Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

An Evaluation of BBRv2 Congestion Control Using NS-3

Killian Lanigan
[redacted]

Supervisor: Prof. Meriel Huggard

April, 2020

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
Master In Computer Science

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed:  Date: 30/4/2020

Abstract

With the ever growing demand for optimal performance of a network, a solution to provide consistent speeds and low latency is through the introduction of new congestion control methods in the transportation of packets across a network. In this work, the second version of Bottleneck Bandwidth and Round Trip Time, a congestion control algorithm developed by Google, is evaluated to see if it stays true to its claims of achieving an optimal use of network resources. This evaluation is performed using the network simulation tool NS-3. This new congestion control algorithm was implemented in NS-3 along with the original version and other variations of the algorithm developed from studies of the original Bottleneck Bandwidth and Round Trip Time algorithm. A network topology was setup in NS-3 and simulations with multiple different network scenarios were tested. The findings from these simulations where that this new version of BBR improved on its predecessor in situations involving networks with multiple flows of the same congestion control algorithm and in contention with a delay-based congestion control algorithm. The new BBR algorithm failed to validate its claims in scenarios with a competing flow consisting of a loss-based algorithm. This evaluation shows that there is a positive progression in the new version of BBR in achieving its goals, however, since it is still in development, there are still some issues that have to be addressed.

Acknowledgements

I would like to thank my supervisor Prof. Meriel Huggard for the support, feedback and guidance throughout the course of this project. I would also like to thank my family and friends for their continued support throughout the duration of my academic journey. Finally, I would like to thank the School Of Computer Science and Statistics for their amazing response during a time of global crisis, showing support for students to continue their work and to stay safe.

Contents

List of Figures

List of Tables

List of Code Listings

1 Introduction

1.1 Background and Motivation

The demand for fast and reliable internet is a constant requirement in modern day telecommunications. The use for these fast speeds is due to the increase in use of real time applications like video streaming which requires a reliable stream of packets varying in size with quality of video and takes up to 65% of downstream traffic in the world [5]. Breaking down these requirements of speed and reliability into telecommunication related terms might see them defined as the utilisation of a bandwidth channel, a high delivery rate of packets, low latency and little to no packet loss. These types of characteristics can be seen to be managed through the transport layer of the OSI internet stack [6] which is represented in Figure 1.1. The layers in this stack make up the fundamentals around transfer and use of information over the internet with each layer serving each other. The transport layer is at the heart of the stack and provides important protocols in order to perform effective data transfer. The main protocol used in transport layer is Transport Control Protocol [7]. Congestion control is a part of Transport Control Protocol and can be seen to dictate the network metrics of bandwidth utilisation, delivery rate, latency and packet loss.

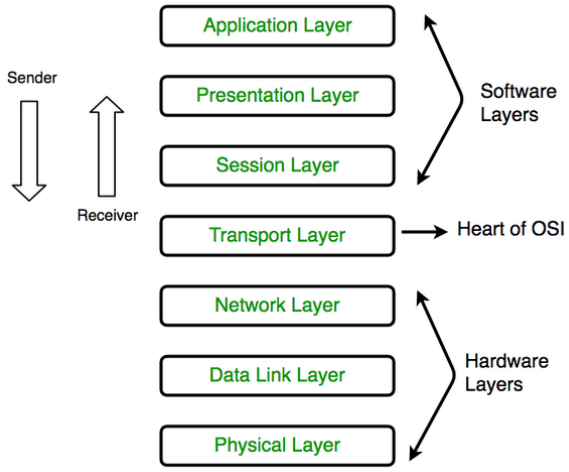

Figure 1.1: OSI Network Model

There exists a number of different congestion control algorithms that aim to provide the optimal balance between these types of metrics in a network to achieve an efficient and reliable connection. In order to keep up with the demand for increasing speeds and reliability of connections, telecommunication teams in service providers like Google have been engaging in research to implement new congestion control algorithms and introduced a new congestion control algorithm to face the influx of increasing internet demands and to improve the performance of all of its services. This congestion control algorithm was called Bottleneck Bandwidth and Round Trip Time (BBR) and initially looked to solve this demand for a high speed reliable connection [8]. This success was short lived as extensive research into the new congestion control algorithm found flaws and undesirable characteristics in its behaviour [9]. This encouraged Google to develop a second version of the algorithm to overcome these issues and stay on track with a goal of increasing reliability of data transfer whilst maintaining high speeds [2]. This new version is still under development by Google and is an open source project. Google encourages researchers and academics to test this new type of congestion control algorithm and contribute to the development. The motivation behind this project is to engage in the current research encouraged by teams at Google to help in the development of this new congestion control algorithm by giving a good performance evaluation of the algorithm and ultimately provide a better internet experience for users in the long term.

1.2 Research Objectives

The main objective of this project is to apply evaluation techniques using a network simulator to assess this new version of the bottleneck bandwidth and round trip time using appropriate network metrics. Since this version of the bottleneck bandwidth and round trip time congestion control algorithm is only very new, there is very little evaluations on its performance and validations of the goals it wishes to achieve. This project aims at validating these claims and giving a clear insight into the performance of the new congestion control algorithm. The original algorithm along with a number of variations suggested from related works will also be evaluated in order to compare their performance with the new version and outline distinct differences between them in functionality. In order to achieve these objectives, these algorithms will be implemented in the network simulator NS-3 and appropriate network scenarios will be set up for testing.

1.3 Dissertation Layout

Chapter 2 of this dissertation will consider the state-of-the-art and related research in the evaluation of BBR congestion control and Network Simulation. Chapter 3 will outline the design and technical implementation of this project. Chapter 4 will present a critical evaluation of the results of this project. Chapter 5 will conclude this dissertation with a summary of the key findings and a discussion of potential future work.

2 Background Research

This chapter gives an insight into the related state of art research into BBR congestion control and Network Simulation. The approaches to BBR evaluation will be examined in the section on related work in BBR. Network Simulation techniques and corresponding tools will also be discussed in the section on Network Simulation and NS-3. Abbreviation reference can be found in Appendix A1.

2.1 Congestion Control

Congestion occurs in a network when the host is unable to process the volume of traffic it is being sent. The premise around the transportation of packets across the internet is a concept that involves a lot of situational complexities with a series of different protocols and techniques that make up the stack of packet transportation. Different network-defined applications use different variations of these techniques to achieve optimal transportation of packets. Congestion Control can be described as a solution to the complexity problem, dealing with the problems around achieving the optimal requirements of packet transportation. The underpinning philosophy of congestion control is to achieve as much utilisation as possible of the network without overloading the network. Packets typically operate in flows that represent a stream of packets from the sender to the receiver, each flow passes through a network of switches and routers known collectively as nodes. These switches and routers often use buffers to process the packets passing through them. The transportation of packets itself is carried out using the Transmission Control Protocol (TCP), which itself adopts an array of different congestion control algorithms. TCP is the most used transport layer protocol in the internet [7]. Such algorithms work by assessing and addressing congestion using different techniques with unique attributes depending on network conditions and network defined metrics.

Congestion control algorithms can be divided into groups depending on their method of congestion detection [10]. Loss-based algorithms are ones that detect congestion when packets are dropped from full buffers. Delay-based algorithms use round trip time

(RTT) measurements where the RTT is the amount of time taken for a packet to go from sender to receiver and for an acknowledgement to be sent back to the sender. A fluctuation in the RTT indicates delays due to increased buffering. Hybrid is the name given to a class of algorithms that exploit both the previous methods for congestion detection. Some Hybrid algorithms might classify their type of congestion control algorithm based on their unique congestion detection method. A table of different congestion control algorithms and their unique benefits can be see below in Table 2.1.

Table 2.1: Congestion Control Algorithms Overview

Category	**Algorithm**	**Benefit**
Loss-Based	BIC	Optimised for long fat networks, binary methods for cwnd [11]
	CUBIC	Enhanced version of BIC, cwnd non-dependent of RTTs [12]
	Reno	3 cwnd management algorithms and 2 loss detection methods [13]
	NewReno	Improves retransmission during fast-recovery phase of TCP Reno [14]
	Scalable	Cwnd managed to benefit high speed networks [15]
	HighSpeed(HSTCP)	Utilises available bandwidth in networks with high BDP [16]
	H-TCP	Maintains consistent fairness in both high and low BDP links [17]
Delay-Based	Vegas	40-70% better throughput than Reno, proactive mechanism [18]
	Vegas+	Aggressive cwnd size mechanism for fairness with TCP Reno [19]
	FAST	Congestion measure is queuing delay, descendant of Vegas [20]
	VFAST	Better Fairness than FAST with smoother RTT measurements [21]
Hybrid	BBR	Detects actual congestion through model based methods [8]
	BBR'	Slightly modified BBR that increases fairness [22]
	BBRPlus	Increased BBR performance by change in pacing gain method [23][22]
	BBR+	Higher throughput than BBR at cost of RTT [24]
	Delay-BBR	Fairness, rate stability and low packet loss compared to BBR [25]
	BBRv2	BBR upgrade in terms of fairness, packet loss and throughput [2][26][27]
	Illinois	Uses both packet loss and queuing delay as detection, high throughput [28]
	Compound TCP	Has 2 different cwnds to increase accuracy of congestion detection [28]

Congestion control algorithms continuously survey the network's state in order to gauge if there is congestion at the bottleneck. TCP Congestion control is window-based, the congestion window (Cwnd) is used to limit the amount of the packets the sender can have in the network. If there is no congestion detected at the bottleneck then the sending rate is increased to make use of the additional bandwidth available, this is done by increasing the congestion window size. The bottleneck is often considered the main congestion point of a network and taken as the maximum delivery rate [29]. Congestion occurs at the bottleneck when the sending rate is increased to a point where it can not handle the incoming packets quickly enough; that is when packets are arriving more quickly than they are being forwarded. Queues eventually will form at each buffer and there is possibility of lost packets if buffer overflow occurs. Increasing the buffer size is not necessarily a warranted solution as it causes bufferbloat which adds delay to the packets due to the increased number of queued packets in the buffer [30]. The different

congestion control algorithms use different methods of increasing and decreasing the congestion window to provide optimal use of network resources.

2.1.1 Network Metrics

Evaluation of congestion control algorithms is based on specific metrics that gauge the overall effectiveness of the algorithm; for example, the sending rate and the throughput are some of the metrics used. The flow of bits per unit time is typically known as the bit rate [31], while the sending rate is the bit rate of a specific flow generated from a source. Throughput represents the amount of bits that are received per unit time over a network link. The ideal throughput is the sending rate; however, this is often not achieved due to factors such as traffic in the system and the sharing of network resources among other sources. Throughput can be viewed as the average utilisation rate of the supplied bandwidth.

Although the data transfer rate might be high it is important to be able to classify the usefulness of the received data. The transportation of packets across a network introduces overhead due to a number of reasons including protocol computations and retransmissions. A way to gauge the usefulness of the received data is to determine how much is transported to the application layer. A general metric used to measure this is the Goodput. This is represented using the formula [10]:

$$Goodput = \frac{(D_s - D_r - D_o)}{\Delta t} \tag{1}$$

where D_s is the number of useful bits transmitted, D_r is the number of bits retransmitted and D_o is the number of overhead bits in a time interval Δt. In essence, the goodput ratio can be seen as the ratio of useful transmitted data to the total amount of data transmitted [10].

Different devices and applications use different congestion control algorithms if they use TCP to transmit data. The way in which these different algorithms interact with each other and share bandwidth is referred to as fairness [32]. Fairness is broken into three categories: (1) intra-fairness which refers to the resource distribution between flows running the same congestion control algorithm; (2) inter-fairness which refers to the resource distribution between flows running a different congestion control algorithm; (3) RTT-fairness which is a property defined by the difference in resource distribution of flows with different RTTs. Jain's index [33] is a quantitative representation of fairness through an index that ranges from 0 to 1, the closer the value is to 1 the more equal the share of resources across the different flows are. The index is based on the sending rate and can be defined as seen in equation 2, where n is the number of competing flows and

$[x_1,x_2,...,x_n]$ represents each flow.

$$J(x_1, x_2, ..., x_n) = (\sum_{i=1}^{n} x_i)^2 / n \sum_{i=1}^{n} x_i^2 \quad (2)$$

2.2 BBR Overview

Bottleneck Bandwidth and Round trip time was introduced as a congestion control algorithm by Google in 2016 [8]. In contrast to other widely used congestion control algorithms such as Cubic and Reno, BBR continuously takes measurements of the maximum available bandwidth (bottleneck bandwidth) and propagated round trip time to gain an overall understanding of the activity of the network. BBR then uses these measurements to gauge the network congestion. These measurements are taken independently to prevent delay when probing for more bandwidth due to the creation of a queue at the bottleneck. The goal of BBR is to operate at Kleinrock's optimal operating point [34] by adjusting the sending rate and manage the number of in-flight packets to reach high throughput and low congestion in order to keep delay at a minimum. BBR operates on the sender side and does not require any changes to the network itself. It uses specific parameters such as pacing gain and congestion window gain to control the sending rate in the algorithm. The pacing rate is the product of the pacing gain and the estimated bandwidth [22]. Pacing gain values greater than 1 increase the volume of in-flight packets and decreases the packet inter-arrival time, a value less than 1 has the opposite effect [8]. BBR is broken up into four distinct operational states as shown in Figure 2.1.

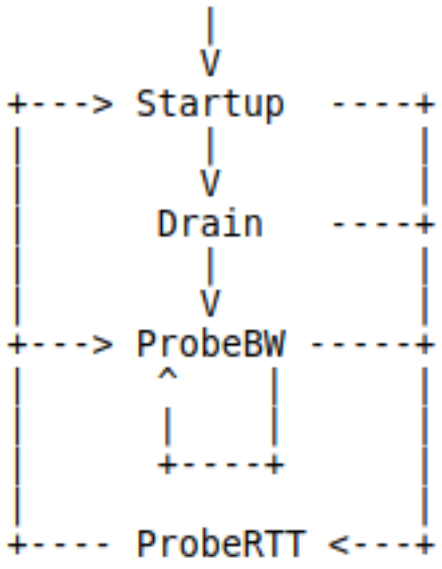

Figure 2.1: BBR State Diagram [1]

2.2.1 Startup Phase

Similar to how the slow start of TCP operates [35] this phase involves an exponential increase in the sending rate where it is doubled every RTT in order to probe the max BtlBw (Bottleneck Bandwidth). A binary search method is used and the pacing gain is set to $\frac{2}{\ln 2}(\sim 2.89)$ in order to allocate this bandwidth probing. An estimate of the maximum BtlBw is deemed to have been achieved when there is no (or very little) increase in the sending rate after 3 RTTs and when the estimated bandwidth is 1.25 times less than the previous bandwidth value. The state switches to Drain once this stage is reached. The expense of this stage is the development of a queue that is twice the BDP (Bandwidth Delay Product) value (that is the propagated RTT x bandwidth).

2.2.2 Drain Phase

In the Drain phase, the goal is to drain the queue that developed during the startup phase of the excessive data that accumulated due to the increased sending rate. The pacing gain is set to the inverse of the value in the startup phase of $\frac{\ln 2}{2}$ and it takes one RTT to drain the queue. The sending rate is also set to 0.75 times the BDP value to stop the same queue build up as in the startup phase. Once the in-flight packets match the BDP value the state is switched to ProbeBW.

2.2.3 ProbeBW

In order to keep up with changing network conditions, BBR uses the ProbeBW phase as a way of keeping the sending rate at a maximum. It does so by increasing the pacing rate to 1.25 the BtlBw (the maximum observed delivery rate during the previous 8 RTTs) for one RTT (the estimated propagated RTT). After which the rate is decreased to 0.75 times the BtlBw in order to drain the queues of excess data. The rate then plateaus for a further 6 RTTs at the normal BtlBw rate. This method of probing bandwidth is known as gain cycling [8] and uses cycles of 8 RTTs with the sequential use of specific pacing gain values. To compensate for the increase in sending rate, the congestion window is set to twice the BDP value to allow for more packets to be sent. BBR spends the majority of its time in this state [8] to allow for high utilisation of bandwidth and throughput.

2.2.4 ProbeRTT

When not in ProbeBW phase, the BBR algorithm spends the rest of its time in the ProbeRTT phase. This phase is entered once there has been no update in the

propagated RTT value during the previous 10 seconds. In order to re-estimate the value the amount of in-flight packets is reduced to 4 per RTT to drain any queue in the intermediate router. The new RTT value is the minimum observed RTT in 200ms plus one RTT of these packets. Upon updating the RTT value the phase is set to either Startup or ProbeBW depending on whether the pipe was filled before entering the state [8].

2.2.5 BBR Variants

Since BBR's release, there has been multiple studies that sought to improve its behaviour by changing certain elements of the algorithm. In this section some of these variants will be described.

2.2.5.1 BBR'

As described before, the ProbeBW phase includes a gain cycling method of updating the pacing gain with a sequence of [1.25,0.75,1,1,1,1,1,1]. In the BBR algorithm there are two different ways of updating the pacing gain between the queue draining cycle (pacing gain = 0.75) and the cruising cycles (pacing gain = 1). The first involves the increase in the cycle offset when the queue draining cycle holds for one RTT, the second is to increase the cycle offset when the in-flight packets are less than or equal to the estimated BDP. The second method suggests that the queue was drained earlier than expected and achieves a lower queue delay. BBR with this second pacing gain update method is referred to as BBR' [22][1]. As seen in line 10 in the Code Listing 2.1 there are the two conditions for which the pacing gain is updated between queue draining cycle and cruise cycle. The variable `prior_inflight` represents the amount of in-flight data before an acknowledgement is processed.

```
1    BBRIsNextCyclePhase():
2       is_full_length = (Now() - BBR.cycle_stamp) > BBR.RTprop
3       if (BBR.pacing_gain == 1)
4         return is_full_length
5       if (BBR.pacing_gain > 1)
6         return is_full_length and
7                   (packets_lost > 0 or
8                    prior_inflight >= BBRInflight(BBR.pacing_gain))
9       else  //  (BBR.pacing_gain < 1)
10        return is_full_length or
11                  prior_inflight <= BBRInflight(1)
```

Code Listing 2.1: BBRIsNextCyclePhase() [1]

2.2.5.2 BBRPlus

BBRPlus is a variation of BBR that was introduced in a blog [23] and follows the method of BBR' in changing the existing ProbeBW attributes of BBR. Similar to BBR', the pacing gain is set to one when the in-flight packets are less than or equal to the estimated BDP, but the main factor of change is the length of the gain cycling. A randomisation of values between 2 and 8 is used to decide the length of the cycles, this combined with the pacing gain optimisation reduces packet loss and improves fairness [23]. Induced packet loss also introduces the queue draining cycle sooner. Line 5 of Code Listing 2.2 shows the randomisation of cycle length where `GAIN_LENGTH` is 8 and `CYCLE_RAND` is 7. The introduction of packet loss as a cycle changing condition can also be seen in Line 13.

```
1    UpdateGainCyclePhase():
2       elapsed = (Now() - BBR.cycle_stamp)
3       if (elapsed > cycle_len * BBR.RTprop):
4         BBR.cycle_stamp = Now()
5         cycle_len = GAIN_LENGTH - rand()%CYCLE_RAND
6         BBR.pacing_gain = 1.25
7       if (BBR.pacing_gain == 1)
8         return
9       if (BBR.pacing_gain < 1 and prior_inflight <= BBRinflight(1)
10        BBR.pacing_gain = 1
11      if (elapsed > BBR.RTprop
12       and prior_inflight > 1.25 * BBRinflight(1)
13       or packets_lost > 0)
14        BBR.pacing_gain = 0.75
```

Code Listing 2.2: UpdateGainCyclePhase() [22]

2.2.5.3 BBR+

The idea around the design of BBR+ originated from an evaluation of TCP over LTE on High-speed Rails (HSR) [24]. It is stated that the BBR algorithm performance is suboptimal in networks with rapidly changing bandwidth and RTT. The changes applied to make up the BBR+ algorithm for a more suitable environment like HSR include the introduction of a different sequence of pacing gains as in a HSR scenario there is a more non stable BtlBw. The sequence is [1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5] which allows for larger fluctuations in the pacing rate. BBR+ also makes changes in how the propagated RTT is changed in the ProbeRTT phase to compensate for the network dynamics of HSR and the fact that there is a conservative nature to the propagated RTT over the last 10 seconds. The formula can be seen below:

$$RTprop = minRTT + \lambda\sqrt{Var(RTT)} \tag{3}$$

It is found that the distribution of different RTT samples is Gamma with a fat tail [24], hence the use of the shape parameter (λ) in the formula for the calculation of propagated RTT. The shape parameter acts as a tunable parameter to trade off between bandwidth and RTT.

2.2.5.4 Delay-BBR

In relation to video streaming, a variant of BBR was introduced to specifically cater for video transmission scenarios [25]. The name given to the algorithm was Delay-BBR. It has the goal of providing lower packet loss rate and lower transmission latency with solid throughput. The authors argue that the undesirable features of BBR that affect any real time video transmission include the probing of bandwidth where there is a vast increase in packets sent with the pacing gain set to 1.25 and decrease in the next cycle at 0.75 which would induce a queue of packets going into the draining cycle that were needed for the previous video frame. Latency increases are also introduced as the sending rate often exceeds the bottleneck capacity and the ProbeRTT phase only sends 4 packets.

To tackle these issues it was proposed that the pacing gain's sequence would change to [1.11,0.9,1,1,1,1,1,1] for the gain cycle to give some stability for the consistent delivery of video frames. The ProbeRTT phase is re-purposed to allow for excess buffered packets to be drained and the frequency of entering this phase is drastically increased meaning the state will be entered not just after the 10 second interval of updating the propagated RTT. The pseudo code in A2.1 - A2.5 offers a look at the changes made including the use of a smoothed RTT signal which allows for the draining of buffers once it reaches a certain threshold. Congestion is detected through the smoothed RTT signal becoming larger than the product of the base RTT (during the ProbeBW phase) and a Beta value. When in the ProbeRTT state the smoother RTT signal is reset and only updated when a sequence number of an acknowledged packet is larger than the sequence number of the last sent packet [25].

2.2.5.5 BBRv2

BBRv2 is Google's ongoing research attempt at making BBR more robust and fair across multiple flows [2][26][27]. The issues with BBR that BBRv2 looks to solve is the low throughput when sharing with Reno or Cubic flows at the bottleneck, the high packet loss rates if there is a bottleneck queue less than 1.5 times the BDP, aggregated flows that result in low throughput and a varied congestion window causing low throughput in ProbeRTT phase. BBRv2 adopts a DCTCP (Data Centre TCP) inspired ECN (Explicit Congestion Notification) which has a main purpose of indicating congestion without dropping packets. It should be noted that general use of ECN in

TCP is optional [36]. BBRv2 introduces changes to the general BBR flow cycle to consider actual packet loss. Upper and lower bounds of the in-flight rate are introduced as part of a three part model to represent the bounds between which the compiled congestion signals reside. These include delivery, loss and ECN [2]. The lower bound for bytes in flight is calculated as per equation 4 when an acknowledgement is received with a loss indication, β is 0.3 [2].

$$inflight\_lo = max(inflight, (1 - \beta) * inflight\_lo \tag{4}$$

The network path model is shown in Figure 2.2, this displays the points in the execution of the algorithm where certain metrics are set. In the startup state the upper bound for in-flight rate is used to estimate the maximum in-flight rate when the loss rate is too high and acts as an exit clause when it is set. The ProbeBW state is typically broken up into probe up, probe down and probe cruise phases. These are altered from their typical behaviour in BBR to provide a more steady variance in sending rate with dependence on the lower and upper bounds for conversion between phases.

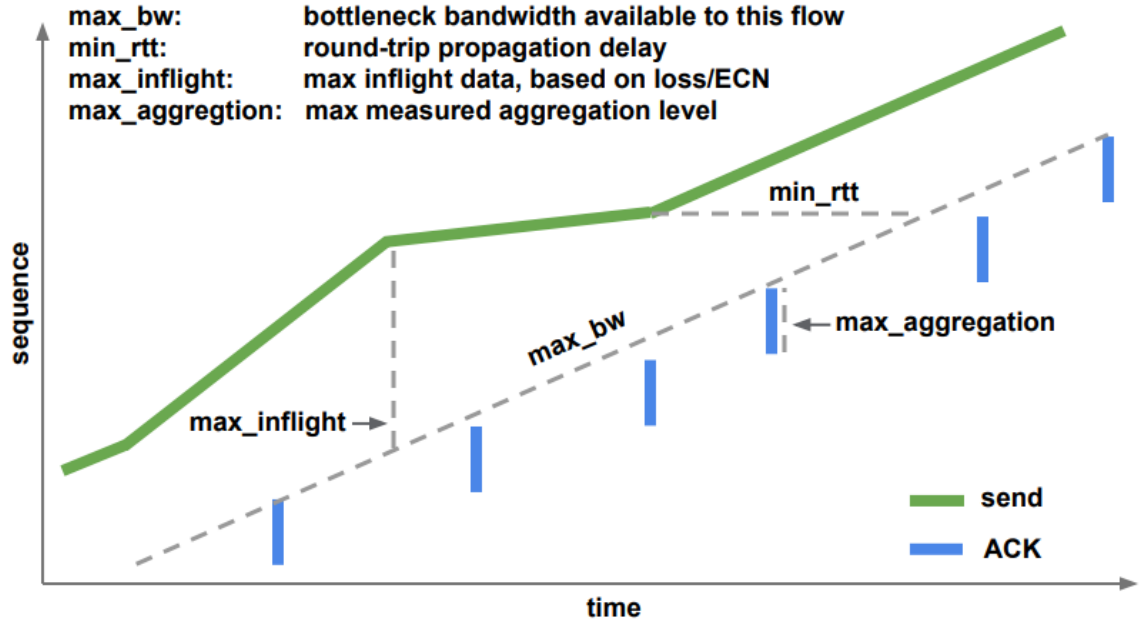

Figure 2.2: BBRv2 Network Path Model [2]

The probe cruise phase has interval adaption per RTT to update the bandwidth and lower bound for in-flight rate using a combination of the loss and ECN signals. The in-flight rate starts at the estimated BDP and, in order to avoid approach to the upper bound, headroom is left as per equation 5 where headroom = 0.15. A probe refill phase is initiated for one RTT to prepare for the probe up phase, the pipe is filled but the queue is not. Probe up is the priority of the ProbeBW state as it is where any unutilised bandwidth is sought. For BBRv2, the probe up phase grows exponentially

after each RTT to utilise any extra bandwidth and it ceases probing when the estimated queue is 1.25 times the estimated BDP or the value of upper in-flight rate bound is set. The upper bound for bytes in flight is set when the loss and ECN rate is too high. The probe down phase takes up its original role of draining any excess queue built up by the probe up phase. In order to reach its goal for a reduction in throughput variation the ProbeRTT state is changed to reduce the congestion window to 50% of the BDP instead of only 4 packets.

$$inflight = (1 - headroom) * inflight\_hi \tag{5}$$

2.3 Related Work In BBR Evaluation

Since its release in 2016, BBR has been carefully studied to assess the veracity of the claims made about it by the varying of different network dynamics. The validation of claims made in the original development of the algorithm are amongst the most popular types of evaluations where comparisons were made to other congestion control algorithms and assessments of its performance in variations of situations were considered. From the literature it can be seen that BBR suffers from a number of problems [9][37], the most notable being in situations where there are multiple flows of both BBR and non-BBR congestion control algorithms. BBR suffers from an RTT unfairness issue which comes from the algorithms inability to give a fair share of the bandwidth when flows exist with different RTTs, with BBR favouring bandwidth allocation to the flow with the highest RTT.

The algorithm also tends to overload the bottleneck link with coexisting flows and buffers that are under the estimated BDP value (shallow buffer) resulting in massive loss which goes undetected by BBR. The sharing of bandwidth amongst flows with different congestion control algorithms has also resulted in poor performance [38][10][39] [40] especially with loss-based congestion control algorithms like Cubic. Buffer Size makes a difference in the behaviour with these other congestion control algorithms due to BBR's dependence on the BDP value.

Buffers that are greater than the estimated BDP value show cyclic behaviour with Cubic with the throughput typically varying from high to low for the competing flows, this is because BBR has an in-flight cap that is used once a fair estimation of the throughput and propagated RTT is found causing a fixed rate of packets at the bottleneck queue, Cubic takes advantage of this to expand its congestion window and causing BBR to reduce its congestion window due to a lower observed throughput at the bottleneck router. ProbeRTT takes hold for BBR every 10 seconds and when a

competing flow like Cubic is present, the queue that would supposedly be drained will stay full due to this Cubic flow and hence cause a large re-estimate of the propagated RTT and thus the BDP value. Consequentially, a large increase in congestion window occurs causing high loss that goes undetected by BBR but picked up by Cubic which in turn backs off, reducing its congestion window. This cyclic behaviour is not ideal for networks, especially ones dependent on consistent goodput, latency and jitter.[39]

Many evaluations use custom topologies with realistic bandwidths to simulate a typical network. The ability to dynamically change the bandwidth and RTT of the bottleneck link allows for the characteristics of BDP to be changed, which in turn impacts on the behaviour of BBR. In order to evaluate the behaviour of the multiple variations of BBR mentioned in this paper, a decision would have to be made on the most appropriate topology and network conditions to test under.

Multiple studies have used a typical dumbbell topology over a simulation of a wired network using tools such as NS-3 or Mininet [22][38][4] but only a limited number of studies have been done on varying types of network. Jinting Lin performed an evaluation of BBR over varied network conditions including LAN (Local Area Networks), WAN (Wide Area Networks) and WLAN (Wireless Local Area Networks) [41]. This showed that BBR does not perform well with wireless networks, in particular, with a low rate of goodput and fairness towards other competing congestion control algorithms. WANs also show poor performance with the ongoing issue of high losses and retransmissions despite having reasonable goodput.

An alternate evaluation approach is to use different content providers as test beds to see how BBR operates in a real world scenario [42]. The behaviour was as expected, as per recent declarations, with poor fairness in bandwidth allocation among different flows and with content providers with conflicting congestion control algorithms. Buffer size proved optimal in deciding BBRs dominance amongst content providers. This, along with the methodologies of the other evaluations, would influence the decision to opt for evaluation by simulation in a wired network where dynamic changes to network attributes such as bandwidth, bottleneck bandwidth and propagation time are possible in order to fully capitalise on BBR's functionality.

The existence of many flaws in BBR's approach to congestion control caused an influx of suggestive changes to solve them, with common approaches targeting one or two specific flaws. The effect of loss was a significant factor identified in the extensive initial evaluation of BBR and some suggestions to solving this issue include a decrease in congestion window when packet loss is above a certain threshold and when the measured RTT is less than the propagated RTT [39]. Other changes sees the introduction of an additional signal to detect loss before it happens [25]. RTT

unfairness is another important flaw that can be tackled by reducing congestion window growth based on the last measured RTT and the propagated RTT to reduce the creation of long queues [43] at the bottleneck.

BBRv2 takes all of these flaws into account with the aims to fix them all [2]. Research and testing with real traffic (YouTube) is ongoing with BBRv2 and the latest release has yet to be evaluated and validated to have solved BBR's problems by sources other than Google. Whilst the evaluations done by its developers proved promising [26], the limited external evaluation provided the initial motivation for the evaluation performed in this report. As of writing, November 2019 was the latest release of information about BBRv2 [27]. A paper by Zhang et al. [22] is one that evaluates an earlier release of BBRv2 alongside different variants of BBR including BBR' [22], BBR+ [24] and BBRPlus [23]. This evaluation found comprehensive improvements in fairness with other congestion control algorithms and consistent performance across all other metrics including elimination of RTT unfairness. This evaluation hopes to achieve similar results with the latest release of BBRv2.

2.4 Network Simulation

Congestion Control methods like BBR need to be tested in order to handle varying changes to network dynamics and to ensure robustness before using real world traffic. Network simulators are analytical modelling tools used in research and development for the interpretation of network behaviour and performance. These simulators can be used to study the current behaviour of a network or predict the outcome of a change in the network by testing new protocols. The difficulty behind analysing network behaviour with the increase in network scale and the development of new technology is made easier through the use of network simulators. Experimental methods were the typical forms of testing network behaviour with the use of a developed test bed with the required network characteristics to carry out research on the network environment, but as networks got increasingly more complex with the mix of wireless and wired architectures, the ability to use experimental methods would prove costly and too strict to its predefined characteristics. These analytical models can also prove expensive in terms of the consumption of energy, memory, processing power and most notably time, especially with larger scale network models. The goals of network simulation can be seen as [44]:

- Predict the performance of current networks and protocols in order to aid technology assessment and capacity planning and to demonstrate fulfilment of customer goals.

- Predict the expected behaviour of new network protocols and designs through qualitative or quantitative estimates of performance or correctness.
- Explore a range of potential protocol designs through rapid evaluation and iteration.

The application of network simulators can be used across various fields of work in both research and development of new network technologies or the validation of current technologies. In the process of software development, network simulators can be used as part of the workflow to enable the correctness of developed work in the necessary network environment. Successful network simulations of a specified network behaviour offer confidence in that protocol operating as expected in a real world specification.

In order to gauge the effectiveness of network simulations and all its software distributions, the results that are produced from a simulation model must be interpreted correctly. Validation is the process of assuring that a model provides meaningful answers to the question being investigated [44]. This is a necessity in terms of providing clarification to the results of a simulation. Grades of validation are based on the questions being asked and the systems being used with levels of abstraction judged from the interpretation of the situation to be simulated. High level of detail in certain networks will require a more complex model of simulation with a look into the effects of the simulation on all aspects of the network through all its corresponding OSI layers.

A comparison between a network simulation of a specified protocol versus a real world implementation of the same protocol is the simplest means of validating the simulation but is very restrictive to the types of protocols that can be compared due the ability to create these protocols in a larger scaled network. Typically network protocols like for example, TCP, have specialised design decisions that in certain cases are embodied by the simulation model and only represent a specific "version" of that protocol. Also with all its different implementations, TCP may have to be validated through simulation against different implementations and any future changes that may impact it. BBR is a prime example of one of those changes.

A robust methodology to validate a network simulation of a specific protocol is to have an established baseline truth about that protocol to base any varied specification comparison against. The use of specified metrics can be applied to each simulation and presented in visualisation or graph plots in order to prove any hypotheses and validate the simulation for the protocol in question. In the case of BBR, statistical measures such as sent and received packets, throughput and round trip time are just a few of the metrics can be used in the appropriate simulation comparisons.

