Investigating WebRTC BBR as an alternative to GCC for live video streaming

Rebecca Drucker
Stony Brook University, Furman University
rdrucker@cs.stonybrook.edu, rebecca.drucker@furman.edu

Aruna Balasubramanian, Anshul Gandhi
Stony Brook University
{arunab,anshul}@cs.stonybrook.edu

***Abstract*—Google Congestion Control (GCC) is the default congestion control algorithm for WebRTC, a popular web application used for live video streaming. BBR, also developed at Google, is commonly used for streaming pre-recorded video on services like YouTube. However, BBR has not been widely deployed for real-time applications like live video streaming. It was implemented for WebRTC in 2018, but it was later deprecated due to poor performance. While GCC performs well under most network conditions, it can be starved by a loss-based TCP flow using the same bottleneck link. In this work, we investigate the possibility of using BBR as an alternative to GCC for WebRTC congestion control. We test it under a variety of network conditions and find that it performs better than GCC when competing with TCP, and it achieves bitrates comparable to GCC's in isolation, except when bandwidth is restricted and the bottleneck buffer is deep. We find that this is because of bandwidth overestimation, a problem which also exists in TCP BBR. While modifying WebRTC BBR's bandwidth estimation fails to improve performance in our experiments, we do find that disabling its recovery state, a unique loss response, improves WebRTC BBR's performance in underprovisioned networks.**

I. Introduction

Live video streaming is a web application that is growing in popularity. Twitch, a gaming-focused live streaming platform, has seen its average number of concurrent viewers increase from 102,000 to 2.42 million between 2012 and 2023 [1]. Established social media platforms such as Facebook, Instagram, and YouTube have added live streaming capabilities as well.

WebRTC is a popular choice for live video streaming applications. Its default congestion control algorithm is Google Congestion Control (GCC). Despite being designed specifically for this application, it can provide degraded video quality when competing with loss-based TCP traffic because the TCP flow starves the GCC flow [2], [3]. Therefore, an alternative congestion control which can effectively share bandwidth with TCP is needed.

TCP BBR has been used for video streaming by services such as YouTube and Netflix. However, it has never been deployed at a large scale for live video, so its effectiveness for live video streaming is unknown.

WebRTC developers implemented BBR as an alternate congestion control option to the default GCC, but they later deprecated it due to poor performance [4]. Outside of WebRTC, BBR has been observed to overestimate the min RTT in some cases [5], [6], which could impact latency-sensitive applications like live streaming. Developers observed an inflated min RTT estimate when using BBR as the congestion control for WebRTC [4], which resulted in poor video streaming performance. However, our experiments suggest that WebRTC BBR is able to achieve higher video quality than GCC when sharing a bottleneck link with a TCP flow. Thus, if its performance issues when run in isolation (without competing flows) can be resolved, it may serve as a viable alternative to GCC.

In this work, we compare the performance of BBR and the default GCC for a live video streaming application under a variety of network conditions. We find that, like TCP BBR, WebRTC BBR underperforms in deep buffers due to bandwidth overestimation. It performs especially poorly in underprovisioned networks.

To address WebRTC BBR's performance issues, we first alter its RTT estimation behavior to mimic that of BBRv2, which includes increased frequency of RTT probes and increased cwnd during the probes to alleviate problems with low throughput. Developers observed issues with RTT estimation for WebRTC BBR as well [7]. However, this change fails to improve BBR's live video quality, suggesting that another factor is responsible for the poor performance.

Next, we investigate WebRTC BBR's bandwidth estimation, since TCP BBR is known to underperform in deep buffers due to inflated bandwidth estimates. We find a negative correlation between BBR's bandwidth estimate and video quality. While both TCP BBR and WebRTC BBR expire bandwidth estimates after a 10-RTT window, WebRTC BBR only keeps three estimates at a time and replaces all three when a new best (maximum) is achieved. This practice only amplifies bandwidth overestimation when compared to TCP BBR.

We first replace all requests for the best bandwidth estimate with requests for the third best, but this degrades performance. We then alter the bandwidth estimation technique so that a new best only replaces the current best, not all three estimates; however, this also results in reduced video quality.

In an attempt to improve WebRTC BBR's video quality in bandwidth-restricted conditions, we observe that WebRTC BBR has a loss response called a recovery state that is distinct from the threshold-based loss response found in BBRv2. Disabling this recovery state indeed improves BBR's video quality by 63%.

Finally, we confirm that, when modified to disable the recovery state, WebRTC BBR's performance not only improves when in isolation, but also outperforms GCC when competing with a TCP flow on the same bottleneck link. These results suggest that, with appropriate modifications, WebRTC BBR may serve as a potential alternative to GCC, particularly for

users sharing bandwidth with TCP traffic.

The rest of this paper is organized as follows. In Section II, we provide the necessary background on live video streaming and the protocols and algorithms employed for live video streaming. Section III discusses the related work on live video streaming. We then describe our experimental setup in Section IV, which we use to analyze the performance of BBR and GCC for live video streaming in Sections V and VI. Section VII details our attempts at improving the video streaming performance of WebRTC BBR. Finally, we conclude in Section VIII.

