DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

An Analysis of the Queueing Delays and Throughput of the TCP BBR Congestion Control in NS-3

XINKAI XIONG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An Analysis of the Queueing Delays and Throughput of the TCP BBR Congestion Control in NS-3

XINKAI XIONG

Master in Computer Science
Date: November 30, 2020
Supervisor: Marco Chiesa
Examiner: Markus Hidell
School of Electrical Engineering and Computer Science
Swedish title: En Analys av Köfördröjningarna och Genomströmningen av TCP BBR Congestion Control i NS-3

Abstract

BBR is a congestion control recently proposed by Google, unlike the traditional congestion control which uses packet loss as the signal of congestion, BBR uses the estimation of bottleneck bandwidth to control the sending rate. However, recent work shows that BBR suffers from a variety of problems such as large queuing delays with multiple flows in the network. Most of the existing work in this area has so far focused on the performance analysis of BBR. Despite these efforts, there exists still a lack of understanding on how to improve the performance of BBR in different scenarios. In this paper, we first present the behaviour of the original BBR in the Network Simulator 3 (ns-3), then, we provide an improvement that carefully adjusts the pacing rate based on the RTT of the flow, finally, to validate our method, we run simulations varying different bottlenecks, latency, and numbers of flows in both small and large buffer size scenarios on ns-3 network simulator. The results show that our improvement can significantly reduce the queuing delay on the bottleneck at a very small cost of throughput in large buffer scenarios, and also achieve less than 0.1% retransmission rate in small buffer scenarios.

Keywords: Congestion Control, TCP, BBR, Latency, ns-3

Sammanfattning

BBR är en typ av stockningskontroll som nyligen föreslagits av Google. Till skillnad från traditionell stockningskontroll som använder paketförlust som stockningssignal använder BBR en uppskattning av bandbredden i flaskhalsen mellan sändare och mottagare för att styra sändningshastigheten. Senare arbete visar dock att BBR lider av olika problem, såsom långa förseningar i paketköer med flera flöden i nätverket. Det mesta av det befintliga arbetet inom detta område har hittills fokuserat på att analysera BBR-prestanda. Trots dessa ansträngningar saknas det fortfarande förståelse för hur man kan förbättra BBR:s prestanda i olika scenarier. I den här rapporten presenterar vi först beteendet hos den ursprungliga BBR i nätverkssimulatorn ns-3. Därefter föreslår vi en förbättring som noggrant justerar sändningstakten enligt RTT (Round Trip Time) för flödet. Slutligen, för att validera vår metod, utför vi simuleringar som varierar olika flaskhalsar, fördröjningar och antal paketströmmar i små och stora buffertstorlekar i ns-3. Resultaten visar att vår förbättring avsevärt kan minska köfördröjningar i flaskhalsar för stora buffertstorlekar till en mycket låg kostnad i genomströmning, samt uppnå mindre än 0.1% omsändningshastighet i scenarier med små buffertstorlekar.

Keywords: Trängsel Kontroll, TCP, BBR, Latens, ns-3

Acknowledgement

First, I would like to express thanks of gratitude to my supervisor, Assistant Professor Marco Chiesa who gave me the precious opportunity to do this project. You always help me whenever I had questions about my thesis. I had learnt so much under your supervision.

Secondly, I would like to send my special thanks to my examiner, Associate Professor Markus Hidell. Thank you for your understanding and constant guidance during the development of this project.

Finally, I would like to thank my families, my friends, whoever helped me through the thesis for their support, love and encouragement. I would have not been able to make it without you.

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

ACK Acknowledgement.

BBR Bottleneck Bandwidth and Round-trip propagation time.

BDP Bandwidth-Delay Product.

NIC Network Interface Controller.

ssthresh slow start threshold.

TCP Transport Control Protocol.

TIMELY Transport Informed by MEasurement of LatencY.

Chapter 1

Introduction

1.1 Background

Most of today's Transport Control Protocol (TCP) congestion controls are loss-based, such as TCP Reno [1] and CUBIC [2]. They use packet loss as the signal of congestion. However, with the evolution of technology, this design can no longer fulfil users' demand. Loss-based congestion controls always tend to fill the buffer of the switches in the network in order to trigger loss signal and subsequent slow-down reactions. Filling buffers with data waiting to be transmitted has two downsides. First, it affects the latency of small latency-sensitive flows, which are queued together with large flows. Second, it reduces the ability of the switch to deal with sudden bursts of data when the buffers are full. One solution for the first problem is to use small buffers so that latency-sensitive flows are not queued for long time in a buffer. However, it is renowned that TCP Reno cannot achieve full-throughput when the bottleneck link has a buffer that is too small [3]. As for the second problem, using even large buffers exacerbates the problem of latency-sensitive flows, which is known as the *bufferbloat* problem.

TCP congestion control has been designed in the 1980s [4]. It uses a congestion window and a congestion control strategy to avoid creating congestion in the network. The congestion window is maintained by the sender and it controls the amount of data that can be sent into the network before receiving an acknowledgement (ACK) that a previously sent packet has been received.

As shown in Figure 1.1, the implementation of TCP Reno contains four phases: slow start, congestion avoidance, fast retransmit, and fast recovery. When a connection starts, it first enters slow start phase, the congestion window is set to one packet, and a slow start threshold (ssthresh) is set. Every

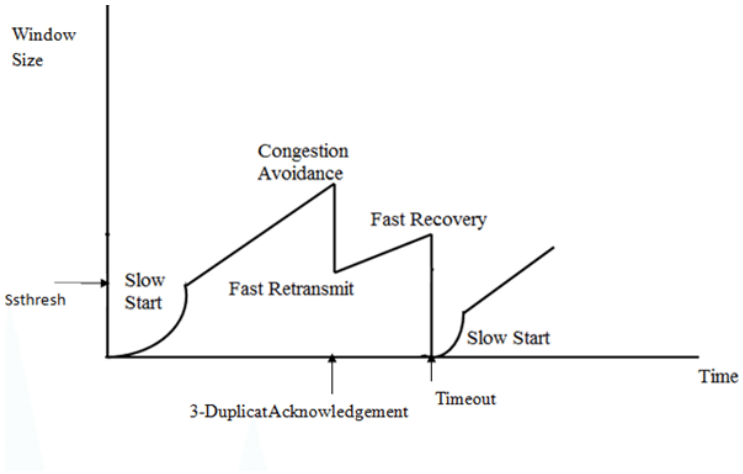


Figure 1.1: TCP congestion control algorithm

time an ACK is received, the congestion window is increased by one packet, doubling the congestion window by each RTT. Once the congestion window reaches ssthresh, it enters congestion avoidance phase. The congestion window will increase by one packet with each RTT. After 3 duplicated ACKs, which denotes a *loss* of a packet, TCP uses fast retransmit algorithm to retransmit the missing packet immediately. Then it enters fast recovery phase, the congestion window and ssthresh are set to half of the current congestion window. If a new ACK is received which means the missing packet is retransmitted successfully, TCP exits fast recovery, and enters congestion avoidance phase.

Bottleneck Bandwidth and Round-trip propagation time (BBR) [3] is a new congestion control that aims to achieve higher throughput and lower latency, it was first proposed by a team in Google in 2016. Figure 1.2 shows the operating points of BBR and loss-based congestion control algorithm. When the delivery rate is smaller than the bottleneck bandwidth, there is no queue in the buffer, so the RTT stays at the minimal RTT, then when the delivery rate reaches the bottleneck bandwidth, the queue starts to accumulate in the buffer, thus the RTT increases. We should notice that at operating point (A), when we have the maximum delivery rate which equals bottleneck bandwidth, we also have the minimal RTT, and that is where BBR operates. Then with the queue growing, at operating point (B), the buffer is full, RTT stops increasing, packets are dropped in the network, which is considered as the signal of congestion by the loss-based congestion control algorithm. Overall, BBR tries to operate before the congestion and the loss-based congestion control operates after the congestion.

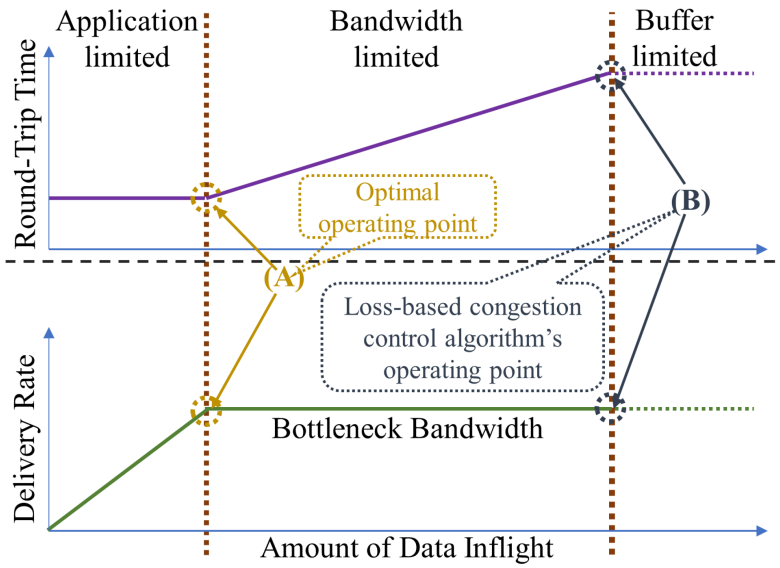


Figure 1.2: Congestion control operating points, from [5]

BBR has been deployed in Google's B4 network and YouTube, it improves YouTube network throughput by 4 percent globally [6]. And it has been supported in Linux since kernel 4.9.