Network Simulation can be broken up based on different techniques of execution. These include; Discrete event simulation, Parallel discrete event simulation and Ultra large scale simulation framework (USSF) [45]. Discrete event simulation models the operation of the system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system [46]. This stateful method of simulation provides an abstract style for which a network simulator emulates the network layers and allows for simple interpretation of the simulators structure.

Parallel discrete event simulation is discussed by R. M. Jujimoto [47] and involves the use of parallelism in the context of discrete event simulation. This allows for the distribution of computational load when simulating larger networks and a general increase in performance of network simulators. Parallelism works well for this type of event driven system as long as systems follow certain sequencing constraints that prevents causality errors which are a result of the parallel execution taking an event at a timestamp that is larger than another that is executing and the larger of the two inflicting a change of state whilst the other is still executing. There are two different mechanisms for parallel discrete event simulation: (1) Conservative, which strictly approaches causality errors and has a method in place to determine if it is safe to execute an event; (2) Optimistic which takes a detection and recovery approach to causality errors with a rollback method to recover from any errors [47].

For larger scaled systems that would usually require very long simulation times, Ultra large scale simulation framework (USSF) was developed. USSF is built on top of two parallel discrete event simulators, WARPED [48] and NOTIME [49]. The framework operates by reusing defined components and replicates the modelling constructs that make up the simulation in order to reduce the size and consumption of the simulation, allowing for a large amount of network nodes in a simulation. Efficient use of memory management is employed to improve performance and the features of USSF can be easily exploited on top of a discrete event simulator requiring only minor changes to the application modules [50].

There are a number of different network simulator software that utilise these techniques, the table below presents a look at a few examples with their unique features.

Table 2.2: Network Simulators Overview

Simulator	Method	Feature
OPNET	(Parallel) Discrete event	Visual/GUI support, object-oriented, large scalability [51]
OMNET++	Discrete event	GUI support, component based (modular) analysis [52]
GloMoSiM	(Parallel) Discrete event	Large scale wireless networks, library-based [53]
QualNet	(Parallel) Discrete event	Commercialised GloMoSim, protocol design environment [54][55]
NetSim	(Stochastic) Discrete event	Development environment, feature rich (layer abstraction) [56][55]
NS-2	Discrete event	Object-oriented (TCL), protocol simulation in virtual time [57]
NS-3	(Parallel) Discrete event	Numerous networks types and protocol detail emphasis [58]

2.5 NS-3

Comparisons between network simulators have been carried out using various different metrics, most notably memory usage, CPU utilisation, scalability and computation time which is presented in a study by Khana et al. [59]. The results show that NS-3 outperforms other simulators such as OMNET++, GloMoSim and NS-2 in all the proposed metrics. This along with the fact NS-3 has a large library of modules, is open source and is very well documented is among the reasons for using it in the evaluation of BBRv2. In this section the operation of NS-3 will be discussed in detail.

As mentioned before, NS-3 is a discrete event simulator with the evolution of a network system modelled through discrete events in time. NS-3 also has the capabilities to use parallelism to increase overall performance of simulations. It is written in C++ but also offers the ability to use Python bindings to write modules and simulator scripts. It was intended as a replacement for NS-2 with a complete rewrite of the core modules and the scrap of O-TCL (Objective Tool Command Language) as the topology design language.

The modules included in NS-3 for network simulation include numerous network devices such as CSMA, LTE and WiFi, internet protocols such as TCP, UDP and IPv4/v6 and routing protocols such as AODV and DSDV [60]. What separates NS-3 from its other simulator counterparts is its attention to realism in the implementation of all its modules. The flow and operation of the majority of its modules would be very similar to the behaviour in a real world kernel implementation.

Given its open source nature, NS-3 allows for contributions to the expansion of its modules making it up to speed with the latest protocols available. The tool has maintainers that approve contributions and look after fixing any bugs that might be discovered by its users. NS-3's large library allows for contributions to be made simple as long as their strict coding style is adhered to [61].

There is a high level of flexibility when combining modules together to form the desired topology and configuration for the network to be simulated. NS-3 includes visualisers and animators to visualise networks in operation of dynamically change elements in a simulation. Tracing is available in the form of pcap files to observe per packet level interactions and other traces can be set up to observe desired metrics of a network.

A typical simulation setup would consist of setting up a network topology first, instantiating any of the network nodes and devices needed on the network and assigning them the desired protocols. Application models are then produced to mimic the creation and transportation of packets across the network setup. The simulation is given a desired time to run and ceases at a user defined time also. Figure 2.3 shows the typical flow of a simulation.

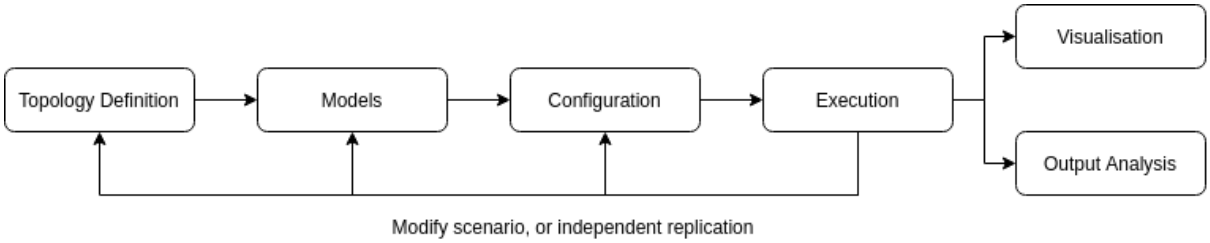

Figure 2.3: NS-3 Simulation Structure [3]

2.6 Congestion Control In NS-3

As part of its internet module, NS-3 has a TCP model to allow for bidirectional communication using TCP, the native implementation in NS-3 supports many different congestion control algorithms and other implementations of TCP. The attributes of TCP are hosted in the `TcpSocket` class which can be reused across implementations. The implementation of TCP greatly resembles the Linux approach by modularising each congestion control implementation into separate classes and having an interface to interact between the socket classes and congestion methods [62].

The TCP state machine is managed along with all the other features such as Selective Acknowledgements (SACK), Fast Retransmit and Recovery in the `TcpSocketBase` class. Congestion control algorithms are implemented as their own classes and inherit the abstract class `TcpCongestionOps` in order to make their custom changes to mimic the specialised behaviours of the congestion control algorithm. The methods that exist in the `TcpCongestionOps` class are matches to the Linux implementation. These include [63]:

- `PktsAcked`: When a packet is acknowledged this method is called and contains relevant timing information including round trip time and the number of segments

acknowledged.

- `IncreaseWindow`: When new segments are acknowledged this method updates the cwnd appropriately.
- `Congestion State Set`: Before the congestion state is changed, this method is triggered to perform different events on certain congestion states, these events vary for different congestion control algorithms.
- `CwndEvent`: This method is called to perform events when there is a cwnd event. These events could include delayed acknowledgements or ECN notifications.
- `GetSsthresh`: When there is a loss event this method retrieves the slow start threshold in order for TCP to recover from the loss event if required.

2.7 Background Summary

Congestion control evaluation is a widely explored topic through academia and research due to the changing nature of the internet and demand for consistent performance. BBR as a recent introduction into the area of congestion control gives opportunity to explore ways of increasing general internet performance from a TCP standpoint and as mentioned before has been made evident in the number of evaluations and proposed variations that look to improve the congestion control algorithm.

The introduction of BBRv2 was discussed in this chapter and as the newest official form of BBR it leaves the ability to explore its performance and features. The manner in which so will be through the use of NS-3 which provides the sufficient architecture to carry out such experiments as explained previously.

3 Implementation

In this chapter the detail of the design and implementation of the simulations required to evaluate BBR, BBRv2 and the other variations in NS-3 will be discussed. All of the following implementation is available on Github with sufficient instructions on how to reproduce simulations[1].

3.1 Overview

Developing on the knowledge received from the research studied in the background section, the approach to this implementation was decided. The process for evaluating congestion control algorithms in NS-3 involves specifying the type of congestion control to use when creating the executable simulation script and following with the desired network setup and characteristics. NS-3 has a large variety of congestion controls to choose from but unfortunately does not have TCP-BBR implemented onto its TCP stack. In order to carry out the desired evaluation, NS-3 has to be extended to include BBR, BBRv2 and all the other variants in its TCP implementation. Adding a congestion control is something that NS-3 does allow and supports through its documentation [62]. Fortunately, Vivek Jain et al. implements BBR in NS-3 using version 3.27 [4]. The implementation follows the NS-3 guidelines in terms of approach to realism and coding style making it an appropriate NS-3 congestion control with the aim of having it merged onto the official NS-3 development distribution. The implementation outlined in this dissertation uses Jain's implementation to include BBRv2 and the other variants discussed in the background in NS-3.

The approach at the development of BBRv2 and the other BBR variants in NS-3 adopts a software engineering methodology in order to allow for separation between variants and flexibility of use. This is done by using Git [64] and Github [65]. The full executable NS-3 environment is placed in a repository and for every BBR variation implemented a separate branch is used and merged onto the master branch after a pull request is self reviewed to ensure code compliance with NS-3 coding standard and

[1] https://github.com/lanigan23/BBRv2-Eval-ns-3

readability. This is to allow for consistency across implementation and follow the realism element that NS-3 adopts to match the corresponding Linux implementations of the congestion control.

3.2 BBR In NS-3

The implementation of BBR in NS-3 involves using the modular architecture of NS-3 and introducing BBR as a class of its own that uses the interface to interact between the relevant TCP classes and methods. As discussed in the background the `TcpCongestionOps` class contains all the overridable methods that BBR needs to inherit in order for it to operate. The layout of the interactions between the classes can be seen in Figure 3.1. `TcpSocketState` is encapsulated by the `TcpSocketBase` class and represents the state of the TCP socket.

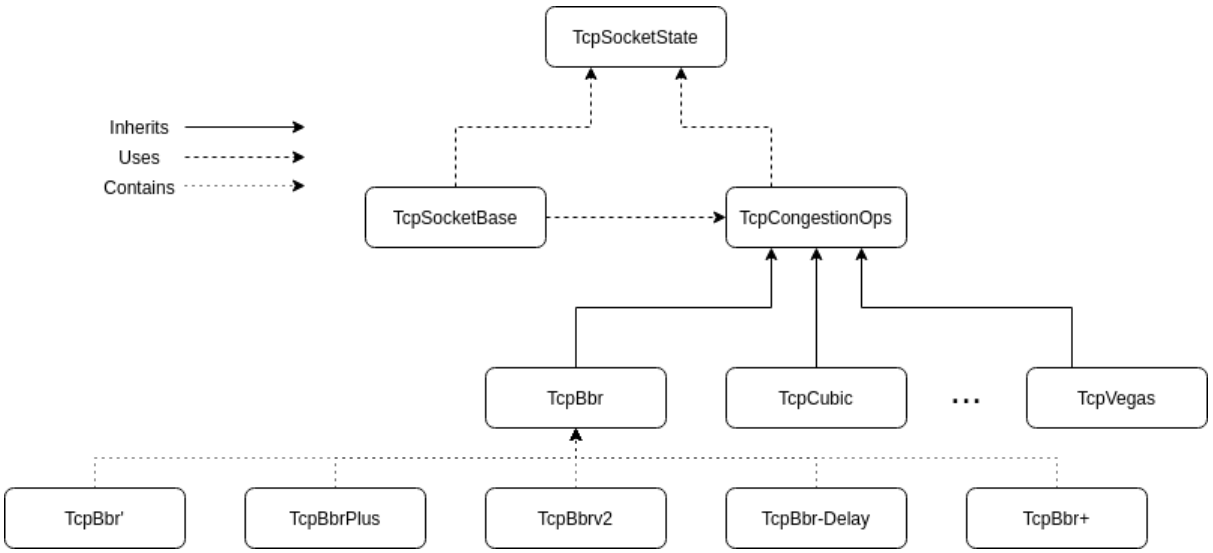

Figure 3.1: BBR Class Interaction

The Linux implementation of BBR contains a function called `cong_control` [66] which upon the acknowledgement of a packet, is used to update the control parameters of BBR including congestion window and pacing rate from the estimated delivery rate. It is also solely responsible for instigation of the stateful functionality of BBR, updating the propagated RTT, bottleneck bandwidth and then the BBR state based on these updates. Unfortunately NS-3 has no such implementation of this function and the `TcpCongestionOps` class needed to be extended to include it. This function replaces the need to use the `IncreaseWindow` function to update the congestion window, a Boolean function `HasCongControl` is used to clarify if the current congestion control is using the `CongControl` function.

3.2.1 Rate Sampling

Another modification that had to be introduced into the TCP flow of NS-3 was the inclusion of delivery rate estimation for BBR. Neal Cardwell et al. (the team behind the introduction of BBR) implements a specialised algorithm to carry out such estimation that contributes particularly to the operation of BBR [67]. A delivery rate sample records the estimated rate at which the network delivered packets for a single flow, calculated over the time interval between the transmission of a data packet and the acknowledgement of that packet.

The delivery rate estimate is used in the calculation of the pacing rate and in turn, the BDP for BBR. The algorithm used to produce the delivery rate does so by using per-connection and per-packet state measurements. Per-connection state information is taken for each connection and consists of the amount of data delivered over the lifetime of the connection as well as the time in which it was last updated. It also contains an indicator if the connection is application limited and a variable to hold either the send time of the packet last marked as delivered if the packets are in flight or the send time of the most recently sent packet if the connection is idle. In NS-3 the `TcpSocketState` class holds this connection information.

When a packet is transmitted or retransmitted the per-packet information is recorded as the same information held by the per-connection state just at the time of the packet being sent. The time at which the packet was sent is also recorded among the per-packet state variables. When an acknowledgement is received a rate sample object is filled based on the previously gathered per-connection and per-packet information. This rate sample object holds the information that is used in BBR calculations. If the packet was newly acknowledged then the existing rate sample is updated with the per-packet information from the time at which the packet was last transmitted.

The delivery rate is calculated as by equation 1 where the *delivered* member is the amount of data delivered over the lifetime of the connection stored in the per-connection state minus the amount delivered from the most recent delivered packet. The *interval* is the maximum value between the send time interval from the most recent delivered packet and the acknowledge time from the most recent delivered packet [67]. These calculations are performed in the `TcpSocketBase` class. In addition to storing this information about the delivery rate, the rate sample object implemented in NS-3 also stores the amount of bytes lost in flight as a packet loss indicator.

$$deliveryRate = delivered / interval \tag{1}$$

3.2.2 Windowed Min-Max Filter

Keeping track of delivery rate estimates is necessary for updating BDP values in BBR especially when it is continuously updated. A windowed min-max filter is used to keep track of these estimates in the Linux implementation of BBR [66]. A windowed filter operates in general by keeping track of three of the estimates of a certain type over a specified window length using a min or max comparison. Every time the filter needs to be updated with a new estimate it checks the previous three estimates against the new estimate using the min or max comparison and updates the stored estimates relevantly. If the window length has expired, the best estimate expires and the second best and third best estimate get promoted to first and second best.

In addition to this, at the quarter point and half way point of the window length, the second best estimate is taken from the second quarter of the window and the third best estimate is taken from the second half of the window respectively. In the case of BBR, the type of value that is the estimate is the delivery rate, the comparison used is the maximum of the rates and the window length is the number of round trips in which the default value is 10 [66]. The implementation of this in NS-3 is taken from a Chromium implementation that is adapted for NS-3 [4][68].

3.2.3 BBR Model

The `TcpBbr` class that inherits `TcpCongestionOps` implements the appropriate methods to follow the state functionality of the algorithm. The corresponding `Enter` and `Exit` methods exist for each of the Startup, Drain, ProbeBW and ProbeRTT phases. The behaviour of these phases is initiated in each of these methods through the use of specific functions for each phase. For example, `AdvanceCyclePhase` is responsible for the gain cycling functionality of adjusting the pacing rate over a fixed number of cycles in order to probe for more bandwidth in ProbeBW. `HandleProbeRTT` maintains the ProbeRTT phase for 200ms plus one RTT whilst a new propagated RTT is being estimated.

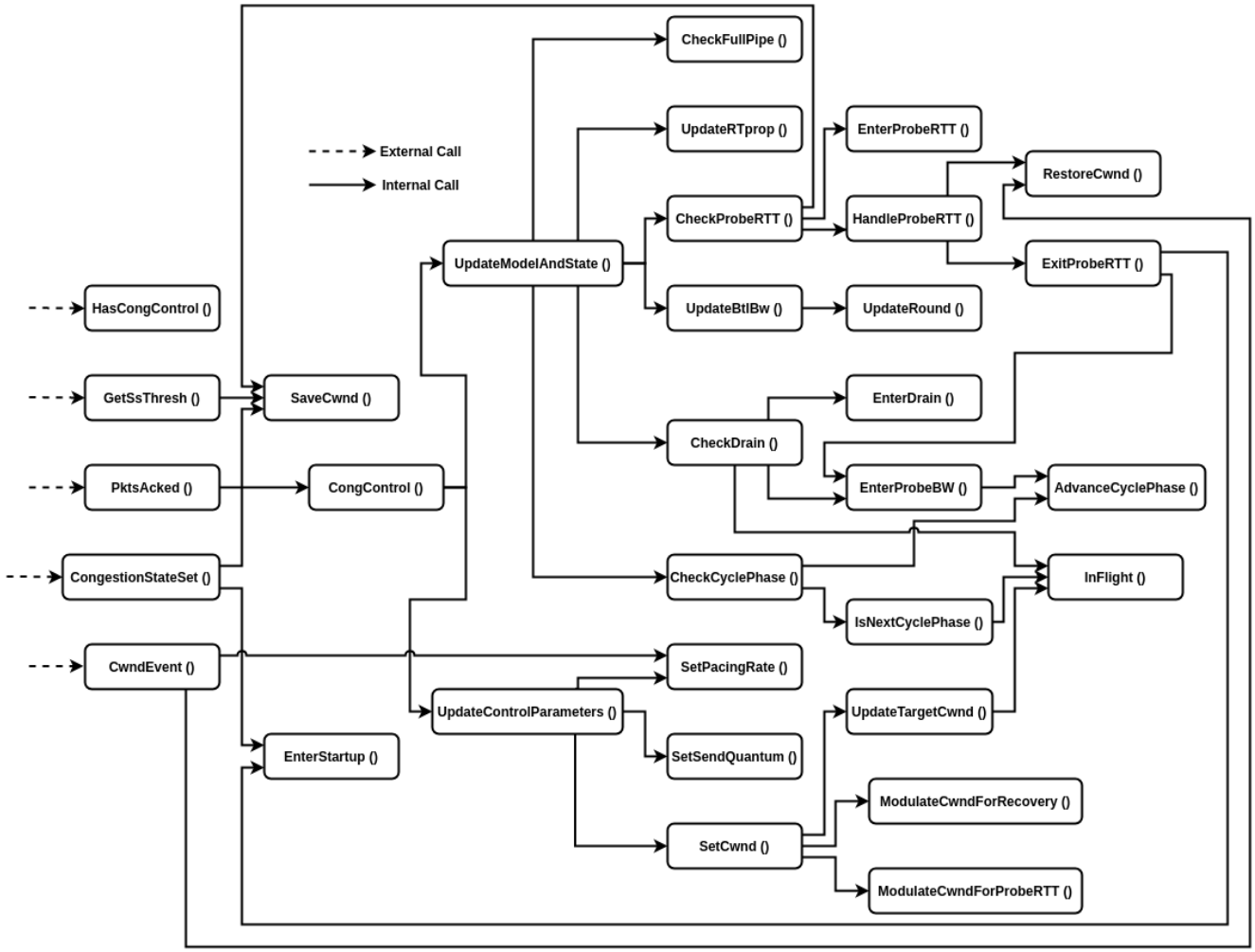

Figure 3.2: BBR Function Graph [4]

A graph for the flow of each BBR specific method can be seen in Figure 3.2, it can be seen that the flow can be separated in to two main functions. The `UpdateModelAndState` method is for updating the BDP of BBR and the instigation of the BBR phases whilst the `UpdateControlParameters` method maintains the pacing rate and congestion window. As made clear by Jain et al. [4] the send quantum functionality of BBR is not supported in the implementation as NS-3 does not have segmentation offloading available.

BBR in NS-3 overrides a number of methods in `TcpCongestionOps` in order to tailor for its specialised behaviour. The `GetSshThresh` method which usually returns the value of the slow start threshold after a loss event but in the case of the BBR implementation returns a maximum integer value as the slow start threshold value does not apply to the functionality of the algorithm. As a reactive method, `CwndEvent` typically gets called when a particular event occurs that affects the congestion window. BBR overrides this function to restore the congestion window to its prior value after a congestion recovery

event. The congestion state machine is an important component to represent the situation of a TCP socket from the point of view of the congestion control. There are different states that represent different congestion situations and the actions that have to be taken based on these situations.

`CongestionStateSet` is the function that decides on the actions to be taken and is overridden by the BBR class to perform BBR specific behaviours on these congestion events. `CA_OPEN` represents the normal state with no congestion events and acts as a point of entry for BBR into the Startup phase, initialising all appropriate RTT and congestion window measurements. `CA_LOSS` is an event based on the timeout of the retransmission timer which can indicate loss. The behaviour of BBR in this event is to save the value of the congestion window at the time of this loss in order to restore the congestion window when it is safe to do so and restart the round. `CA_RECOVERY` is triggered when there are Selective Acknowledgements (SACKs) or duplicate acknowledgements and these type of acknowledgements reach a certain threshold or when loss is indicated which initiates fast retransmission of packets. BBR saves the value of the congestion window at the time of this event and then adjusts the congestion window accordingly to cater for the retransmission of packets.

3.3 BBRv2 And Variants In NS-3

The approach taken to implement the version two of BBR and its variants outlined in the background into NS-3 was to contain each variant within the BBR class mainly because the functionality of each variant does not change from the overall stateful behaviour of BBR. This allows for a global attribute to be used to decide between what variation to set at simulation time and gives a clear distinction between what variation is in use. This section describes the necessary changes needed in the BBR and TCP classes to implement BBRv2 and the other variations.

3.3.1 BBR'

The adaption of the BBR' variant involves the lesser changes out of all the variations since it only involves the separation between the two conditions for which the pacing gain cycle offset in ProbeBW is updated. The condition that distinguishes the variation to be BBR' is updating the offset when the prior bytes in flight are less than or equal to the current in flight bytes rather than updating the offset after one round trip. This change is applied to the `IsNextCyclePhase` function which is responsible for checking if it is time to advance the pacing gain cycle.

3.3.2 BBR+

There are two major changes required in the behaviour of BBR to be made in order to match the BBR+ variation used in the study of TCP over LTE in a HSR scenario [24]. Firstly, the values for the pacing gain cycle were updated to those used in the study. The sequence changes from [1.25, 0.75, 1, 1, 1, 1, 1, 1] to [1.5, 0.5, 1.5, 0.5, 1.5, 0.5, 1.5, 0.5] with the pacing gain being set to these values during ProbeBW when the variation is set to BBR+. Secondly, the way the propagated RTT is calculated needed to be changed based on the variance of the RTT and a shape parameter lambda to make the distribution of RTT samples a Gamma one. The lambda value is made to be a tunable attribute of the BBR class that could be set at simulation time.

Before the implementation of BBR+, the RTT information estimated in the TCP socket classes were only the minimum RTT and most recent RTT through the use of the `RTTEstimator` class which estimates and stores different information about the RTT samples. Fortunately, this class also stores information on the variance of the RTT samples which is needed in the calculation of the propagated RTT in BBR+. The `TcpSocketBase` class is updated to take RTT variance as a measurement as well as the minimum RTT and most recent RTT. Line 11 in the Code Listing 3.1 shows the use of the RTT variance in the calculation of the propagated RTT based off equation 3 outlined in the background.

```
1  void
2  TcpBbr::UpdateRTprop (Ptr<TcpSocketState> tcb)
3  {
4    NS_LOG_FUNCTION (this << tcb);
5    m_rtPropExpired = Simulator::Now () > m_rtPropStamp +
       m_rtPropFilterLen);
6    if (tcb->m_lastRtt >= Seconds (0) && (tcb->m_lastRtt <= m_rtProp ||
       m_rtPropExpired))
7      {
8        if (m_variant == Bbr::BBR_HSR)
9          {
10           double rttVar = sqrt((tcb->m_rttVar).GetDouble ());
11           m_rtProp = tcb->m_lastRtt + m_lambda * MilliSeconds (rttVar);
12         }
13       else
14         {
15           m_rtProp = tcb->m_lastRtt;
16         }
17       m_rtPropStamp = Simulator::Now ();
18     }
19 }
```

Code Listing 3.1: UpdateRTprop()

3.3.3 BBRPlus

The BBRPlus variant is implemented as a patch of the Linux C file of BBR in the blog in which it was introduced [23]. The underlying functional changes in the Linux file had to be adapted for NS-3 in its native C++ language. In order to capture the behaviour of the BBRPlus variant of BBR in NS-3, changes to the behaviour of the pacing gain cycle had to be made. BBRPlus changes the assignment of the pacing gain from a sequential action to a non-sequential one through the `SetCycleIndex` method. This allows for the probe up, probe down and probe cruise phases of ProbeBW to be dynamically changed based on certain conditions. The function `DrainToTargetCycling` replaces the `AdvanceCyclePhase` flow in the case of BBRPlus to carry out these assignments of pacing gain.

The other major change to the activity of the pacing gain cycle was the randomisation of the length of the cycle between the values of two to eight cycles. This length is set in the initial probe up phase of ProbeBW. This can result in a variance of the amount of packets to be sent in a round trip. A different approach to how the probe down phase of ProbeBW is entered is also implemented in the `DrainToTargetCycling` method where any packet loss detected in the rate sample causes the pacing gain cycle to enter probe down phase as a condition as well as the expiry of the cycle stamp or the prior bytes in flight being less than current bytes in flight.

3.3.4 Delay-BBR

As a variant of BBR tailored for the purpose of video transmission [25], Delay-BBR offers significant changes to the operation of BBR in NS-3 needed to emulate its behaviour. The first is the change in the pacing gain values to probe more conservatively to allow video frames to be less broken up upon a queue forming at the bottleneck. The sequence is changed to [1.11, 0.9, 1, 1, 1, 1, 1, 1] with these values assigned to the pacing gain when the variant is set to Delay-BBR.

$$srtt > (baseRtt * beta) \tag{2}$$

Secondly the introduction of a congestion delay signal involving a new smoothed RTT signal value and its estimation was a new component as a part of the implementation of Delay-BBR in NS-3. This new component of the algorithm updates the overall operation of BBR in NS-3 with the purpose of entering the ProbeRTT phase more frequently. The `CheckCongestionDelay` function is responsible for setting this new congestion signal during ProbeBW phase. The condition for which it is set is based off equation 2 where the $srtt$ is the smoothed RTT, the $baseRTT$ is the base line RTT during ProbeBW and $beta$ is a value of 1.2 as outlined in the implementation by Zhang

et al. [25]. When this congestion signal is set, the phase will change to ProbeRTT which is instigated by `CheckProbeRTT` where it checks if the congestion delay signal is set. The process for updating the smoothed RTT signal takes place in the `CheckCongestionDelay` method also as per equation 3 where *alpha* is a fixed value of 0.9 [25], *oldSrtt* is the previous stored smoothed RTT signal and *newRTT* is the most recent RTT. The base line RTT is updated with the value of the smoothed RTT when it is less than the current base line.

$$newSrtt = (1 - alpha) * oldSrtt + alpha * newRtt \tag{3}$$

The smoothed RTT signal is only updated on the condition that the sequence number of the last acknowledged packet is greater than the sequence number of the most recent sent packet. Unfortunately only the sequence number of the last acknowledged packet is recorded by the TCP socket classes. In order to overcome this, the `TcpSocketBase` class needed to be modified to record the sequence number of the most recently sent packet. Fortunately, A buffer is used to store the sequence numbers of the most recently sent packets, so the head of this buffer is retrieved and stored once a packet is sent. Upon setting the congestion delay signal in `CheckCongestionDelay`, the sequence number of the most recently sent packet is set to be used in comparison with the sequence number of the last acknowledged packet. The last changes that needed to be made to make Delay-BBR compatible was the adjusting of the congestion window during ProbeRTT to having a capacity based off of 0.75 times the current BDP value instead of limiting it to just 4 packets for the duration of the ProbeRTT phase. The smoothed RTT signal value is reset when ProbeRTT has concluded.

3.3.5 BBRv2

As the integral part of this evaluation, BBRv2 involves the most changes to the BBR workflow in NS-3. BBRv2 keeps the core principles of BBR in its stateful manner but adds a lot more characteristics that makes its distinction from the original algorithm. There are two different versions of BBRv2 that are implemented, both are based off the Linux kernel code from the official repository on Github[2]. These consist of the alpha release and the alpha release with experimental changes. BBRv2 is implemented on top of the original BBR code base with the new functions filtered out at the bottom of the source code in the kernel implementation. This style is followed in the NS-3 implementation of the congestion control algorithm. The explanation of the adaption of BBRv2 on top of BBR will be broken up into appropriate sections representing the changes needed to the TCP socket classes and the corresponding BBR states. An

[2]https://github.com/google/bbr

updated function call diagram can be seen in Figure A3.7 showing the additional BBRv2 methods added to the flow.

3.3.5.1 Explicit Congestion Notification Signal

BBRv2 adds the use of an Explicit Congestion Notification Signal which enables the ability to have another congestion condition to adjust the network model appropriately to. The basic operation of ECN entails having flags in the IP header and TCP header to enable the signal and notify if congestion is experienced. ECN Capable Transport (ECT bit) and Congestion Experienced (CE bit) for IP headers and Congestion Window Reduced (CWR) and ECN Echo (ECE) for TCP headers. NS-3 implements ECN in its versions 3.30 and above, strictly following the Request For Comments (RFC) guidelines [36]. Since the version used in this implementation of BBRv2 in NS-3 is 3.27, the ECN component had to be added, adapted from the later versions of NS-3.

The operation of ECN in NS-3 mainly takes place in the TCP socket classes with the enabling of the component done through an attribute of the class. The steps for initialising ECN takes place in the three-way handshake of TCP. The sender first sends an empty packet with the CWR and ECE bits as well as the usual synchronize (SYN) bit and the receiver sends the same with an acknowledgement in place of the CWR bit. The sender then sends an acknowledgement upon receipt of the receivers acknowledgement. This causes ECT bits to be set in the IP header of every packet sent by the sender enabling use of ECN.

ECN in NS-3 has stateful properties when reacting to congestion, transitioning from different states in the TCP socket classes in response to the TCP headers bits being set. The state transitions operate as follows as per NS-3 documentation [62]:

1. Before initialisation the sender and receiver have the ECN state set to disabled.
2. After the TCP three-way handshake is successful, the ECN state is set to idle for both sender and receiver meaning they are ready to react to receipt of ECE or CWR bits.
3. When the receiver receives a packet with the CE bit set in the IP header, the ECN state is set as CE received. The receiver sends an ACK with the ECE bit set causing the ECN state to switch to the sending ECE state. This state is then changed to ECN idle once a packet with the CWR bit is received from the sender.
4. The senders ECN states transition first from idle to ECE received once the ACK with ECE bit set is obtained from the receiver. The state then moves to CWR sent when it sends a packet with the CWR bit set, retaining this state until another ACK with an ECE is received changing back to ECE received.

These state transitions impact the behaviour of BBRv2 in its implementation with the tracking of the amount of ECN occurrences in a round trip and reactions to congestion window events caused by ECN, all contributing to the updating of the network model for BBRv2. An ECN alpha value is calculated at the start of every round trip in the implementation of BBRv2 in NS-3 to be used in handling ECN signals. The alpha value is calculated as per equation 4 where *ecnGain* is a predetermined value of 1/16 taken from the kernel source code. The *ecnAlpha* is the previous value of the alpha value (1 if not initialised). *EceRatio* is calculated as per equation 5 where *deliveredEce* is the amount of data delivered so far with an ECE mark minus the previous delivered bytes with an ECE marked when the ECN alpha was last updated. *Delivered* is the total amount of data delivered so far minus the previous delivered bytes when the ECN alpha was last updated. The amount of data delivered so far with an ECE mark is set when there is a congestion window event triggered by a CE mark.

$$ecnAlpha = (1 - ecnGain) * ecnAlpha + ecnGain * eceRatio \tag{4}$$

$$eceRatio = deliveredEce/delivered \tag{5}$$

3.3.5.2 Three Part Model

As discussed in the background, BBRv2 uses a new model in which it operates from in both bandwidth and in-flight bytes. This model is used to adapt the bandwidth and in-flight bytes appropriately when probing for bandwidth. The bounds themselves are set based on the congestion signals consisting of a mixture of loss and ECN. In the implementation of BBRv2 it uses three operating points from the model which are outlined in the table below.

Table 3.1: BBRv2 Three Part Model

Operating Point	**Description**	**Variables**
Latest	Latest measurement from the current round trip	`m_bwLatest`, `m_inflightLatest`
Upper Bound	Robust, optimistic, long-term upper bound	`m_bwHi[2]`, `m_inflightHi`
Lower Bound	Robust, conservative, short-term lower bound	`m_bwLo`, `m_inflightLo`

The `UpdateCongestionSignals` method updates the congestion signals throughout the round trip based off information from the rate sample. The amount of loss and ECN in a round trip is recorded and the latest bandwidth and in-flight bytes are updated as part of the model. If there is loss or ECN the lower bounds for bandwidth are adjusted in response. The lower bounds can not be changed if the network is being probed for bandwidth and these bounds adjusted differently depending on if loss or ECN is

experienced.

$$ecnInflightLo = inflightLo * (1 - (ecnAlpha * ecnFactor)) \tag{6}$$

For the case of ECN, the lower bound for in-flight bytes is set as per equation 6 where the *ecnFactor* variable is a predefined value of 1/3 determined from the kernel source code and the *ecnAlpha* value is calculated as outlined in the previous section. *Inflightlo* is the current value of the lower bound for in-flight rate and is set to the current congestion window if it has not been initialised before.

$$bwLo = max(bwLatest, (bwLo * (1 - beta))) \tag{7}$$

$$inflightLo = max(inflightLatest, (inflightLo * (1 - beta))) \tag{8}$$

When there is packet loss in the round, the lower bound for bandwidth and in-flight bytes are set as per equation 7 and 8. In the equations, the variables *bwLatest* and *inflightLatest* are the latest bandwidth and in-flight samples from the latest round trip, *bwLo* and *inflightLo* are the previous values for the bandwidth and in-flight lower bounds and *beta* is a constant set to 3/10 taken from the official source code for BBRv2. If they have not been used before, the values for *bwLo* and *inflightLo* in the equation are set to the maximum bandwidth in the bandwidth filter and the congestion window value respectively. The resulting value of the lower bound for in-flight bytes is the minimum between the in-flight bound calculated in the case of ECN and the case of packet loss.