II. Background

A. Live video streaming

In large-scale deployments, the broadcaster is unlikely to stream video directly to each viewer. Instead, the broadcaster will stream to a CDN server to which viewers will connect and stream the video. The raw video is first uploaded to an ingest server which transcodes it into multiple bitrates, which are then transferred to CDN servers where clients can fetch them.

In contrast to DASH video, live streamed video is sent to the viewer as it is created. This means that low latency is the main goal, and there is little time to run algorithms that choose an appropriate bitrate for each video chunk.

B. WebRTC and RTP

While WebRTC is most commonly used for video conferencing, it is also used for low-latency live video streaming because it:

1) often uses UDP, which has less overhead than TCP;
2) uses RTP, a protocol optimized for real-time data transmission;
3) pushes video chunks to the client as they become available rather than having the client request each chunk [8]; and
4) supports adaptive bitrate streaming and is compatible with many browsers and video players [9].

WebRTC is primarily able to achieve low latency by using Real-Time Transmission Protocol (RTP) at the transport layer. TCP is not well suited to real-time applications because it guarantees complete and in-order delivery of all packets, but it does not make any guarantees about latency. In contrast, UDP makes no guarantees, but packets are never retransmitted, so the client never waits for a packet. RTP offers a compromise by providing sequence numbers and timestamps so that the application can detect if a packet is missing and use data contained in packets at the right time. RTP is used for data, and its companion protocol RTCP is used to send QoS information to all receivers and to keep track of participants in the session. There are separate RTP sessions for video and audio, and RTCP synchronizes them [10].

C. GCC

Google Congestion Control (GCC) [11] is the default congestion control option for WebRTC. This algorithm, introduced in a 2011 IETF draft [12], was designed specifically for real-time applications. Unlike popular TCP congestion control algorithms like Cubic, GCC uses a combination of loss and delay signals to determine its sending rate, rather than relying on loss alone.

GCC has been well-documented to underperform when sharing a bottleneck link with TCP traffic [2], [3]. Because GCC uses a combination of loss and delay signals to determine an appropriate sending rate, in the presence of congestion, it decreases its sending rate before loss-based TCP congestion control algorithms. This results in bandwidth starvation for the GCC flow in the presence of competing loss-based TCP flows.

D. BBR

BBR (Bottleneck Bandwidth and Round-trip propagation time) is a congestion control algorithm introduced by Google in 2016 to alleviate bufferbloat, a condition of inflated RTTs caused by loss-based congestion control approaches such as Reno and Cubic filling large buffers in the network. Since its introduction, two updates to BBR, called BBRv2 and BBRv3, have been released by Google. BBR and its variants have been successfully deployed in Google's networks and others' for applications like video streaming.

Rather than responding to loss, BBR relies on estimates of the available bandwidth and the propagation delay (represented by the maximum delivery rate within a 10-RTT window and the measured RTT after draining the bottleneck buffer, respectively) to set its sending rate, approximating the optimal operating point of the network.

In 2018, a version of BBR was implemented for WebRTC and introduced into its source code. It was deprecated soon thereafter due to poor performance [4]. While TCP BBR and WebRTC BBR operate similarly, WebRTC BBR contains some features which do not exist in any TCP version of BBR. The most notable of these is a unique loss response, which we investigate in Section VII-B. WebRTC BBR also has a more limited window for bandwidth and RTT samples than TCP BBR, which we discuss in Section VII-A.

III. Related Work

Many works investigate live streaming and protocols associated with it. Jansen et al. conduct a performance evaluation of WebRTC as a video conferencing tool and find cases where it underperforms [13]. In their measurement study, Deng et al. map out Twitch's network infrastructure to understand how they deliver live video to millions of users [14]. They find the locations of servers and investigate Twitch's strategy for allocating broadcasts and viewers to the servers. Kim et al. present LiveNAS, a system that uses machine learning to improve live video QoE. Their video ingest server employs a deep neural network that up-samples lower-resolution video frames to improve visual quality without consuming the bandwidth needed to transfer a higher-resolution chunk [15]. Salsify [16]

is a low-latency video system design that combines a compressed video codec and UDP-based transport protocol with a congestion control strategy similar to WebRTC's GCC and Sprout. It encodes video frames to maximize quality while minimizing latency. Several works also focus on improving BBR's performance on real-time applications [17]–[19].

IV. Experimental Setup

We now briefly describe our experimental setup. Figure 1 shows the WebRTC video streaming testbed used in our experiments. One machine acts as a broadcaster, sending video frames and audio, while the other acts as a viewer, sending feedback to the broadcaster. The broadcaster and viewer are connected via a Linksys WRT1900ACS router with OpenWRT 19.07.1 installed.

Large-scale live video streaming applications also use either a STUN or TURN server to allow broadcasters inside private networks to stream video to viewers inside their own private networks. When a STUN server is unable to allow the broadcaster and viewer to connect directly, a TURN server must be used. All traffic between the broadcaster and the viewer must be routed through this TURN server. In our experiments, we use a STUN server.

We use the most recent version of WebRTC which includes BBR as a congestion control option. This version was released in 2018.

All experiments use a five-minute video. The maximum average bitrate achieved when streaming this video across all of our experiments was approximately 1.3Mbps. All experimental conditions were run at least ten times.