1.2 Problem

BBR intends to prevent congestion before it happens, however, the mechanism BBR uses overestimates the available bandwidth when there are multiple competing flows in the network, which leads to a large queuing delay on the bottleneck, and causes massive packet loss when the buffer at the bottleneck is small.

The thesis statement of this work is the following: *"The TCP BBR congestion control suffers from high buffer occupancy, a problem that can be mitigated in simple test cases by re-configuring some of its parameters"*.

1.3 Purpose

We aim to provide a method to adjust the overestimated bottleneck bandwidth, and the users of BBR can customize the behaviour of BBR according to different network conditions. The detail of our improvement is presented in Chapter 3. We believe our method can provide some inspirations to the development

of BBR and benefit the users of BBR.

1.4 Goals

The goal of the thesis is to develop an improvement of BBR for reducing the queuing delays and packet loss. This has been divided into the following sub-goals:

1. Studying recent researches about BBR's behaviour, improvements, and performance in different network scenarios.
2. Understanding the source code of the implementation of BBR for ns-3.
3. Developing a method that improves the performance of BBR without changing its core mechanism.
4. Designing a testbed and simulation metrics for different scenarios.
5. Running simulations in different scenarios and comparing the performance between the original BBR and our improved BBR.

1.5 Deliverables

1. A project proposal.
2. A git repository containing the source code, our modification, and the testing scripts.
3. A written thesis containing the project background, the details of the method and the implementation, and the results analysis.
4. An oral presentation of the project.

1.6 Research Methodology

In this project, we collected data from designed simulations on ns-3 to evaluate the performance of our improvement for BBR and compared it with the original BBR's performance in different scenarios. Thus, a quantitative method [7] is used in this thesis.

1.7 Delimitations

In this project, we conduct experiments only in a simulated environment. The network designed for the simulations is very simple, only contains a single bottleneck. Our improvement focuses on the performance of queuing delays and packet loss. Other performances of BBR, such as RTT fairness are out of the scope of this project. We rely on an existing implementation of BBR in ns-3, which may not be representative of the existing ones in Linux.

1.8 Benefit, Ethic, and Sustainability

The experiments in this project are running in a simulated environment, so there will be no damage to the real network or machines. All tools we use in this project are open source, therefore, there will be no ethic problems on licenses. Also, this project is aimed to benefit the users of BBR, it is completely harmless to any user or organization.

This paper provides a new method to improve BBR's performance on queuing delay and packet loss. The users can have better experiences by having a low latency and packet loss connection. Better services are able to be provided by using our improvement, the network can be more efficient which means less equipment is needed. Less equipment means less pollution, power consumption and the cost of money which makes it both sustainable and economic. This thesis does not study whether the proposed modification to BBR is "friendly" to existing TCP congestion control schemes.

1.9 Structure of the thesis

Chapter 2 presents relevant background information of congestion control algorithm and recent researches of BBR. Chapter 3 provides the simulation setup and details of our modification on BBR. Chapter 4 illustrates the results of original and improved BBR and their comparison. Chapter 5 concludes our findings, discusses the limitations and future work.

Chapter 2

Background

2.1 TCP NewReno

TCP NewReno [8] is a modification to TCP's fast recovery algorithm. TCP Reno's fast recovery improves the throughput and robustness after a packet loss, however, it only considers the situation of single packet loss. In a real network, once the congestion happens, there will be a lot of packet loss. If we use TCP Reno, it will consider that there are multiple network congestion, thus the congestion window and ssthresh will be halved multiple times, causes extremely low throughput.

We noted the ACK that sender received in fast recovery as ACKx, and noted the packet with the largest sequence number that had been sent by the sender when receiving ACKx as PKTy. If ACKx is not the acknowledgement of PKTy, then it is a partial ACK; If not, it is a recovery ACK. Partial ACKs are used in TCP NewReno to avoid halving the congestion window too aggresively.

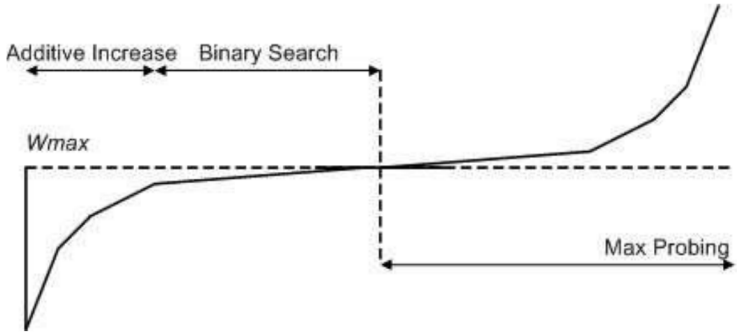
TCP New Reno deals with the single packet loss as same as TCP Reno, but when multiple packets are dropped in the same period, the sender exits fast recovery and enters congestion avoidance only when receiving the recovery ACK, i.e. all lost packets are retransmitted and the ACKs are received.

2.2 BIC-TCP

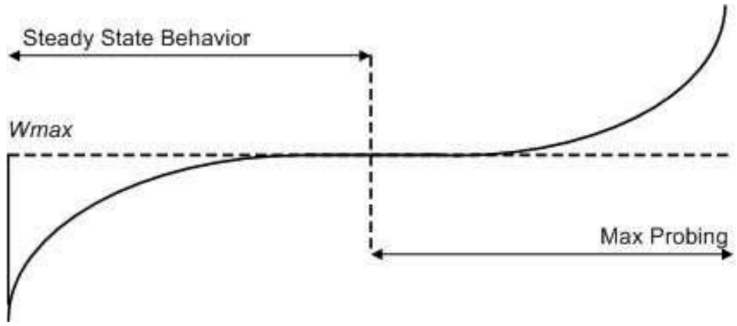
BIC-TCP [9] is a congestion control designed for fast long-distance network. The main feature of BIC-TCP is its window growth function as shown in Figure 2.1(a). When a packet loss event happens, BIC-TCP will reduce its congestion window, the window size before the reduction is set to W_{max}, the window size

after the reduction is set to W_{min}. Then BIC-TCP uses a binary search to set the congestion window to W_{mid}, the middle of W_{max} and W_{min}. In case of the step from W_{min} to W_{mid} is too large, BIC-TCP set a constant S_{max}, when the step is larger than S_{max}, the next window size is set to W_{min} + S_{max} instead of W_{mid}. Then it will update W_{min} until the step is smaller than S_{max}, if there is no packet loss. In the meanwhile, there is another constant S_{min}, when the window increment is smaller than S_{min}, the current window size will be set as the W_{max}.

If the window is larger than W_{max}, which means there could be more packets in the network. A new W_{max} needs to be set. BIC-TCP will then enter a new phase, Max Probing. In Max Probing, it uses a window growth function that is symmetrical with the Additive Increase and Binary Search phases.



(a) BIC-TCP window growth function



(b) CUBIC window growth function

Figure 2.1: Window growth function of BIC-TCP and CUBIC

2.3 CUBIC

CUBIC is the next version of BIC-TCP. It is less aggressive than BIC-TCP especially when RTT is short in the network. CUBIC uses a cubic function, both concave and convex parts, as the window growth function, as shown in Figure 2.1(b). The congestion window is calculated as:

$$cwnd = C(T - K)^3 + W_{max} \tag{2.1}$$

where:

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{2.2}$$

β is a multiplicative decrease factor, W_{max} is the window size just before the last reduction, T is the time elapsed since the last window reduction, C is a scaling constant, and $cwnd$ is the congestion window at the current time.

The key feature of CUBIC is that its window growth depends only on the time between two consecutive congestion events, so the window growth is independent of RTT since the function is defined in real-time. This allows CUBIC to perform good RTT fairness and TCP friendliness.

2.4 BBR

BBR uses a state machine that contains 4 states (Startup, Drain, ProbeBW, and ProbeRTT) to control its behaviour, as shown in Figure 2.2. BBR operates at the optimal operating point as we can see in Figure 1.2. To do that, BBR constantly estimates the available bandwidth $\widehat{BW}_{est}$ and the minimal RTT $\widehat{RTT}_{min}$. BBR calculates a path's available Bandwidth-Delay Product (BDP) via:

$$\text{BDP} = \widehat{BW}_{est} \times \widehat{RTT}_{min}$$

The pacing_gain is a primary parameter of BBR that controls how fast BBR sends packets in each state. The cwnd_gain is another important parameter that controls the size of the congestion window. The congestion window is set as:

$$\text{CWND} = \text{BDP} \times \text{cwnd\_gain}$$

BBR sets the pacing rate on Acknowledgements (ACKs), when an ACK arrives, the pacing rate is calculated as:

$$\text{pacing rate} = \widehat{BW}_{est} \times \text{pacing\_gain}$$

We provide an overview of BBR's behaviour in each state and the transition among them in the following.

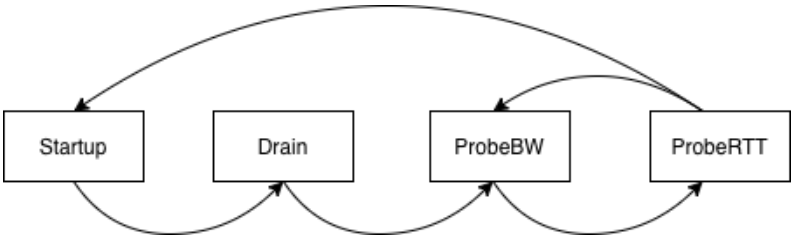


Figure 2.2: BBR state machine

2.4.1 Startup

When a BBR flow starts, it first enters the Startup state. In this state, the pacing_rate increases rapidly to fulfil the bottleneck. The pacing_gain is set to $2/\ln 2$, this almost doubles the sending rate every RTT. If there are three rounds where BBR tries to increase its delivery rate but actually gets less than 25% increase, BBR estimates that the network pipe is full and enters Drain.