3.3.5.3 Startup

The goal of startup in BBRv2 remains the same as with BBR, rapidly discover the available bandwidth in the network. A pacing gain of $\frac{2}{\ln 2}$ is set as normally set with BBR. The conditions for which startup exits is changed to include loss and ECN as well as having the pipe at capacity for 3 round trips. The `CheckExcessiveLossStartup` method keeps a tally of loss events at the end of each round trip in the startup phase through the packet loss recorded by the rate sample. The packet loss is checked against the bytes in flight rate multiplied by a loss threshold constant of 2/100, another value taken from the official implementation by Google. Along with this, the amount of data marked with an ECE mark over the rate sampling interval is compared against the total amount delivered multiplied by an ECN threshold (similar to the loss threshold) of 1/2. If either of these conditions hold, then the in-flight rate is deemed as too high.

This behaviour takes place in the `IsInflightTooHigh` method and if this method returns true and the loss events tally in startup equal or exceed a value of 8 then

startup is exited. This method of exiting startup causes the upper bound of in-flight bytes to be set to the current BDP value. As well keeping a tally of loss events in startup to exit upon, a tally of ECN flags is kept when the ECN alpha value is updated at the start of every round trip. This tally is increased when the *eceRatio*, seen in equation 5, is greater than the ECN threshold. If this tally equals or exceeds a value of 2, then startup is exited and similar to the loss event exit, the upper bound in-flight rate is set to the current BDP value.

3.3.5.4 Drain

The Drain phase for BBRv2 remains the same for the majority of the functionality in comparison to the Drain phase for BBR in that the pacing gain is set to $\frac{\ln 2}{2}$ in order to drain any excess packets from the queue at the buffer. This is still done until the bytes in flight are less than or equal to the current BDP value. The only difference between the adaption of the later version of BBR is that the congestion signals, including the loss and ECN counts for the round trip, are reset to a zero value in order to be used again once the Drain phase is over.

3.3.5.5 ProbeBW

The most substantial implementation changes that had to take place to adopt the behaviour of BBRv2 was to the ProbeBW phase of BBR. This phase is completely re-worked in order to efficiently probe for more bandwidth whilst maintaining low loss and ECN signals. This is done by leaving behind the fixed cyclic pacing gains and dynamically switching through the ProbeBW phases of probe up, probe down, probe cruise and probe refill. The `UpdateCyclePhase` function is solely responsible for updating the phases of ProbeBW in the implementation of BBRv2 in NS-3. The maintaining of the upper bounds of the three part model also takes place in the ProbeBW phase of BBRv2 and occurs every time `UpdateCyclePhase` is initiated in a round trip. In fact, the ProbeBW is only advanced if the upper bounds were adjusted successfully. A number of tracking variables are used in the phase to handle different scenarios in which the upper bounds are adjusted. An acknowledgement phase indicator is used to show the relation between the ACK stream and the bandwidth probing done in ProbeBW. These are set in each ProbeBW phase based on the type of probing being done, for example, in probe up, the indicator is set to probe starting and so on.

The use of these acknowledgement phase indicators comes in to play when the upper bounds are being adjusted at the start of a round trip. When probing is indicated to be stopping, the bandwidth filter is advanced so that the older bandwidth samples can be forgotten. Along with this, tracked variables that are used to assess if a probe is risky and if the previous probe was too high are checked and if it is deemed that the probe is

risky and the previous probe was not too high then the refill phase is entered. The upper bounds themselves are set based on if the in-flight rate is too high or not. If the in-flight rate is too high and this is as a result of further probing by the probe refill phase then this is handled. Consequently, the previous probe is deemed too high and the upper bound for in-flight bytes is set as per equation 9. *BytesInFlight* is the current amount of in-flight rate, *BDP* is the estimated bandwidth multiplied by the propagated RTT and the *beta* value is a constant of 3/10 used before in the calculation of the lower bound. The probe down phase is entered upon the *inflightHi* being set in this case.

$$inflightHi = max(bytesInFlight, BDP * (1 - beta)) \tag{9}$$

In the case that the in-flight rate is not deemed too high, the upper bound for in-flight bytes is set to the current bytes in flight rate given that it is greater than the previous value for the upper in-flight bound. If the ProbeBW phase is in probe up during this assignment then the upper bound is further increased. The method for which this is done is presented in the Code Listing 3.2 where `m_bwProbeUpAcks` is a count of how many packets were selectively acknowledged per increase in the upper bound of in-flight bytes set by the last selectively acknowledged bytes in the rate sample. The `m_bwProbeUpCount` variable holds the amount of delivered packets per increase in the upper bound of in-flight bytes. This variable is set upon the start of the round trip as per equation 10 where *initialCwnd* is the initial value of the congestion window and *bwProbeUpRounds* is a count of how many round trips are spent in probe up which is incremented every time this equation is executed. Upon entering the ProbeBW phase in BBRv2 for the first time in an execution of the congestion control algorithm, the probe down phase is the first to be entered. This happens when the ProbeBW is re-entered again but only for one round trip as it is followed by the probe cruise phase.

```
1 m_bwProbeUpAcks += rs->m_lastAckedSackedBytes;
2 if (m_bwProbeUpAcks >= m_bwProbeUpCount)
3   {
4     m_bwProbeUpAcks -= (m_bwProbeUpAcks / m_bwProbeUpCount) *
    m_bwProbeUpCount;
5     m_inflightHi += m_bwProbeUpAcks / m_bwProbeUpCount;
6   }
```

Code Listing 3.2: ProbeInflightHighUpward()

$$bwProbeUpCount = max(initialCwnd/bwProbeUpRounds, 1) \tag{10}$$

Probe Down - The probe down phase behaviour is the same as in BBR with the draining of excess packets from the queue at the buffer as a result of probing for more

bandwidth. The application of this phase in BBRv2 involves resetting the congestion signals (like in the Drain phase), the count of round trips spent in the probe up phase and the timestamp for the gain cycle. A variable is used to track the number of round trips there has been since the last probe of bandwidth. In this phase, this is reset to a random variable between 0 and 2, following from the methodology in the official kernel implementation. This tracking variable is incremented at the start of every round trip.

$$roundsSinceProbe >= min(bwProbeMaxRounds, BDP * renoGain) \tag{11}$$

Probe down exits based on two conditions in the `UpdateCyclePhase` method. `IsTimeToProbe` is a function that checks if it is reasonable to start probing for more bandwidth based on if a round trip has passed or an equation used specifically for Reno coexistence holds. This equation can be seen in equation 11 with the *roundsSinceProbe* being the count of round trips there has been since the last probe of bandwidth, *bwProbeMaxRounds* is a constant for the maximum number of rounds that bandwidth can be probed which is set as the same value of the kernel implementation of 63. The *BDP* is the BDP value at the time and *renoGain* is set to 1. The probe refill phase is entered if either of these conditions hold.

`IsTimeToCruise` checks if the network model is in a state to transition from probe down to probe cruise. This decision is made to be true when the prior bytes in flight are less than or equal to the BDP value at the time or if a round trip has passed. The prior bytes in flight are also compared against the upper bound for in-flight bytes with headroom taken away to determine if there is enough headroom left for cruising at a pacing gain of one. The method returns false when the prior bytes in flight are greater than this upper bound minus headroom meaning there is not enough headroom left. Headroom is calculated as seen in equation 12 where the *inflightHi* is the upper bound for in-flight bytes and *inflightHeadroom* is a fixed constant of 15/100, a value determined by the official implementation by Google.

$$headroom = inflightHi * inflightHeadroom \tag{12}$$

Probe Up - When the probe up phase of ProbeBW is entered, the actions for probing for more bandwidth begin. The pacing rate is set to 1.25 the BDP and the tracker of probe up rounds is incremented along with the count of delivered packets per probe up round as calculated by equation 10. The timestamp for the ProbeBW cycle is reset in this phase as it may take a number of rounds to probe for more bandwidth until one of the exit conditions for the phase is hit. These exit conditions include if the previous probe was a risky one and if there is a queue forming at the bottleneck. A probe is gauged as risky if the previous probe was too high i.e. loss or ECN was

detected and if the prior bytes in flight go above the upper bound for in-flight bytes. A queue can be seen as forming at the bottleneck if a full round trip has passed and the prior bytes in flight are greater than the BDP value at a pacing gain of 1.25. Consequently, the probe down phase is entered if these conditions are met in order to drain any excess packets from the queue as a result of the bandwidth probing.

Probe Refill - The responsibility of the probe refill phase is to prepare the network model for that of the probe up phase in the discovery of more bandwidth with the three part model coming into play. The lower bounds get reset in order to set them to new values based on the success of the probing of bandwidth. The phase enters the probe up phase immediately after preparing the model for probing.

Probe Cruise - When entered from the probe down phase the probe cruise phase sets the pacing rate back to a value of one of the BDP value. In the implementation of BBRv2, it also sets the value for the lower bound of the in-flight rate in order to set the appropriate interval between that lower bound and the upper bound as a result of the previous probing for bandwidth. During this phase, the `UpdateCyclePhase` method constantly checks if the network is in a state to probe for more bandwidth through the use of the `IsTimeToProbe` function and transitions to the probe refill phase if the conditions in this method hold.

3.3.5.6 ProbeRTT

The behaviour of the ProbeRTT phase in the implementation of BBRv2 in NS-3 is, for the most part, the same as that of BBR apart from the regulation of the congestion window which is controlled based on the three part model. The `BoundCwndForInflightModel` method is responsible for performing this behaviour on the congestion window. When in ProbeRTT the congestion window is set as in equation 13 where the *inflightHeadroom* variable is the upper bound of the in-flight rate minus the headroom calculated in equation 12. *InflightLo* is the lower bound of the in-flight rate, the *minCwnd* is the minimum value the congestion window can be (4 packets) and the *cwnd* is the current value of the congestion window.

$$cwnd = min(max(min(inflightHeadroom, inflightLo), minCwnd), cwnd) \tag{13}$$

3.3.5.7 Experimental changes

The experimental changes made to BBRv2 are also implemented in NS-3 and can be set to be on or off in simulations through a configuration attribute. The changes include an update to the adaption of the lower and upper bounds for the three part model. The

lower bound for bandwidth is initialised to be the maximum bandwidth in the bandwidth filter and if ECN is experienced it is set based on equation 14 where *bwLo* is the previous lower bound for bandwidth, *ecnAlpha* is the alpha value from the ECN calculations and *ecnFactor* is set to 1/2 in the case of the experimental changes. This applies to the *ecnFactor* used in all other calculations involving the constant for the experimental changes.

$$ecnBwLo = bwLo * (1 - (ecnAlpha * ecnFactor)) \tag{14}$$

In the adaption of the upper bounds of the three part model, the upper bound for bandwidth is set to the delivery rate from the most recent rate sample if it is higher than the current upper bound for bandwidth. This applies when the in-flight rate is not deemed as too high. If it is the case that the in-flight rate is too high and needs to be handled as a result of refilling the pipe, the upper bound for bandwidth is set as per equation 15 where *bwHi* is the previous value for the upper bound for bandwidth, *ecnAlpha* is the alpha value based on ECN calculations and *ecnFactor* is 1/2 as mentioned before.

$$bwHi = bwHi * (1 - (ecnAlpha * ecnFactor)) \tag{15}$$

The final experimental change that was adapted in the implementation of BBRv2 in NS-3 was made to both the upper bound for the bytes in flight and the bandwidth when assessing if the network is in a state to probe for more bandwidth. If there is an ECN signal detected in the ProbeBW cycle but there is no ECN mark in the acknowledgement of the most recent packet then the upper bound for bytes in flight is incremented and the upper bound for bandwidth is set as seen in equation 16. The *bwHi* variable is the current value for the upper bound for bandwidth and *rtProp* is the value of the propagated RTT. As a result of these changes to the two upper bounds, the probe refill phase of ProbeBW is entered.

$$bwHi = bwHi + (bwHi/rtProp) \tag{16}$$

3.4 Simulation Setup

The simulation for evaluating BBRv2 and the other variants of BBR in NS-3 is encapsulated in one executable script that is configured appropriately to simulate the desired scenario. NS-3 allows for traces to be set up for each network node in order to record certain network metrics over the simulation time. The traced metrics in each simulation include congestion window size, round trip time, retransmission timeout time

and bytes in flight. A file stream is set up to output each metric to a file for each scenario. For packet level analysis NS-3 includes the ability to generate pcap (packet capture) files for each flow. Statistics like packet size, flags, arrival time and contents can be seen using these files. `FlowMonitor` is a class in NS-3 that retrieves attributes of a flow in a simulation such as delay, loss, jitter, number of sent and received packets and times packets where forwarded. This means that the likes of throughput, transmission delay and packet loss ratio can be traced throughout the simulation and also outputted to a file using a file stream.

3.4.1 Topology

The topology adopted in the simulations was a dumbbell topology which is broken up into a number of sender and receiver leaf nodes linked to a router. The two routers are connected at the bottleneck link. A representation of the topology can be seen in Figure 3.3 where there are two leaf nodes on each side. NS-3 provides helper classes for creating dumbbell topologies, by simply including the number of left and right leaves along with a point-to-point device to represent the leaves. IP addresses are assigned to each leaf after the devices are installed. Point-to-point devices represent the network nodes for a simple point-to-point network and contain attributes like data rate and delay that can be set in order for specific behaviour to be performed in the network. In the case of this evaluation TCP applications are assigned to the sender and receiver nodes with distinct start and stop times to send and receive packets. Following this, the routing tables are populated for each of the nodes in the network in order for distinct routes to be set up between all the nodes.

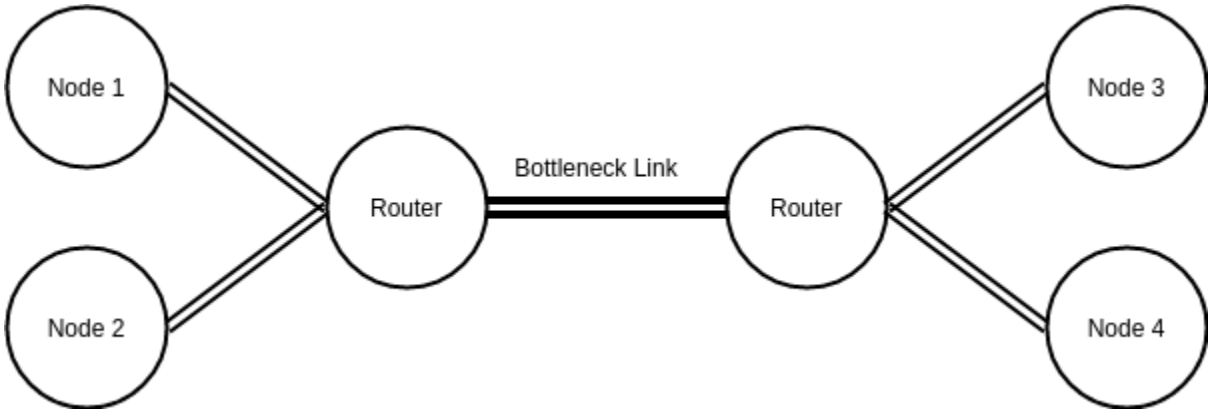

Figure 3.3: Simulation Topology

3.4.2 Attributes

The network attributes are managed through a number of variables that can be changed depending on the scenario to be simulated. The bandwidth and delay for both the

access links between nodes, routers and the bottleneck links are set and assigned to the point-to-point devices representing both. The attributes for the TCP application including segment size, buffer size, initial congestion window size, minimum retransmission timeout and the congestion control algorithm itself are controlled through these variables and configured before the creation of the TCP applications. In the case of this implementation, the simulation script includes the ability to switch between variations of BBR including BBRv2 by changing an attribute for the congestion control. Other attributes that are configured especially for the evaluation of BBRv2 and its variants include the number of leaf nodes to be used so that the number of flows can be increased or decreased in a simulation, the control of the lambda shape parameter value for the BBR+ variant and the enabling of ECN and experimental changes for BBRv2. All these attributes allow for a wide number of scenarios to be simulated in the evaluation of the different BBR variations.

3.4.3 TCP Cubic

As the default congestion control in the Linux kernel and a widely used one in implementations of TCP across the internet, TCP Cubic is an essential congestion control algorithm to simulate against, especially in the case of this evaluation of BBRv2 and the other variants. Unfortunately, NS-3 does not have an implementation for TCP Cubic as one of its congestion control modules by default. Fortunately, in 2014 Brett Levasseur et al. designed and implemented TCP Cubic for NS-3 with appropriate validation with the Linux version [69]. This implementation of TCP Cubic in NS-3 is adapted and used in this evaluation of BBRv2 and its variants.

4 Results and Evaluation

This chapter will evaluate and discuss the results obtained from running simulations in NS-3 using BBRv2 congestion control and the other variations discussed. The details of each simulation will be outlined with an analysis into the results of the simulation in correspondence to the variant of BBR congestion control used in that simulation. The figures that show the results are plotted using R [70]. Every configuration that is ran is plotted using a number of different network metrics. Plots that are not displayed in this section can be found in the results folder of the source code for this project in the Github repository[1]. The results and evaluation are broken up into the following sections:

- Single Flow - Evaluation of a network with one flow of BBRv2 or a variant.
- Intra-Protocol - Evaluation of a network with multiple flows of BBRv2 or a variant.
- Inter-Protocol - Evaluation of a network with multiple flows of BBRv2 or a variant with a loss-based or delay-based algorithm.

4.1 Single Flow

A network environment for a single TCP flow transportation over the network is used to analyse the characteristics of a congestion control algorithm with no contention for bandwidth. A number of single flow scenarios are set up in NS-3 using different network attributes. In order to keep a consistency across each simulation, a number of network attributes in the simulation scenarios in NS-3 are kept the same. These are displayed in Table 4.1. The access bandwidth and delay correspond to the bandwidth and delay of each leaf node access link. The value of 40Mbps and 1ms was chosen as a consistent value across all simulations with no extreme variation in bandwidth between the access links and bottleneck links. The buffer size is kept consistent at 1.5 times the BDP value. This value is chosen to keep the buffer at a medium size to show the optimal

[1] https://github.com/lanigan23/BBRv2-Eval-ns-3

behaviour of the congestion control being simulated in this single flow scenario.

Table 4.1: Scenario Configuration

Attribute	**Value**
Access Bandwidth	40 Mbps
Access Delay	1 ms
Simulation Time	100 Seconds
Segment Size	536 Bytes
Initial Cwnd	10 Segments
Buffer Size	1.5 * BDP
Minimum RTO	200 ms

The network attributes that are changed for each simulation include the bottleneck bandwidth and bottleneck delay. The values for these attributes for this single flow scenario are outlined in Table 4.2. These values are chosen to show any distinction in behaviour of the congestion control algorithm in different variations of bandwidth and delay at the bottleneck. A number of algorithm specific attributes are also set for particular algorithms including BBR+ and BBRv2. These include the lambda shape parameter value for BBR+ which is involved in the calculation of propagated RTT value for that algorithm. This lambda value is set the same as demonstrated in the introduction of the algorithm [24] with the value of 0.5 and 0.125. The simulations are ran twice to compensate for this change in value and to distinguish any difference between them. The attribute for BBRv2 which is enabled at simulation time is the enabling of the experimental changes for the algorithm, which as discussed before, makes changes to the three-part model of the algorithm. Like BBR+, the simulations are ran twice with and without this attribute set.

Table 4.2: Single Flow Bottleneck Configuration

Bandwidth	**Delay**
5 Mbps	10 ms
5 Mbps	40 ms
5 Mbps	100 ms
10 Mbps	10 ms
10 Mbps	40 ms
10 Mbps	100 ms

Bandwidth utilisation is looked at through the outputted throughput and goodput of the congestion control algorithm being used in a simulation. The comparison of

throughput over time compared with the actual bottleneck bandwidth is a good indication of an algorithms performance. The average goodput is taken through NS-3's packet sink which eliminates the overhead in the transported packets to output the useful bits transported and forwarded on for use in the application layer of the network stack. BBR has the goal of a highly consistent throughput across the network, Figure 2 of [8] shows a high bandwidth utilisation in the official verification of the algorithm in 2017. The variations of BBR follow suit in aiming to achieve high bandwidth utilisation without drastically changing the characteristics of the algorithm. The Table 4.3 shows the average throughput and goodput achieved in each simulation scenario for every variation of BBR including BBRv2. Particularly for BBR, the RTT value obtained gives insight into how the propagated RTT is having an effect on the re-estimation of the BDP value and the transition between states of the algorithm.

4.1.1 BBR and BBR'

BBR and BBR' contain few differences in functionality with the major difference being in the advancement in the gain cycling when probing for more bandwidth. For a single uncontested flow, both show a high utilisation of bandwidth with minuscule difference between the two types of algorithm. The trend of a high throughput rate can be seen throughout the 100 second simulation in Figures 4.1 and 4.2. The distinct curve of the line shows the phases of BBR. The startup phase is shown by the sharp increase at the start of the simulation all the way up to the bottleneck bandwidth. A small decrease in bandwidth shows the draining of the queue followed by further probing up to the estimated bottleneck bandwidth. This pattern continues for the remainder of the simulation with shorter drain times and a gradual convergence at the estimated bottleneck bandwidth.

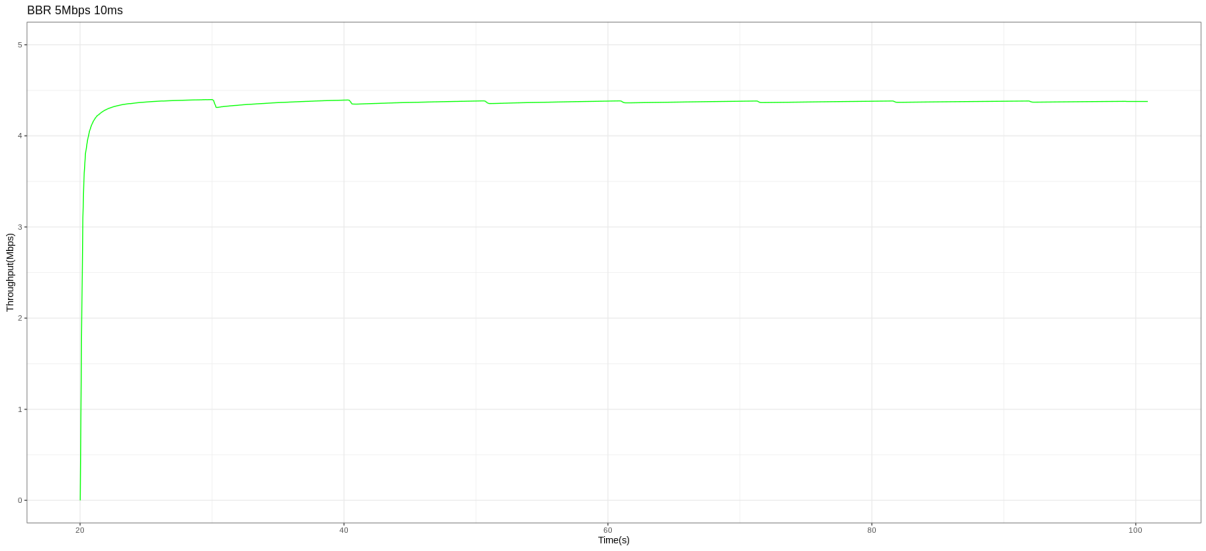

Figure 4.1: BBR Single Flow Throughput

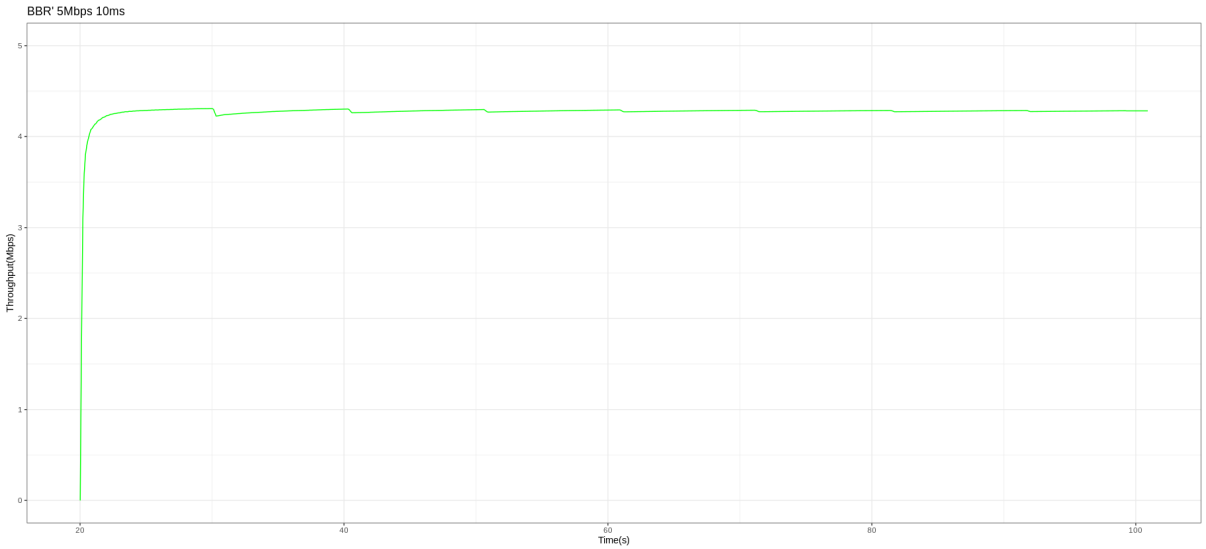

Figure 4.2: BBR' Single Flow Throughput

An interesting observation is the utilisation of only 50% bandwidth for a 100ms delay in the bottleneck for a 10Mbps flow. This is due to the inability to probe at a sufficient rate in the startup phase because of the increased delay at the bottleneck. The estimated bottleneck value is only set to half of the utilised bandwidth in this case and is unable to probe beyond this during the ProbeBW phase.

Table 4.3: Single Flow Results

Algorithm	Bandwidth	Delay	Average Throughput	Average Goodput
BBR	5 Mbps	10 ms	4.38 Mbps	3.80 Mbps
	5 Mbps	40 ms	4.20 Mbps	3.52 Mbps
	5 Mbps	100 ms	4.25 Mbps	3.39 Mbps
	10 Mbps	10 ms	8.99 Mbps	7.25 Mbps
	10 Mbps	40 ms	9.01 Mbps	6.89 Mbps
	10 Mbps	100 ms	5.14 Mbps	3.91 Mbps
BBR'	5 Mbps	10 ms	4.28 Mbps	3.80 Mbps
	5 Mbps	40 ms	4.05 Mbps	3.52 Mbps
	5 Mbps	100 ms	4.25 Mbps	3.39 Mbps
	10 Mbps	10 ms	8.84 Mbps	7.25 Mbps
	10 Mbps	40 ms	9.00 Mbps	6.89 Mbps
	10 Mbps	100 ms	5.13 Mbps	3.91 Mbps
BBR+	5 Mbps	10 ms	3.48 Mbps ($\lambda = 0.125$)	3.80 Mbps ($\lambda = 0.125$)
			3.48 Mbps ($\lambda = 0.5$)	3.80 Mbps ($\lambda = 0.5$)
	5 Mbps	40 ms	4.50 Mbps ($\lambda = 0.125$)	3.51 Mbps ($\lambda = 0.125$)
			4.50 Mbps ($\lambda = 0.5$)	3.51 Mbps ($\lambda = 0.5$)
	5 Mbps	100 ms	4.23 Mbps ($\lambda = 0.125$)	3.38 Mbps ($\lambda = 0.125$)
			4.23 Mbps ($\lambda = 0.5$)	3.38 Mbps ($\lambda = 0.5$)
	10 Mbps	10 ms	9.02 Mbps ($\lambda = 0.125$)	7.25 Mbps ($\lambda = 0.125$)
			9.02 Mbps ($\lambda = 0.5$)	7.25 Mbps ($\lambda = 0.5$)
	10 Mbps	40 ms	9.03 Mbps ($\lambda = 0.125$)	6.89 Mbps ($\lambda = 0.125$)
			9.03 Mbps ($\lambda = 0.5$)	6.89 Mbps ($\lambda = 0.5$)
	10 Mbps	100 ms	5.15 Mbps ($\lambda = 0.125$)	3.92 Mbps ($\lambda = 0.125$)
			5.15 Mbps ($\lambda = 0.5$)	3.92 Mbps ($\lambda = 0.5$)
BBRPlus	5 Mbps	10 ms	4.38 Mbps	3.80 Mbps
	5 Mbps	40 ms	4.50 Mbps	3.51 Mbps
	5 Mbps	100 ms	4.23 Mbps	3.40 Mbps
	10 Mbps	10 ms	9.02 Mbps	7.25 Mbps
	10 Mbps	40 ms	9.03 Mbps	6.89 Mbps
	10 Mbps	100 ms	5.16 Mbps	3.93 Mbps
Delay-BBR	5 Mbps	10 ms	4.60 Mbps	3.82 Mbps
	5 Mbps	40 ms	4.44 Mbps	3.60 Mbps
	5 Mbps	100 ms	4.60 Mbps	3.49 Mbps
	10 Mbps	10 ms	9.31 Mbps	7.33 Mbps
	10 Mbps	40 ms	9.33 Mbps	7.06 Mbps
	10 Mbps	100 ms	5.26 Mbps	3.97 Mbps
BBRv2	5 Mbps	10 ms	4.16 Mbps	3.80 Mbps
			4.16 Mbps (exp)	3.80 Mbps (exp)
	5 Mbps	40 ms	3.90 Mbps	3.51 Mbps
			3.90 Mbps (exp)	3.51 Mbps (exp)
	5 Mbps	100 ms	4.21 Mbps	3.40 Mbps
			4.21 Mbps (exp)	3.40 Mbps (exp)
	10 Mbps	10 ms	8.77 Mbps	7.25 Mbps
			8.77 Mbps (exp)	7.25 Mbps (exp)
	10 Mbps	40 ms	8.97 Mbps	6.89 Mbps
			8.97 Mbps (exp)	6.89 Mbps (exp)
	10 Mbps	100 ms	5.11 Mbps	3.89 Mbps
			5.11 Mbps (exp)	3.89 Mbps (exp)

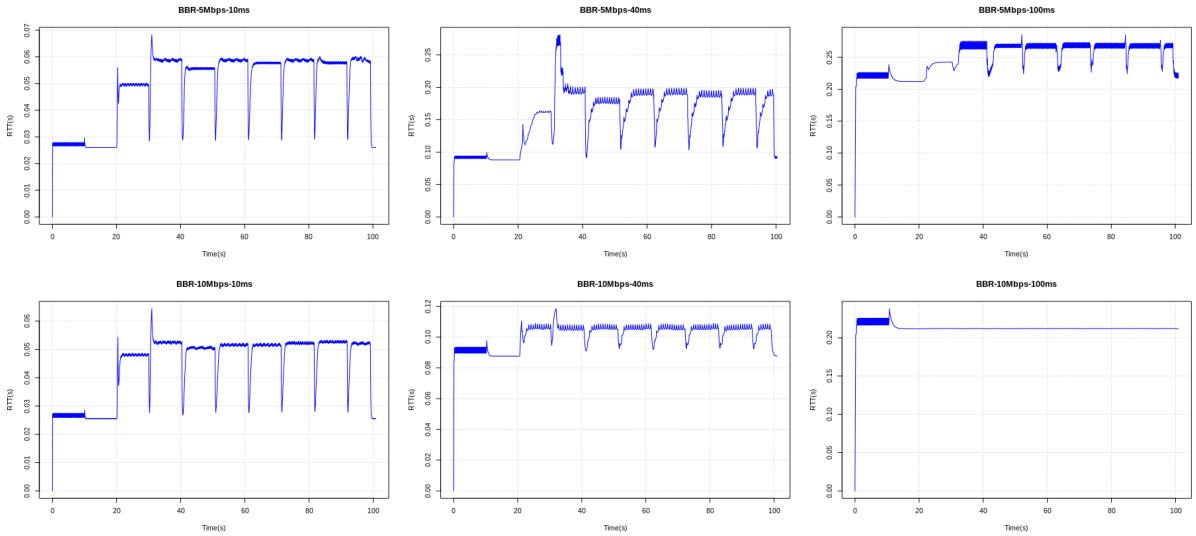

Figure 4.3: BBR Single Flow RTT

The round trip times of each configuration can be seen for both BBR and BBR' in Figures 4.3 and 4.4. The variation of the RTT in comparison between the corresponding configurations of the two algorithms show the difference that the change in gain cycling method has with a greater variation in BBR' when the state is in ProbeBW phase indicated with the area between the dips in the line. The re-estimation of the propagated RTT can also be seen in a dip in the line every 10 seconds when ProbeRTT phase is entered. For configurations with higher delay at the bottleneck the RTT values are consistently higher and vary less because of the added delay at the bottleneck.