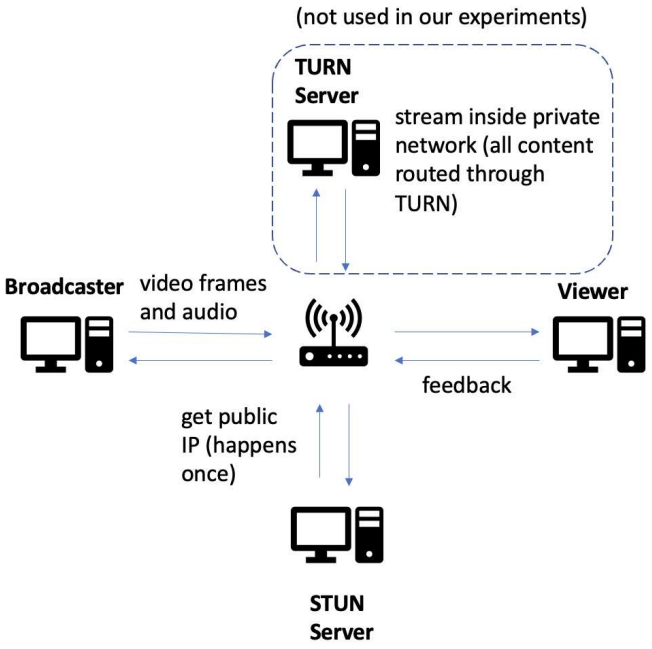


Fig. 1: Illustration of our experimental setup, showing the relationships between broadcaster, viewer, and WebRTC-specific servers in the testbed used in our experiments.

V. Performance of BBR and GCC

We first compare the performance of BBR and GCC when sharing a bottleneck link with a TCP flow to determine whether BBR could be a suitable alternative to GCC. Then, we compare the two congestion control options on their own under a variety of network conditions. To the best of our knowledge, no prior work has empirically compared the performance of WebRTC BBR and GCC with and without TCP competition.

A. Performance under TCP competition

As early as 2013, GCC was known to perform poorly when sharing a bottleneck link with a TCP flow [2], [3]. To determine whether this problem still exists in more recent versions of GCC, as well as whether WebRTC BBR has the same issue under TCP competition, we streamed live video using both GCC and BBR with a 100ms RTT, 5 Mbps bandwidth, and a 1 MB buffer while running iPerf3 on the same bottleneck link. The iPerf3 sender uses Linux default Cubic for congestion control. Figure 2 shows the average bitrate achieved by GCC and BBR when competing with the TCP flow across all experiments. Each congestion control's achieved bitrate without TCP competition is also shown for comparison.

While GCC and BBR perform similarly in isolation, GCC's bitrate decreases by 96% when sharing the bottleneck link

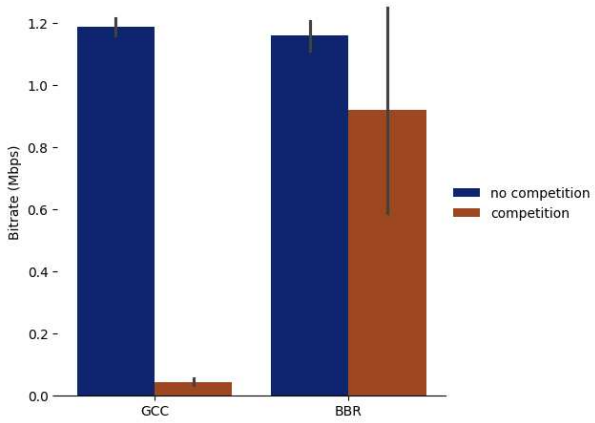


Fig. 2: Average achieved bitrate of GCC and BBR under 100 ms RTT, 5 Mbps bandwidth, and a 1 MB buffer, with and without a competing TCP flow on the same link.

with a TCP flow. In contrast, BBR's bitrate decreases by only 21% under competition. Note that some decrease in bitrate may be necessary when sharing bandwidth with a competing TCP flow. Furthermore, while both BBR and GCC have high average RTTs in competition because the competing TCP flow uses Cubic, a loss-based congestion control algorithm which is known to fill deep buffers (resulting in inflated RTTs [20]), we find that BBR maintains a slightly lower average RTT than GCC (1.2 seconds for BBR vs. 1.5 seconds for GCC).

Since BBR achieves bitrates 95% higher than GCC and maintains similar RTTs when streaming live video under

RTT	Bandwidth
20 ms, 50 ms, 100 ms	500 kbps, 1 Mbps, 2 Mbps, 5 Mbps

TABLE I: Network conditions used in our experiments on each congestion control algorithm in isolation.

competition from TCP, we posit that BBR may be a viable alternative to GCC when a live video stream must share the link with TCP flows. In the sections that follow, we subject WebRTC BBR (as well as GCC, for comparison) to live video streaming experiments under a variety of network conditions to determine its suitability for live video streaming applications.