2.4.2 Drain

In Drain, the pacing_gain is set to the inverse value used in Startup to drain the queue resulting from the Startup. When the inflight packets number equals the estimated BDP, BBR will exit Drain and enter ProbeBW.

2.4.3 ProbeBW

BBR spends most of its time in ProbeBW. Every time an ACK arrives, BBR will calculate and store the delivery rate, set the pacing rate and also store the RTT. BBR uses a max filter to estimate the available bottleneck bandwidth, it keeps the delivery rate calculated in the latest ten RTTs. On every ACK, the filter is updated and the max value in the filter is considered as the estimated bandwidth $\widehat{BW}_{est}$. And BBR calculates the estimation of minimal RTT, $\widehat{RTT}_{min}$, using a minimal filter, the length of the filter window is ten seconds. The minimal RTT in the filter in the last ten seconds is selected as $\widehat{RTT}_{min}$.

In ProbeBW, BBR use a gain_cycle to probe for bandwidth, it has eight phases with the following values: 5/4, 3/4, 1, 1, 1, 1, 1, 1. Each phase lasts for one RTT. BBR uses a pacing_gain of 5/4 to probe for more bandwidth and a pacing_gain of 3/4 to drain the queue creates by the higher rate. If the increased sending rate results in an increased delivery rate, the new delivery rate will be used as the sending rate immediately. During ProbeBW, the inflight

limit is 2 $\times$ BDP as the cwnd_gain is set to 2.

2.4.4 ProbeRTT

BBR periodically enters ProbeRTT to measure $\widehat{RTT}_{min}$, if $\widehat{RTT}_{min}$ is not updated for ten seconds. In ProbeRTT, the amount of inflight data is reduced to 4 packets for at least 200ms or one RTT. This should drain the queue in the buffer very quickly. Then BBR switches to ProbeBW if the bandwidth is filled, otherwise, it enters Startup to refill the bottleneck.

2.5 BBR'

BBR' [10] is an implementation of BBR for ns-3 [11], a network simulator for network systems. BBR' has similar behaviour and performance of BBR. However, there are some features in BBR that are not supported by BBR'. For example, after ProbeRTT, BBR' only switches to ProbeBW no matter the queue is drained or not.

2.6 The ns-3 network simulator

The ns-3 simulator is a discrete-event network simulator written in C++, it is mainly used on Linux or macOS systems [12]. ns-3 is open-source, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use. ns-3 provides users a simulation engineer and modules of how packet-level networks work and performance. ns-3 is not backwards compatible with ns-2, it is a new simulator that does not support the ns-2 APIs. One of the goals in the design of ns-3 is to make the implementation of the modules closer to the actual software implementation.

The ns-3 simulator has models for all network elements as in real computer network, e.g., network nodes, network devices, and communication protocols. In addition to the modules of network elements, ns-3 also has helper objectives that are not directly modelled in the simulations, they assist the execution of the simulation [11].

2.7 Related work

In this section, we introduce some related research about BBR. 2.7.1 introduces the experimental evaluation of BBR. 2.7.2 introduces other researches

about BBR completing with other congestion control algorithms. In 2.7.3, a congestion control designed by Google is presented. Some patches for the development of BBR are introduced in 2.7.4. 2.7.5 introduce researches about BBR fairness issue.

2.7.1 Experimental Evaluation of BBR

In [5], an experimental evaluation of BBR is introduced, the experiments are running on real machines using BBR's Linux kernel 4.9 implementation. They observed that BBR works as intended for single flow at the bottleneck. However, when there are multiple flows in competing at the bottleneck, the behaviour does not meet BBR's goal. They presented experiments for large buffer and shallow buffer scenarios. Large queueing delays were produced when the buffer is large, a huge amount of packet loss was observed when the buffer is small.

2.7.2 BBR versus other congestion control algorithms

Since BBR was published, there are many other researches studying how BBR interacts with other congestion control algorithms over the past years. When BBR is competing with loss-based congestion control algorithms, it becomes window-limited, the sending rate is limited by its 'in-flight cap' [13]. In [5], experiments for 1 BBR flow and 1 CUBIC flow when the buffer is large are presented, results show that they oscillate sharing the bottleneck equally. They also conducted experiments for 2 BBR flows and 2 CUBIC flows competing in the network when the buffer size is small, BBR flows will take most of the bandwidth of the bottleneck. Several studies have reproduced these results [14, 15, 16]. In [15], experiments for up to 10 CUBIC flows and up to 10 BBR flows competing in a large buffer network are presented, they found that BBR flows can always take at least 35% of the bottleneck bandwidth no matter how many number of BBR and CUBIC flows are competing.

In [16], existing congestion control algorithms are divided into three groups: loss-based, delay-based, and hybrid. They then run tests that flows of the same group and across groups competing over the bottleneck, they find that BBR does not suit for a typical network, where flows relay on loss-based algorithms. In addition, results also show that BBR flows with higher RTTs would have a large share of the bandwidth. [17] also discusses when and when not to use BBR, they found that BBR works well for the network with shadow buffers though it will cause high retransmissions, and loss-based congestion control

algorithms are more suited for deep buffer networks. In [18], they measured BBR versus CUBIC on high-speed 4G LTE networks, and found that "CUBIC and BBR generally have similar throughputs, but BBR has significantly lower self-inicted delays than CUBIC.

2.7.3 An RTT-based congestion control

Transport Informed by MEasurement of LatencY (TIMELY) [19] is an RTT-based Congestion Control designed for datacenter by Google. They showed that Network Interface Controller (NIC) hardware makes it possible to accurately measure RTT, and with microsecond accuracy, these RTTs can be used to measure switch queuing. Then they presented how TIMELY adjusts the sending rate to keep a low latency while having a high throughput.

2.7.4 BBR patches

BBR is still under development. Several updates have been presented at IETF conferences since 2016 [20, 21, 22, 23]. Additionally, internet drafts are given in [24, 25]. In [20], the direction and plans for improving BBR are presented. in [21], Linux TCP BBR patches for higher wifi throughput and lower queuing delays were released. The new algorithm for BBR v2 was described in [22]. In [23], BBR v2 open source "alpha/preview" was released.

2.7.5 Fairness of BBR

Several researches [5, 26, 27, 28] have found that BBR has fairness issue when studying the performance of BBR. Flows with long RTT will take more bandwidth than flows with short RTT. In [26], they introduced Modest BBR which enables better fairness than loss-based congestion control algorithms such as CUBIC. In [27], a derivative of BBR is proposed to guarantee flow fairness, it can be implemented in the Linux kernel. In [28], they improve fairness between flows with different RTTs and ensure better throughput with there BBR-E.

Chapter 3

Methods

3.1 Topology

The experiments were run in the ns-3 simulator using the topology shown in Figure 3.1. The topology includes 6 senders and 1 receiver that directly connect to the router. The bottleneck is between the router and the receiver.

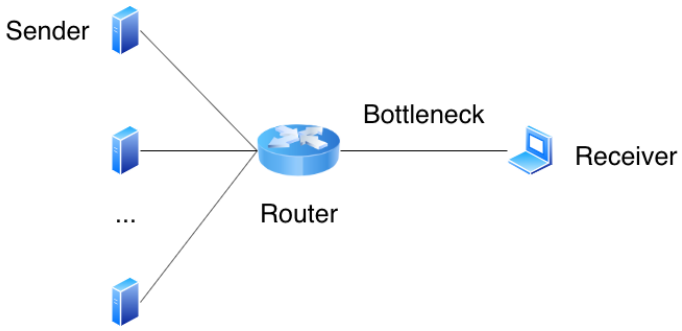


Figure 3.1: Testbed topology

3.2 Our improvement of BBR

BBR uses the pacing_cycle to control the pacing rate in ProbeBW, however, the static value of pacing_gain could lead to a large queuing delay especially when there are multiple competing flows in the network. For example, two BBR flows both send at 50 Mbps through a 100 Mbps bottleneck link. When the first flow probes for more bandwidth, the sending rate increases to 1.25 times of the initial sending rate, which is 62.5 Mbps, while the other flow

remains its sending rate. So the first flow will have more fair share, it will get a higher delivery rate although the bottleneck is fully utilized. And this higher delivery rate will be set as the new sending rate due to the maximum filter of BBR. Thus, the bottleneck becomes overloaded.

Thereby we use an RTT-based gain_cycle to adjust the pacing rate. The pacing_gain is adjusted by RTT instead of using constant values. An example code is shown below in Listing 3.1.

BBR may transit more packets than expected due to the overestimation of the available bandwidth, so, at the probe phase of pacing_cycle, when the RTT hits $1.25 \times min\_rtt$, BBR will stop probing for more and switch to the drain phase earlier. Then in the steady phase, we use $\alpha \times rtt\_min/rtt$ instead of the constant value, 1, as the pacing_gain. α is a coefficient whose range is limited to $(0, 1]$, rtt_min is the minimal RTT in the last 10 seconds, rtt is from the ACK indicating the current RTT of the flow.