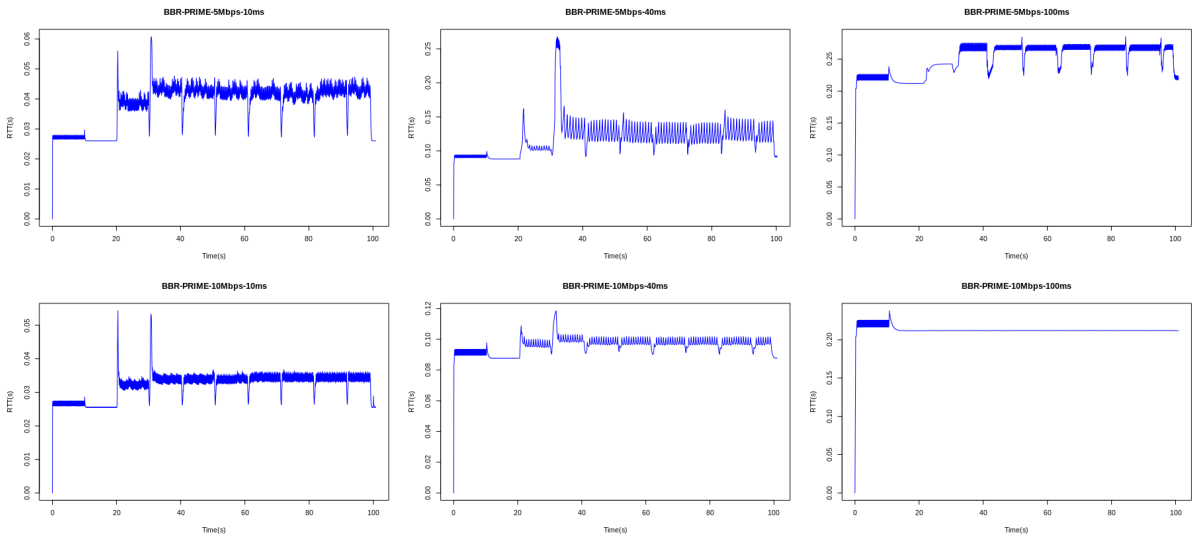

Figure 4.4: BBR' Single Flow RTT

4.1.2 BBR+

The changes made to make BBR more adaptable to high speed environments are reflected in the simulation of BBR+ in a single flow scenario. The outputted throughput and goodput however for each different lambda value contrast that of the evaluation over high speed rails [24], with virtually no difference between the simulations ran with different lambda values as seen in Table 4.3. However, there is some difference in comparison to the other variations of the BBR algorithm with the changes to the gain cycling and the estimation of the propagated RTT taking effect as see in Figure 4.5 when the lambda value is 0.5. The RTT varies drastically in ProbeBW due to the sharp increase in pacing rate when probing for more bandwidth with BBR+ and also through the use of the lambda value to estimate the propagated RTT every 10 seconds.

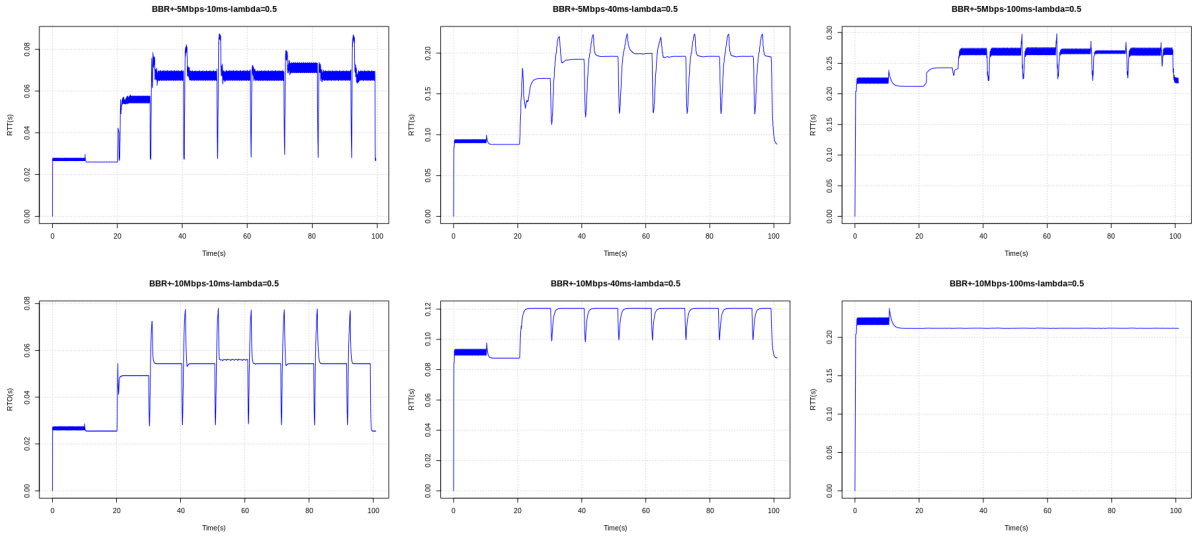

Figure 4.5: BBR+ Single Flow RTT

The values obtained for the throughput over time for each value of lambda used in the simulation of BBR+ congestion control can be seen in Figures 4.6 and 4.7. This supports that there is no difference in throughput variance over time for the two different values of lambda used contradictory of that in the evaluation over high speed rails [24]. The behaviour of the line in the plots of the throughput reflects the change in the BDP estimation since the estimation for the propagated RTT is different. After the startup phase reaches a throughput of over 4 Mbps seen in the plots it can be seen draining to a throughput of just under 3 Mbps due to this BDP estimation.

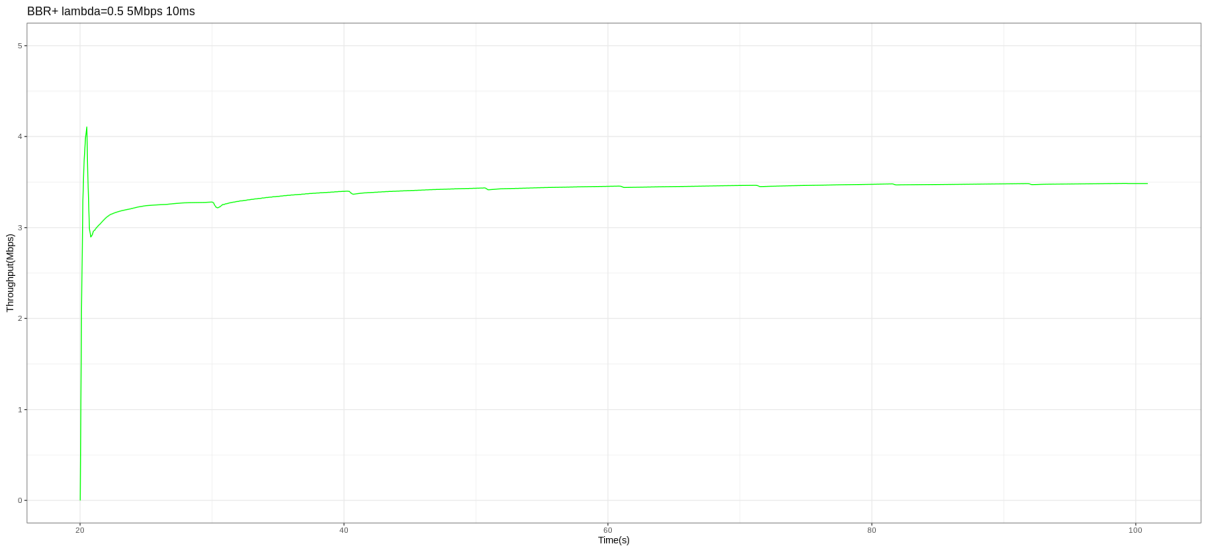

Figure 4.6: BBR+ Single Flow Throughput $\lambda = 0.5$

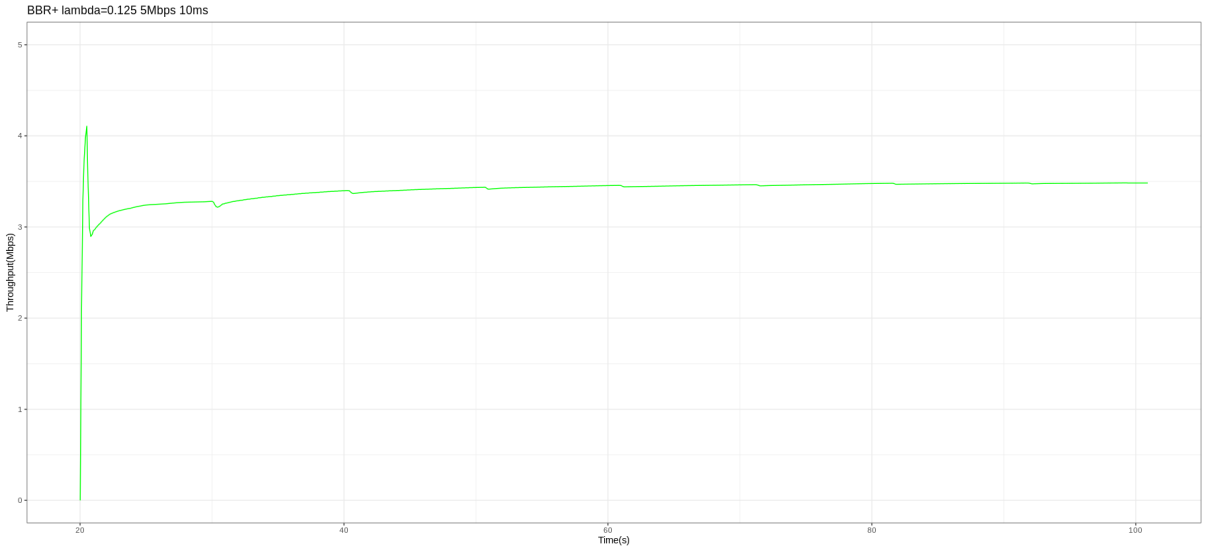

Figure 4.7: BBR+ Single Flow Throughput $\lambda = 0.125$

4.1.3 BBRPlus

The performance of the BBRPlus variant of BBR in terms of throughput and goodput shows high bandwidth utilisation as per the values in the Table 4.3. The Figure 4.8 shows a consistent probing for more bandwidth using the updated methodology of the algorithm which dynamically changes through the phases of ProbeBW.

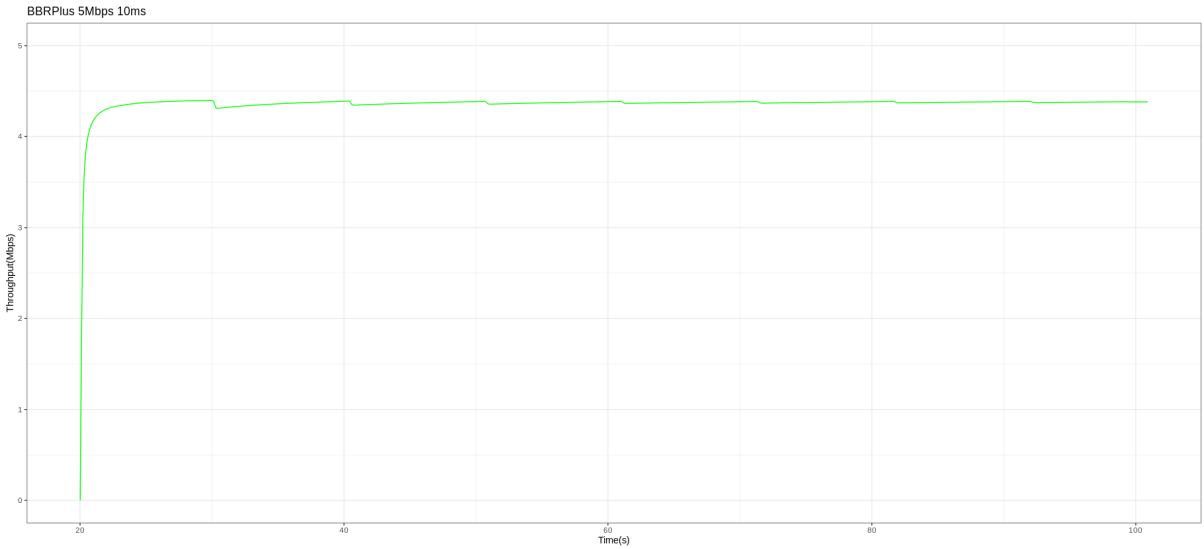

Figure 4.8: BBRPlus Single Flow Throughput

The RTT plots also reflects the good performance of the variant in Figure 4.9 with a consistent round trip time during the ProbeBW phase before the propagated RTT is re-estimated every 10 seconds. One of the goals of this variation was to achieve a higher throughput for wifi and other paths including wired networks [23]. These results reflect a slight improvement over the original algorithm in this scenario of an uncontested single flow network.

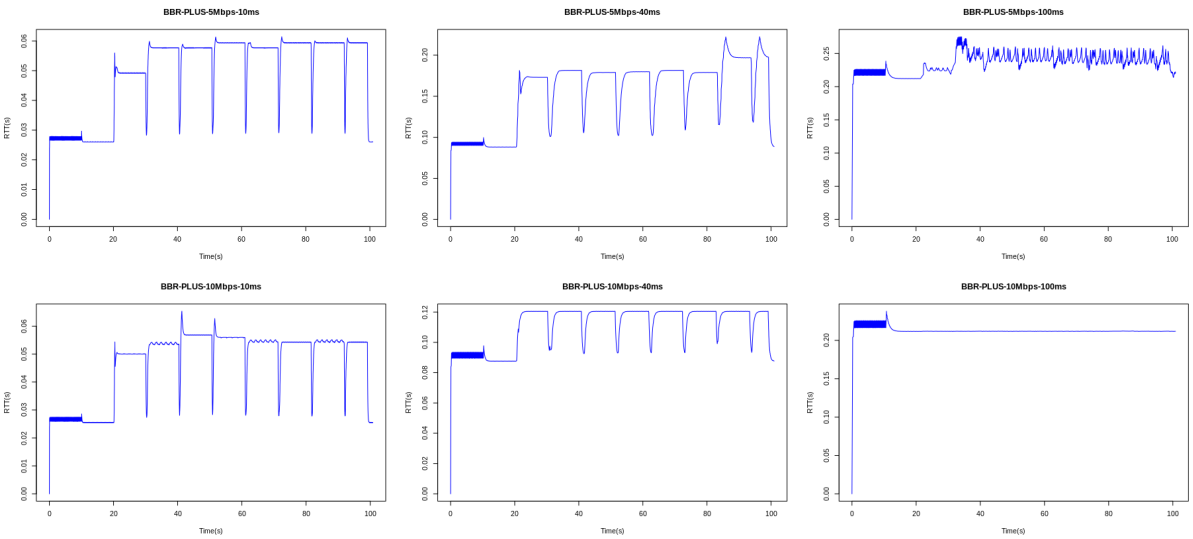

Figure 4.9: BBRPlus Single Flow RTT

4.1.4 Delay-BBR

This variation of BBR, which tailors towards video streaming, performs with the greatest increase in throughput over the original BBR algorithm, as seen in the values

of each simulation of Table 4.3. The Figure 4.10 of the throughput over time for the configuration of 10Mbps bottleneck bandwidth with a 10ms delay gives evidence of the change in behaviour of the variation of the algorithm. This is through the consistent line at approximately 9Mbps which deviates very slightly every 10 seconds. This is due to the changed values for the pacing gain cycle which are far more conservative when probing for bandwidth and draining the queues at the buffer.

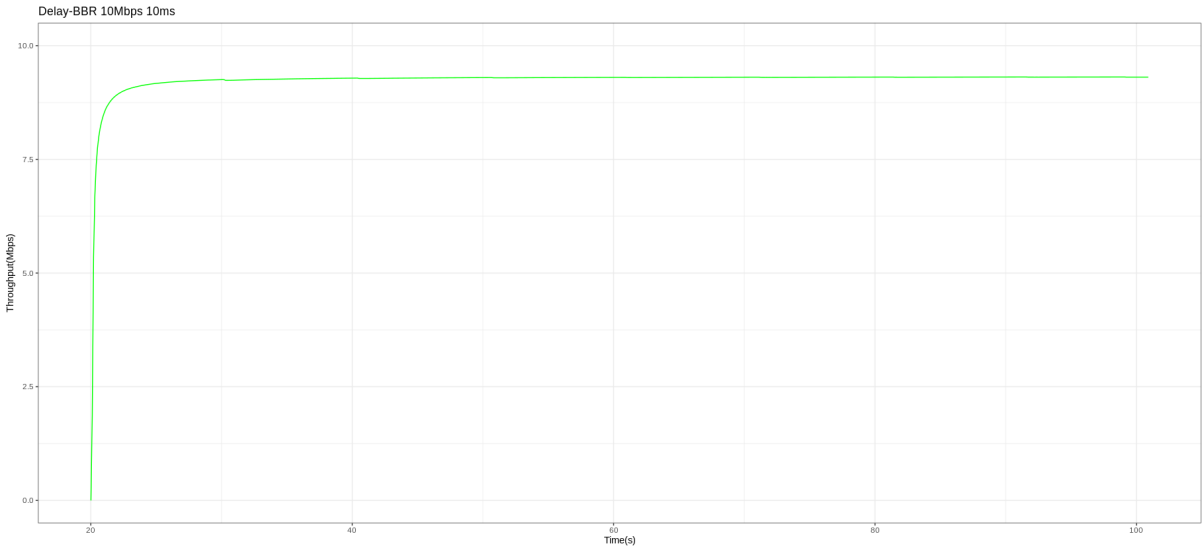

Figure 4.10: Delay-BBR Single Flow Throughput

This convincing performance of the Delay-BBR comes at cost of higher RTT values, as seen in Figure 4.11, where values are far greater than that of BBR or any other variation of the algorithm. The use of the smoothed round trip signal causes this since the ProbeRTT phase is entered more frequently and instead of allowing only 4 packets in this phase the congestion window is set to have a value of 3/4 the BDP value. This causes queues to stay built up at the buffer when typically this phase is for draining those queues. Consequently, this leads to longer round trip times. This is not captured in the evaluation of the variant by Zhang et al. [25]. Despite this, the high throughput output of the algorithm correlates with Figure 7 of the evaluation of the proposed algorithm by Zhang et al.

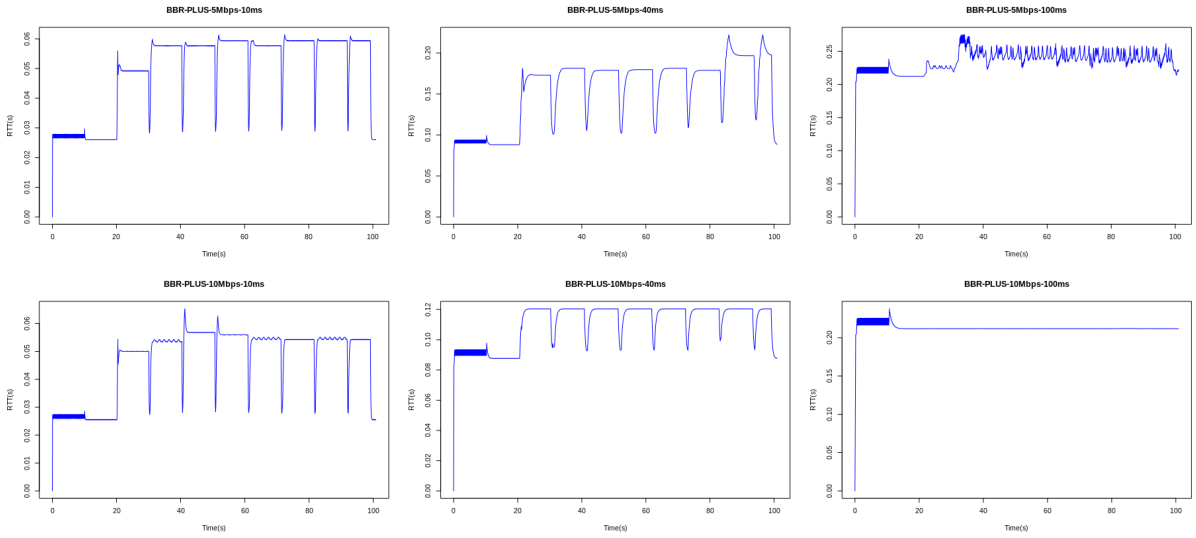

Figure 4.11: Delay-BBR Single Flow RTT

4.1.5 BBRv2

The results obtained for throughput for the second version of the BBR algorithm reflects the claims made in the introduction of the algorithm. Figure 1 of the evaluation by the team behind the development of the algorithm shows that the algorithm under-performs that of the original algorithm in a single flow test [26]. This is the case in this evaluation of the implemented algorithm in NS-3 as outlined in Table 4.3. Although not significantly lower than that of the original, the throughput is still relatively high in terms of bandwidth utilisation and the goodput rate remains in or around the same as BBR. BBRv2 with experimental changes is also simulated using the same scenarios as the algorithm without these changes. The results show little to no effect to the simulation with the experimental alterations. This is due to the experimental changes being related to the use of the ECN signal which is not experienced in these scenarios. There is no congestion experienced so there is no evidence of the scenarios of congestion for which BBRv2 deals with. The trends of the Figures of the throughput of BBRv2 4.12 and 4.13 are similar to that of the plots depicted in the introduction of the algorithm [2]. The ProbeBW phases are identified with the subtle dips in the line corresponding to the probe down phase. The increase in bandwidth after this represents the probe refill and probe up phase exploring more bandwidth. The narrowing of the area between every probe as the simulation continues shows the increase of the lower and upper bounds of the three part model of the algorithm.

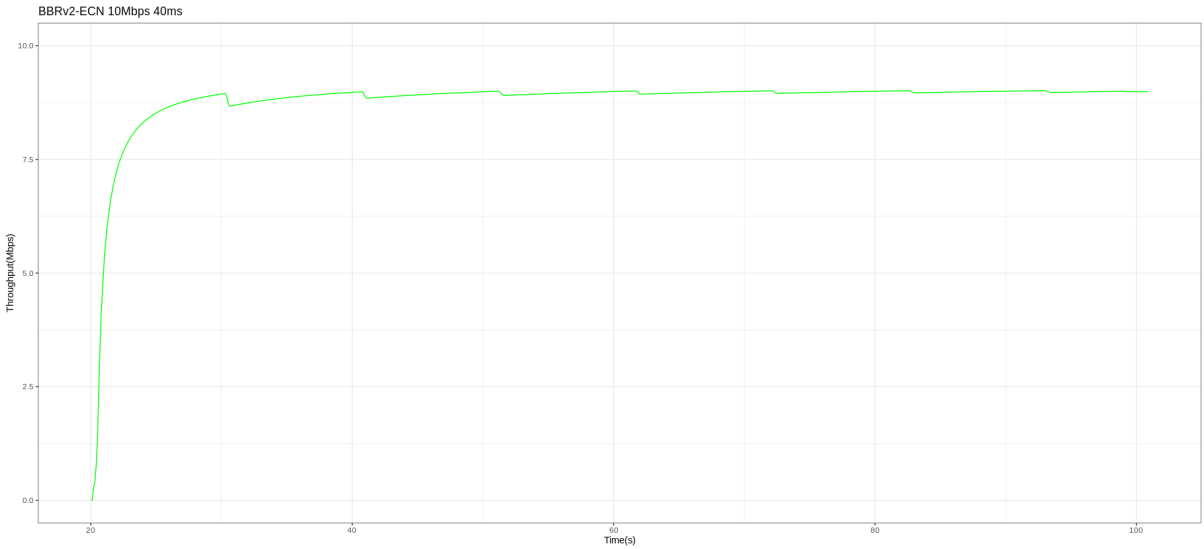

Figure 4.12: BBRv2 Single Flow Throughput

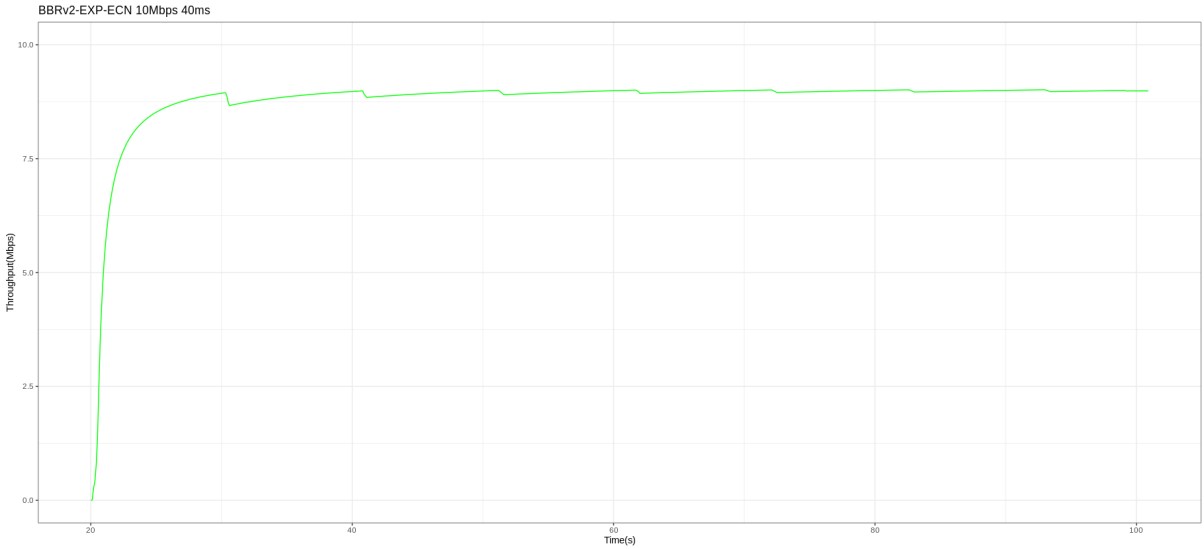

Figure 4.13: BBRv2-EXP Single Flow Throughput

The RTT values obtained are displayed for each configuration in Figure 4.14. These values show a different trend to the other variations of BBR with a consistently low RTT value after a large spike in the 20-30 second range. This spike is down to the initial exponential startup phase and this can be seen to calm to a consistent RTT value after the Drain phase is entered and the propagated RTT is updated.

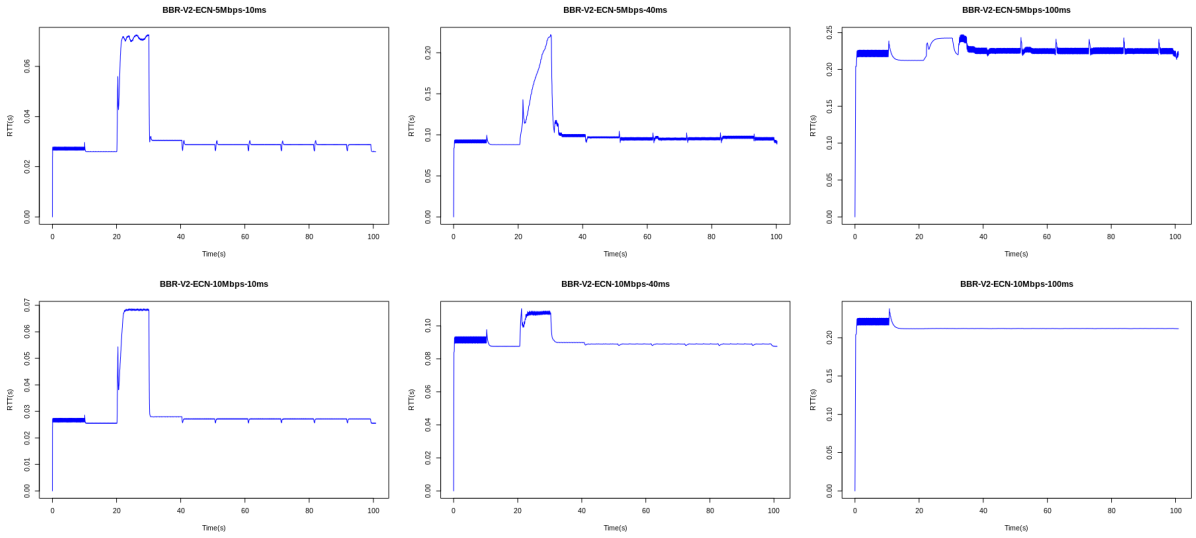

Figure 4.14: BBRv2 Single Flow RTT

4.1.6 Bandwidth Variation Response

In the official evaluation of the BBR congestion control algorithm, a scenario that is evaluated in order to discover the responsiveness of the algorithm is one that entails changing the bandwidth dynamically for a single flow. The results for this scenario are in Figure 3 of the paper [8]. This scenario is simulated for each variation of BBR in this project in NS-3 in order to validate the behaviours of algorithms against their original claims. An increase from 10Mbps to 20Mbps is used for the variation in bottleneck bandwidth in these simulation scenarios. The bottleneck bandwidth and delay configurations for these simulations are outlined in the Table 4.4. All of the other network characteristics that were used in the single flow simulations are kept the same including queue size.

Table 4.4: Bandwidth Variation Bottleneck Configuration

Bandwidth	**Delay**
10 Mbps	10 ms
10 Mbps	40 ms
10 Mbps	100 ms

The Figure 4.15 shows the behaviour of all the variants of BBR explored in this evaluation and their response to the bandwidth doubling when the bottleneck bandwidth is initially 10Mbps and the bottleneck delay is 10ms. The event of increasing the bandwidth was scheduled for 50 seconds into the simulation and as seen in the plot every algorithm instantaneously explores the new available bandwidth quickly, reaching

15Mbps in the next 50 seconds. This behaviour shows similarity to that of the Figure 3 of [8] where the estimated bandwidth is doubled.

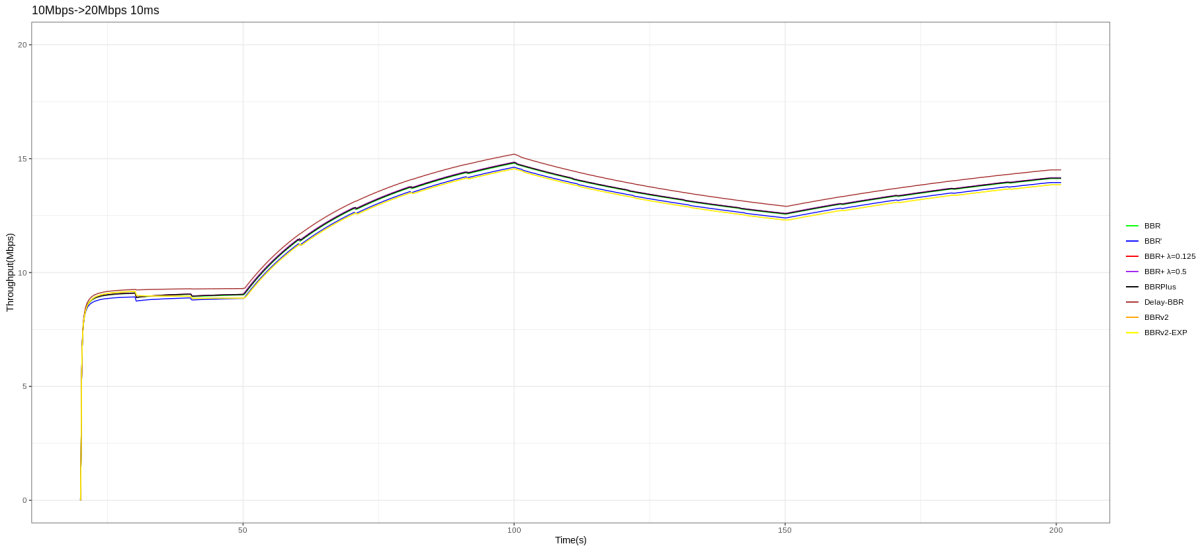

Figure 4.15: BBR Variants Responsiveness

The most important adaption of this behaviour is the minimal effects on the RTT when this change in bandwidth occurs. The only activity that happens to the RTT estimates is a spike when the new maximum bottleneck bandwidth estimate is reached. The RTT values in this scenario for BBR and BBRv2 are shown in Figures 4.16 and 4.17. The spike in RTT mentioned can be seen at the 100 second mark when the new maximum bandwidth estimate is reached and the pipe is full. The RTT values quickly stabilise back to their original values, showing each of the algorithms abilities to adapt to this specific scenario. The same RTT activity can be seen for the other variations in Figures A3.1 - A3.6.

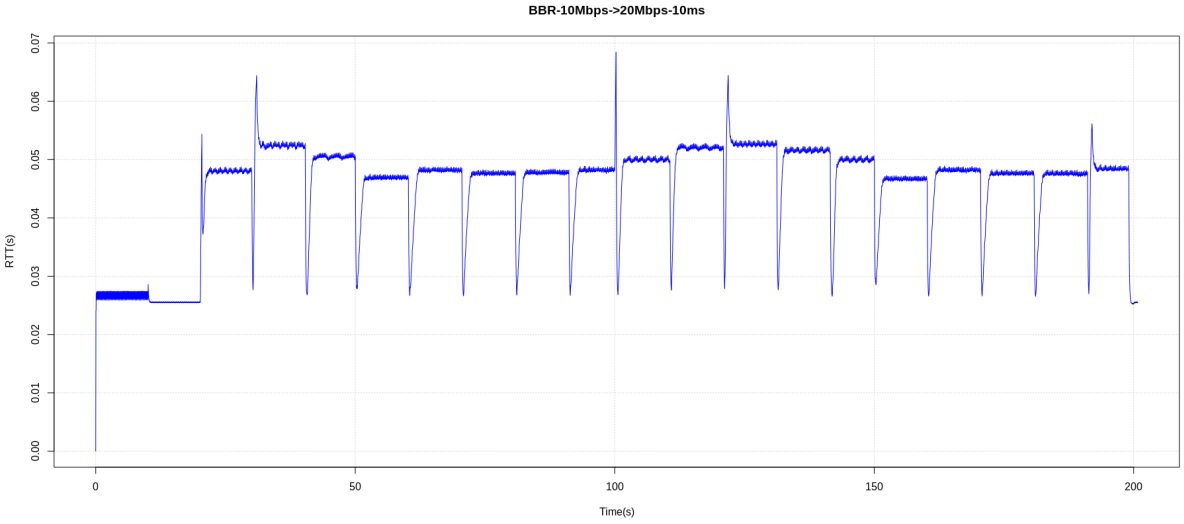

Figure 4.16: BBR RTT Responsiveness

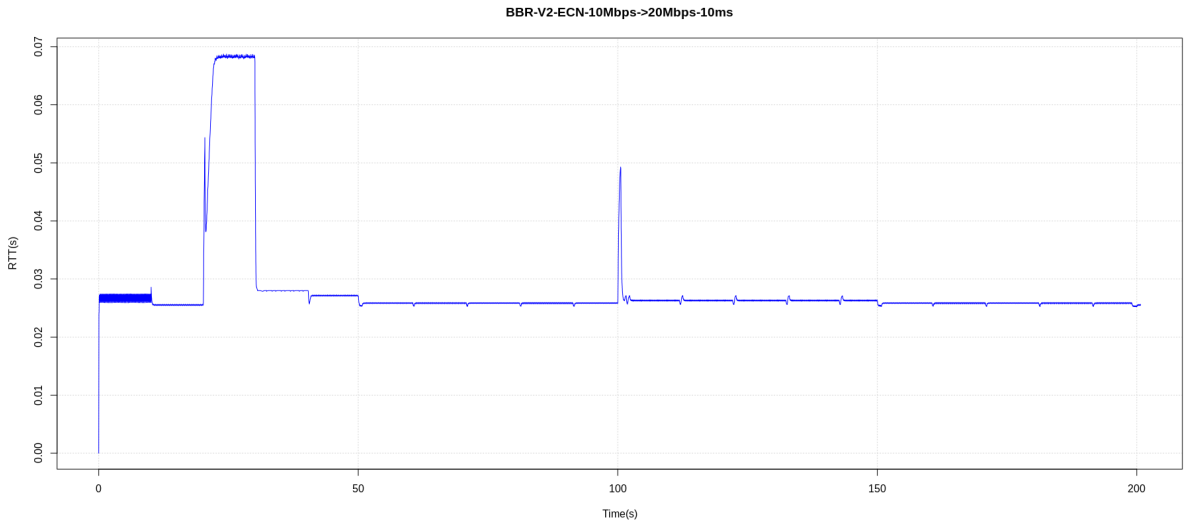

Figure 4.17: BBRv2 RTT Responsiveness

4.1.7 Single Flow Summary

The results of BBRv2 and each variant in a single flow uncontested network show values that respectively reflect the claims made by each algorithm in their unique differences from the original BBR algorithm. BBRv2 especially stays on track with its goal of keeping the similarities of the original algorithm in terms of bandwidth utilisation. The decreased RTT values of BBRv2 in comparison to BBR and the other variants shows a good improvement of the change in the ProbeBW phase of the algorithm to efficiently probe for more bandwidth without the sacrifice of RTT times. The single flow scenario itself shows the trends between results when the variable bandwidth and delays are changed. The high RTT values for configurations with higher delay apply for all variations as seen in their respective figures of RTT over the simulation time. These higher delay configurations gives an insight into a more realistic LAN network scenario where delays may vary at the bottleneck link. The results of the dynamic variance in bandwidth in the middle of a simulation to test the responsiveness of the algorithms further validate BBRv2 in keeping to the same core behaviours of the original BBR algorithm and show the consistence across all the variants.