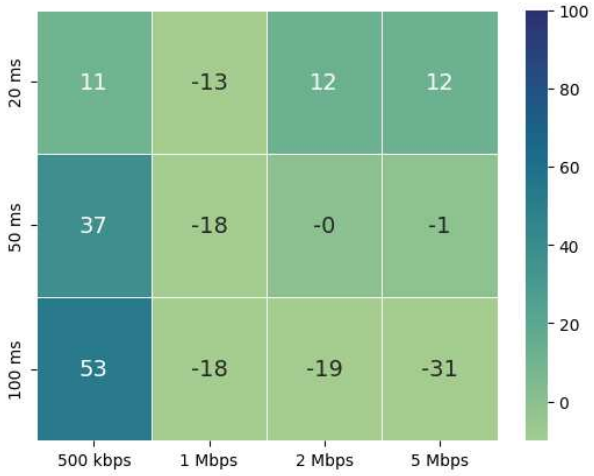
B. Performance in isolation

Next, we compare WebRTC BBR and GCC by studying their performance in isolation across a variety of RTTs, bandwidths, and buffer sizes. Table I shows the network conditions that were used in our experiments. For each combination of the three RTTs and the four bandwidths, we chose two buffer sizes: one which would be shallow in comparison to the BDP, and one which would be deep.

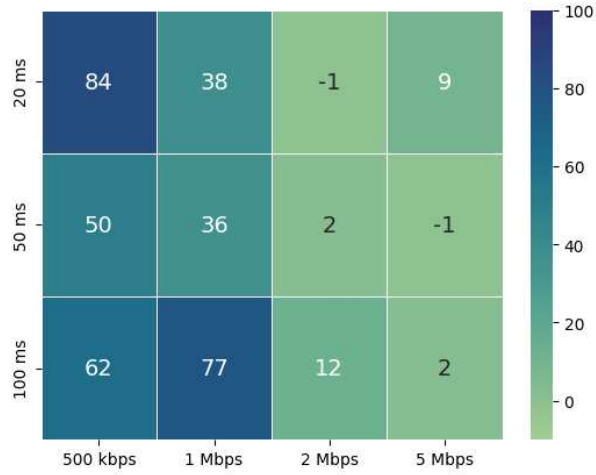
Figure 3 shows the percentage difference in achieved bitrate between BBR and GCC under the network conditions we tested. Positive values indicate that GCC achieved the higher bitrate under a particular condition, while a negative value indicates that BBR achieved the higher average bitrate. Overall, GCC and BBR achieve similar bitrates across all of our experiments, except under two sets of conditions: low bandwidth/high BDP in the shallow buffer condition, and low bandwidth in the deep buffer condition.

In our experiments, the maximum average bitrate achieved in live video streams using either congestion control algorithm we tested is approximately 1.3Mbps. Providing additional bandwidth beyond this threshold does not increase the average bitrate. Thus, the 500 kbps and 1 Mbps bandwidth conditions represent an *underprovisioned* network, while the 2 Mbps and 5 Mbps bandwidth conditions represent an *overprovisioned* network. It is only in the underprovisioned scenario that WebRTC BBR consistently achieves lower video bitrates than GCC.

Furthermore, under deep buffers, GCC achieves video bitrates up to 84% greater than those achieved by BBR for the two lower-bandwidth conditions. Previous studies have found that TCP BBR can underperform in deep buffers [20] due to bandwidth overestimation associated with the max filter it uses to choose its bandwidth estimate [21]; a similar effect may be taking place in the deep buffer scenario for WebRTC BBR.



(a) Shallow buffer



(b) Deep buffer

Fig. 3: Percent difference between the average video bitrate for GCC and BBR under (a) a shallow buffer and (b) a deep buffer.

VI. Explaining BBR's low video quality

We have now confirmed that while BBR outperforms GCC when competing with TCP flows, it achieves much lower bitrates than GCC when bandwidth is restricted, especially when the bottleneck buffer is deep. To solve this problem, we must first determine the root cause of BBR's poor performance under these conditions.

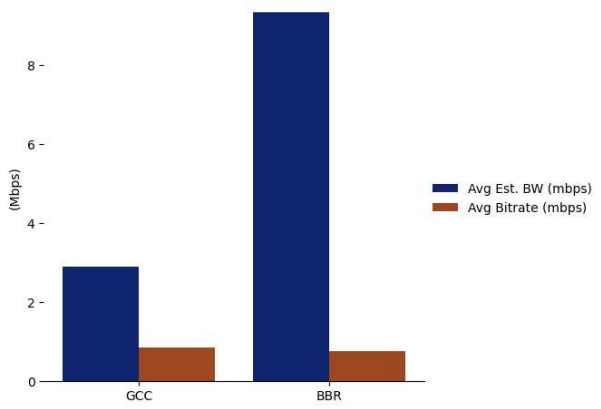
Since BBR relies on its bandwidth and RTT estimates to set its sending rate, inaccurate estimates can lead to bandwidth underutililization or excessive sending, either of which can result in application performance degradation and a poor user experience.

A. Inaccurate RTT and bandwidth estimation

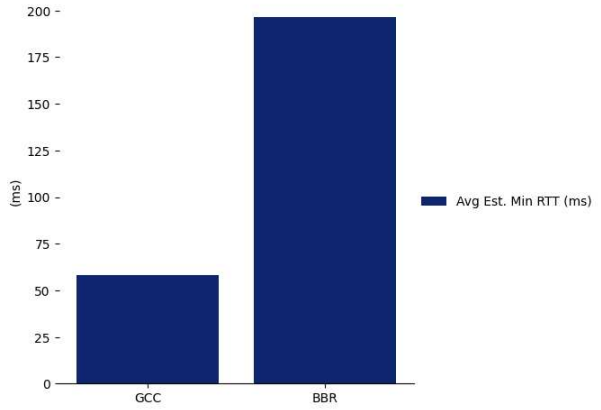
We first investigate WebRTC BBR's RTT estimation, which was noted by its developers to be "inflated" for bidirectional video streams [4]. While our tests only involve sending in one direction (broadcaster to viewer), we suspect this problem may exist in our experiments as well.

