwidth, and the sender would be totally unaware of the situation. This could lead to worse outcomes if all the receivers compete and try to inflate their RTT constantly.

4.2. Unresponsive to packet loss

BBR overestimates the bottleneck bandwidth and according to the analysis done by Hock et al. in [9], the

 doi: 10.2313/NET-2023-11-1_16



(a) Phases of BBRv1 & BBRv2 (b) PROBE_BW in BBRv2

Figure 2: Phases of BBR and BBRv2 algorithm [10]

Amount Inflight is between 2 BDP and 2.5 BDP, most of the time. This could lead to problems when the buffer size is less than 1 BDP since BDP + Buffersize, in this case,would be less than 2 BDP. Due to the overestimation BBR might start operating only at 2BDP, whereas loss has already started occurring at BDP + Buffersize and BBR would not reduce the amount of data sent since it does not respond to loss, leading to further retransmissions and delays.

4.3. Fairness between BBR and other algorithms

When BBR operates together with loss-based congestion like CUBIC, the share of the bandwidth varies according to the buffer size. When the buffer is shallow, BBR occupies more bandwidth and in the case of deep buffers CUBIC occupies a major share. In each of these cases, this behaviour leads to starvation of flows using the other algorithm leading to high retransmissions.

5. BBRv2

BBR was proposed by Google in 2016 [3] as an alternative to traditional loss and delay-based algorithms, promising maximum throughput with minimum queuing delay. To achieve this, BBR tries to work at Klienrock's optimal point. It was deployed in Google B4 WAN, Google.com and Youtube video servers [6]. B4 is a high-speed Wide Area Network from Google, and in B4, it was to be seen that BBR had a throughput 2 to 25 times greater than CUBIC, and when the receiver window was raised, BBR was 130 times faster than CUBIC. In Google.com, web page downloads were faster mainly in the developing world. On Youtube, playbacks using BBR had less rebuffering, and BBR was also able to achieve a median RTT of $53\,\%$ and $80\,\%$ in developing world.

Even though BBR showed promising results in the deployments, it still came with flaws. Apart from the ones discussed earlier, BBR also does not respond to ECN signals (Internet Protocol provides an extension called Explicit Congestion Notification(ECN) to notify TCP about network congestion without dropping the packet, thus avoiding retransmission [11]), and it has low throughput for paths with high aggregation(wifi). BBRv2, the newest version of BBR with modifications, was developed by Google in 2018 to solve these problems [12]. To coexist better with algorithms like CUBIC, it adapts bandwidth probing. BBRv2 also responds to loss and uses DCTCP-style ECN. Finally, it estimates the recent degree of aggregation to avoid low throughput problems [13].

	CUBIC	**BBR**	**BBRv2**
Parameters	None	BtlBW, RTT	BtlBW, RTT, max aggregation,max inflight
Response to loss	Yes	No	Yes
Response to ECN	Yes	No	Yes
Startup	slow until RTT rise or any loss	slow start until threshold	slow start until threshold or ECN/loss rate > target
cwnd in Probe RTT phase	N/A	4 packets	BDP/2

TABLE 1: Comparison table based on [12], [16]

A current draft for the BBRv2 is available on the internet [14] and as per the draft BBRv2 is currently available publicly for Linux TCP and QUIC. QUIC is a transport layer protocol like TCP implemented by Google and deployed in 2012. It is supported in major web browsers like Google Chrome, Firefox, and Microsoft Edge. It tries to establish a connection with less latency than TCP. QUIC also aims at bandwidth estimation in both directions, which in turn helps the BBRv2 at BtlBW estimations [15].

5.1. Algorithm

Explanation of the algorithm according to [14] and [10]. BBRv1 and BBRv2 algorithms work similarly in most cases. BBRv2 has some additional modifications on how it responds to loss and ECN. The Probe_Bandwidth phase is adaptive and therefore helps to coexist better with algorithms like CUBIC solving the interprotocol fairness problem. BBRv2 keeps tabs on the current estimate(.._latest), lower bound(.._lo), and higher bound(.._hi) values of the bandwidth and amount of data in flight. The congestion window is bounded by these lower and upper bound values instead of the inflight cap of 2 BDP, like in BBRv1. BBRv2 also has some minor changes with the congestion window in the drain phase.

Both BBR and BBRv2 exit the startup phase and enter the drain phase when there has been no significant increase of a minimum of $25\,\%$ in the bandwidth in the last three RTTs. In addition to this condition, BBRv2 also goes to the drain phase if there is packet loss or when ECN marks $50\,\%$ of the delivered packets for two consecutive RTTs.