4.2 Intra-Protocol

The simulation of more than one TCP flow in a network shows the competition between these flows for the available bandwidth. The use of the same congestion control algorithm can give an insight into how the same behaviour of these algorithms can work with each other with the given network resources. This type of simulation is performed in NS-3 with 4 flows of the same variation of BBR. This is achieved through the

increasing of the number leaves in the dumbbell topology setup. The network characteristics are kept the same as seen in Table 4.1 apart from the buffer size and simulation time. The buffer size is varied in order to test the interaction of multiple flows in a shallow, medium and large buffer. In previous studies of BBR, it is known that the algorithm experiences high packet loss in shallow buffers [9][37]. This is tied into the over estimation of the bottleneck bandwidth when probing for bandwidth in these multiple flow scenarios. The simulation time is set to 200 seconds to allow for more time for the flows to run a desirable amount of round trips.

Table 4.5: Multiple Flows Bottleneck Configuration

Bandwidth	**Delay**	**Buffer Size**
10 Mbps	10 ms	0.5 * BDP
10 Mbps	10 ms	1.5 * BDP
10 Mbps	10 ms	3 * BDP
10 Mbps	40 ms	0.5 * BDP
10 Mbps	40 ms	1.5 * BDP
10 Mbps	40 ms	3 * BDP
10 Mbps	100 ms	0.5 * BDP
10 Mbps	100 ms	1.5 * BDP
10 Mbps	100 ms	3 * BDP

The Table 4.5 displays the configurations ran in the simulations of the multiple flows of BBRv2 and the other BBR variants. The flows themselves have separated starting times of 20 second intervals. This is done to observe the convergence towards the fairness line and responsiveness of the simulated variation of BBR. Fairness itself is the most important metric taken away from these simulations with the fairness line corresponding to the throughput value that represents the equal share between the number of flows and the available bandwidth. Jain's index is used as a metric to represent fairness as a numeric value. This is calculated using equation 1 where n is the number of competing flows and $[x_1,x_2,...,x_n]$ represents the average throughput of each flow. The closer the result is to 1 the greater the fairness between the flows competing for bandwidth is.

$$J(x_1, x_2, ..., x_n) = (\sum_{i=1}^{n} x_i)^2 / n \sum_{i=1}^{n} x_i^2 \quad (1)$$

Table 4.6 displays the obtained results for these multi flow simulations, including metrics such as average throughput for each flow, goodput of all combined 4 flows as well as the total packet loss and Jain's index of fairness between the flows.

Table 4.6: Multiple Flows Results

Algorithm	Bandwidth	Delay	Average Throughput 0.5BDP	Average Throughput 1.5BDP	Average Throughput 3BDP	Goodput 0.5BDP	Goodput 1.5BDP	Goodput 3BDP	Packet Loss 0.5BDP	Packet Loss 1.5BDP	Packet Loss 3BDP	Jain's Index 0.5BDP	Jain's Index 1.5BDP	Jain's Index 3BDP
BBR	10 Mbps	10 ms	3.13 Mbps	3.35 Mbps	3.07 Mbps	8.17 Mbps	8.16 Mbps	8.15 Mbps	662 Packets	242 Packets	414 Packets	0.91	0.97	0.84
	10 Mbps	40 ms	2.45 Mbps	2.40 Mbps	3.22 Mbps	6.95 Mbps	7.64 Mbps	7.89 Mbps	1081 Packets	254 Packets	0 Packets	0.92	0.70	0.93
	10 Mbps	100 ms	1.71 Mbps	3.03 Mbps	3.03 Mbps	3.53 Mbps	7.39 Mbps	7.39 Mbps	55 Packets	0 Packets	0 Packets	0.42	0.97	0.97
BBR'	10 Mbps	10 ms	1.97 Mbps	3.34 Mbps	3.23 Mbps	8.14 Mbps	8.17 Mbps	8.16 Mbps	725 Packets	152 Packets	290 Packets	0.95	0.98	0.96
	10 Mbps	40 ms	2.67 Mbps	2.72 Mbps	3.28 Mbps	7.28 Mbps	7.70 Mbps	7.91 Mbps	964 Packets	333 Packets	0 Packets	0.95	0.90	0.94
	10 Mbps	100 ms	2.45 Mbps	3.06 Mbps	3.06 Mbps	6.09 Mbps	7.38 Mbps	7.38 Mbps	55 Packets	0 Packets	0 Packets	0.62	0.98	0.98
BBR+ ($\lambda = 0.5$)	10 Mbps	10 ms	3.00 Mbps	2.29 Mbps	2.26 Mbps	8.06 Mbps	8.16 Mbps	8.12 Mbps	3155 Packets	2268 Packets	1384 Packets	0.93	0.97	0.97
	10 Mbps	40 ms	2.76 Mbps	2.77 Mbps	3.00 Mbps	7.06 Mbps	7.64 Mbps	7.88 Mbps	3017 Packets	892 Packets	146 Packets	0.94	0.89	0.96
	10 Mbps	100 ms	2.58 Mbps	3.05 Mbps	3.05 Mbps	7.17 Mbps	7.40 Mbps	7.40 Mbps	153 Packets	0 Packets	0 Packets	0.77	0.97	0.97
BBR+ ($\lambda = 0.125$)	10 Mbps	10 ms	3.00 Mbps	2.29 Mbps	2.26 Mbps	8.06 Mbps	8.16 Mbps	8.12 Mbps	3155 Packets	2268 Packets	1384 Packets	0.93	0.97	0.97
	10 Mbps	40 ms	2.76 Mbps	2.77 Mbps	3.00 Mbps	7.06 Mbps	7.64 Mbps	7.88 Mbps	3017 Packets	892 Packets	146 Packets	0.94	0.89	0.96
	10 Mbps	100 ms	2.58 Mbps	3.05 Mbps	3.05 Mbps	7.17 Mbps	7.40 Mbps	7.40 Mbps	153 Packets	0 Packets	0 Packets	0.77	0.97	0.97
BBRPlus	10 Mbps	10 ms	3.30 Mbps	3.55 Mbps	2.13 Mbps	8.11 Mbps	8.15 Mbps	8.10 Mbps	973 Packets	683 Packets	666 Packets	0.96	0.98	0.97
	10 Mbps	40 ms	2.09 Mbps	3.06 Mbps	3.27 Mbps	6.25 Mbps	7.63 Mbps	7.89 Mbps	1075 Packets	249 Packets	0 Packets	0.76	0.92	0.95
	10 Mbps	100 ms	1.91 Mbps	3.01 Mbps	3.01 Mbps	5.44 Mbps	7.37 Mbps	7.37 Mbps	139 Packets	0 Packets	0 Packets	0.73	0.97	0.97
Delay-BBR	10 Mbps	10 ms	2.98 Mbps	3.18 Mbps	3.15 Mbps	8.21 Mbps	8.17 Mbps	8.19 Mbps	534 Packets	658 Packets	438 Packets	0.74	0.90	0.88
	10 Mbps	40 ms	2.81 Mbps	2.77 Mbps	3.32 Mbps	7.01 Mbps	7.57 Mbps	8.03 Mbps	472 Packets	224 Packets	81 Packets	0.88	0.66	0.87
	10 Mbps	100 ms	3.50 Mbps	3.27 Mbps	3.27 Mbps	6.24 Mbps	7.57 Mbps	7.57 Mbps	19 Packets	0 Packets	0 Packets	0.87	0.97	0.97
BBRv2	10 Mbps	10 ms	3.54 Mbps	3.47 Mbps	3.35 Mbps	8.08 Mbps	8.13 Mbps	8.17 Mbps	319 Packets	364 Packets	40 Packets	0.98	0.96	0.99
	10 Mbps	40 ms	2.78 Mbps	3.53 Mbps	3.53 Mbps	7.68 Mbps	7.92 Mbps	7.95 Mbps	369 Packets	0 Packets	0 Packets	0.92	0.99	0.99
	10 Mbps	100 ms	3.28 Mbps	3.28 Mbps	3.28 Mbps	7.31 Mbps	7.32 Mbps	7.32 Mbps	1 Packets	0 Packets	0 Packets	0.99	0.99	0.99
BBRv2 (EXP)	10 Mbps	10 ms	3.54 Mbps	3.47 Mbps	3.33 Mbps	8.08 Mbps	8.13 Mbps	8.08 Mbps	319 Packets	364 Packets	40 Packets	0.98	0.96	0.97
	10 Mbps	40 ms	2.78 Mbps	3.63 Mbps	3.63 Mbps	7.68 Mbps	7.78 Mbps	7.78 Mbps	369 Packets	0 Packets	0 Packets	0.92	0.99	0.99
	10 Mbps	100 ms	3.50 Mbps	3.50 Mbps	3.50 Mbps	7.28 Mbps	7.28 Mbps	7.28 Mbps	0 Packets	0 Packets	0 Packets	0.99	0.99	0.99

4.2.1 BBR and BBR'

It is clear from the results gathered in Table 4.6 for BBR and BBR' that despite the small difference in functionality between them, these multi flow simulations display different results. The problems around BBR suffering in lower sized buffers remain true [9][38], as seen in the high packet loss rates of both BBR and BBR' for a buffer of size half of the BDP value. The BBR' variant shows better fairness overall with high Jain index values representing a good share of the bandwidth between the four flows.

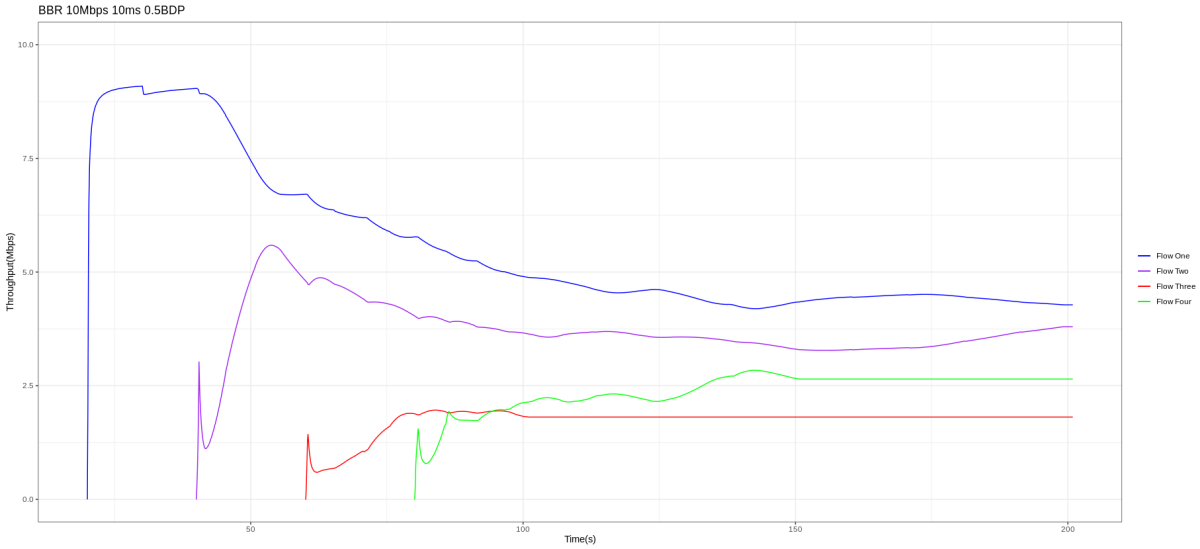

Figure 4.18: BBR Multi Flow Throughput

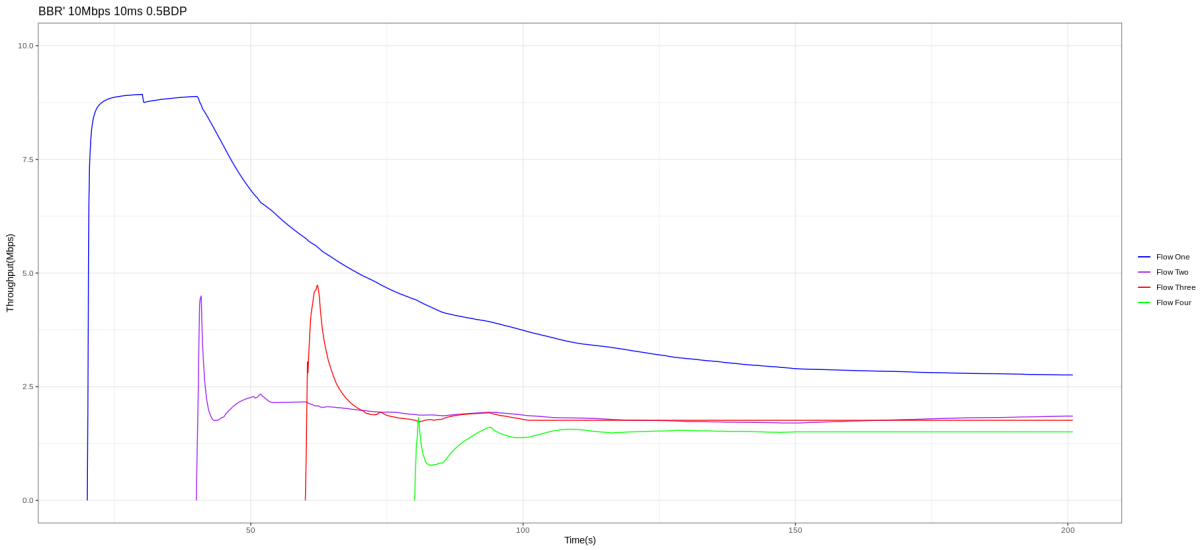

Figure 4.19: BBR' Multi Flow Throughput

The difference in the time between the advancement of the gain cycling in the ProbeBW phase in both algorithms is the root cause of the difference in fairness

between the two algorithms with BBR taking one whole round trip to advance in comparison to the case of BBR' where the in-flight rate is seen to have a decrease on the previous round trip. This means that the queue was drained earlier in BBR' resulting in a lower queue delay. The overestimation of the bottleneck issue that is found to be a big flaw of the BBR algorithm in multi flow scenarios [9] can be seen in Figures 4.18 and 4.19. The throughput values over time of both algorithms show the sum of all flows operating above the actual bottleneck bandwidth of 10Mbps. This results in an increased amount of packet loss since BBR does not consider loss as a method of congestion detection in the network.

Hence, there is no response to packet loss which occurs because of a full queue at the buffer. This loss over time for each flow for both BBR and BBR' can be seen in Figures 4.20 and 4.21.

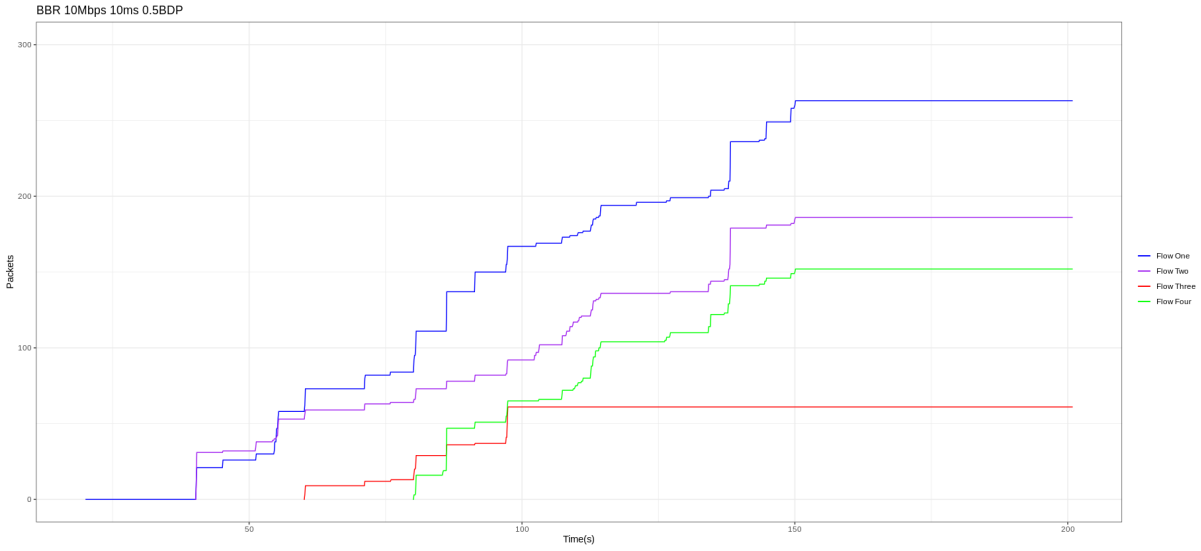

Figure 4.20: BBR Multi Flow Loss

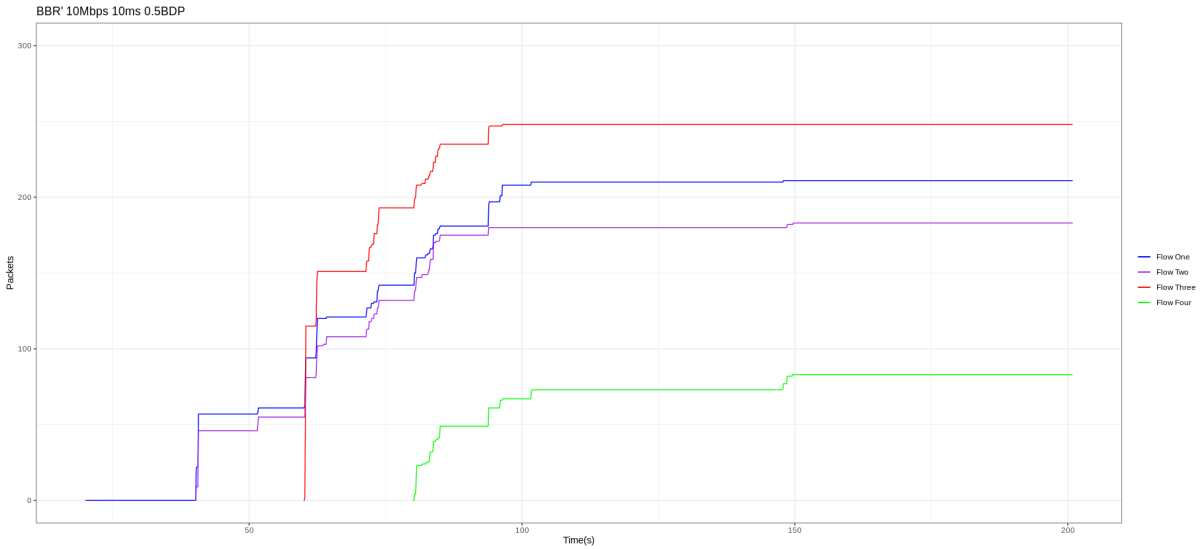

Figure 4.21: BBR' Multi Flow Loss

4.2.2 BBR+

BBR+ exhibits massive packet loss in the results of the simulation of the algorithm in a multiple flow scenario. This is due to the aggressiveness of its probing strategy having gain cycle values of 1.5 every second cycle, meaning the pacing rate is the estimated bandwidth multiplied by 1.5 every 4 cycles of the 8 cycle ProbeBW phase. Since there is no change in consideration for packet loss in this variation of the BBR algorithm, packets are dropped at an alarming rate. Figures 4.22 and 4.23 displays this over the 200 second simulation time.

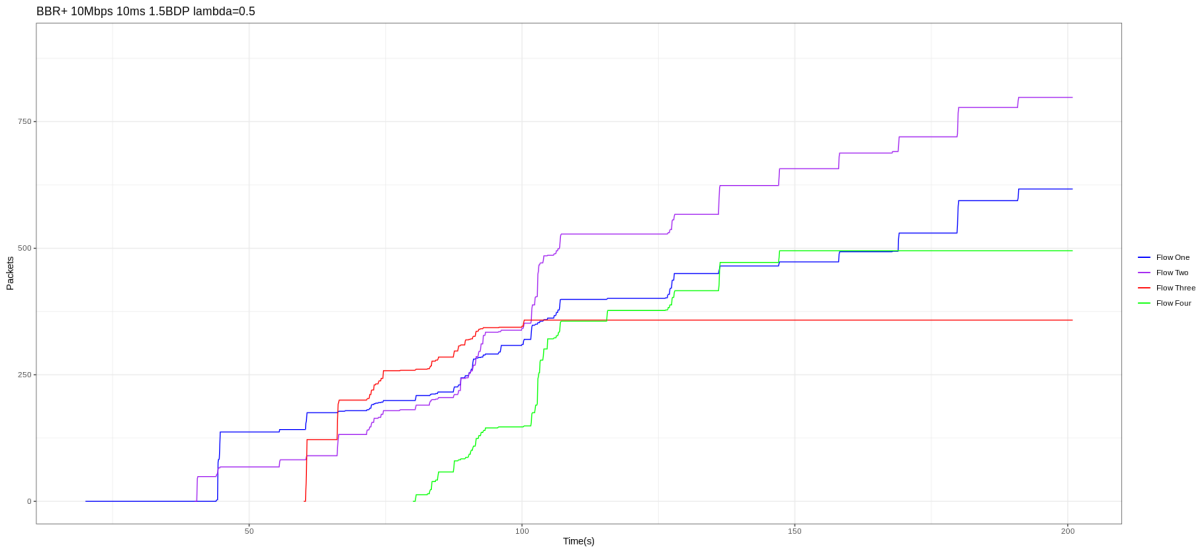

Figure 4.22: BBR+ Multi Flow Loss $\lambda = 0.5$

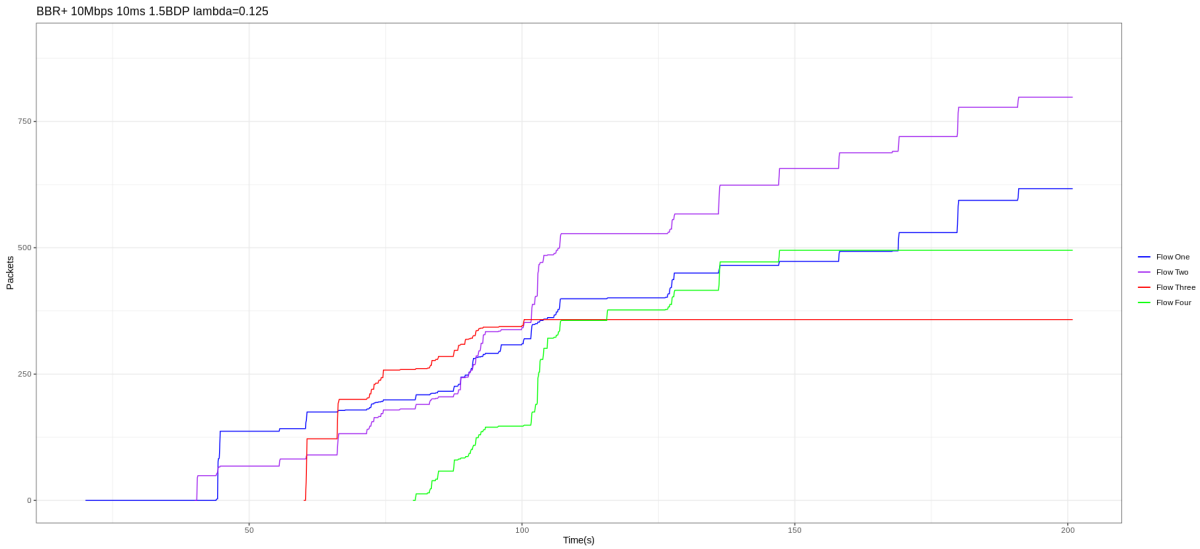

Figure 4.23: BBR+ Multi Flow Loss $\lambda = 0.125$

Once again, there is no difference in the change of the lambda shape parameter value involved in the estimation of the propagated RTT for BBR+, contrary of the results of the evaluations performed in the introduction of the variation [24]. It is believed that the different estimations of the RTT by the two shape parameter values do not have a impact on the variance of the BDP and give the same outcome. In Zhang et al.'s evaluation of BBR+ the results of the rate dynamics of BBR+ for a medium sized buffer shown in Figure 11 of the paper [22] match that of the results obtained in this evaluation, as seen in Figures 4.24 and 4.25, validating the behaviours of the algorithm in this type of scenario.

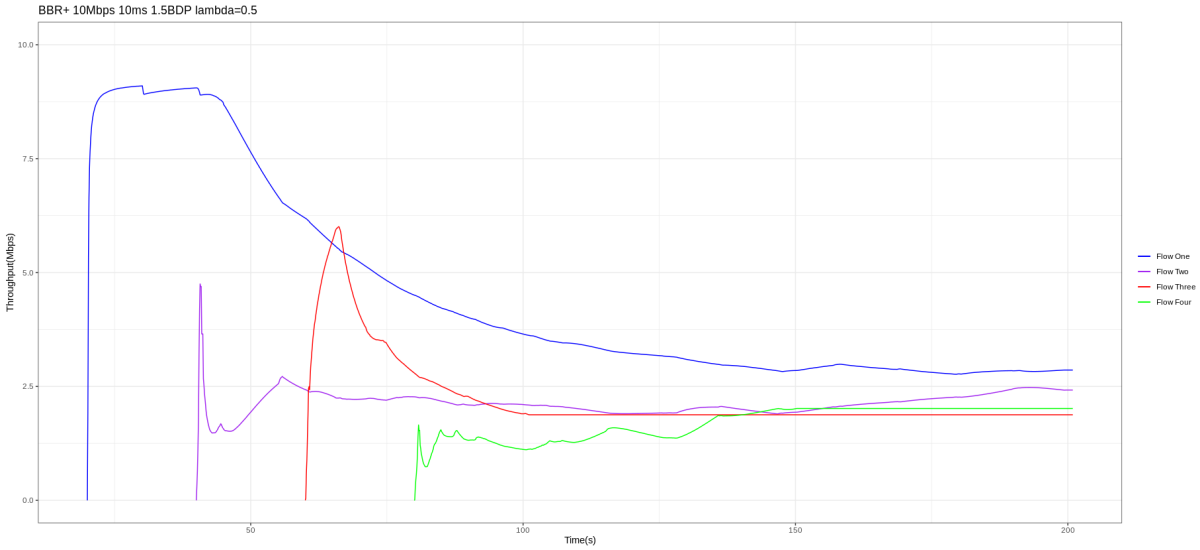

Figure 4.24: BBR+ Multi Flow Throughput $\lambda = 0.5$

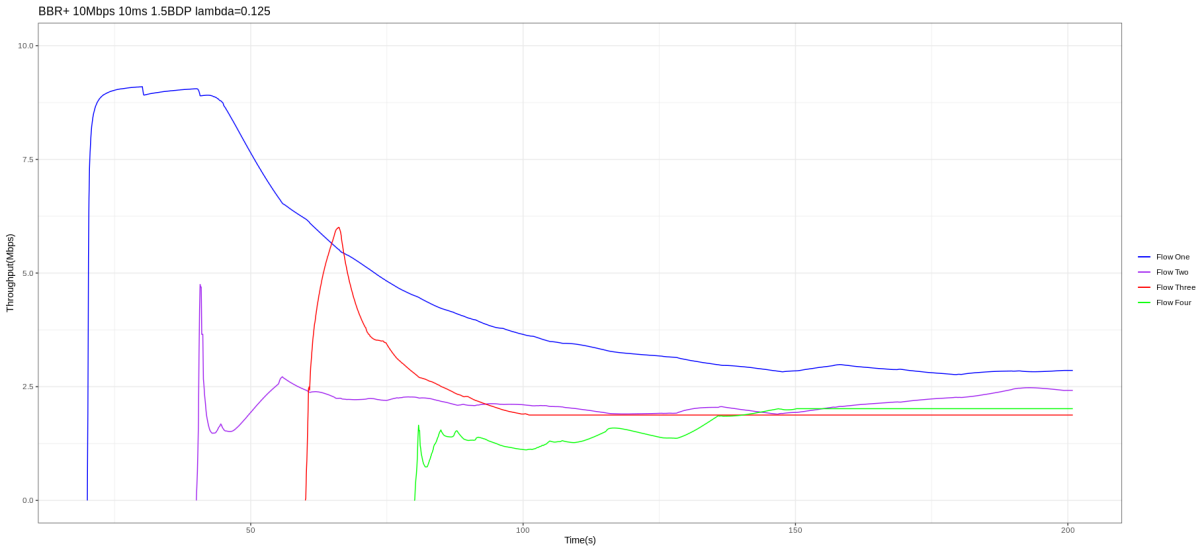

Figure 4.25: BBR+ Multi Flow Throughput $\lambda = 0.125$

4.2.3 BBRPlus

The evaluation of the BBRPlus variation of BBR over a network with multiple flows with the same congestion control give a good indication of a steady convergence towards the fairness line as outlined in Figure 4.26 but with the sacrifice of dropping packets similar to that of BBR. The change in the behaviour of the algorithm includes the consideration of packet loss as a condition to reduce the sending rate in the ProbeBW phase but this has ill-effect as seen in the values of packet loss in Table 4.6. Good fairness shown by Jain's index is clouded by the poor packet loss of this variation of BBR, not reaching the goal set in its introduction to reduce the packet loss of the congestion control [23].

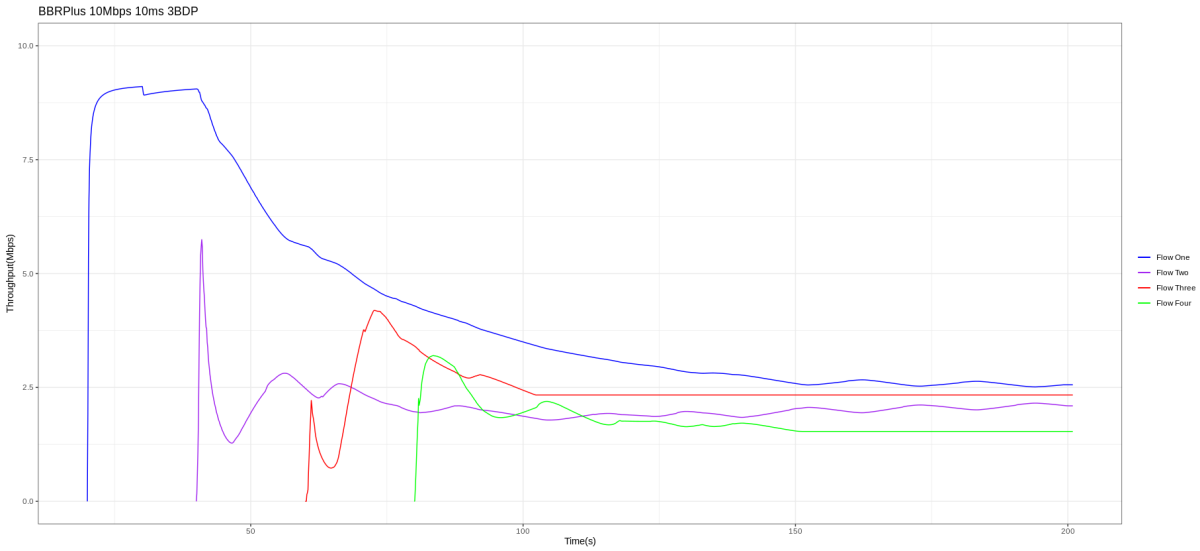

Figure 4.26: BBRPlus Multi Flow Throughput

4.2.4 Delay-BBR

In Figure 6 and 8 of the introductory paper of the Delay-BBR variation as an alternative to BBR for video streaming, the rate dynamics of the algorithm are shown [25]. These results show stability in the multiple flows and a convergence towards the fairness line. The evaluation performed in this project using the same algorithm achieves similar results, with a high degree of fairness indicated by Jain's index and a visual convergence of the flows, shown in Figure 4.27, matching the plots in the original paper. Delay-BBR shows some improvement in the overestimation of the bottleneck bandwidth in shallow and medium sized buffers in the values for the average throughput but unfortunately it is not enough to convince that the variation has solved the ongoing issue found by Hock et al. [9].