So, when the packets start to accumulate in the queue, the RTT increases, which sets pacing_gain smaller than 1 and leads to draining the queue. The larger the queue, the smaller the pacing_rate.

```
1  // Set pacing rate (in Mb/s), adjusted by gain.
2
3  // α is a adjustable coefficient
4  double α = 0.75;
5
6  // Probe phase
7  if (m_pacing_gain == 1.25){
8    if (rtt.GetSeconds() / min_rtt.GetSeconds()) >=
     1.25){
9      m_pacing_gain = 0.75;
10   }
11 }
12
13 // Steady phase
14 if (m_pacing_gain == 1){
15   m_pacing_gain = α * min_rtt.GetSeconds() / rtt.
    GetSeconds();
16 }
17
18 // Get the estimation of the available bandwidth
19 double max_bw = getBW();
20
21 // Set the pacing rate
22 double pacing_rate = max_bw * m_pacing_gain;
```

Listing 3.1: Example code of our improvement

3.3 Experimental design

In 3.3.1, the network metric we use to conduct the simulations are introduced, 3.3.2 provides the information about the hardware we use in the project.

3.3.1 Test environment

To evaluate the performance of the original BBR and our improved BBR, we designed some simulations using the setup shown in Table 3.1 and Table 3.2. The setup covers different network scenarios.

Table 3.1: Simulation setup for original BBR

Bottleneck	$\widehat{RTT}_{min}$	Buffer size	Flow number
20 Mbps	20 ms	large	2, 4, 6
20 Mbps	30 ms	large	2
20 Mbps	40 ms	large	2
20 Mbps	60 ms	large	2
50 Mbps	20 ms	large	2
200 Mbps	20 ms	large	2
20 Mbps	20 ms	small	2
50 Mbps	20 ms	small	2
200 Mbps	20 ms	small	2

For original BBR, we set up simulations for both large buffer and small buffer scenarios. In large buffer scenarios, we have at most 6 BBR flows completing at the bottleneck, we setup bottleneck to 20 Mbps, 50 Mbps, and 200 Mbps for both small and large buffer simulations, the RTT without buffer for large buffer simulations are 20 ms, 30 ms, 40 ms, and 60 ms. We only set 20 ms of $\widehat{RTT}_{min}$ and 2 flows for simulations in small buffer scenarios.

As for the improved BBR, we also conduct simulations for small and large buffer scenarios. In order to compare the results with the original BBR, the setup in bottleneck, $\widehat{RTT}_{min}$, and flow numbers are almost the same as the setup for original BBR. Besides, we choose 1 and 0.75 as α and 0.75 in large buffer simulations and 0.75 and 0.5 in small buffer simulations.

Table 3.2: Simulation setup for improved BBR

Bottleneck	$\widehat{RTT}_{min}$	Buffer size	α	Flow number
20 Mbps	20 ms	large	1	2, 4, 6
20 Mbps	20 ms	large	0.75	2, 4, 6
20 Mbps	30 ms	large	1	2
20 Mbps	40 ms	large	1	2
20 Mbps	60 ms	large	1	2
50 Mbps	20 ms	large	1	2
50 Mbps	20 ms	large	0.75	2
200 Mbps	20 ms	large	1	2
20 Mbps	20 ms	small	0.75	2
20 Mbps	20 ms	small	0.5	2, 4, 6
50 Mbps	20 ms	small	0.75	2
50 Mbps	20 ms	small	0.5	2
200 Mbps	20 ms	small	0.75	2
200 Mbps	20 ms	small	0.5	2

3.3.2 Hardware to be used

In order to run the experiments of each scenario in the simulated environment, one x86 64 physical machine with 16 cores was used. The CPU model of this physical machine was Intel Haswell Xeon E5-2667 v3 3.20GHz. The second machine was accessed remotely using SSH.

3.4 Planned data analysis

In this section, we introduce how we collect data from the experiments and the technique and tools we used to analyze them.

3.4.1 Data Analysis Technique

We use tracing sources of ns-3 to keep track of the queue size in the buffer. All necessary data (RTT, packet loss, inflight packets, etc.) are documented in the log file. We extract these data from the log file and export them into a CSV file. Then use Plotly [29], an open-source plotting python library to plot figures of the parameters we want to observe. The retransmission rate is collected from the pcap file generated by ns-3.

3.4.2 Software Tools

The implementation of BBR for ns-3 (version 3.27) is taken from [30]. We use the Jupyter Notebook of Anaconda Navigator to plot the RTT figures of our simulation results and calculate the average RTT, and we use Excel to record the average RTT and total throughput of each test.

3.5 Reliability

In order to make sure our results are reliable, each test was repeated for 5 times, and each simulation lasted enough time to get a relatively stable result. Our experiments are repeatable by following the topology, network metric and the script we use.

Chapter 4

Results

In this chapter, we present the results of our experiments running on the ns-3 simulator. The simulations varied in latency, bottleneck bandwidth, flow numbers, and buffer size.

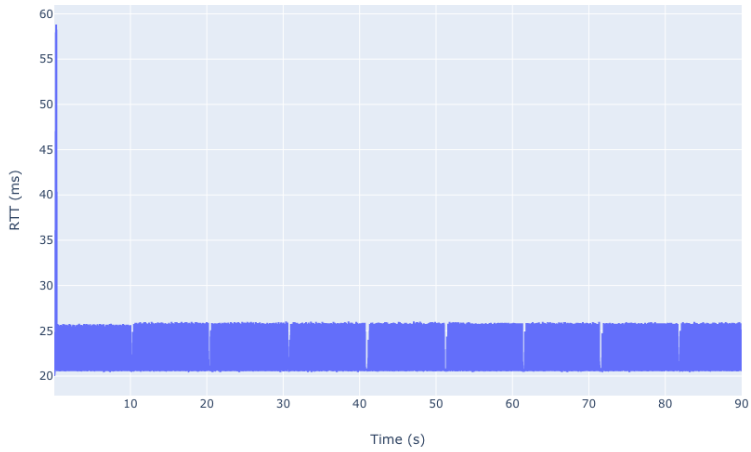
Our results align with the main thesis statement, which is *"The TCP BBR congestion control suffers from high buffer occupancy, a problem that can be mitigated in simple test cases by re-configuring some of its parameters"*. We study the original behaviour of TCP BBR in Sect. 4.1, how our improvement of BBR performs in similar scenarios in Sect. 4.2, and a comparison between the two mechanisms in Sect.4.3.

Each simulation is run for 90 seconds. Each flows is started 3 seconds after the previous one. The large buffer is set to 80 $\times$ BDP to avoid packet loss and the small buffer is set to 0.8 $\times$ BDP. Each test was repeated for five times, we only show the result of a representative run for clarity.

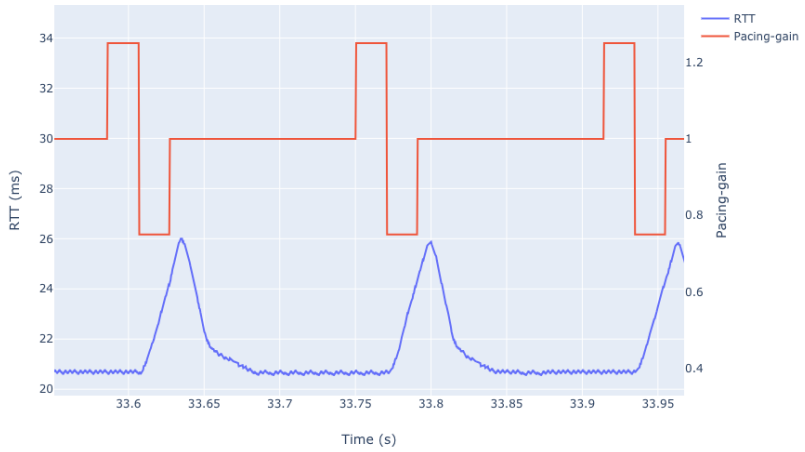
4.1 Original BBR‘s behaviour

We first present the original BBR's behaviour. Figure 4.1(a) shows that a single BBR flow performs as intended. The measured minimal RTT is a little above 20ms because of the processing time of the emulator. In Startup state, the RTT rapidly increases to around 58 ms as the pacing_gain = 2.89. Then in Drain state, the queue built in Startup is drained. The RTT drops regularly because BBR enters ProbeRTT state every 10 seconds. In ProbeBW state, as shown in Figure 4.1(b), the RTT changes with the pacing_gain with one RTT delayed, when the pacing_gain = 1.25, a queue is built at the bottleneck, the RTT increases to around 25 ms, then the queue is drained when the pacing_gain = 0.75, and as the available bandwidth is estimated correctly, there

are no additional queuing delay when the pacing_gain = 1.