Since previous studies of TCP BBR have shown that it overestimates bandwidth under deep buffers, it may be the case that this problem also exists in the WebRTC version.

Figure 4 shows that, across all tested network conditions, BBR's estimated bandwidth and estimated min RTT are considerably higher than GCC's estimates of the same values under the same conditions, suggesting that BBR is overestimating both values. However, as found in prior work on



(a) Estimated bandwidth and bitrate



(b) Estimated RTT

Fig. 4: (a) Average achieved bitrate and estimated bandwidth, and (b) average estimated min RTT of GCC and BBR, across all our experiments.

TCP BBR performance for pre-recorded video streaming, high RTT estimates may be a symptom of the problem rather than the root cause, which has been found to be bandwidth overestimation [21]. The developers of WebRTC BBR also observed "slight" bandwidth overestimation in their tests [4]. We will next determine whether the bandwidth overestimation we observe above is responsible for the low achieved bitrates of WebRTC BBR in Figure 3.

B. Relationship between bandwidth estimate and achieved bitrate

Ideally, a higher bandwidth estimate should indicate that network conditions are favorable, and a higher video bitrate may be sent. This would be evident in a strong positive correlation between the average bandwidth estimate and the average video bitrate.

Table II shows the Pearson correlation between BBR and GCC's bandwidth estimates and their video bitrates across all of our experiments. Overall, there is a strong positive correlation between estimated bandwidth and bitrate for GCC, and a weak negative correlation for BBR.

Goodput (Mbps)	overall	bitrate $\geq$ 700 kbps	bitrate $<$700 kbps
GCC	0.87	0.94	-0.18
BBR	-0.24	0.59	-0.59

TABLE II: Pearson correlations between the average estimated bandwidth and the average bitrate for GCC and BBR, across all runs, and separated by runs with better ($\geq$700 kbps) and worse ($<$700 kbps) average bitrates.

We also separate experiments with higher average bitrates from those with lower average bitrates and recalculate the correlations. For both GCC and BBR, the correlation between estimated bandwidth and video bitrate is positive when bitrates are high and negative when bitrates are low. However, the positive correlation for higher-bitrate experiments is stronger for GCC, and the negative correlation for lower-bitrate experiments is stronger for BBR. This suggests that, statistically speaking, even when BBR performs well, its bandwidth estimates are less accurate than GCC's. High bandwidth estimates are also more strongly related to low bitrates for BBR than for GCC, likely due to BBR's reliance on accurate bandwidth estimates to set its sending rate appropriately.

This result confirms that for BBR in particular, not only are its bandwidth estimates frequently inaccurate (specifically, that BBR is *overestimating* the available bandwidth), but that these inaccurate estimates are associated with low video bitrates.

VII. Improving BBR's video quality

In Section VI, we established that under restricted bandwidth and deep buffer conditions, WebRTC BBR overestimates both the available bandwidth and the min RTT.

In this section, we make three alterations to WebRTC BBR's RTT and bandwidth estimation techniques in an attempt to improve its video quality when bandwidth is restricted and the bottleneck buffer is deep. We also investigate its *recovery state*, a response to packet loss which is not found in any TCP variant of BBR.

A. Bandwidth and RTT estimation

We must first determine which changes to make to WebRTC BBR's bandwidth and RTT estimation techniques to improve its performance. For TCP BBR, an updated version called BBRv2 has been shown to perform nearly as well as BBRv1 while maintaining desirable fairness properties [22]. Thus, a natural first step in improving WebRTC BBR's performance is to mimic the behavior of BBRv2. Since WebRTC BBR was created prior to the introduction of BBRv2 in 2019, no WebRTC version of BBRv2 exists. We instead aim to replicate two of BBRv2's most prominent features: a modified RTT estimation technique and a response to packet loss.

WebRTC BBR already responds to loss, a feature which we will investigate further in Section VII-B. We discuss a modification to WebRTC BBR's RTT estimation in this subsection.

In BBRv2, the developers of BBR altered its RTT estimation technique. BBRv1 entered the PROBE_RTT state every 10

seconds and reduced its cwnd to 4 packets. This resulted in undesirable variations in throughput [23].

Change #1. We implement a BBRv2-like RTT estimation technique.

We make three changes to WebRTC BBR to make its RTT estimation behave similarly to BBRv2: first, we allow it to choose the number of packets in flight during RTT probes to be based on the current estimated BDP; then, we set the cwnd during RTT probes to half of the estimated BDP; and finally, we expire RTT estimates after 5 seconds rather than 10 seconds.

WebRTC BBR's RTT estimation also has a component not found in any other version of BBR: it does not initiate an RTT probe if the most recent RTT sample is within 12.5% of the current min RTT estimate. Thus, it will reduce the sending rate to measure the current min RTT less often than BBRv2. We leave this behavior in place for all of our experiments.