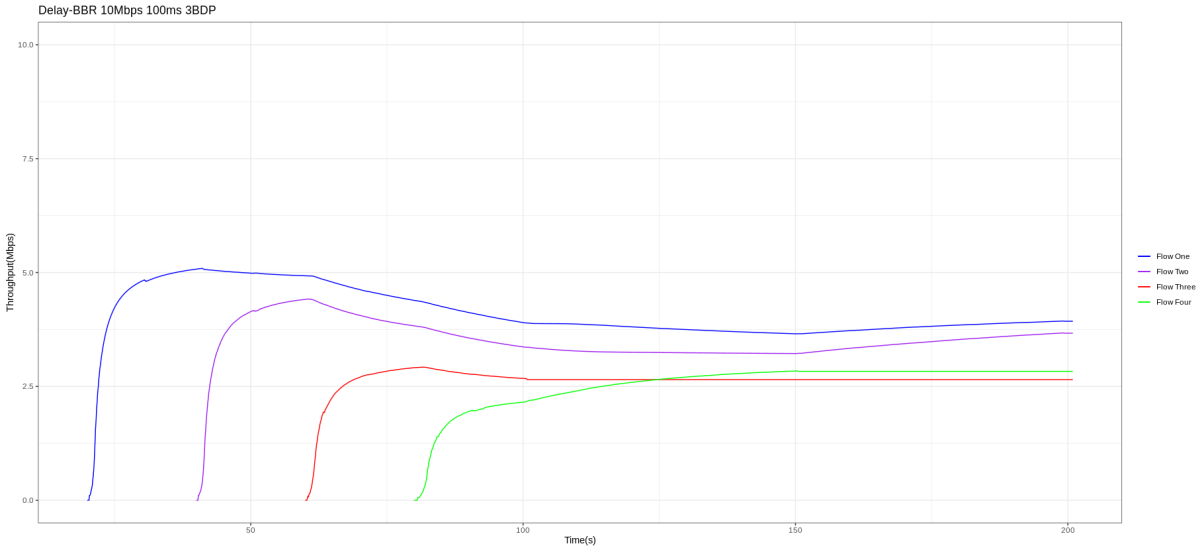

Figure 4.27: Delay-BBR Multi Flow Throughput

This BBR variation shows a good improvement of a decrease in packet loss experienced at the bottleneck when the buffer is under the estimated BDP value. This is presented in the plot 4.28 where in comparison to the variations there is a less aggressive increase in the loss of packets. This is due to the combination of the use of the additional congestion signal based on round trip values and the less progressive pacing gain values in the ProbeBW phase. When the queue at the buffer fills up, the round trip times increase and in the case of Delay-BBR this enables the smoothed RTT value that is monitored during the ProbeBW phase to increase and reach a point where it is greater than the base line RTT in the ProbeBW phase, enabling the congestion delay signal. The ProbeRTT phase is entered consequently with the pacing rate set to drain any excess packets at the buffer hence preventing the excess packet loss.

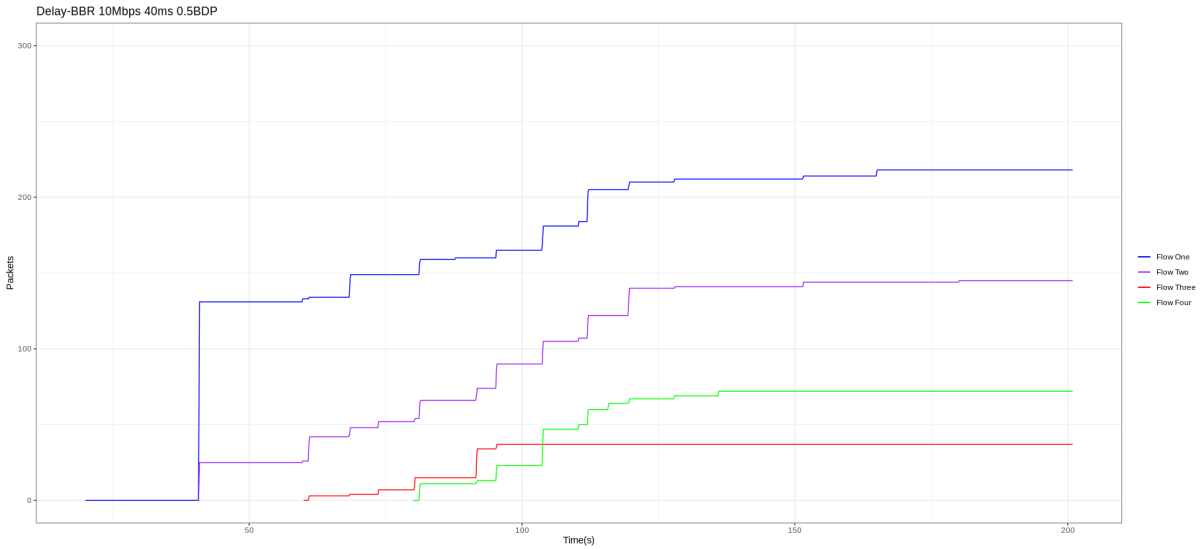

Figure 4.28: Delay-BBR Multi Flow Loss

4.2.5 BBRv2

The balance of probing for more bandwidth whilst avoiding excessive bottleneck congestion is something that BBRv2 manages very well in the simulation tests with multiple flows of the same congestion control. Table 4.6 shows the lowest packet loss values and impressive fairness indexes for all buffer sizes, outperforming all the other variations and most importantly the original BBR algorithm. Since the introduction of the new version of BBR, one of the main goals was to overcome the issue that BBR has with packet loss in shallow buffers [2]. The Figures 4.29 and 4.30 of the packet loss over a buffer that is less than the BDP value reflects the improvement to reach this goal. This is down to the introduction of both packet loss and ECN as an additional signal for congestion detection in the algorithm, reducing the sending rate appropriately when either is detected at the bottleneck link.

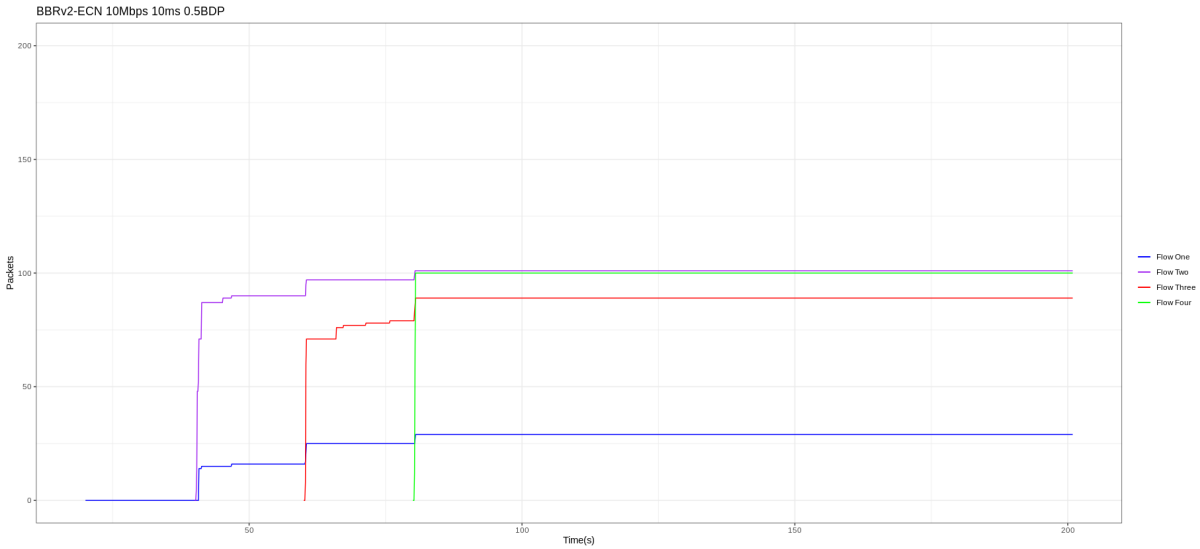

Figure 4.29: BBRv2 Multi Flow Loss

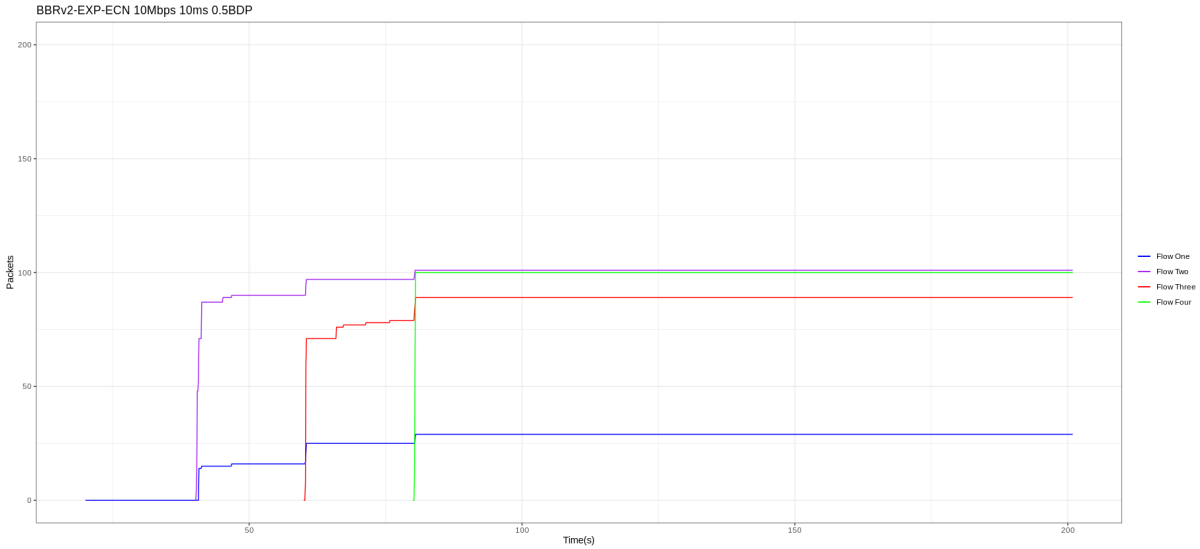

Figure 4.30: BBRv2-EXP Multi Flow Loss

The experimental changes are enabled and ran with the same simulation configuration to compare the differences made. The results show that there is little to no difference in performance similar to that of the single flow test cases. The upper and lower bounds that are part of the three part model in the newest version of BBR are used to manage the area in which the bandwidth and in-flight rate can operate. They are set to converge on the fairness line with a good distribution of the available bandwidth between all the flows in these simulated scenarios. Figures 4.31 and 4.32 shows the fairness between the flows of BBRv2 for a small sized buffer. These plots show how the lines representing each flow converge towards each other, as each flow is introduced, to fairly share the available bandwidth.

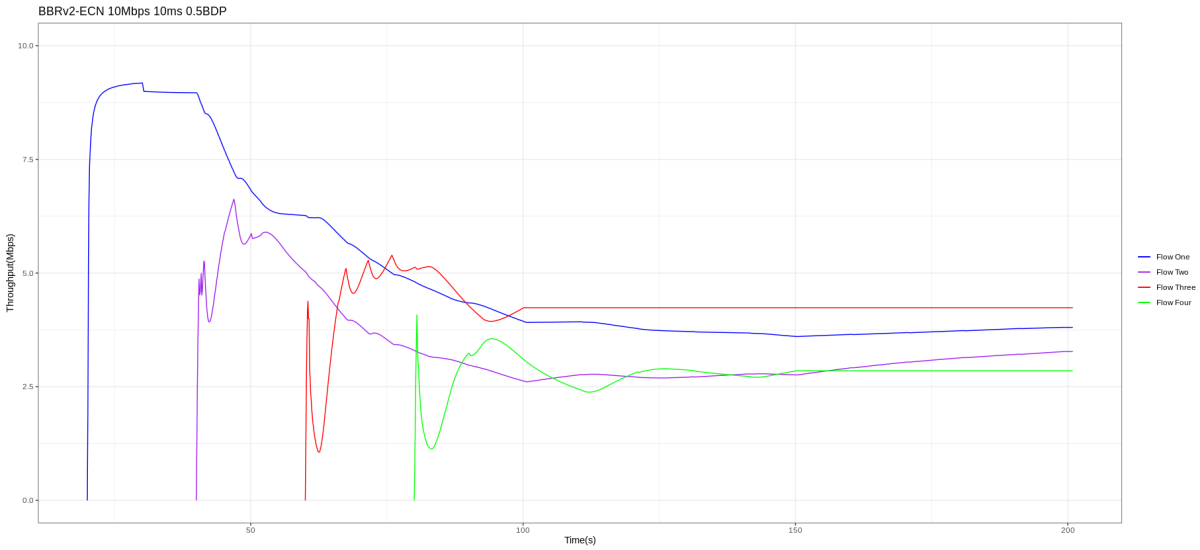

Figure 4.31: BBRv2 Multi Flow Throughput

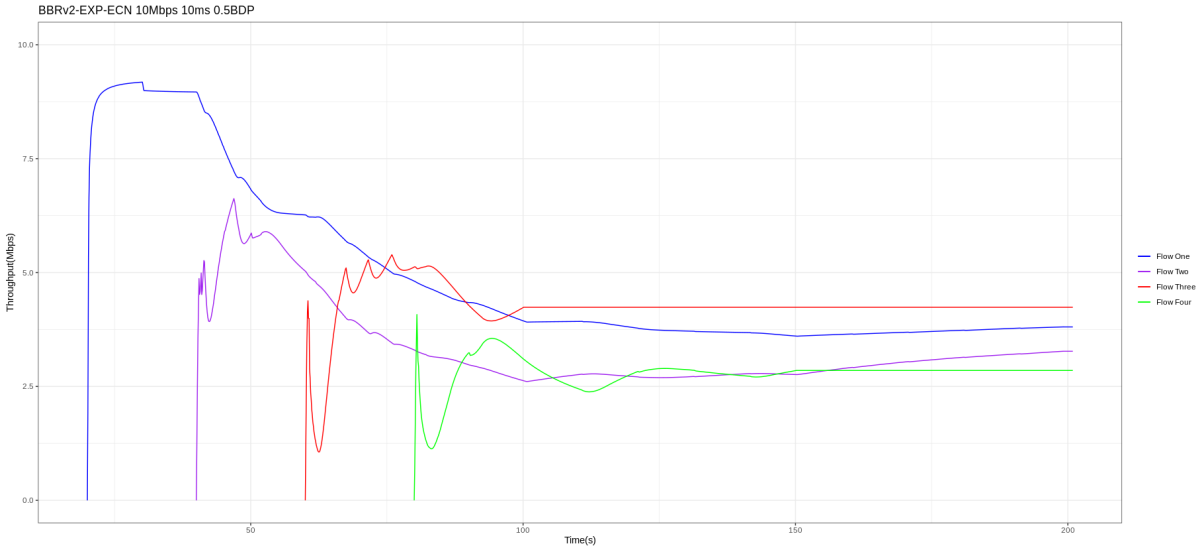

Figure 4.32: BBRv2-EXP Multi Flow Throughput

4.2.6 Intra-Protocol Summary

The simulations of these contested networks using the same congestion control algorithm displays results reflecting the claims of these algorithms along with the residing issues of some. Delay-BBR stays consistent in being a good improvement over the original BBR algorithm showing an all round increase in fairness and less packet loss. BBRv2 takes the spotlight in showing vast improvement over BBR. The increase of delay in the simulated scenarios sees a trend in a decrease in packet loss and greater fairness across the results for BBRv2 and all the other variations. This increase in delay allows for more retransmissions of packets with no received acknowledgements allowing for the reduction in drop rate for those packets. The goodput is consistently high for all

the variations of BBR despite there being difference in packet loss and fairness between the results of the simulations. The existence of the problems of performance in a shallow buffer still persists in the majority of the variations apart from BBRv2, this gives a good indication of the effectiveness of the changes the algorithm has. Packet loss occurs in the other algorithms because of the inability to deal with the filling of the queue at the buffer which forces the dropping of these packets when the buffer overflows. The great performance of BBRv2 in these multi flow scenarios overcoming the fairness problems and the shallow buffer issues of its predecessor shows the progression towards its initial goals of increased fairness across multiple flows and becoming an overall improvement over BBR [2]. The only issue that is predominant with BBRv2 and all the other variations of BBR is the overestimation of the bottleneck bandwidth value as seen in the product of the average throughput and number of flows in each configuration in Table 4.6. This is why some packet loss still exists in low delay networks, despite BBRv2 explicitly having loss as a clear indicator of its need to adjust the three part model that it adopts. This feature is included ultimately to reduce the sending rate to a point that stabilises the bandwidth consumption of any flows using the new version of BBR as a congestion control.

4.3 Inter-Protocol

Real world networks can have TCP connections using numerous different types of congestion control algorithms. It is important for these congestion controls to operate well together and share the available resources evenly. An inter-protocol simulation scenario is set up in NS-3 to evaluate BBRv2 and the other BBR variants against a loss-based and a delay-based congestion control algorithm. The loss-based and delay-based algorithms of choice respectively are TCP Cubic [12] and TCP Vegas [18]. One flow of a chosen BBR variant is pitched against one flow of Cubic or Vegas.

Table 4.7: Vegas Results

Algorithm	Bandwidth	Delay	Average Throughput (BBR/Vegas Ratio)	Goodput	Packet Loss	Jain's Index
BBR	10 Mbps	10 ms	4.54 Mbps (17.37)	8.06 Mbps	0 Packets	0.56
	10 Mbps	40 ms	4.55 Mbps (21.91)	7.80 Mbps	0 Packets	0.55
	10 Mbps	100 ms	4.39 Mbps (1.42)	7.15 Mbps	0 Packets	0.97
BBR'	10 Mbps	10 ms	4.51 Mbps (9.07)	8.07 Mbps	0 Packets	0.61
	10 Mbps	40 ms	4.56 Mbps (14.02)	7.81 Mbps	0 Packets	0.57
	10 Mbps	100 ms	4.39 Mbps (1.42)	7.15 Mbps	0 Packets	0.97
BBR+ ($\lambda = 0.5$)	10 Mbps	10 ms	4.54 Mbps (19.00)	8.04 Mbps	61 Packets	0.55
	10 Mbps	40 ms	4.55 Mbps (25.18)	7.80 Mbps	0 Packets	0.54
	10 Mbps	100 ms	4.26 Mbps (1.51)	7.14 Mbps	0 Packets	0.96
BBR+ ($\lambda = 0.125$)	10 Mbps	10 ms	4.54 Mbps (19.00)	8.04 Mbps	61 Packets	0.55
	10 Mbps	40 ms	4.55 Mbps (25.18)	7.80 Mbps	0 Packets	0.54
	10 Mbps	100 ms	4.26 Mbps (1.51)	7.14 Mbps	0 Packets	0.96
BBRPlus	10 Mbps	10 ms	4.55 Mbps (18.20)	8.06 Mbps	0 Packets	0.55
	10 Mbps	40 ms	4.56 Mbps (21.21)	7.81 Mbps	0 Packets	0.55
	10 Mbps	100 ms	4.38 Mbps (1.42)	7.14 Mbps	0 Packets	0.97
Delay-BBR	10 Mbps	10 ms	4.71 Mbps (0.06)	7.54 Mbps	0 Packets	0.56
	10 Mbps	40 ms	4.72 Mbps (21.02)	8.02 Mbps	0 Packets	0.55
	10 Mbps	100 ms	4.45 Mbps (1.44)	7.22 Mbps	0 Packets	0.97
BBRv2	10 Mbps	10 ms	4.60 Mbps (1.79)	8.10 Mbps	0 Packets	0.93
	10 Mbps	40 ms	4.62 Mbps (4.30)	7.84 Mbps	0 Packets	0.72
	10 Mbps	100 ms	4.36 Mbps (1.46)	7.11 Mbps	0 Packets	0.97
BBRv2 (EXP)	10 Mbps	10 ms	4.60 Mbps (1.79)	8.10 Mbps	0 Packets	0.93
	10 Mbps	40 ms	4.62 Mbps (4.30)	7.84 Mbps	0 Packets	0.72
	10 Mbps	100 ms	4.36 Mbps (1.46)	7.11 Mbps	0 Packets	0.97

The network attributes stay consistent to those set in the single flow simulations, seen in Table 4.2, with the difference of a longer simulation time of 200 seconds to allow for the contesting flows to fully settle before being observed. The constraints for the bottleneck bandwidth and delay are set to the configurations, seen in Table 4.4, for each test case. The contesting Cubic or Vegas flow is set to begin 20 seconds after the BBR variant of choice has began. The network metrics being observed in these test cases match those studied in the intra-protocol scenarios with Jain's index being estimated using equation 1. An additional observation to the results obtained is the ratio of the two throughputs against each other in order to gauge a gap in the throughput values with respect to the assigned bottleneck bandwidth. Tables 4.7 and 4.8 show the gathered results for these simulations for contested networks with Vegas and Cubic respectively.

Table 4.8: Cubic Results

Algorithm	Bandwidth	Delay	Average Throughput (BBR/Cubic Ratio)	Goodput	Packet Loss	Jain's Index
BBR	10 Mbps	10 ms	4.89 Mbps (1.14)	8.14 Mbps	1056 Packets	0.99
	10 Mbps	40 ms	4.96 Mbps (0.98)	8.02 Mbps	0 Packets	1
	10 Mbps	100 ms	4.64 Mbps (0.97)	7.46 Mbps	0 Packets	1
BBR'	10 Mbps	10 ms	4.95 Mbps (0.63)	8.12 Mbps	1109 Packets	0.95
	10 Mbps	40 ms	5.01 Mbps (0.69)	8.02 Mbps	0 Packets	0.97
	10 Mbps	100 ms	4.64 Mbps (0.97)	7.46 Mbps	0 Packets	1
BBR+ ($\lambda = 0.5$)	10 Mbps	10 ms	3.36 Mbps (1.39)	8.10 Mbps	2348 Packets	0.97
	10 Mbps	40 ms	4.96 Mbps (0.97)	8.02 Mbps	0 Packets	1
	10 Mbps	100 ms	4.63 Mbps (0.97)	7.45 Mbps	0 Packets	1
BBR+ ($\lambda = 0.125$)	10 Mbps	10 ms	3.36 Mbps (1.39)	8.10 Mbps	2348 Packets	0.97
	10 Mbps	40 ms	4.96 Mbps (0.97)	8.02 Mbps	0 Packets	1
	10 Mbps	100 ms	4.63 Mbps (0.97)	7.45 Mbps	0 Packets	1
BBRPlus	10 Mbps	10 ms	4.76 Mbps (1.06)	8.14 Mbps	1240 Packets	1
	10 Mbps	40 ms	4.96 Mbps (1.03)	8.02 Mbps	0 Packets	1
	10 Mbps	100 ms	4.63 Mbps (0.96)	7.45 Mbps	0 Packets	1
Delay-BBR	10 Mbps	10 ms	4.82 Mbps (0.04)	7.46 Mbps	369 Packets	0.54
	10 Mbps	40 ms	4.98 Mbps (1.07)	8.04 Mbps	0 Packets	1
	10 Mbps	100 ms	4.73 Mbps (0.99)	7.58 Mbps	0 Packets	1
BBRv2	10 Mbps	10 ms	5.09 Mbps (0.24)	8.12 Mbps	822 Packets	0.73
	10 Mbps	40 ms	5.12 Mbps (0.26)	8.02 Mbps	0 Packets	0.74
	10 Mbps	100 ms	4.66 Mbps (0.80)	7.46 Mbps	0 Packets	0.99
BBRv2 (EXP)	10 Mbps	10 ms	5.09 Mbps (0.24)	8.12 Mbps	822 Packets	0.73
	10 Mbps	40 ms	5.12 Mbps (0.26)	8.02 Mbps	0 Packets	0.74
	10 Mbps	100 ms	4.66 Mbps (0.80)	7.46 Mbps	0 Packets	0.99

4.3.1 BBR and BBR'

4.3.1.1 Delay-Based

Previous work that has evaluated BBR with a delay-based algorithm like TCP Vegas has shown poor performance of the algorithm with low fairness of bandwidth [38]. This is replicated in the simulations performed in this evaluation represented by the low

fairness index values for both BBR and BBR' in Table 4.7. Figures 4.33 and 4.34 show the poor distribution of bandwidth between the two flows for BBR and BBR'. The split of bandwidth for BBR' is slightly better than BBR but with not enough impact to register as a fair distribution.

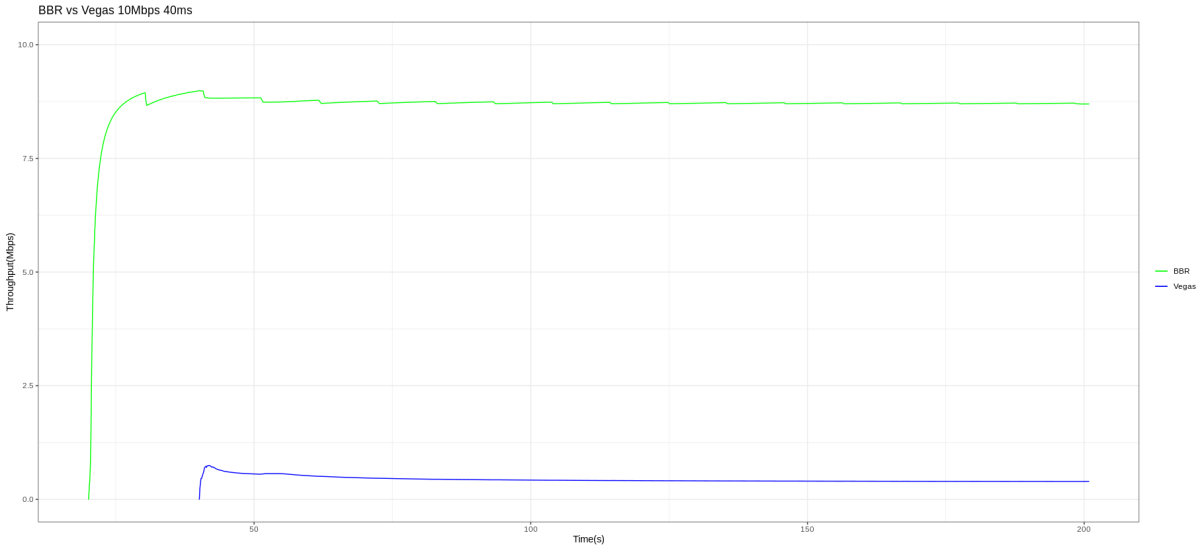

Figure 4.33: BBR vs Vegas Throughput

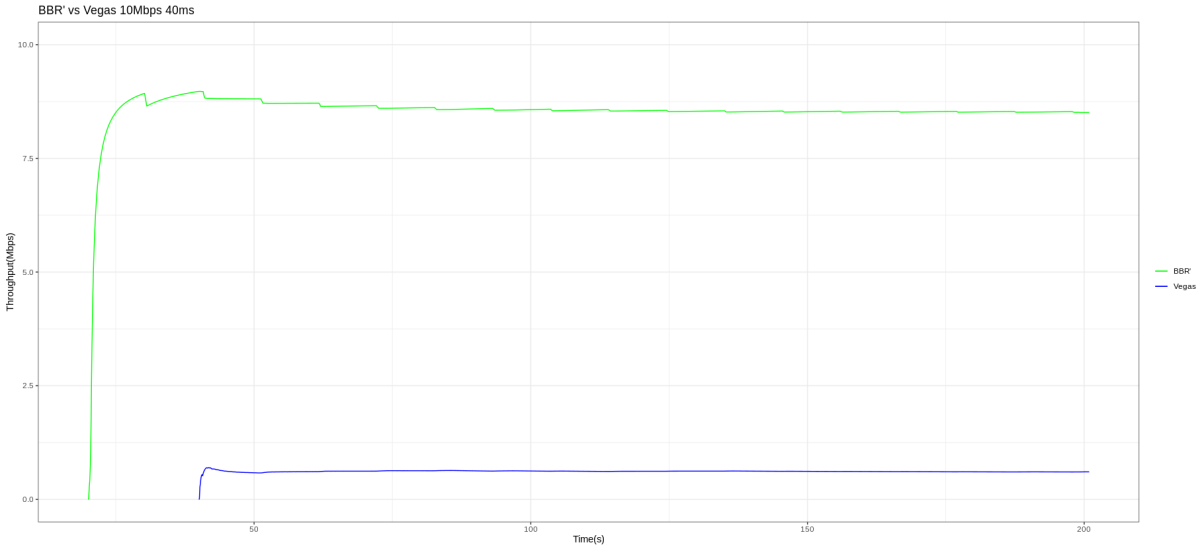

Figure 4.34: BBR' vs Vegas Throughput

This type of behaviour occurs with Vegas due to a constant queue being maintained at the bottleneck, with BBR sending at a high estimated bottleneck bandwidth value that is measured due to Vegas lowering its congestion window when delay is detected. This queue is unable to be drained as when BBR is in ProbeRTT, Vegas increases its congestion window size. This is because it measures lower RTT values so it deems it safe to do so. This increase in congestion window size along with BBR returning to a

high sending rate after the ProbeRTT phase keeps the queue constantly filled. Thus, fairness values are excessively low between the two congestion control algorithms.

4.3.1.2 Loss-Based

BBR and Cubic are the main two congestion control algorithms put together for evaluations of inter-protocol behaviour in previous studies [9][38][39][10][37][40]. The conclusion from these studies is that in shallow buffer scenarios, BBR and Cubic exhibit poor fairness and have a trend to show an oscillating behaviour between the competing flows. In the case of medium to large buffer sizes (1.5 BDP and above) there is fairness between the competing congestion control algorithms. The results obtained in this simulation reflect that since a medium sized buffer is used. There is a very high Jain's index in the case of both BBR and BBR' but with BBR having a more even ratio split of throughput between the low delay scenarios seen in Table 4.8.

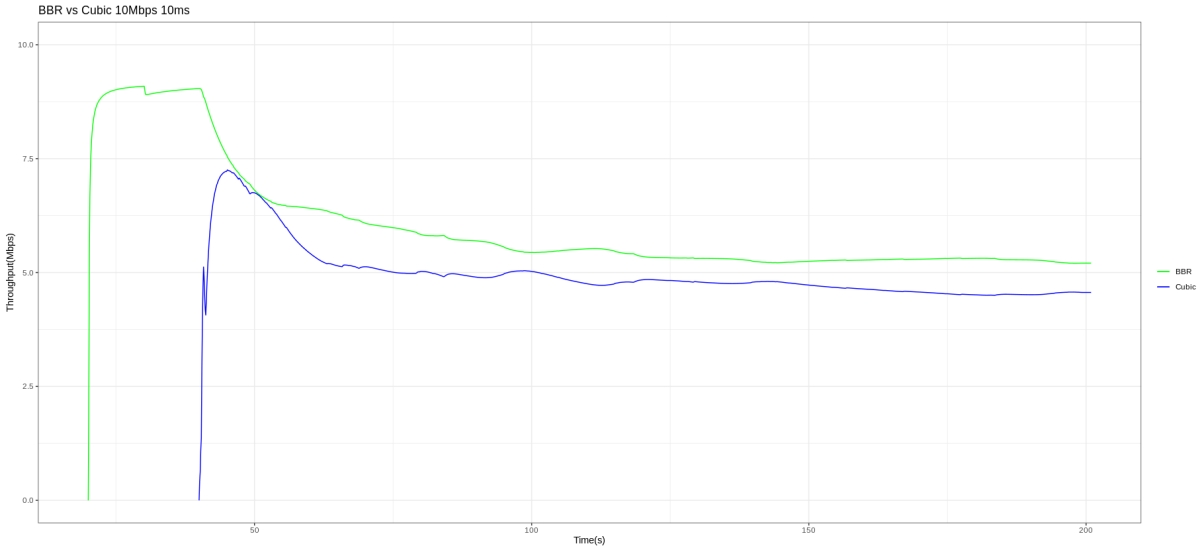

Figure 4.35: BBR vs Cubic Throughput

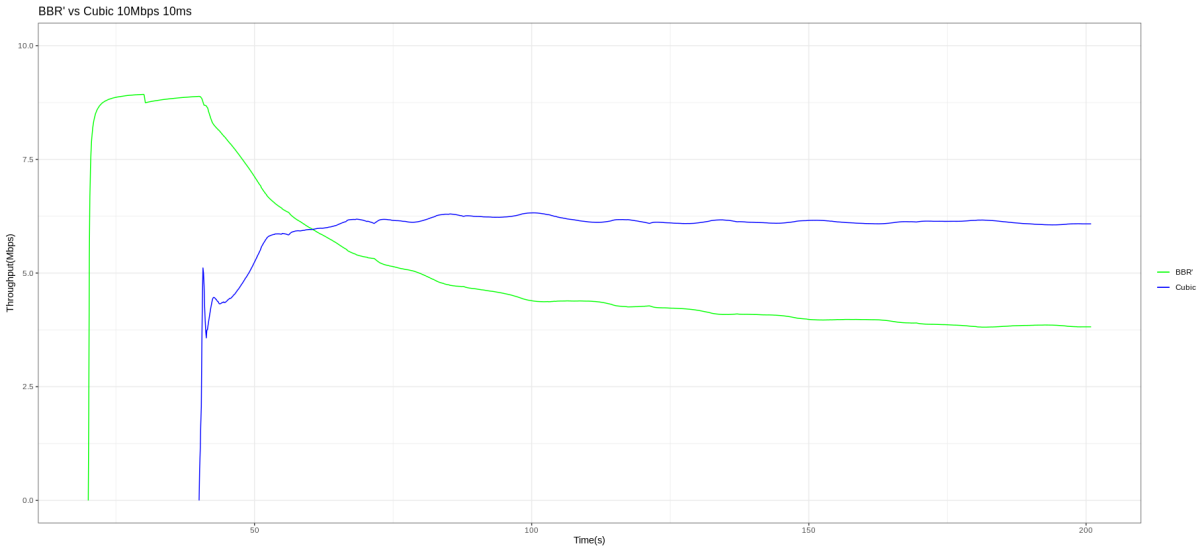

Figure 4.36: BBR' vs Cubic Throughput

High packet loss is evident due to conflicts between the two different bandwidth probing strategies, Cubic fills up the buffers quickly whilst BBR takes measurements of the propagated RTT, which ends up being a high value as the buffers are full. This results in packet loss and a reduced sending rate for Cubic whilst queues at the buffers are being drained. The constant filling up of the queues at the bottleneck leads to increased time gap between BBR's ability to probe for more bandwidth in the ProbeBW phase. Since the primary difference between BBR and BBR' is the methodology in advancing the cycle of probing, the reason why BBR outperforms BBR' is because the queues are already full in the case of BBR' when probing and it relies on the previous bytes in flight to be lower than that of the current bytes in flight as a condition to advance. This condition will not hold because the queues are full and results in a longer time between the advancement in the gain cycle. Figures 4.35 and 4.36 shows the difference in the throughput ratio between BBR and the BBR' variant.

4.3.2 BBR+

4.3.2.1 Delay-Based

BBR+ has its functionality changed to deal with high speed environments but does not contain changes to deal with the contesting of network resources with other congestion control algorithms like Vegas. Along with this, BBR+ has a more aggressive gain cycle increase in ProbeBW phase so the results reflect a worsened fairness metric than that of BBR. A higher maximum bottleneck bandwidth estimate value is set because the pacing gain probe up value is 1.5 and this bandwidth estimate is the constant value that the sending rate is set too leaving little bandwidth for Vegas to use. The performance of BBR+ also introduces packet loss in a low delay bottleneck from the

buffer overflowing due to the constant queue. The gap between the flows is displayed in the Figures 4.37 and 4.38 of BBR+, each with different values for lambda used in the calculation of the propagated RTT. Following from the trend of the previous results, the lambda has no effect on the obtained results for BBR+ with Vegas.