(a) RTT

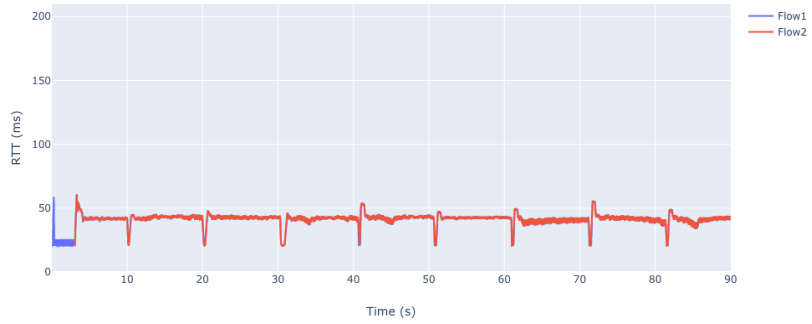


(b) RTT (zoomed)

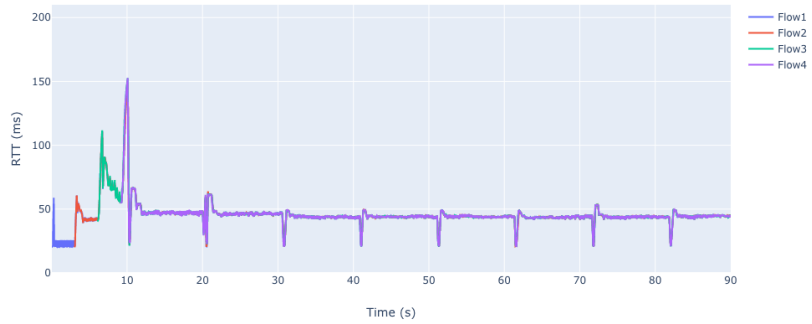
Figure 4.1: RTT of a single original BBR flow, bottleneck = 20 Mbps, RTT_{min} = 20 ms

BBR overestimates the available bandwidth at the bottleneck in multiple flows scenarios and the cwnd_gain is set to 2 at ProbeBW state, which doubles the RTT when the buffer is large as we observed in Figure 4.2. The peak at

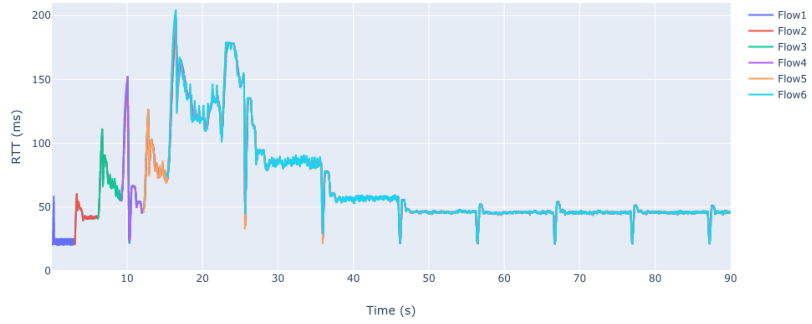
the beginning is caused by the large pacing_gain at the Startup state. And with more flows, the peak goes higher and the convergence time gets longer since the queue starts to accumulate after the second flow joins. Figure 4.3 shows the queue and inflight of 2 original BBR flows with 20 Mbps bottleneck, when the first flow starts, the inflight increase to 14000 Bytes ($2/\ln 2 \times$ BDP) at Startup state, but then the queue is drained at Drain startup. Before Flow 2 joins, everything works as intended, the inflight is controlled by pacing. However, after that, when the available bandwidth at the bottleneck is overestimated, the inflight is not limited by pacing anymore, it is limited by the congestion window which is $2 \times$ BDP. At ProbeBW state, the queue built in probe phase can not be drained up in drain phase, so the queue stays at around 50 packets. As we can see in Figure 4.3(b), the inflight decreases to about 4000 Bytes (4 packets) every 10 seconds due to the reduction of the congestion window at the ProbeRTT state. And the queue is completely drained at the ProbeRTT state. In 4.3(c), we can see how congestion window and inflight of Flow 1 changes. When there is only one flow, the inflight is stable at 1 BDP, the congestion is twice of the inflight since the cwnd_gain is 2 at ProbeBW state. The congestion window and the inflight overlaps after Flow 2 joins because the overestimation of bandwidth, the inflight is limited by the congestion window which also means that the pacing rate is limited by the congestion window.



(a) 2 flows

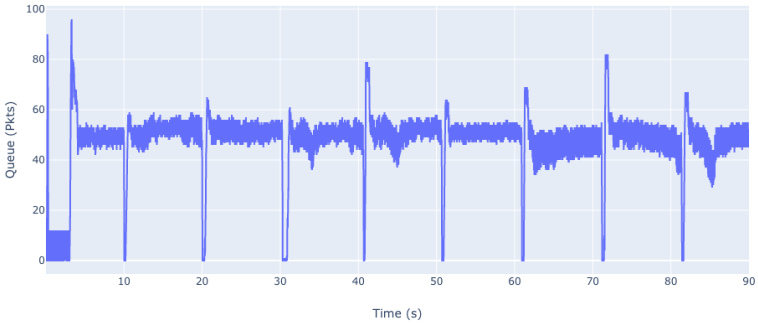


(b) 4 flows

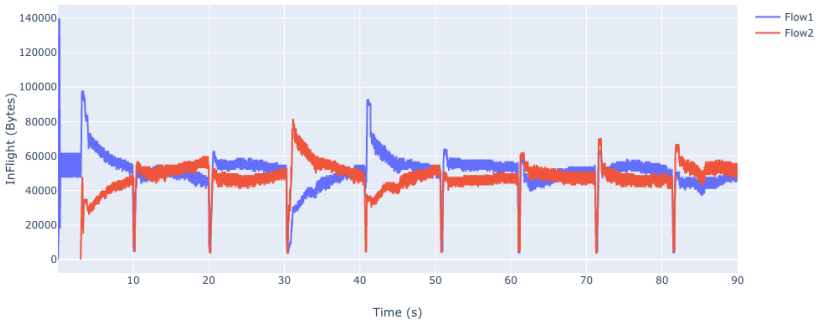


(c) 6 flows

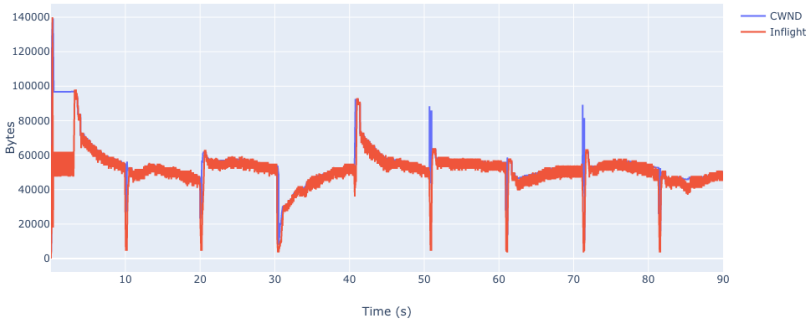
Figure 4.2: RTT of multiple original BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, large buffer



(a) Queue



(b) Inflight



(c) CWND and inflight of Flow 1

Figure 4.3: 2 original BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, large buffer

When the buffer is small, the observed RTT is almost constantly above 36 ms (20 ms RTT_{min} + 16 ms maximum queuing delay) as shown in Figure 4.4. This indicates that the operating point of BBR in small buffer is actually crossed (B) (cf. Figure 1.2) instead of at (A). In our simulations, at a certain point, some flows were killed because of experiencing large TCP retransmissions, for example, in Figure 4.4, flow 2 is killed at 25.8s.

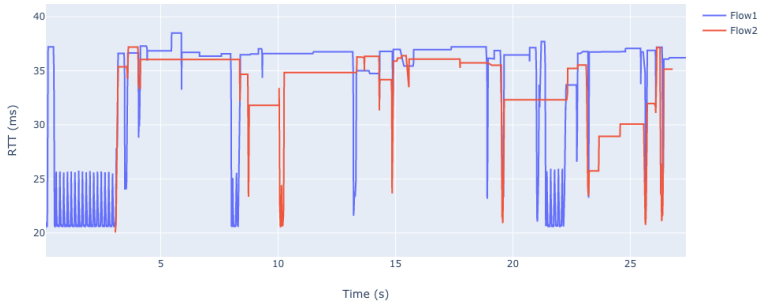
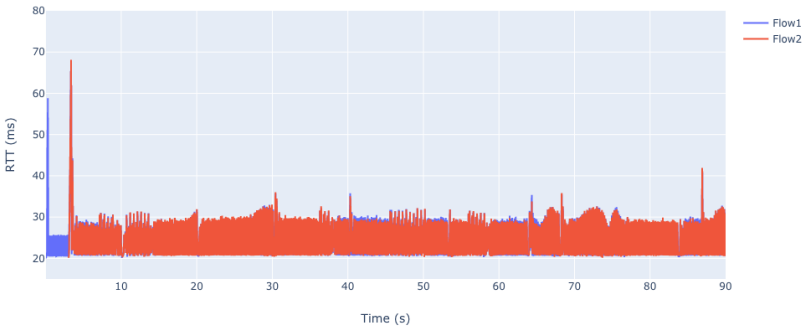


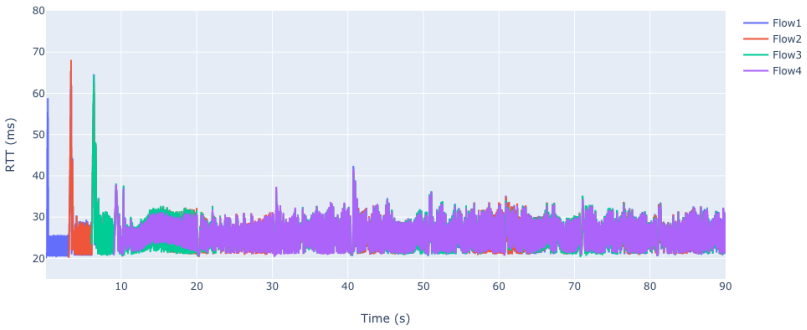
Figure 4.4: RTT of 2 original BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, small buffer