This results in a version of WebRTC BBR that behaves similarly to BBRv2, except that its loss response differs.

The next two changes relate to WebRTC BBR's bandwidth estimation.

Change #2. We use the third-best bandwidth sample to set the sending rate.

Like TCP BBR, WebRTC BBR uses the maximum observed delivery rate as its bandwidth estimate. This practice can lead to bandwidth overestimation, as we observe in our experiments. One proposed solution is to use a sample lower than the maximum as the bandwidth estimate [21]. However, rather than keeping a window of bandwidth samples that covers the past 10 RTTs, WebRTC BBR keeps only the top three samples at a time. To mitigate WebRTC BBR's overestimation, we alter WebRTC BBR to use the third-highest bandwidth sample as its estimate rather than the highest bandwidth sample.

Change #3. We alter the sample filter to replace only the highest bandwidth sample with the new maximum, rather than replacing all three samples.

The default behavior of WebRTC BBR's sample filter, when a new sample arrives, is to replace all samples less than or equal to that sample with the new sample. For example, if a bandwidth sample arrives which is greater than the current second-highest sample, but still less than the maximum sample, the second-highest and third-highest samples will *both* be replaced with the new sample. If the sample were to exceed the current highest estimate, all three samples would be replaced with the new sample. Thus, new samples are favored by WebRTC BBR, and a large sample can quickly replace all other samples in the window, potentially leading to long-lasting bandwidth overestimation.

Change #3 alters this behavior. Instead of replacing all samples less than or equal to a new sample, only one sample is replaced. In our previous example, in which a sample arrives which is greater than the second-highest sample but less than the highest sample, the new sample will replace the previous second-highest sample, and the previous second-highest sample will replace the previous third-highest sample.

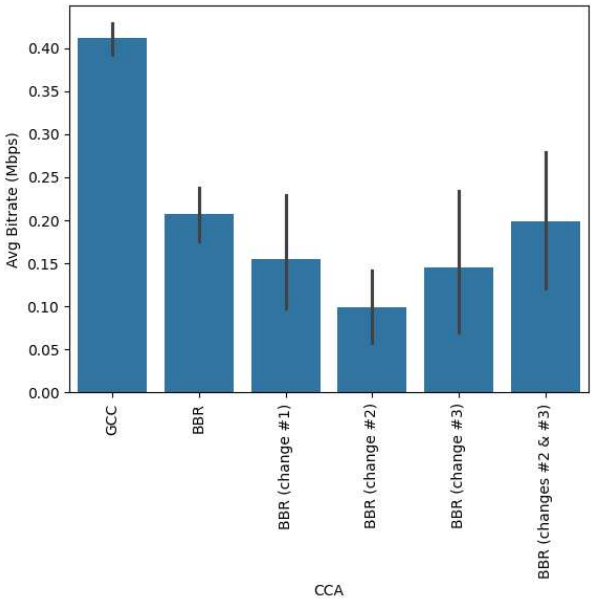


Fig. 5: Comparison of average bitrates achieved by altered versions of BBR, as well as unmodified WebRTC BBR and GCC, under a 50 ms RTT, 500 kbps bandwidth, and a (deep) 100 kb buffer.

Note that WebRTC BBR uses the same sample filter for its bandwidth and RTT estimates. This means that any change to the sample filter affects not only bandwidth samples, but also RTT samples.

Figure 5 shows the results of these changes under a low-bandwidth, deep-buffer network condition. We test each of the changes in isolation, and we also combine changes #2 and #3, which affect bandwidth estimation. Average bitrates for GCC and unmodified BBR under the same network condition are shown for comparison.

Only the version of WebRTC BBR with both of the bandwidth estimation changes achieves a bitrate greater than or equal to that of unmodified BBR. The other changes in isolation achieve even lower bitrates. None of the changes result in an improvement in WebRTC BBR's performance in an underprovisioned network with a deep bottleneck buffer.

B. Recovery state

Since all three changes to WebRTC BBR's bandwidth and RTT estimation failed to improve video bitrates in underprovisioned, deep buffer network conditions, we next investigate a unique property of WebRTC BBR not found in any of its TCP variants: the recovery state.

Like BBRv2 and BBRv3, WebRTC BBR responds to packet loss. However, unlike those TCP BBR variants, it does not use a threshold-based approach. WebRTC BBR's recovery state is a response to loss that begins on a single packet loss. WebRTC BBR exits recovery if there are no losses in a round. The recovery state consists of four sub-states: CONSERVATION, MEDIUM_GROWTH, GROWTH,

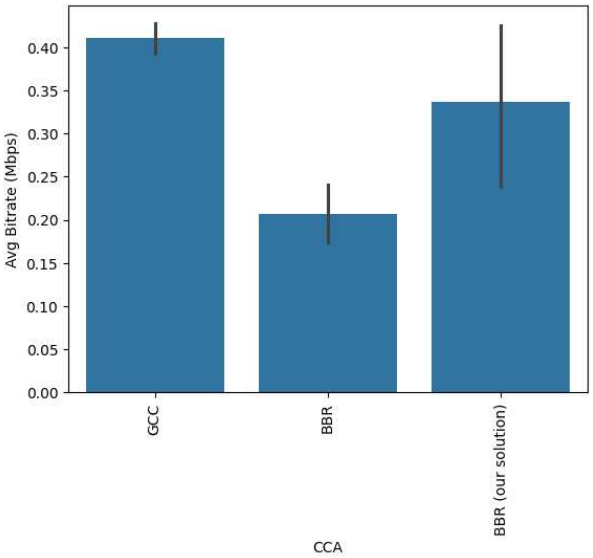


Fig. 6: Comparison of average bitrates achieved by the version of BBR with the recovery state disabled, as well as unmodified WebRTC BBR and GCC, under a 50 ms RTT, 500 kbps bandwidth, and a 100 kb buffer.