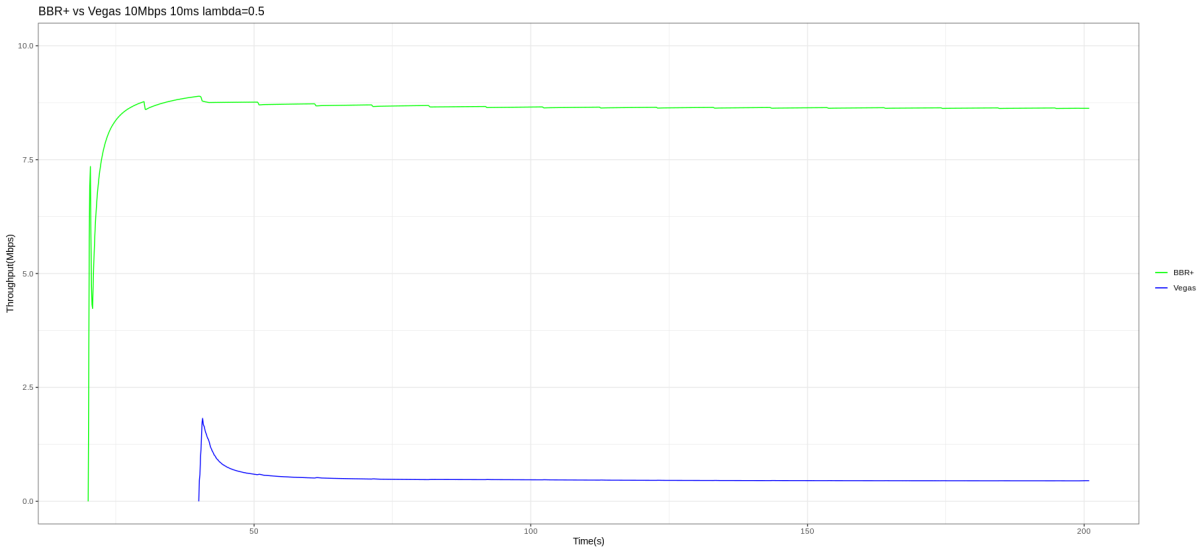

Figure 4.37: BBR+ vs Vegas Throughput $\lambda = 0.5$

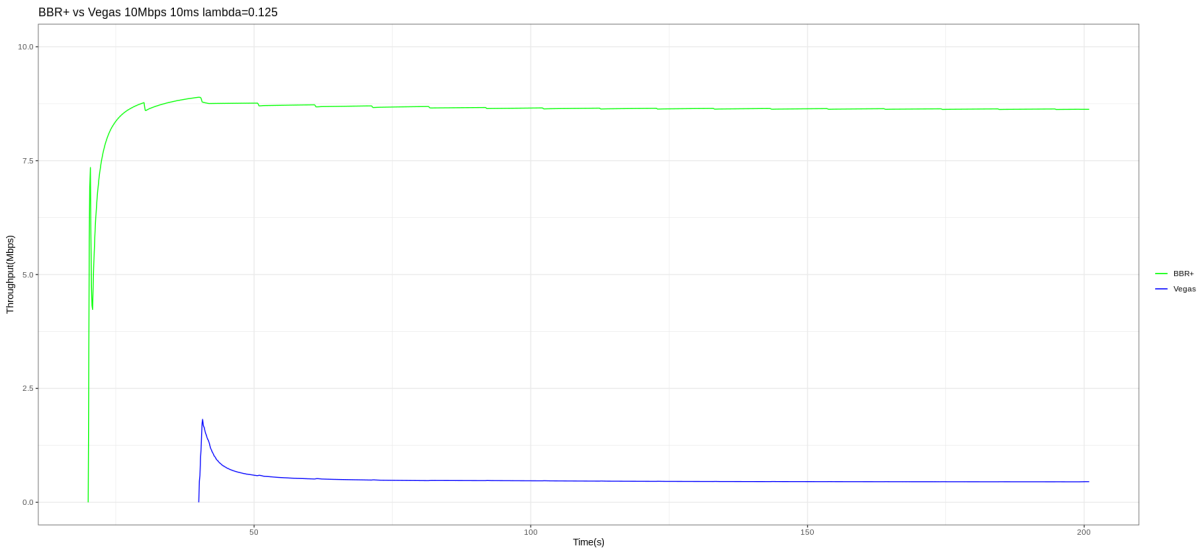

Figure 4.38: BBR+ vs Vegas Throughput $\lambda = 0.125$

4.3.2.2 Loss-Based

When simulated with a Cubic flow, BBR+ continues the trend of amplifying on the results obtained for BBR due to its more aggressive nature, shown in Table 4.8. Very high packet loss is exhibited in the scenario where there is a 10 ms delay but keeps a high fairness index. The scenarios with a higher delay at the bottleneck provide an ideal

state of no packet loss and a perfect amount of fairness. This can be seen in Figures 4.39 and 4.40 where both flows converge very close to the fairness line. Higher delay scenarios are producing these results as more time is being spent in the ProbeRTT phase, allowing for any packets built up in the queues at the buffers to be drained.

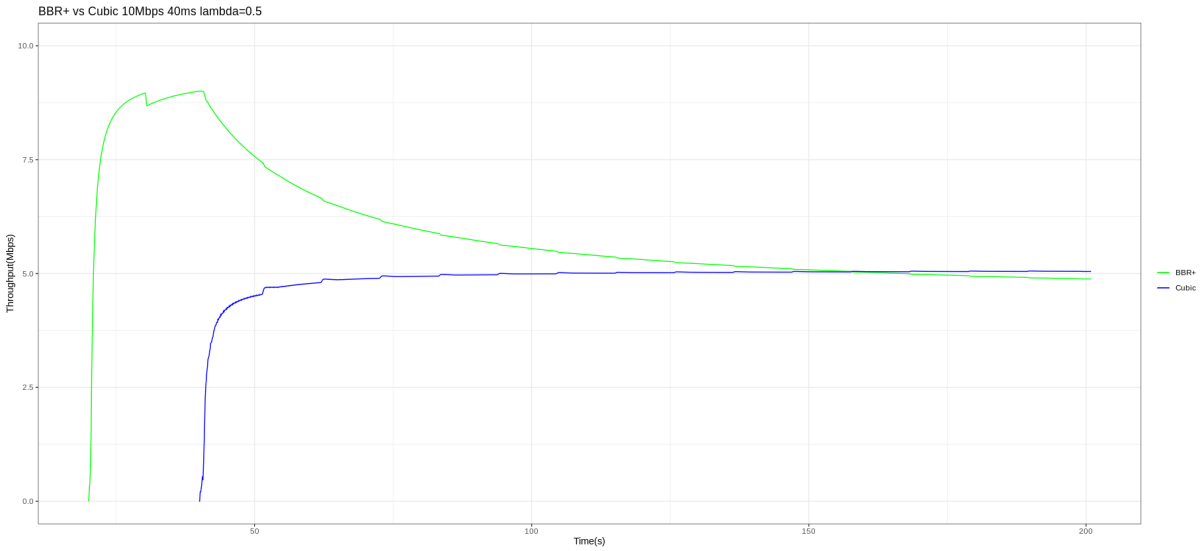

Figure 4.39: BBR+ vs Cubic Throughput $\lambda = 0.5$

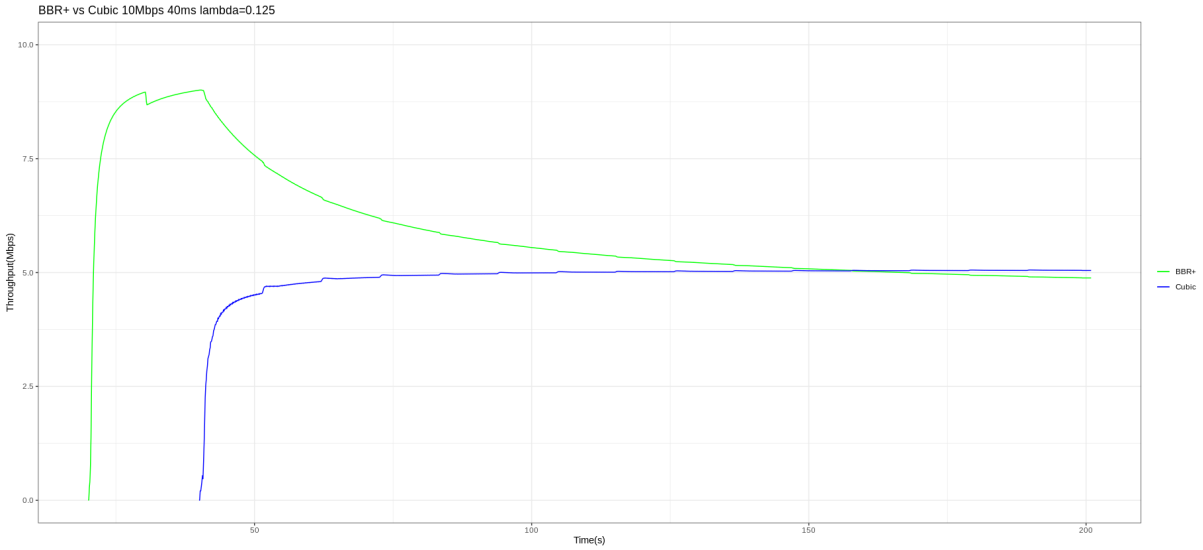

Figure 4.40: BBR+ vs Cubic Throughput $\lambda = 0.125$

4.3.3 BBRPlus

4.3.3.1 Delay-Based

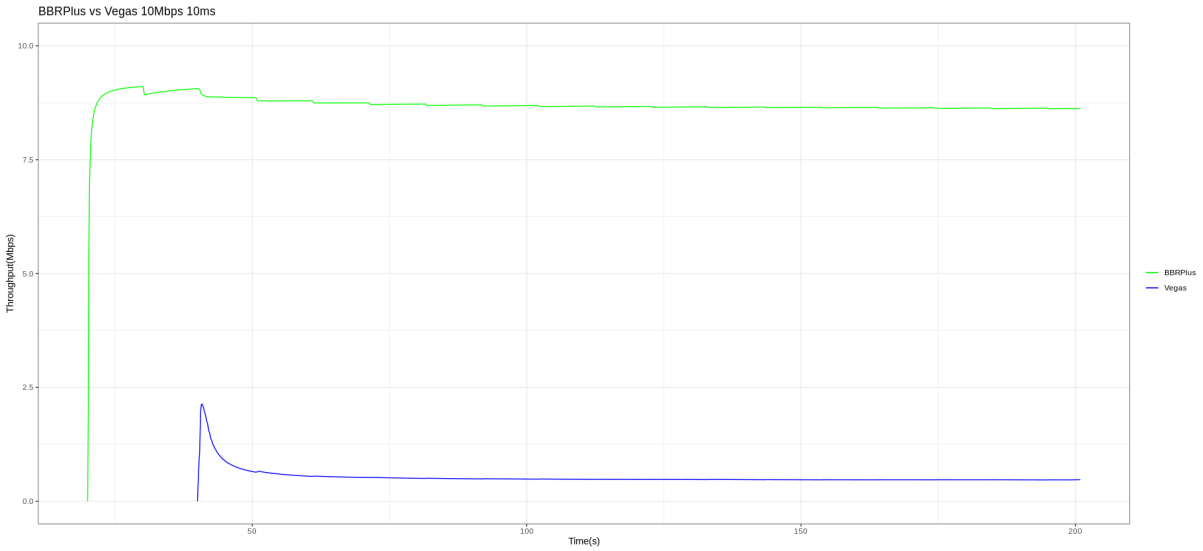

Figure 4.41: BBRPlus vs Vegas Throughput

Similar to that of BBR, BBRPlus obtains values of poor fairness in its performance with Vegas, as seen in Table 4.7. Since Vegas is a delay-based algorithm, the changes made to BBRPlus with the inclusion of packet loss as a congestion detection come to no effect as before packet loss is induced by the full queue at the bottleneck. Vegas reduces its congestion window when an alteration in RTT is detected. Thus, the same effects that BBR has with Vegas take hold with BBRPlus sending at a constant high estimated bandwidth whilst Vegas struggles to compete for a share of bandwidth. The Figure 4.41 highlights this with a stark similarity in the behaviour of the flows to the plots of BBR.

4.3.3.2 Loss-Based

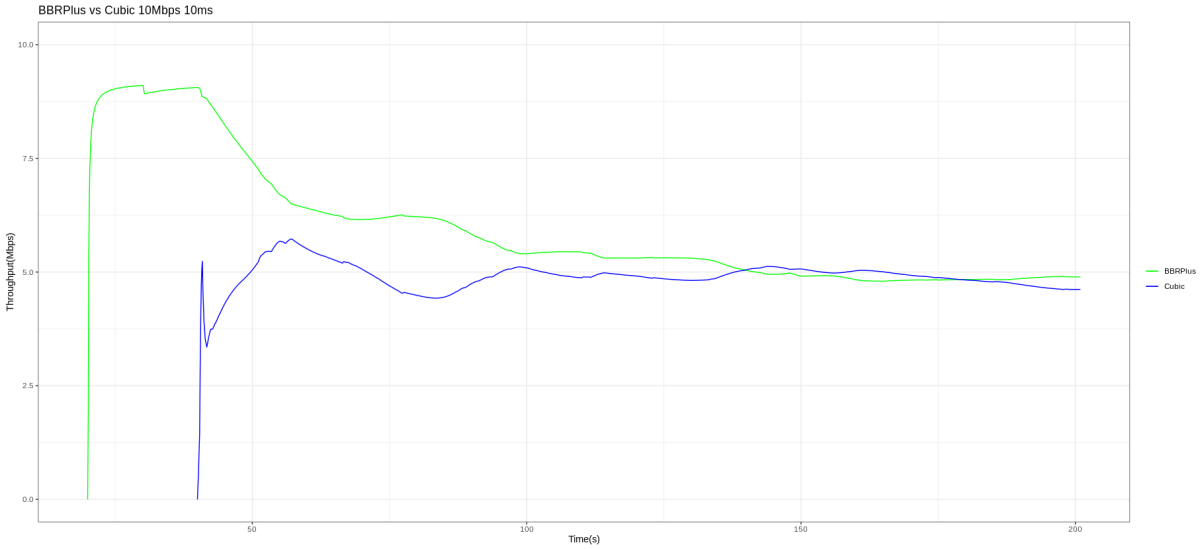

Figure 4.42: BBRPlus vs Cubic Throughput

BBRPlus again captures similar behaviour to that of BBR when competing with Cubic in the simulations ran, but with more positive results than that of the contest with Vegas. An even ratio of throughput across all cases with the highest possible Jain index is evident, however, the large amount of packet loss clouds this with the algorithms inability to deal with packet loss effectively, even though it was a key part of the changes that separated BBRPlus from BBR. Figure 4.42 shows the response that BBRPlus has when Cubic is introduced into the network. Cubics aggressive probe for bandwidth forces BBRPlus to decrease its sending rate after the queues fill up at the buffer shown by the decrease in the BBRPlus flow in the plot of throughput over time. This can be seen to then plateau out to a fair share of bandwidth with some small overlapping of the flows due to the increase and decrease of their respective congestion windows when Cubic and BBRPlus are trying to probe for more bandwidth.

4.3.4 Delay-BBR

4.3.4.1 Delay-Based

Since the introduction of Delay-BBR, there has been no evaluation of this variation in a network contested with a delay-based algorithm [25]. Contrary to the belief that this variation might contain few differences between it and the original BBR, the results obtained reflect the use of the congestion signal used to show interesting behaviour. Vegas bases its congestion detection off delay in an increase in RTT values and the introduced congestion signal that operates in Delay-BBR is set based on a condition that uses the variation of RTT values with a maintained RTT value in ProbeBW. For a

low delay bottleneck there is a high variation of RTT values and in competition with Vegas, BBRPlus is unable to operate at a high capacity and gives up the majority of bandwidth to Vegas. This scenario is displayed in Figure 4.43 where after the introduction of Vegas, represented by the blue line, it quickly increases its sending rate to a high constant value. This is due to the more frequent visiting of the ProbeRTT phase by BBRPlus and since the algorithm initially set a low estimated bottleneck bandwidth combined with Vegas taking up the majority of the bandwidth, when it exits ProbeRTT it can never recover to a reasonable sending rate. Once the bottleneck delay is set to a higher value like the 40ms simulation scenario, the common BBR behaviour with Vegas returns due to less of a variation in the RTT values consequently leading to a less frequent decrease of Vegas's congestion window. Poor fairness and throughput ratio is then shown again.

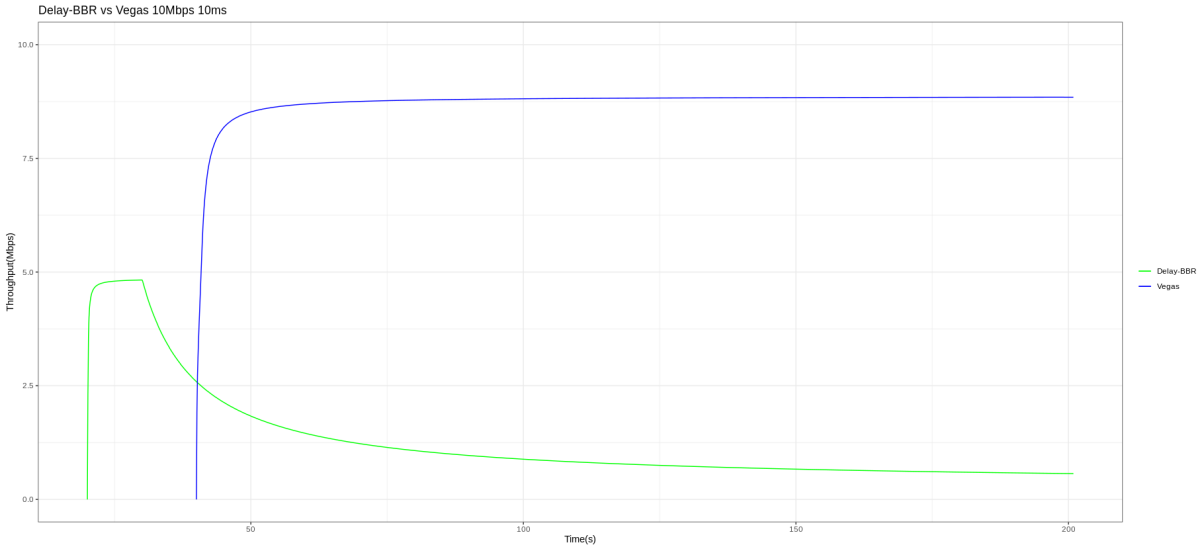

Figure 4.43: Delay-BBR vs Vegas Throughput

4.3.4.2 Loss-Based

In comparison to the results found in the contention of Delay-BBR and Vegas, the same behaviour occurs for this variation of BBR in a low delay bottleneck. The variation in the RTT values deviates the congestion delay signal and Delay-BBR suffers a very low bandwidth utilisation which is capitalised on by the Cubic flow as seen in Figure 4.44. A lower packet loss is achieved in this case as the queues at the bottleneck do not fill as quickly in comparison with a more contending variation of BBR. The increase in delay for the other simulated cases sees a more similar behavioural outcome to that of BBR with high fairness between these flows in a medium sized buffer.

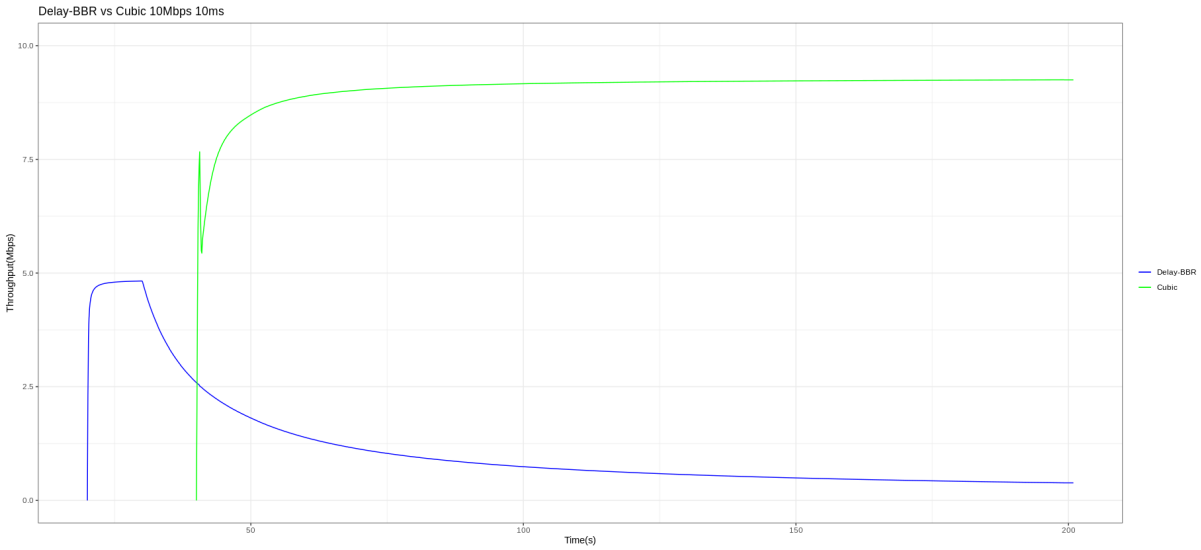

Figure 4.44: Delay-BBR vs Cubic Throughput

4.3.5 BBRv2

4.3.5.1 Delay-Based

The coexistence of the newest version of BBR with a delay-based congestion control algorithm is a type of situation not evaluated by the development team, as outlined by the introductory slides [2][26][27]. As a relevant issue with the original BBR algorithm it is important that the newer versions address it. The obtained results on the BBRv2 simulation reveal an improvement in the fairness between the two different types of congestion control algorithms represented by the high Jain index values in Table 4.7. Figure 4.45 shows the convergence towards the fairness line of 5 Mbps by the two flows. Although BBRv2 still dominates the share of bandwidth, Vegas is allowed to explore more bandwidth. This is because of the changes to the activity in the ProbeRTT phase which sets the congestion window size to adapt to the monitored bounds for the bytes in flight rate as part of the three part model introduced in BBRv2. This congestion window value increases the draining of the constant queue that forms at the bottleneck, allowing for more probing of bandwidth.

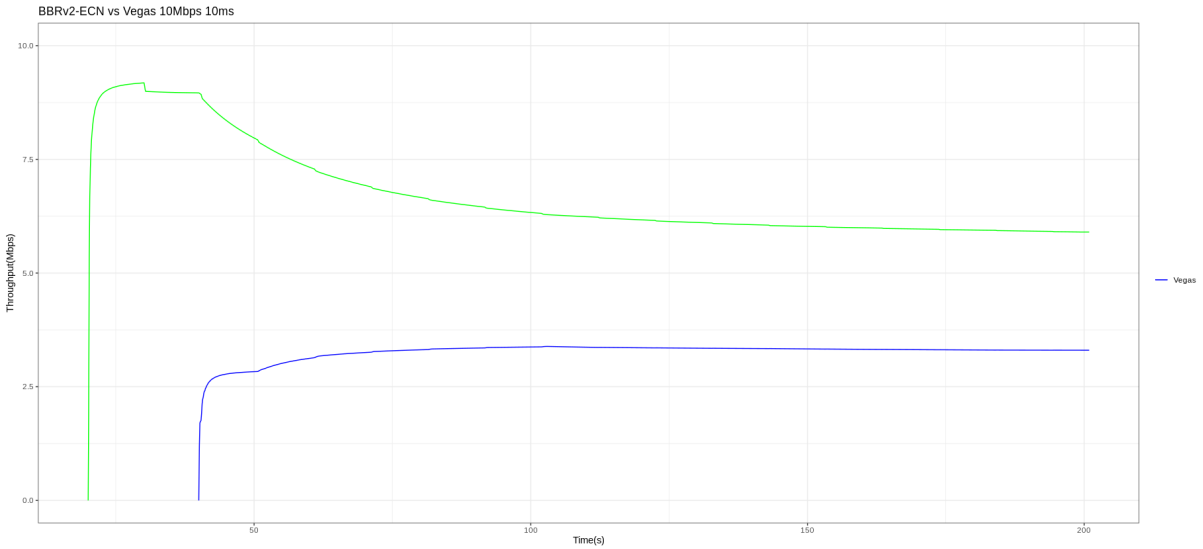

Figure 4.45: BBRv2 vs Vegas Throughput

The experimental changes introduced to BBRv2 contained relevant changes to updating the bandwidth bounds of the three-part model, especially when ECN signals occur in a network. It appears in these test cases with Vegas in contention that there is no instigation of ECN signals when there is congestion in the network, which make the changes irrelevant, as made clear in the identical values of BBRv2 and BBRv2-EXP in Table 4.7.

4.3.5.2 Loss-Based

BBRv2 exclusively makes changes in order to co-exist with loss-based algorithms like Cubic in a shared network. These include having a more conservative approach towards probing for more bandwidth in the ProbeBW phase by appropriately setting the upper bound and lower bounds of the in-flight rate and taking packet loss and ECN into consideration when looking to probe for more bandwidth. There is also specific a condition tailored for loss based co-existence introduced to check when it is reasonable to probe for bandwidth based on if the number of rounds since the last probe is greater than the BDP value. The results obtained show little effect of these changes towards increasing the co-existence between these different types of congestion control algorithms. A moderate fairness is achieved with an uneven ratio of throughput between the two flows. This is highlighted in the plot of the two flows 4.46 where it can be see upon introduction, Cubic dominates the share of resources. It is to be believed that this is down to the loss detected when Cubic probes for bandwidth causing the ProbeBW phase of BBRv2 to be set to probe down continuously for the duration of the simulation.

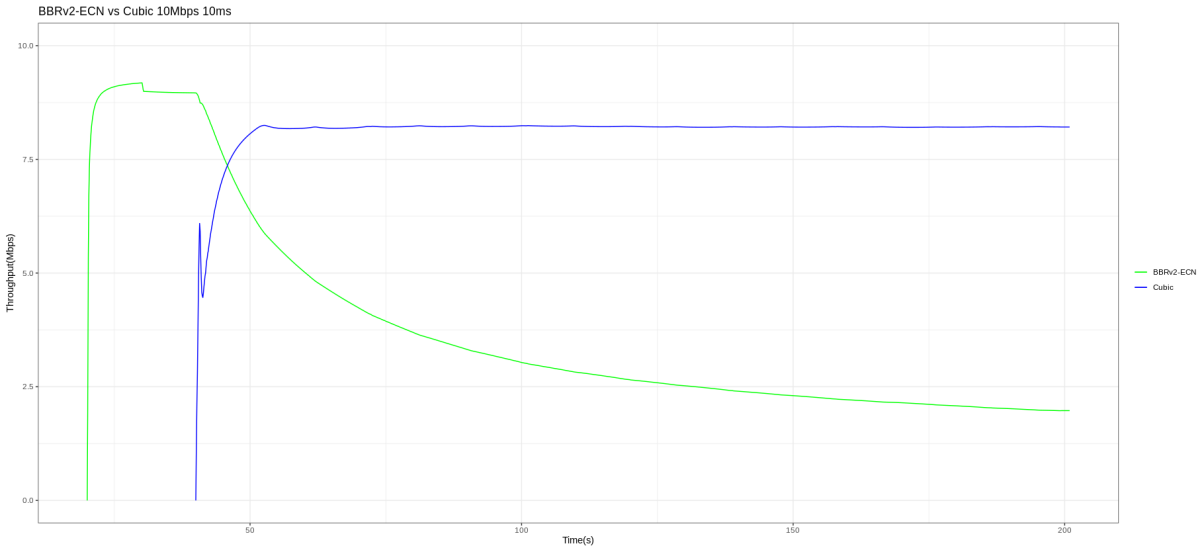

Figure 4.46: BBRv2 vs Cubic Throughput

On the initial transition into the ProbeBW phase the probe down state of ProbeBW is entered which drains any excess packets at the buffer and resets the congestion signals that BBRv2 operates on. When Cubic is introduced, it is not deemed safe to probe for more bandwidth but the flow is forced out of the probe down phase after one round trip. The probe refill state is entered and consequently the probe up phase which then quickly transitions back into the probe down phase as the loss detected from Cubic causes the in-flight rate to be deemed too high. This constant loop continues as long as Cubic is operating and is seen by the decrease of throughput by BBRv2 in Figure 4.46. The Table 4.8 shows the inability of the experimental changes to have an effect on this behaviour similar to the previous simulated scenarios.

4.3.6 Inter-Protocol Summary

The inter-protocol scenarios are set up to emulate the potential real world behaviour of the interactions between BBRv2 and the other variants with common delay-based and loss-based algorithms that are widely used across the internet. BBRv2 shows progression in increasing fairness towards delay-based algorithms but does not obtain its goal of a fair share of bandwidth with loss-based algorithms like Cubic. Problems still persist with the algorithms inability to react proactively with the introduction of a loss-based algorithm into the network. The reproduction of BBR's behaviour with delay-based algorithms stays consistent with claims of previous studies [38] whilst the simulations with Cubic only account for a specific scenario with a medium buffer. This highlights the positive interactions with loss-based algorithms whereas the problems with loss-based algorithms lie in shallow buffer scenarios [9]. The other variants overall show little improvement over the original behaviour of BBR with the likes of BBRPlus

displaying disappointing results for its interactions with Cubic despite having a focus on preventing packet loss. Delay-BBR exhibits unique behaviour in low delay scenarios due to the use of a round trip time dependent congestion signal. BBR+ is situationally specific and shows its inability to perform with other types of algorithms in these simulations. The collection of algorithms again show the underutilisation of bandwidth in high delay scenarios, seen in the values in Tables 4.7 and 4.8. The inability to fill the buffer at the bottleneck and probe for more bandwidth leads to only a 50% utilisation of bandwidth, however, this bodes well for the inter-protocol scenarios allowing both Vegas and Cubic to get a fair share of the bandwidth.

5 Conclusion

5.1 Overview

In this project an evaluation of BBRv2 was performed along with BBR and variants of the congestion control algorithm. A network simulation was set up using the network simulator NS-3 and a number of different test cases where simulated to evaluate the algorithms in different situations. The different algorithms where implemented into the NS-3 architecture. Evaluating BBRv2 with the different variations of BBR gave an insight into the functional differences between the algorithms and outlines the effects that these differences have in comparison with each other. Delay-BBR was a variant of BBR that displayed its conservative methods of probing by showing an increase in performance over BBR in single flow and intra-protocol scenarios. BBRPlus adopted techniques that looked to improve BBR in situations that involved packet loss but this was not reflected in the results as there was little change in behaviour, especially when the variation contested with delay-based and loss-based congestion control algorithms. BBR+ performed as expected, producing poor performance in the wired network simulation. This variation was designed to operate in an environment that involved high speed transportation and an LTE connection, so the functional changes that separated it from BBR came to ill-effect in the simulation setup. BBR' contains little difference with the original BBR algorithm but this difference proved to have an effect in the outcome of results. BBR' shows slight improvement in intra-protocol scenarios, giving evidence that the changing in the strategy in which the gain cycle advances gives a shorter time between the draining of the queues at the bottleneck. Otherwise, BBR' shows similar results to that of BBR in the single flow and inter-protocol test cases.

In terms of simulation outcome, BBRv2 was found to excel in some areas but not others. Single flow simulations showed validation of a well rounded performance of BBRv2 with a slight decrease in bandwidth utilisation in comparison with its predecessor. A consistent bandwidth variation response shows the stability of the core features of the algorithm. The improvements that can be seen in BBRv2 over BBR and its variations is through the competing of network resources with multiple flows of itself

and also the delay-based algorithm Vegas. There is clear contrast in the values obtained for fairness in the simulated situations for these types of intra-protocol and inter-protocol test cases that gives a positive impression of BBRv2. The disappointment in performance is the new algorithms fairness with the loss-based algorithm Cubic, one issue that resided in the previous version and was made a main goal of the new version. The possible flaw found in the ProbeBW phase of the algorithm could be narrowed down to the implementation of the algorithm in the simulation architecture. This implementation was taken from a Linux kernel adaption written in the C language. Disparity between the translation in the C code to NS-3 style C++ could result in values having different effects in the network simulator in comparison to the Linux kernel implementation. Testing a congestion control in a network simulator like NS-3 was a difficulty in this project and there is no certainty that all the new parts of BBRv2 work effectively. That of the Explicit Congestion Notification which works between two layers of the network stack and could very well operate differently in a real test-bed in comparison to a network simulator.

This also could not be the case as it is seen in single flow plots that the core behaviours work effectively, with efficient alteration between the probing of more bandwidth and draining of the bottleneck queue. BBRv2 remains very youthful in its development with further evaluations needed to verify these behaviours in different network situations. A general evaluation like the one performed in this project gives a good idea of the core functionalities of the new algorithm. Whilst a performance evaluation is important for a new algorithm like this, it can be said that a greater in-depth analysis of the algorithm is needed before suggested changes can come to light. This could include a look into more of the algorithms specific behaviours in each of its phases rather than an overall view of the performance. The objective of this project was to evaluate the congestion control algorithm BBRv2 and pitch it against its predecessor and variants of that. An insight was gained into the new algorithms distinct differences between the original version and the variations in that of the operation to prevent congestion in a network and the effects it has to ultimately achieve the goal of a high throughput and low delay network.

5.2 Future Work

It has to be noted that the BBRv2 evaluated in this project is an alpha version of the algorithm and the final algorithm has not been released at the time of writing this dissertation. The types of simulations performed in this project only scrape the surface of the performance and operation scenarios to be tested on a congestion control algorithm, especially one as complex as BBRv2. This was a general evaluation with a

high-level examination of network metrics that are typically considered when looking at congestion control algorithms. Component specific tests are examples of further evaluations that can be done with the introduction of many different components in BBRv2 including Explicit Congestion Notification Signal which is a part of TCP that can be dynamically set. In turn, it can be seen if the effects of Explicit Congestion Notification have a big effect on the network model of BBRv2. Different network constraints can be included in future tests including wireless and wide area networks which bring different network dynamics into play when simulating network protocols. The configurations used in these simulations are one of many with multiple different variations of values to set the bandwidth, delay, buffer size and other network characteristics. Especially with inter-protocol scenarios where buffer size can be seen to have an effect in previous iterations of BBR. NS-3 is used in this project but different network simulators that are mentioned in the background can be used to validate results across simulators. Along with this, real world evaluations of BBRv2 external from that at Google are yet to be performed using a physical test bed consisting of routers and devices. It is expected that further evaluations of all these different types will be performed on this new version of BBR as development continues and upon final release. At the time of writing this Google has tested the latest alpha version on its Youtube service for a small group of users and continues to test it internally with active work on preparation for a full scale deployment [27].