4.2 Improved BBR's performance

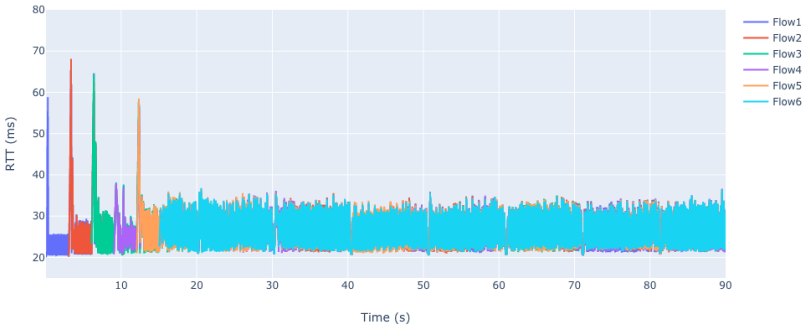
The results of multiple improved BBR flows with large buffer are provided in Figure 4.5. As we expected, the RTT significantly decreases with our improvement of BBR. However, the queue is not completely removed, because the pacing_rate is adjusted after the RTT gets higher than the $\widehat{RTT}_{min}$, which means the queue will be built first, then removed, and then built again. For example, as Figure 4.6 shown, the RTT starts to increase in the probe phase of the gain_cycle. However, due to the overestimation of the available bandwidth, when it exits the probe phase, the queue is still growing until the RTT reaches around 30 ms, then with our method, the pacing_rate slows down and the queue is drained, then it enters another cycle, the queue is built again. Our improvement only adjusts the pacing_gain in ProbeBW state, so the queue built in Startup state is still there. However, the convergence time is much shorter than with the original BBR because the queue does not accumulate after more flows joining in.



(a) 2 flows



(b) 4 flows



(c) 6 flows

Figure 4.5: RTT of multiple improved BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, α = 1, large buffer

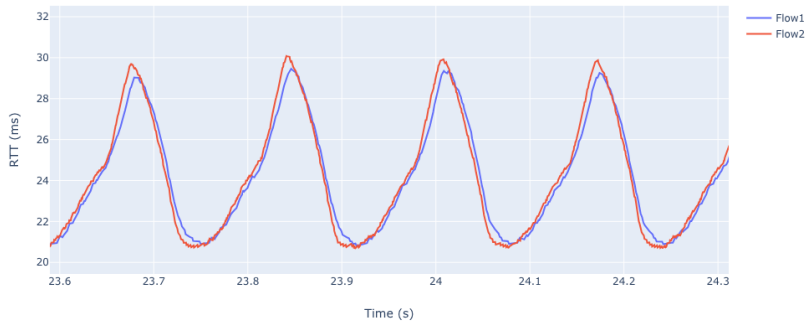


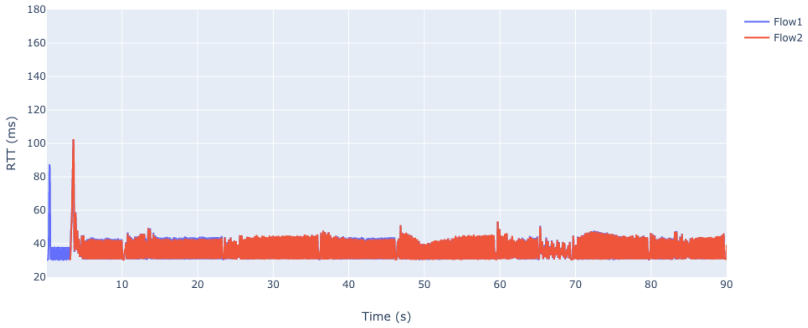
Figure 4.6: RTT of 2 improved BBR flows corresponding Figure 4.5(a) (zoomed)

Figure 4.7 shows the RTT of our improved BBR with different RTT_{min} when the buffer is large. It can be observed that our improvement also works for different RTT_{min}. The RTT can increase up to 1.5 $\times$ RTT_{min}, but this can be narrowed down by a smaller coefficient α. The results with more flow numbers have a similar trend, so here we only show the results of 2 flows.

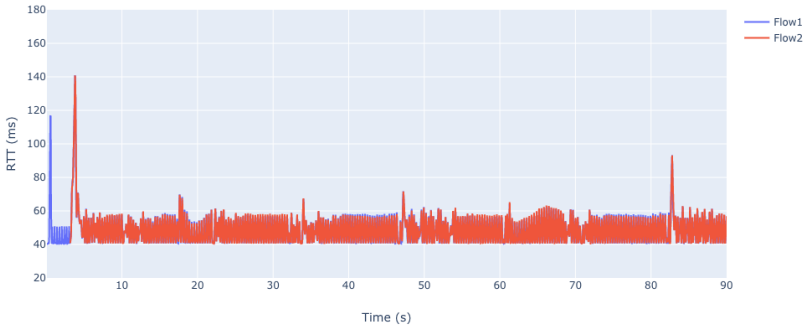
The comparison of different bottlenecks of our improved BBR in large buffer is presented in Figure 4.8. Different bottleneck does not affect how RTT changes. So, the results are almost the same because they have the same RTT_{min}.

When the buffer is small, the improved BBR also performs good. As shown in Figure 4.9, the RTT in Startup increased to about 36 ms (20 ms RTT_{min} + 16 ms queuing delay), it is limited by the buffer instead of the pacing_gain. In ProbeBW, the queue does not keep building in the bottleneck, the RTT stays below 36 ms.

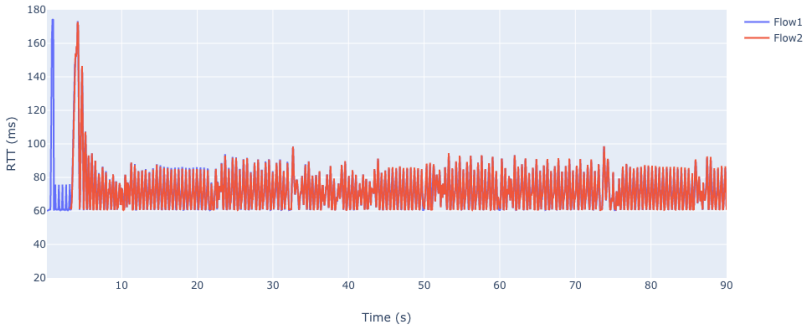
Figure 4.10 shows the RTT of 2 improved BBR flows of different RTT_{min}, in Startup, the RTT can only reach to 1.8 $\times$ RTT_{min} because of the limit of 0.8 $\times$ BDP buffer size. In ProbeBW state, the RTT is also controlled below 1.8 $\times$ RTT_{min} thanks to the dynamic pacing cycle.



(a) 30 ms

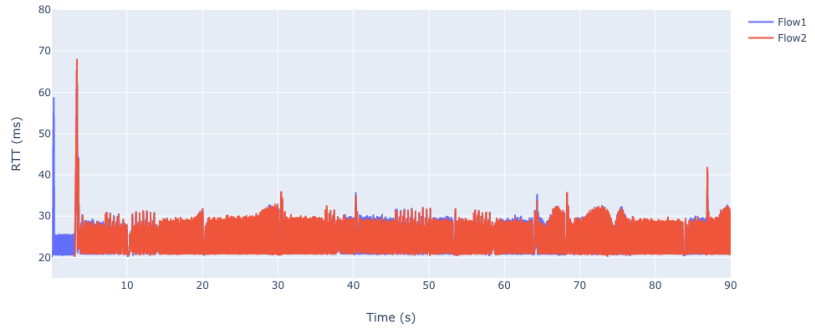


(b) 40 ms

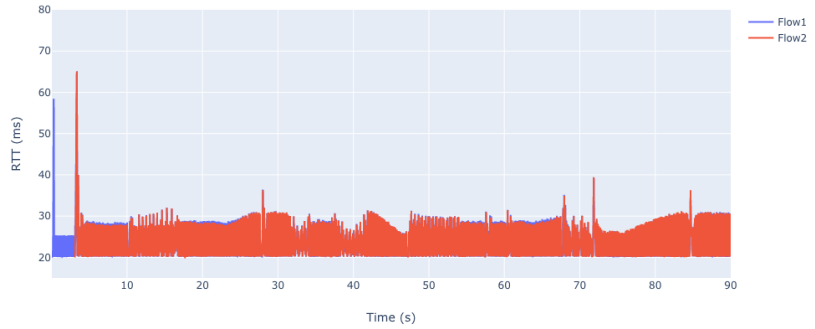


(c) 60 ms

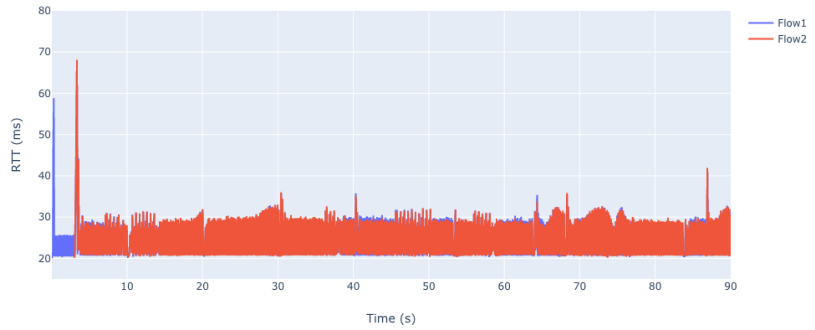
Figure 4.7: RTT of 2 improved BBR flows, bottleneck = 20 Mbps, comparison of different RTT_{min}, α = 1, large buffer