The main difference in the algorithm is in the Probe_Bandwidth phase. According to Nandagiri et al. in [10], BBRv2 further breaks Probe Bandwidth into four sub-phases, as shown in Figure 2. In the Probe_Down phase BBR tries to reduce the amount of data in flight by lowering the sending rate to $90\,\%$ of the bottleneck

 doi: 10.2313/NET-2023-11-1_16

bandwidth. It exits this state when there is free headroom. So if the inflight_hi value is set, it remains in this state until the volume of inflight data is less than inflight_hi. The inflight_hi helps prevent loss and leaves space for other flows or cross traffic. Also, any queues created at the bottleneck have to be drained. Both of these conditions have to be met to exit this phase. In the Probe_Cruise phase, BBRv2 tries to send data at the same rate as the delivery rate, i.e., data is sent at 100 % capacity of bottleneck bandwidth. In this phase, it responds to loss and ECN by reducing bandwidth_lo and inflight_lo, indicating that the delivery rate and amount of data inflight need to be reduced. This phase holds adaptively and exits when it needs to probe for more bandwidth. In the Probe_Refill phase, BBRv2 has a goal of refilling the pipe. It attempts to send at 100 % of bottleneck bandwidth for just one more RTT, with bandwidth_hi and inflight_hi restraining the connection. In the Probe_Up phase, BBRv2 tries to probe for more bandwidth. The *pacing gain* is set to 1.25 which when multiplied by the bottleneck bandwidth, gives the current sending rate. When higher bandwidth and amount inflight values are measured, the bandwidth_hi and inflight_hi values are updated. It exits this phase when there is either a loss of 2 % or when the queue is high enough that the flow judges that it has probed adequately.

In the Probe RTT phase, which BBRv1 enters every 10 seconds to estimate the latest RTprop, BBRv1 sends only four packets to drain the queues created in Probe_BW whereas BBRv2 enters the phase every 5 seconds and maintains the throughput by draining it only to half the BDP in this phase.

Table 1, based on [16] and [12], shows the difference in how CUBIC, BBR, and BBRv2 work.

5.2. Performance analysis of BBR v2

Response to loss. As it can be seen in Figure 3 from Song et al. in [12], CUBIC is overly sensitive and starts reducing throughput from a loss of 0.001 %, BBR is loss agnostic and continues to deliver maximum throughput up to around 15 % and BBRv2 provides a good middle ground providing maximum throughput until about 1 % loss.

Inter-protocol Fairness. Experiments were conducted by Nandagiri et al. in [10] to check the Inter-protocol fairness. In the experiment, ECN was disabled since CUBIC and BBRv2 work with different types of ECN. In shallow buffers, BBRv1 had more share than CUBIC as pointed out earlier. But in the case of BBRv2, the sharing of bandwidth with CUBIC is fairer than BBRv1. This behaviour attributes to BBRv2 being bounded by lower and upper bound values and not having an inflight cap like BBRv1, preventing losses in the startup phase, helping CUBIC to maintain its throughput and not reduce the congestion window. After Startup both BBRv2 and CUBIC reduce throughput when they encounter loss. During the probing phase, the bandwidth of BBRv2 remains somewhat constant, but CUBIC on the contrary, increases its bandwidth after each RTT leading to having a slightly larger share.

In the case of deep buffers BBRv2 has a throughput way less than its fair share. Both CUBIC and BBRv2 have an equal throughput in the beginning and reduce

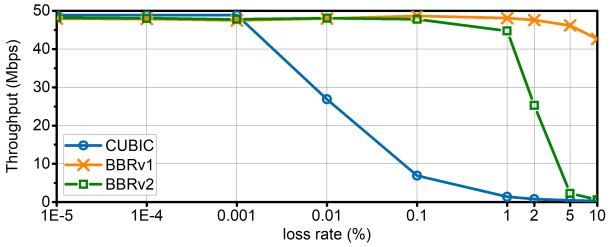


Figure 3: Throughput vs loss for CUBIC,BBR and BBRv2 [12]

throughput encountering a loss. But when CUBIC starts increasing its sending rate BBRv2 is still in the drain phase, trying to reduce the queues created at Startup phase. By the time BBRv2 enters the Probe_Bandwidth phase, it experiences loss due to CUBIC reaching the threshold forcing BBRv2 to reduce its bandwidth's higher bound further.