Bibliography

[1] Yuchung Cheng, Neal Cardwell, Van Jacobson, and Soheil Yeganeh. BBR Congestion Control, 2016. URL `https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00`.

[2] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. BBR v2 A Model-based Congestion Control IETF 104 Update. *IETF*, page 36, 2019. `https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00`.

[3] Mathieu Lacage. An ns-3 tutorial, `https://www.nsnam.org/tutorials/ns-3-tutorial-tunis-apr09.pdf`, last accessed on 2020-03-18, 2009.

[4] Vivek Jain, Viyom Mittal, and Mohit P. Tahiliani. Design and implementation of TCP BBR in ns-3. In *Proceedings of the 10th Workshop on ns-3 - WNS3 '18*, pages 16–22, Surathkal, India, 2018. ACM Press. ISBN 978-1-4503-6413-3. doi: 10.1145/3199902.3199911. URL `http://dl.acm.org/citation.cfm?doid=3199902.3199911`.

[5] Sandvine. The mobile internet phenomena report, february 2020. `https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020/Phenomena/Mobile%20Phenomena%20Report%201H%202020%2020200219.pdf`, February 2020.

[6] J. D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[7] DongJin Lee, Brian E Carpenter, and Nevil Brownlee. Media streaming observations: Trends in udp to tcp ratio. *International Journal on Advances in Systems and Measurements*, 3(3-4), 2010.

[8] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, January 2017. ISSN 00010782. doi: 10.1145/3009824. URL `http://dl.acm.org/citation.cfm?doid=3042068.3009824`.

[9] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, October 2017. doi: 10.1109/ICNP.2017.8117540.

[10] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv:1903.03852 [cs]*, March 2019. URL `http://arxiv.org/abs/1903.03852`. arXiv: 1903.03852.

[11] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524 vol.4, March 2004. doi: 10.1109/INFCOM.2004.1354672. ISSN: 0743-166X.

[12] HaSangtae, RheeInjong, and XuLisong. CUBIC. *ACM SIGOPS Operating Systems Review*, July 2008. URL `https://dl.acm.org/doi/abs/10.1145/1400097.1400105`.

[13] Vladimir Kokshenev and Sergey Suschenko. Analytical Model of the TCP Reno Congestion Control Procedure through a Discrete-Time Markov Chain. In Vladimir Vishnevsky, Dmitry Kozyrev, and Andrey Larionov, editors, *Distributed Computer and Communication Networks*, Communications in Computer and Information Science, pages 124–135, Cham, 2014. Springer International Publishing. ISBN 978-3-319-05209-0. doi: 10.1007/978-3-319-05209-0_11.

[14] Sally Floyd, Andrei Gurtov, Yoshifumi Nishida, and Tom Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm, 2012. URL `https://tools.ietf.org/html/rfc6582`.

[15] Tom Kelly. Scalable TCP: improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83, April 2003. ISSN 01464833. doi: 10.1145/956981.956989. URL `http://portal.acm.org/citation.cfm?doid=956981.956989`.

[16] Sally Floyd <floyd@acm.org>. HighSpeed TCP for Large Congestion Windows, 2003. URL `https://tools.ietf.org/html/rfc3649#page-25`.

[17] Grenville Armitage, Lawrence Stewart, Michael Welzl, and James Healy. An independent H-TCP implementation under FreeBSD 7.0: description and observed

behaviour. *ACM SIGCOMM Computer Communication Review*, 38(3):27, July 2008. ISSN 01464833. doi: 10.1145/1384609.1384613. URL http://portal.acm.org/citation.cfm?doid=1384609.1384613.

[18] Lawrence S. Brakmo, Sean W. O'malley, and Larry L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *In SIGCOMM*, 1994.

[19] G. Hasegawa, K. Kurata, and M. Murata. Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet. In *Proceedings 2000 International Conference on Network Protocols*, pages 177–186, November 2000. doi: 10.1109/ICNP.2000.896302.

[20] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, December 2006. ISSN 1558-2566. doi: 10.1109/TNET.2006.886335.

[21] Salem Belhaj and Moncef Tagina. VFAST TCP: An improvement of FAST TCP. In *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, pages 88–93, April 2008. doi: 10.1109/UKSIM.2008.50.

[22] Songyang Zhang. An Evaluation of BBR and its variants. *arXiv:1909.03673 [cs]*, September 2019. URL http://arxiv.org/abs/1909.03673. arXiv: 1909.03673.

[23] dog250. TCP BBR v2.0, which keeps people waiting for a long time, is coming soon!, 2018. URL https://blog.csdn.net/dog250/article/details/80629551.

[24] Jing Wang, Yuanjie Li, Xiufeng Xie, Yi Sun, Zhongfeng Wang, Yufan Zheng, Yunzhe Ni, Chenren Xu, Feng Qian, Wangyang Li, Wantong Jiang, Yihua Cheng, and Zhuo Cheng. An Active-Passive Measurement Study of TCP Performance over LTE on High-speed Rails. In *The 25th Annual International Conference on Mobile Computing and Networking - MobiCom '19*, pages 1–16, Los Cabos, Mexico, 2019. ACM Press. ISBN 978-1-4503-6169-9. doi: 10.1145/3300061.3300123. URL http://dl.acm.org/citation.cfm?doid=3300061.3300123.

[25] Songyang Zhang, Weimin Lei, Wei Zhang, Yunchong Guan, and Hao Li. Congestion Control and Packet Scheduling for Multipath Real Time Video Streaming. *IEEE Access*, 7:59758–59770, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2913902. URL https://ieeexplore.ieee.org/document/8701688/.

[26] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Ian Swett, Victor Vasiliev, Bin Wu, Matt Mathis, and Van Jacobson. BBR

v2: A Model-based Congestion Control IETF 105 Update. *IETF*, page 21, 2019. https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00.

[27] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, Matt Mathis, and Van Jacobson. BBR v2: A Model-based Congestion Control Performance Optimizations IETF 106 Update. *IETF*, page 32, 2019. https://datatracker.ietf.org/meeting/106/materials/slides-106-iccrg-update-on-bbrv2.

[28] Douglas J Leith, Lachlan L H Andrew, Tom Quetchenbach, Robert N Shorten, and Kfir Lavi. Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms. *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2008)*, page 6, 2008.

[29] Jean-Yves Le Boudec. Rate adaptation, Congestion Control and Fairness: A Tutorial. *EPFL*, page 54, 2019.

[30] Jim Gettys. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing*, 15(3):96–96, May 2011. ISSN 1941-0131. doi: 10.1109/MIC.2011.56.

[31] P.C. GUPTA. *DATA COMMUNICATIONS AND COMPUTER NETWORKS*. PHI Learning, 2006. ISBN 9788120328464. URL https://books.google.ie/books?id=-kNn_p6WA38C.

[32] Christos Douligeris and Lakshmana N Kumar. Fairness issues in the networking environment. *Computer Communications*, 18(4):288 – 299, 1995. ISSN 0140-3664. doi: https://doi.org/10.1016/0140-3664(95)93446-B. URL http://www.sciencedirect.com/science/article/pii/014036649593446B.

[33] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *arXiv:cs/9809099*, September 1998. URL http://arxiv.org/abs/cs/9809099. arXiv: cs/9809099.

[34] L. Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Conference Record, International Conference on Communications*, pages 43.1.1–43.1.10, Boston, Massachusetts, June 1979.

[35] Ethan Blanton and Mark Allman. TCP Congestion Control, 2009. URL https://tools.ietf.org/html/rfc5681#page-4.

[36] Sally Floyd, K. K. Ramakrishnan, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP, 2001. URL https://tools.ietf.org/html/rfc3168.

[37] Kouto Miyazawa, Kanon Sasaki, Naoki Oda, and Saneyasu Yamaguchi. Cycle and Divergence of Performance on TCP BBR. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–6, October 2018. doi: 10.1109/CloudNet.2018.8549411.

[38] Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, and Georg Carle. Reproducible measurements of TCP BBR congestion control. *Computer Communications*, 144:31–43, August 2019. ISSN 01403664. doi: 10.1016/j.comcom.2019.05.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0140366419303470.

[39] Saahil M Claypool. Sharing but not Caring - Performance of TCP BBR and TCP CUBIC at the Network Bottleneck. *WPI*, page 55, 2019.

[40] 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 on - IMC '19*, pages 137–143, Amsterdam, Netherlands, 2019. ACM Press. ISBN 978-1-4503-6948-0. doi: 10.1145/3355369.3355604. URL http://dl.acm.org/citation.cfm?doid=3355369.3355604.

[41] Jinting Lin, Lin Cui, Yuxiang Zhang, Fung Po Tso, and Quanlong Guan. Extensive evaluation on the performance and behaviour of TCP congestion control protocols under varied network scenarios. *Computer Networks*, 163:106872, November 2019. ISSN 13891286. doi: 10.1016/j.comnet.2019.106872. URL https://linkinghub.elsevier.com/retrieve/pii/S1389128618311265.

[42] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. An Empirical View on Content Provider Fairness. *arXiv:1905.07152 [cs]*, May 2019. URL http://arxiv.org/abs/1905.07152. arXiv: 1905.07152.

[43] Geon-Hwan Kim, Yeong-Jun Song, Imtiaz Mahmud, and You-Ze Cho. Enhanced BBR Congestion Control Algorithm for Improving RTT Fairness. In *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 358–360, July 2019. doi: 10.1109/ICUFN.2019.8806064. ISSN: 2165-8528.

[44] John Heidemann, Kevin Mills, and Sri Kumar. Expanding Confidence in Network Simulation. *IEEE*, page 9, 2000.

[45] Gypsy Nandi Gayatry Borboruah. A Study on Large Scale Network Simulators, 2014. URL https://pdfs.semanticscholar.org/17b1/7b6fd993dea3b7acc5ad7400e95c9f03e733.pdf.

[46] Stewart Robinson. Simulation: The practice of model development and use. In *Simulation: The Practice of Model Development and Use*, 2004.

[47] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84545. URL https://doi.org/10.1145/84537.84545.

[48] Dale Martin, Timothy McBrayer, and Philip Wilsey. Warped: Time warp simulation kernel for analysis and application development. In *WARPED: Time Warp Simulation Kernel for Analysis*, volume 1, pages 383–386, 01 1996. doi: 10.1109/HICSS.1996.495485.

[49] Dhananjai Madhava Rao, Narayanan V. Thondugulam, Radharamanan Radhakrishnan, and Philip A. Wilsey. Unsynchronized parallel discrete event simulation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, page 1563–1570, Washington, DC, USA, 1998. IEEE Computer Society Press. ISBN 0780351347.

[50] Dhananjai M Rao and Philip A Wilsey. An ultra-large-scale simulation framework. *Journal of Parallel and Distributed Computing*, 62(11):1670–1693, November 2002. ISSN 07437315. doi: 10.1016/S0743-7315(02)00003-5. URL https://linkinghub.elsevier.com/retrieve/pii/S0743731502000035.

[51] Chang Xinjie. Network simulations with opnet. *WSC'99. 1999 Winter Simulation Conference Proceedings. 'Simulation - A Bridge to the Future' (Cat. No.99CH37038), Simulation Conference Proceedings, 1999 Winter*, 1:307, 1999. ISSN 0-7803-5780-9. URL https://ieeexplore.ieee.org/document/823089?arnumber=823089.

[52] András Varga. The omnet++ discrete event simulation system. *Proc. ESM'2001*, 9:247, 01 2001. URL http://www.sfu.ca/~ljilja/ENSC835/Spring08/News/Presentations/OMNeT++/usman.pdf.

[53] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233), Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pages 154 – 161, 1998. ISSN 0-8186-8457-7. URL https://ieeexplore.ieee.org/document/685281.

[54] Scalable Networks. Qualnet - network simulation software, https://www.scalable-networks.com/qualnet-network-simulation-software-tool, last accessed on 2020-03-18, 2020.

[55] Mohammed Kabir, Syful Islam, Md Hossain, and Sazzad Hossain. Detail comparison of network simulators. *International Journal of Scientific and Engineering Research*, 5:203, 11 2014.

[56] Tetcos. Netsim, https://www.tetcos.com/index.html, last accessed on 2020-03-18,, 2020.

[57] LBL, Xerox PARC, UCB, and USC/ISI. Ns-2, https://www.isi.edu/nsnam/ns/, last accessed on 2020-03-18,, 2020.

[58] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[59] Atta ur Rehman Khana, Sardar M. Bilalb, and Mazliza Othmana. A Performance Comparison of Network Simulators for Wireless Networks. *arXiv:1307.4129 [cs]*, July 2013. URL http://arxiv.org/abs/1307.4129. arXiv: 1307.4129.

[60] Tom Henderson (University of Washington). Ns-3 introduction, https://www.nsnam.org/docs/ns-3-overview.pdf, last accessed on 2020-03-18, 2014.

[61] Nsnam. Ns-3 coding style, https://www.nsnam.org/develop/contributing-code/coding-style/, last accessed on 2020-03-18, 2020.

[62] Nsnam. Tcp models in ns-3, https://www.nsnam.org/docs/models/html/tcp.html, last accessed on 2020-03-19, 2020.

[63] Nsnam. `TcpCongestionOps` class reference, https://www.nsnam.org/doxygen/classns3_1_1_tcp_congestion_ops.html#aed9a4df8aa78e5746830709f2c72d972, last accessed on 2020-03-19, 2020.

[64] Git. Git, https://git-scm.com/, last accessed on 2020-03-20, 2020.

[65] Github, Inc. Github, https://github.com/, last accessed on 2020-03-20, 2020.

[66] C. Stephen Gunn Soheil Hassas Yeganeh Van Jacobson Neal Cardwell, Yuchung Cheng. tcp_bbr.c, 2016. URL https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/tree/net/ipv4/tcp_bbr.c.

[67] Soheil Yeganeh, Yuchung Cheng, Neal Cardwell, and Van Jacobson. Delivery Rate Estimation, 2017. URL https://tools.ietf.org/html/draft-cheng-iccrg-delivery-rate-estimation-00. Library Catalog: tools.ietf.org.

[68] David Schinazi QUICHE team. windowed_filter.h, 2019. URL https://cs.chromium.org/chromium/src/net/third_party/quiche/src/quic/core/congestion_control/windowed_filter.h.

[69] Brett Levasseur, Mark Claypool, and Robert Kinicki. A TCP CUBIC implementation in ns-3. In *Proceedings of the 2014 Workshop on ns-3 - WNS3 '14*, pages 1–8, Atlanta, Georgia, 2014. ACM Press. ISBN 978-1-4503-3003-9. doi: 10.1145/2630777.2630780. URL http://dl.acm.org/citation.cfm?doid=2630777.2630780.

[70] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/.

A1 Abbreviations

BBR	**B**ottleneck **B**andwidth And **R**ound Trip Time
RTT	**R**ound **T**rip **T**ime
TCP	**T**ransmission **C**ontrol **P**rotocol
CWND	**C**ongestion **W**i**ND**ow
BDP	**B**andwidth **D**elay **P**roduct
BtlBw	**B**ot**tl**neck **B**and**w**idth
HSR	**H**igh **S**peed **R**ails
ECN	**E**xplicit **C**ongestion **N**otification
USSF	**U**ltra Large **S**cale **S**imulation **F**ramework
OPNET	**OP**timized **N**etwork **E**ngineering **T**ool
OMNET++	**O**bjective **M**odular **NE**twork **T**ool in C++
GloMoSim	**Glo**bal **Mo**bile Information Systen **Sim**ulator
ACK	**ACK**nowledgement
SACK	**S**elective **ACK**nowledgement
ECT	**E**CN **C**apable **T**ransport
CE	**C**ongestion **E**xperienced
CWR	**C**ongestion **W**indow **R**educed
ECE	**E**CN **EC**ho

A2 Code Listings

```
1    Require timestamp(now), sequence(seq) and payload length of sent
    packet
2    OnPacketSent():
3       info.sent_ts = timestamp
4       info.bytes = payload
5       sent_packets_map.insert(seq, info)
6       inflight = inflight + payload
7       last_sent_packet = seq
```

Code Listing A2.1: OnPacketSent()[25]

```
1    Require ack received timestamp(now) and acked sequence number(seq)
2    OnAck():
3       UpdateRttAndInflight(now, seq)
4       congested = CheckIfCongested()
5       if (now - min_rtt > kMinRttExpiry):
6         min_rtt_expired = 1
7       DrainExcessBuffer(now, min_rtt_expired, congested)
```

Code Listing A2.2: OnAck()[25]

```
Require ack received timestamp(now) and acked sequence number(seq)
UpdateRttAndInflight():
   info = sent_packets_map.get
   rtt = info.sent_ts - now
   inlflight = info.bytes
   if (rtt < min_rtt or min_rtt == 0):
     min_rtt = rtt
     min_rtt_ts = now
   if (rtt < kSimilarMinRtt * min_rtt):
     min_rtt_ts = now
   if (seq > seq_at_backoff):
     if (rtt < base_line_rtt):
       base_line_rtt = rtt
       srtt = rtt
     srtt = (1 - α) * srtt + α * rtt
```

Code Listing A2.3: UpdateRttAndInflight()[25]

```
CheckIfCongestion():
   if (srtt == 0 or base_line_rtt == +∞):
     return 0
   if (mode == ProbeBW and srtt > β * base_line_rtt):
     return 1
```

Code Listing A2.4: CheckIfCongestion()[25]

```
(now, min_rtt_expired, congested)
DrainExcessBuffer():
   if (mode != PRobeRTT and (min_rtt_expired or congested)):
     mode = ProbeRTT
     seq_at_backoff = last_sent_packet
     srtt = 0
     base_line_rtt = +∞
     pacing_gain = 0.75
     bdp = bw * min_rtt
   if (mode == ProbeRTT):
     if (inflight < bdp):
       EnterProbeBwMode()
```

Code Listing A2.5: DrainExcessBuffer()[25]

A3 Figures

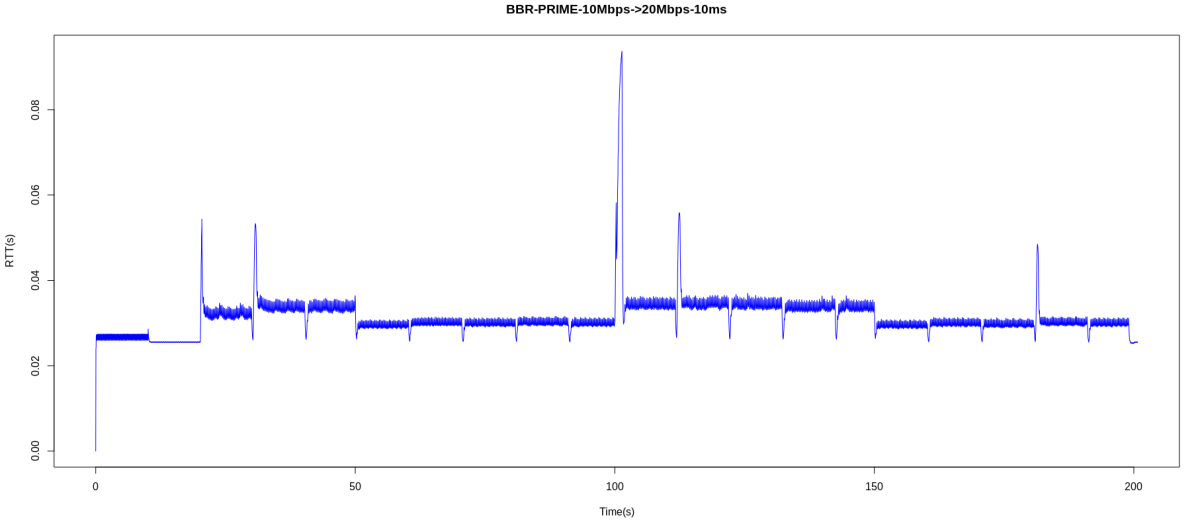

Figure A3.1: BBR' RTT Responsiveness

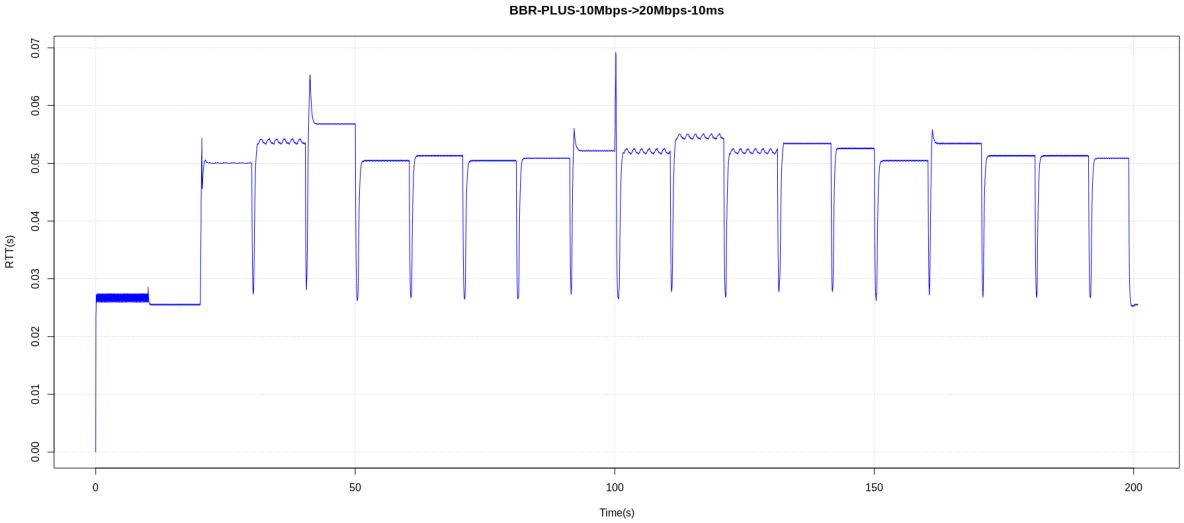

Figure A3.2: BBRPlus RTT Responsiveness

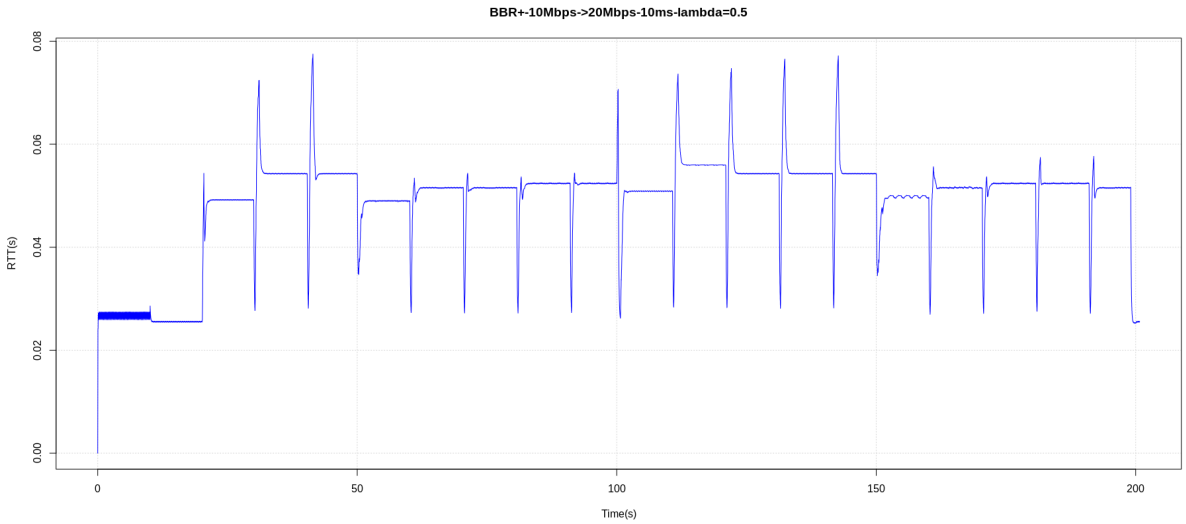

Figure A3.3: BBR+ RTT Responsiveness $\lambda = 0.5$

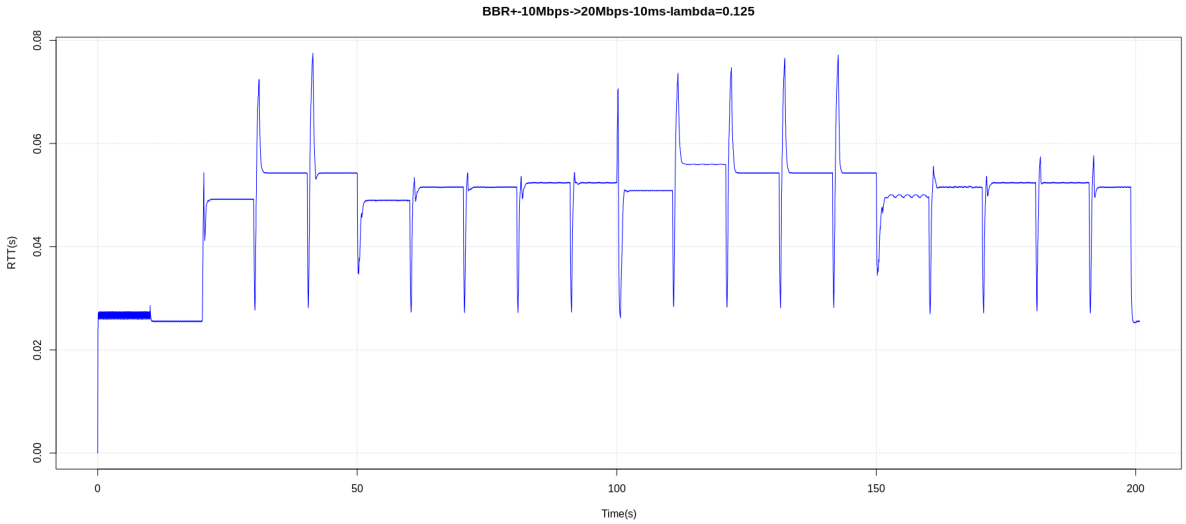

Figure A3.4: BBR+ RTT Responsiveness $\lambda = 0.125$

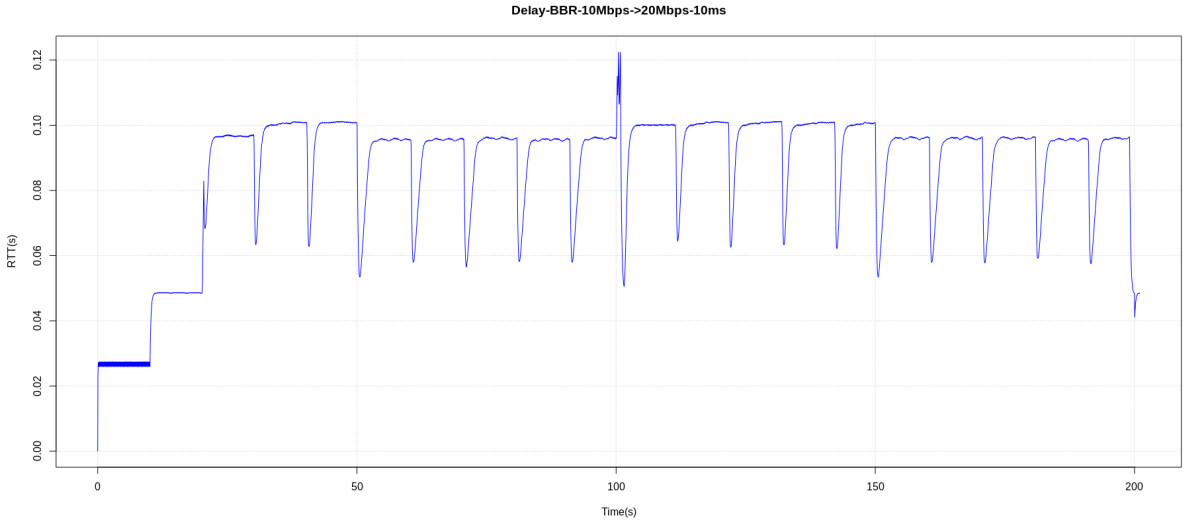

Figure A3.5: Delay-BBR RTT Responsiveness

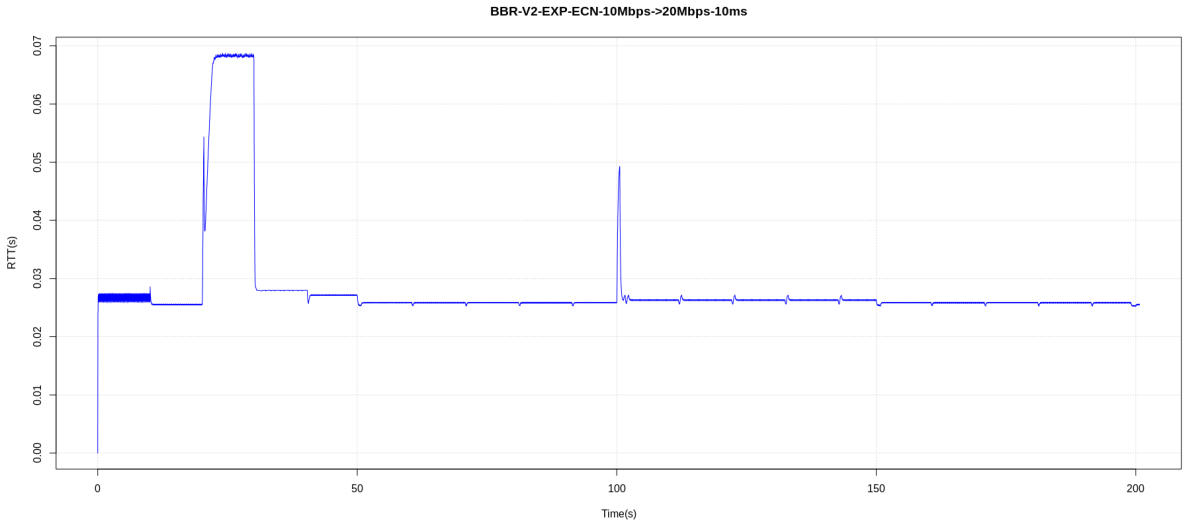

Figure A3.6: BBRv2-EXP RTT Responsiveness

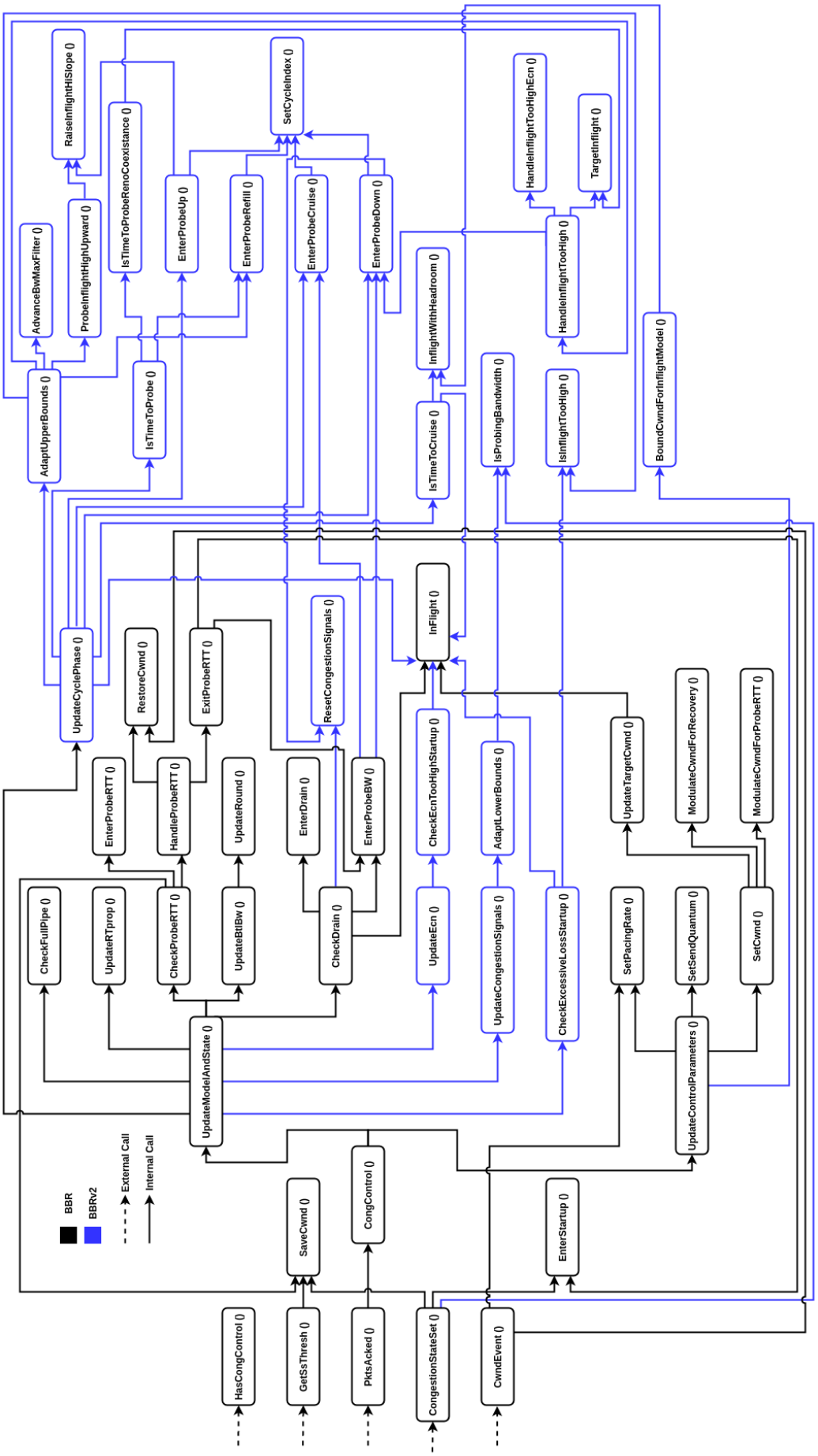

Figure A3.7: BBRv2 Function Graph