(a) 20 Mbps

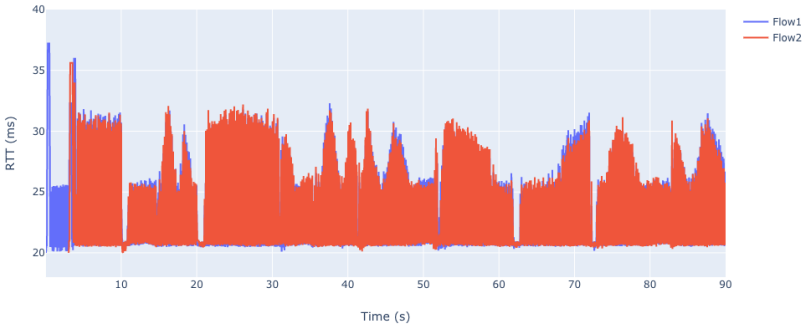


(b) 50 Mbps

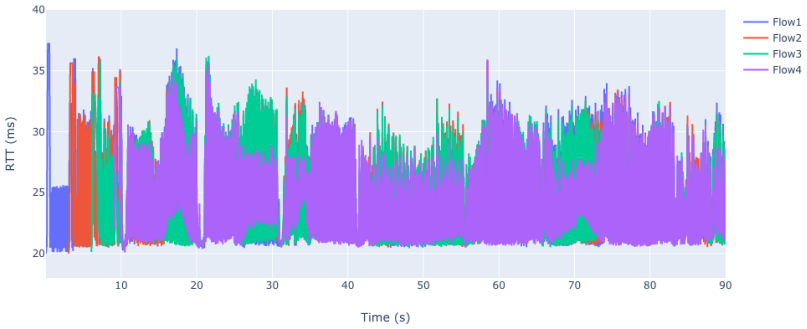


(c) 200 Mbps

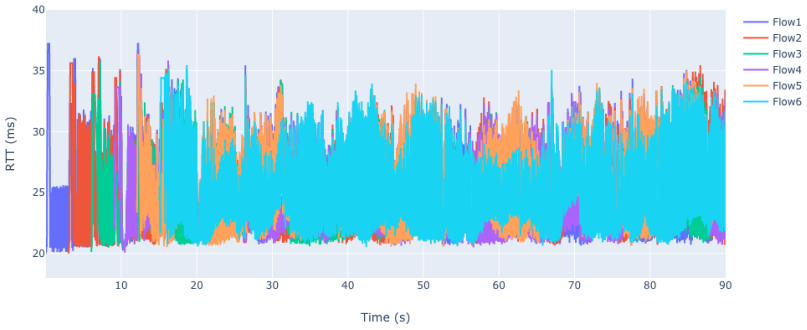
Figure 4.8: RTT of 2 improved BBR flows, RTT_{min} = 20ms, comparison of different bottleneck, α = 1, large buffer



(a) 2 flows

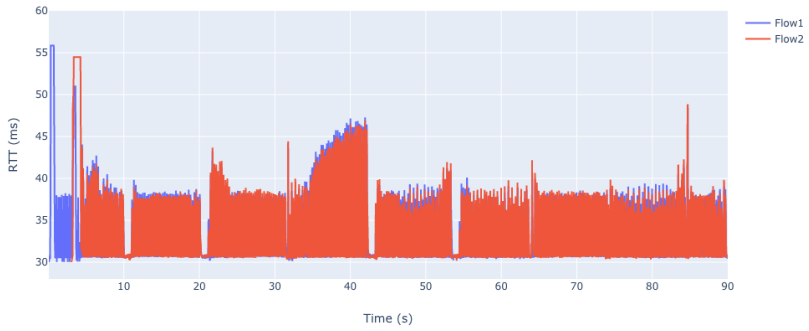


(b) 4 flows

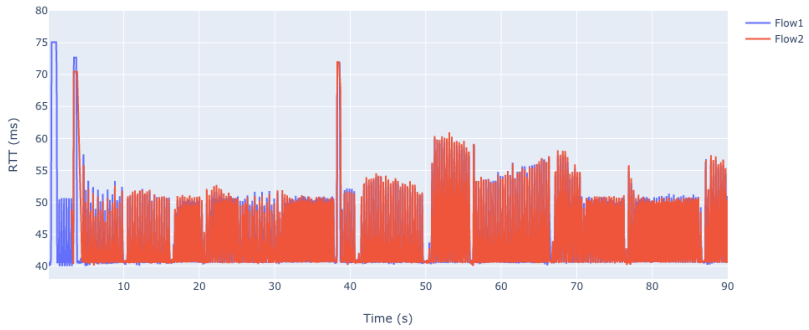


(c) 6 flows

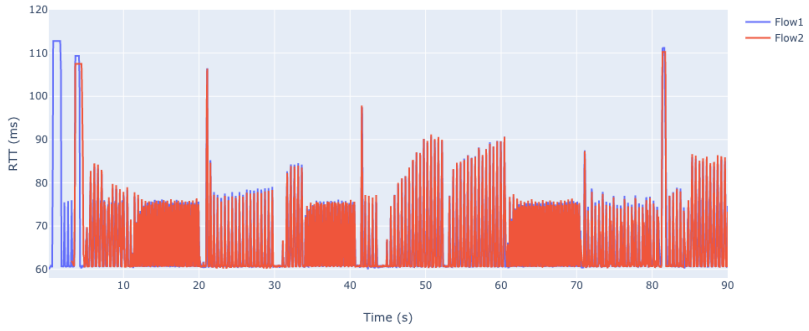
Figure 4.9: RTT of multiple improved BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, α = 0.5, small buffer



(a) 30 ms



(b) 40 ms



(c) 60 ms

Figure 4.10: RTT of 2 improved BBR flows, bottleneck = 20 Mbps, comparison of different RTT_{min}, α = 0.5, small buffer

The first flow's retransmission of 2 improved BBR flows when the buffer is small is shown in Figure 4.11. When the first flow starts, it enters Startup state which has large pacing gain, since the buffer is smaller, retransmission happens. Then in Drain state, the queue is drained, thus there is no retransmission packet. After 3 seconds, the second flow joins and enters Startup state, so retransmission occurs again, but after that, no more retransmissions thanks to the adjustment to the pacing rate. Since retransmission only happens at Startup state, and BBR only spends very little time at Startup, the packet loss is extremely small.

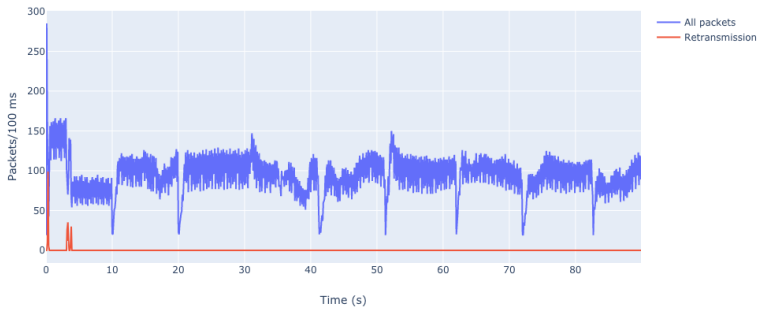


Figure 4.11: The first flow's retransmission of 2 improved BBR flows, RTT_{min} = 20 ms, α = 0.5, small buffer

4.3 Comparison between original BBR and our improved BBR

In this section, we compare the performance between the original BBR and improved BBR with different coefficients α. We focus on the comparison of total throughput and average RTT when the buffer is large, and the comparison of retransmission rate when the buffer is small. The log file recorded the RTT when each ACK arrives, thus, we calculated the average RTT by using the RTTs in the log file. We calculated total throughput by using total received data divided by the running time. In each simulation, we used Wireshark to monitor each flow, we calculate the retransmission rate by using total retransmitted packet number divided by total sent packet number.

Figure 4.12 shows the comparison of average RTT between original BBR and our improvement. We only present the results when the bottleneck is 20 Mbps and RTT_{min} is 20 ms since the results in other scenarios have a similar

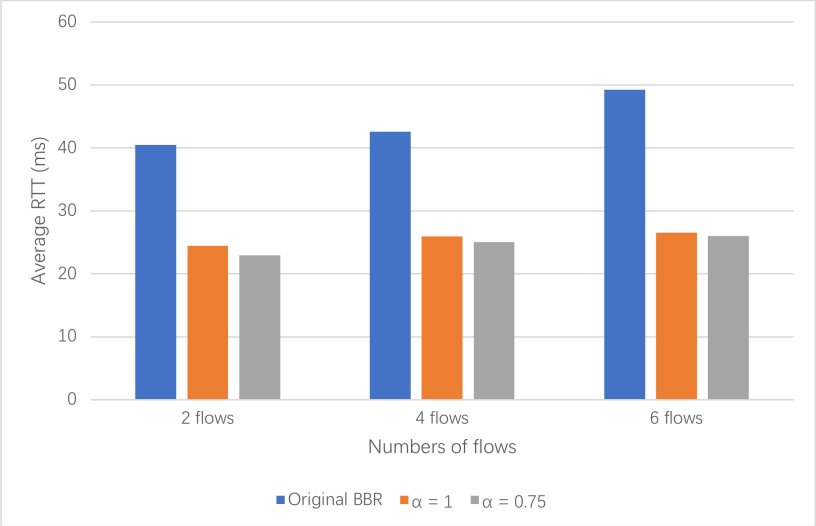


Figure 4.12: Average RTT of different number of BBR flows, bottleneck = 20 Mbps, RTT_{min} = 20 ms, large buffer

trend. The average RTT is 2 to 2.5 times of the RTT_{min} with original BBR due to the overestimation of available bottleneck bandwidth and 2 BDP limit of the congestion window. Our improvement significantly reduces the average RTT to around half of the original cases. And the smaller the α is, the lower the average RTT is.