RTT fairness and Intra-protocol fairness. Experiments were conducted by Song et al. in [12] to determine the RTT and Intra-protocol fairness of BBRv2. It was done with two flows, with the first flow having a fixed RTT of 30 ms and the second flow with varying RTT. In BBRv1 flows with larger RTT received a bigger share of bandwidth than the shorter flows. In BBRv2 a similar behaviour is observed. For flows with the same RTT, the flow starting first gets a bigger share of throughput, but according to Nandagiri et al. in [10], similar flows have better fairness when ECN is enabled. Experiments on RTT fairness were also conducted by Gomez et al. in [17] with 50 flows of 10 ms RTT and 50 flows of 50 ms RTT. It is shown that both BBRv1 and BBRv2 allocate more bandwidth to flows with bigger RTT when buffer size is above 0.6 BDP, but when the buffer size is above 12 BDP, BBR v2 has a very high fairness index, whereas BBRv1 is still unfair.

6. Conclusion and future Work

BBRv2 is the best version of BBR, as of now. BBRv2 when deployed on Youtube had lesser RTT than CUBIC and BBRv1. It is currently deployed as the default TCP congestion control for internal Google traffic [16]. It provides maximum throughout till 1% loss, is more efficient using ECN signals, reduces queuing delay, and has better throughput in wifi. BBRv2 alpha is the current version which has some shortcomings which Google is working on fixing with the final version. BBRv2 has a fairness issue when ECN is disabled and is a bit too complex to deploy in WAN like b4 due to its dependency on DCTCP like ECN [10]. Although RTT fairness issues are better compared to BBRv1, it has not been solved completely. The same situation remains with inter-protocol fairness in the case of deep buffers. According to [18], Google is trying to add BBR.Swift extension to BBRv2. It aims to use delay also as a parameter without changing the BBR core. Google is also planning for a full-scale rollout in their company.

References

[1] GeeksforGeeks, "Transport Layer Responsibilities," [accessed 24-May-2023]. [Online]. Available: https://www.geeksforgeeks.org/transport-layer-responsibilities/

[2] Engagement and P. operations center, "20200504 - Neal Cardwell - BBR: A Model-based Congestion Control," https://www.youtube.com/watch?v=mpbWQbkl8_g, [accessed 17-July-2023].

[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*. IEEE, 2018, pp. 1–9.

[4] GeeksforGeeks, "TCP Congestion control," [accessed 3-June-2023]. [Online]. Available: https://www.geeksforgeeks.org/tcp-congestion-control

[5] Y.-J. Song, G.-H. Kim, and Y.-Z. Cho, "Bbr-cws: improving the inter-protocol fairness of bbr," *Electronics*, vol. 9, no. 5, p. 862, 2020.

[6] 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.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev, and V. Jacobson, "Bbr congestion control: Ietf 99 update," in *Presentation in ICCRG at IETF 99th meeting*, 2017. [Online]. Available: https://www.ietf.org/proceedings/99/slides/slides-99-iccrg-iccrg-presentation-2-00.pdf

[8] S. Ma, J. Jiang, W. Wang, and B. Li, "Fairness of congestion-based congestion control: Experimental evaluation and analysis," *arXiv preprint arXiv:1706.09115*, 2017.

[9] 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.

[10] A. Nandagiri, M. P. Tahiliani, V. Misra, and K. K. Ramakrishnan, "Bbrvl vs bbrv2: Examining performance differences through experimental evaluation," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, 2020, pp. 1–6.

[11] Wikipedia, "Explicit Congestion Notification," accessed 12-June-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Explicit_Congestion_Notification

[12] Y.-J. Song, G.-H. Kim, I. Mahmud, W.-K. Seo, and Y.-Z. Cho, "Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm," *IEEE Access*, vol. 9, pp. 37 131–37 145, 2021.

[13] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "Bbr v2 a model-based congestion control," 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

[14] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/

[15] Wikipedia, "QUIC," accessed 12-June-2023]. [Online]. Available: https://en.wikipedia.org/wiki/QUIC

[16] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao *et al.*, "Bbrv2: A model-based congestion control performance optimization," in *Proc. IETF 106th Meeting*, 2019, pp. 1–32. [Online]. Available: https://lafibre.info/testdebit/linux/201911_bbr_v2_doc_ietf106.pdf

[17] J. Gomez, E. Kfoury, J. Crichigno, E. Bou-Harb, and G. Srivastava, "A performance evaluation of tcp bbrv2 alpha," in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, 2020, pp. 309–312.

[18] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, V. Jacobson, N. Dukkipati, and G. Kumar, "Bbr update : 1:bbr.swift; 2:scalable loss handling," 2020. [Online]. Available: https://datatracker.ietf.org/meeting/109/materials/slides-109-iccrg-update-on-bbrv2-00