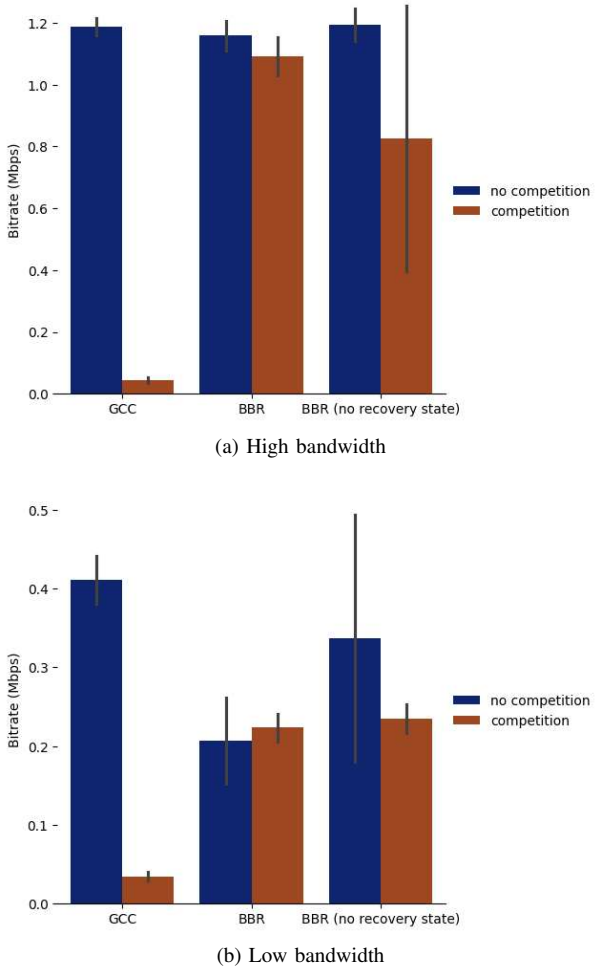
and NOT_IN_RECOVERY. In the CONSERVATION state, the cwnd is reduced by the number of bytes lost. In the GROWTH state, cwnd increases by the number of bytes acked, and in MEDIUM_GROWTH, cwnd increases by half the number of bytes acked.

We hypothesize that WebRTC BBR's loss response is especially harmful to performance in potentially high-loss conditions such as the restricted bandwidth conditions in our experiments. Suppose that all, or nearly all packets are lost during a round trip. This would result in the cwnd immediately being reduced to zero or near-zero. This possibility motivates our next change to WebRTC BBR.

Our solution. We disable BBR's recovery state. This is achieved by ensuring that BBR's recovery state is always set to NOT_IN_RECOVERY.

Figure 6 compares the average bitrate achieved by GCC and unmodified BBR with the average bitrate achieved by BBR with the recovery state disabled (our solution). Our modified version of BBR with the recovery state disabled outperforms unmodified BBR. We also tested a version with the third best bandwidth estimate used, the sample filter altered, and the recovery state disabled (changes #2, #3, and our solution). However, the version with changes #2 and #3 included does not perform as well as the version with our solution alone, suggesting that the alterations to the bandwidth estimation essentially undo any benefits provided by the disabling of the recovery state.

With the recovery state disabled, WebRTC BBR is able to achieve an average video bitrate only 18% less than that of GCC; this translates to a 63% higher average video bitrate over the unmodified WebRTC BBR. If this version of WebRTC



(a) High bandwidth

(b) Low bandwidth

Fig. 7: Average achieved bitrate of GCC, BBR, and BBR with recovery state disabled under (a) 100 ms RTT, 5 Mbps bandwidth, and a 1 MB buffer, and (b) 50 ms RTT, 500 kbps bandwidth, and a 100 kb buffer, with and without a competing TCP flow on the same link.

BBR maintains its favorable properties under competition from TCP, it may serve as a viable alternative to GCC.

C. Improved WebRTC BBR under TCP competition

Since we are investigating WebRTC BBR as a potential alternative to GCC under TCP competition, we next test our modified WebRTC BBR sharing the bottleneck link with a TCP flow, as described in Section V-A. Figure 7 shows the average video bitrate of WebRTC BBR with the recovery state disabled, with and without TCP competition, under high and low bandwidth network conditions. GCC and unmodified WebRTC BBR results are shown for comparison. While our modified WebRTC BBR does achieve lower average bitrates under TCP competition than unmodified WebRTC BBR in the high bandwidth condition, it vastly outperforms GCC under the same conditions. Our modified WebRTC BBR does not achieve an average bitrate as high as GCC in the low band-

width condition when TCP competition is absent; however, it outperforms unmodified WebRTC BBR by 63%. Furthermore, in the presence of a competing TCP flow, WebRTC BBR continues to vastly outperform GCC under low bandwidth with our modification.