The total throughput of the original and improved BBR is provided in Figure 4.13. We also tested with other bottlenecks and RTT_{min}, they have a similar trend of what we show here. The total throughput of original BBR is around 47 Mbps with a 50 Mbps bottleneck. With our improvement, the throughout is a bit lower since the pacing_rate is slower than in the original BBR. And the smaller the α is, the lower the total throughput is. However, the cost is only about 5% even when α is 0.75, which is acceptable.

As we can see from the above results, when the buffer is large, the average RTT cut in half with our improvement, with a very small sacrifice of the total throughput. However, with more flows in the network, the cost becomes smaller, but the queuing delay becomes larger.

When the buffer is small, as shown in Table 4.1, the original BBR has extremely large retransmission rate, while the improved BBR only has around 0.1% retransmission rate in most cases.

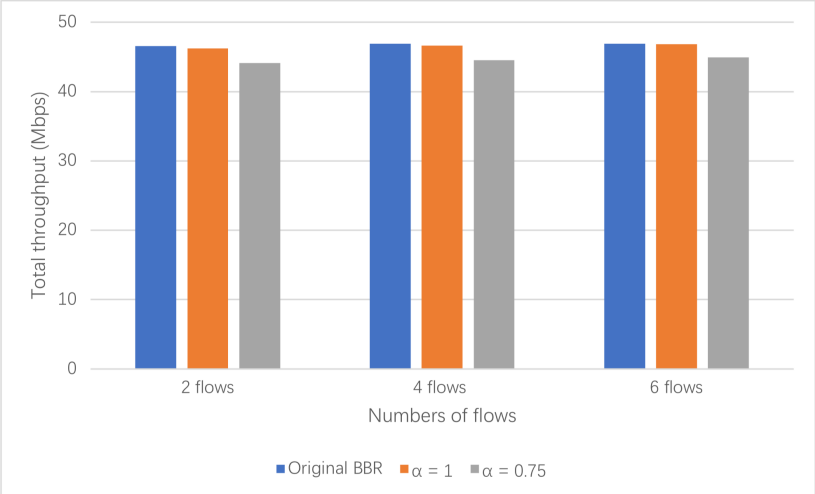


Figure 4.13: Total throughput of different number of BBR flows, bottleneck = 50 Mbps, RTT_{min} = 20 ms, large buffer

Table 4.1: Retransmission rate of 2 BBR flows, RTT_{min} = 20 ms, comparison of different bottleneck, small buffer

	Original BBR	$\alpha = 0.75$	$\alpha = 0.5$
20 Mbps	36.3%	0.1%	0.1%
50 Mbps	28.4%	0.1%	1.7%
200 Mbps	20.8%	0.1%	0.1%

Chapter 5

Conclusions and Future work

5.1 Conclusions

In this thesis, we introduce our improvement of BBR for reducing the queuing delay and packet loss. The results of our experiments showed that the original BBR works well with a single flow but experiences large queuing delay and massive retransmission when there are multiple flows. Our improvement to the original BBR can significantly reduce the additional queue on the bottleneck while keeping a high throughput, and when the buffer is smaller, the retransmission rate is able to reduce to less than 0.1%. However, our method does not remove the queue completely, we slow down the pacing rate as soon as the queue is built on the bottleneck link.

BBR is still under developing, many people are also studying and improving BBR. We believe our work could bring some insights into the further development of BBR.

5.2 Limitations

Our experiments are running on the network simulator instead of the real machine. The implementation of BBR for Linux is not exactly the same as the implementation for ns-3. So the behaviour of BBR on ns-3 could be different from on the real machine in certain scenarios. Also, the code we use to run our experiments haven't been updated since 2018, there could be potential bugs causing unusual results.

5.3 Future work

There are many other scenarios of the network that have been left for future work due to the lack of time. We only tested our improvement on a simple network topology, larger and more complicated topology can be tested in future studies. Future work concerns a deeper analysis of BBR's behaviour and improvement in other aspects.

During our research, we also found some interesting problems of BBR, for example, the RTT fairness which was also discussed in [5]. However, this thesis focus on the problem of queuing delay and packet loss. How our improvement works on RTT fairness remains to study.

We also tried to use the amount of inflight packet as an indicator to adjust the pacing rate during our research. Unfortunately, the results did not present the outcome that we expected in terms of improvements. We believe that finding a way to incorporate the number of inflight packets into the adjustment of the sending rate in BBR remains an open problem.

References

[1] Mark Allman, Vern Paxson, Wright Stevens, et al. “TCP congestion control”. In: (1999).

[2] Sangtae Ha, Injong Rhee, and Lisong Xu. “CUBIC: a new TCP-friendly high-speed TCP variant”. In: *ACM SIGOPS operating systems review* 42.5 (2008), pp. 64–74.

[3] Neal Cardwell et al. “BBR: Congestion-Based Congestion Control”. In: *ACM Queue* 14, September-October (2016), pp. 20–53.

[4] Van Jacobson. “Congestion avoidance and control”. In: *ACM SIGCOMM computer communication review* 18.4 (1988), pp. 314–329.

[5] Mario Hock, Roland Bless, and Martina Zitterbart. “Experimental evaluation of BBR congestion control”. In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE. 2017, pp. 1–10.

[6] Neil Cardwell et al. “TCP BBR congestion control comes to GCP your Internet just got faster”. In: *Google Cloud Platform Blog, July* 20 (2017).

[7] Anne Håkansson. “Portal of research methods and methodologies for research projects and degree projects”. In: *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*. CSREA Press USA. 2013, pp. 67–73.

[8] Sally Floyd, Tom Henderson, Andrei Gurtov, et al. “The NewReno modification to TCP’s fast recovery algorithm”. In: (1999).

[9] Lisong Xu, Khaled Harfoush, and Injong Rhee. “Binary increase congestion control (BIC) for fast long-distance networks”. In: *IEEE INFOCOM 2004*. Vol. 4. IEEE. 2004, pp. 2514–2524.

[10] Mark Claypool, Jae Won Chung, and Feng Li. “BBR’: an implementation of bottleneck bandwidth and round-trip time congestion control for ns-3”. In: *Proceedings of the 10th Workshop on ns-3*. ACM. 2018, pp. 1–8.

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

[12] NS-3 development team. *Introduction - Tutorial*. 2019. URL: https://www.nsnam.org/docs/release/3.30/tutorial/html/introduction.html.

[13] Ranysha Ware et al. “Modeling BBR’s Interactions with Loss-Based Congestion Control”. In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 137–143.

[14] Mo Dong et al. “{PCC} vivace: Online-learning congestion control”. In: *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 2018, pp. 343–356.

[15] Dominik Scholz et al. “Towards a deeper understanding of tcp bbr congestion control”. In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE. 2018, pp. 1–9.

[16] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. “Fifty shades of congestion control: A performance and interactions evaluation”. In: *arXiv preprint arXiv:1903.03852* (2019).

[17] Yi Cao et al. “When to use and when not to use BBR: An empirical analysis and evaluation study”. In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 130–136.

[18] Feng Li et al. “TCP CUBIC versus BBR on the Highway”. In: *International Conference on Passive and Active Network Measurement*. Springer. 2018, pp. 269–280.

[19] Radhika Mittal et al. “TIMELY: RTT-based Congestion Control for the Datacenter”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 537–550.

[20] Neal Cardwell et al. *BBR Congestion Control: An Update*. Presentation at the ICCRG session at IETF 98. 2017. URL: https://www.ietf.org/proceedings/98/slides/slides-98-iccrg-an-update-on-bbr-congestion-control-00.pdf.

[21] Neal Cardwell et al. “BBR Congestion Control Work at Google: IETF 101 Update”. In: *Proc. ICCRG at IETF 102th Meeting*. 2018.

[22] Neal Cardwell et al. *BBR v2: A Model-based Congestion Control*. Presentation at the ICCRG session at IETF 104. 2019. URL: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00.

[23] Neal Cardwell et al. *BBR v2: A Model-based Congestion Control IETF 105 Update*. Presentation at the ICCRG session at IETF 105. 2019. URL: https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00.

[24] Neal Cardwell et al. *BBR Congestion Control, draft-cardwell-iccrg-bbr-congestion-control-00*. 2017. URL: https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00.

[25] Yuchung Cheng et al. *Delivery Rate Estimation, draft-cheng-iccrg-delivery-rate-estimation-00*. 2017. URL: https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00.

[26] Yuxiang Zhang, Lin Cui, and Fung Po Tso. “Modest BBR: Enabling better fairness for BBR congestion control”. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2018, pp. 00646–00651.

[27] Shiyao Ma et al. “Fairness of congestion-based congestion control: Experimental evaluation and analysis”. In: *arXiv preprint arXiv:1706.09115* (2017).

[28] Geon-Hwan Kim et al. “Enhanced BBR Congestion Control Algorithm for Improving RTT Fairness”. In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. 2019, pp. 358–360.

[29] Plotly. *Modern Analytic Apps for the Enterprise*. 2020. URL: https://plot.ly/.

[30] Claypool Mark. *GitHub: mark-claypool/bbr*. 2018. URL: https://github.com/mark-claypool/bbr.

TRITA-EECS-EX-2020:855