Thus, we posit that, with the recovery state disabled, WebRTC BBR can perform well both under competition from TCP flows and in restricted-bandwidth, deep buffer network conditions.

VIII. Conclusion and Future Work

This work presents a modified version of WebRTC BBR as a potential alternative to the default GCC for live video streaming applications. With the recovery state disabled, WebRTC BBR achieves an average video bitrate 6.8× higher than GCC under competition from TCP flows, and its performance in low-bandwidth, deep-buffer network conditions improves by 63% when compared to unmodified WebRTC BBR. However, since the root cause of WebRTC BBR's poor performance is its overestimation of bandwidth, more extensive changes to its bandwidth estimation technique would be required for it to serve as a viable alternative to GCC. Modifications to the bandwidth estimation might include a larger window for bandwidth samples, similar to that used by TCP versions of BBR.

Other future work in this space could focus on developing an appropriate response to loss, perhaps using a threshold-based approach such as that used in TCP BBRv2 and BBRv3. Further studies could also evaluate video QoE for WebRTC BBR and GCC, rather than using QoS metrics like average bitrate and RTT, as we did in this work. Furthermore, WebRTC BBR and any modifications to it should be evaluated under 5G network conditions to determine whether it would be suitable for use in modern cellular networks.

Acknowledgments

We thank the reviewers for their constructive feedback. This work was supported in part by NSF grant CNS-1909356.

References

[1] "Twitch statistics and charts," https://twitchtracker.com/statistics, Jan 2023.

[2] L. De Cicco, G. Carlucci, and S. Mascolo, "Experimental investigation of the google congestion control for real-time flows," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking*, ser. FhMN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 21–26. [Online]. Available: https://doi.org/10.1145/2491172.2491182

[3] ——, "Understanding the dynamic behaviour of the google congestion control for rtcweb," in *2013 20th International Packet Video Workshop*, 2013, pp. 1–8.

[4] "Question about applying bbr on video streaming," https://groups.google.com/g/bbr-dev/c/1EPG5UwBANo, Mar 2021.

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

[6] K. Miyazawa, K. Sasaki, N. Oda, and S. Yamaguchi, "Cycle and divergence of performance on tcp bbr," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018, pp. 1–6.

[7] "Running webrtc with bbr," https://groups.google.com/g/bbr-dev/c/siLJCN9DlnM/m/dLwaJPBzFgAJ, Jan 2023.

[8] "6 ways webrtc solves ultra low latency streaming," https://www.red5pro.com/blog/6-ways-webrtc-solves-ultra-low-latency-streaming/, Dec 2019.

[9] E. Krings, "Rtmp vs. hls vs. webrtc: Comparing the best protocols for live streaming," https://www.dacast.com/blog/rtmp-vs-hls-vs-webrtc/, Jun 2022.

[10] A. Durresi and R. Jain, "Rtp, rtcp, and RTSP - internet protocols for real-time multimedia communication," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–11.

[11] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2910017.2910605

[12] H. Lundin, S. Holmer, and H. T. Alvestrand, "A google congestion control algorithm for real-time communication on the world wide web," Working Draft, IETF Secretariat, Internet-Draft draft-alvestrand-rtcweb-congestion-01, October 2011.

[13] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman, "Performance evaluation of webrtc-based video conferencing," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, p. 56–68, mar 2018. [Online]. Available: https://doi.org/10.1145/3199524.3199534

[14] J. Deng, G. Tyson, F. Cuadrado, and S. Uhlig, "Internet scale user-generated live video streaming: The twitch case," 02 2017, pp. 60–71.

[15] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 107–125. [Online]. Available: https://doi.org/10.1145/3387514.3405856

[16] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein, "Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol," in *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'18. USA: USENIX Association, 2018, p. 267–282.

[17] R. Kumar, A. Koutsaftis, F. Fund, G. Naik, P. Liu, Y. Liu, and S. Panwar, "Tcp bbr for ultra-low latency networking: Challenges, analysis, and solutions," in *2019 IFIP Networking Conference (IFIP Networking)*, 2019, pp. 1–9.

[18] S. Najmuddin, M. Asim, K. Munir, T. Baker, Z. Guo, and R. Ranjan, "A bbr-based congestion control for delay-sensitive real-time applications," *Computing*, pp. 1–23, 2020.

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

[20] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to use and when not to use bbr: An empirical analysis and evaluation study," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 130–136. [Online]. Available: https://doi.org/10.1145/3355369.3355579

[21] S. Vargas, R. Drucker, A. Renganathan, A. Balasubramanian, and A. Gandhi, "Bbr bufferbloat in dash video," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 329–341. [Online]. Available: https://doi.org/10.1145/3442381.3450061

[22] R. Drucker, G. Baraskar, A. Balasubramanian, and A. Gandhi, "Bbr vs. bbrv2: A performance evaluation," in *2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2024, pp. 379–387.

[23] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, and M. M. V. Jacobson, *BBR v2: A Model-based Congestion Control*, IETF-104 : iccrg, Mar 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00