Networked Systems
ETH Zürich — seit 2015

Make them buffer!

Bachelor Thesis

Author: Spencer Tellini

Tutor: Alexander Dietmüller

Supervisor: Prof. Dr. Laurent Vanbever

March 2023 to June 2023

Abstract

One of the prime applications in the networks of today is video streaming. Key factors for the user-perceived quality-of-experience (QoE) are: the average playback bitrate, the duration of rebuffering, the change in bitrate and the startup delay. Adaptive bitrate (ABR) algorithms try to optimize the QoE. But we lack reliable benchmarks to understand whether an algorithm is good or not. In this report, we investigate what parts of communication networks make video streaming more challenging. We first have a look at the background and the work of two related research studies: Puffer and Pantheon. Pantheon has created six emulation environments for their real world paths. After that, we explain how the emulation environment has been designed, how to use it and what can be done with it. Further, we experiment with three ABR algorithms (Linear BBA, MPC and Fugu) and the emulation environments provided by Pantheon. We find that the Nepal-India environment is the most challenging of the Pantheon environments. Important factors why Nepal-India is more challenging are the total throughput, the packet queue size and the loss rate. We focused on the packet queue size and found that the increased queue size first has a positive impact on the algorithms, but the larger the queue becomes the slower is the reaction time for the algorithms. At a certain size the simple Linear BBA drops again in performance, where on the other hand the more sophisticated Fugu doesn't lose performance as it has a better understanding of the network dynamics. This indicates that larger queue size leads to slower reaction times for the algorithms, which makes an environment more challenging.

Contents

Chapter 1

Introduction

Video streaming is one of the prime applications in today's networks and is responsible for over 65% of the traffic [9]. The objective of video streaming application is to maximize their users experience.

The quality-of-experience (QoE) is an indicator for the viewers perception of the video quality. Some of the key factors for the QoE are the average playback bitrate, the amount of rebuffering (the time where the video stalls), the change in bitrate (if the bitrate changes too much or too often we notice this as the video doesn't run smooth) and startup delay (time it takes for the video to start running) [13].

Most commonly, the video is available as chunks that are encoded at a variety of bitrates. Adaptive bitrate (ABR) algorithms help optimizing the QoE, as they try to predict the state of the network and with that choose a bitrate for the next chunk.

ABR algorithms range from simple buffer based [4], which picks its next bitrate depending on the buffer occupancy, to control-theoretic approaches like MPC [13] to new algorithms leveraging AI [5].

But we lack reliable benchmarks to understand whether a new algorithm is good or not. Evaluating these algorithms turns out to be a challenge. The options for the evaluation are either a network simulation/emulation or evaluating on real world network environments. The simulation and emulation are fast but can be unreliable, because it is difficult to properly mimic the behaviour of real networks [7, 10].

Therefore, the algorithms performance in emulated environments may not translate effectively to the real-world Internet conditions [3].

Evaluating on real world environments takes a lot of time and effort to set up and doesn't guarantee a challenging environment, which would give enough indication if an algorithm is good or not. For example when we compare Puffer's results from the past with the results today. Over time the puffer environment has become seemingly easier, with the same algorithms performing better than in the past [1]. This shows that just because it is real, it is not guaranteed to be a testbed.

In order to create a more challenging simulation/emulation environment, we first have to better understand what parts of communication networks make video streaming challenging. To achieve this, we emulate real world environments, compare their characteristics and how the algorithms perform on them, to further test and conclude what contributes to a challenging environment.

The remainder of this report is structured as follows. Chapter 2 gives more general information on how video streaming works, how adaptive bitrate (ABR) algorithms work and why it is complex to choose the optimal bitrate. After that we have a closer look at emulation and simulation, where we see some advatages and limitations.

In chapter 3 we introduce the work of research projects we build on. First Puffer [11], they have built a streaming website to measure the behaviour of ABR algorithms. Second Pantheon [12], they have built real network paths all over the world and created an emulation environment for each path.

In chapter 4 we explain how we have built a docker-based setup opting as client and server, with a Mahimahi [6] emulation shell around the client.

In chapter 5 we experiment with the different algorithms and environments to compare the characteristics and performances. With the conclusions drawn from the experiments, we test further to gather additional information.

Chapter 6 gives a short summary about the report and an outlook for future testing.

Chapter 2

Background

2.1 Video Streaming

Today, HTTP-streaming is a common form of video delivery. The videos are stored as chunks on a server which are normally the size of just a few seconds. These chunks are encoded in different bitrates: the higher the bitrate the higher the quality and also the larger the chunk.

Adaptive bitrate (ABR) algorithms can choose at every chunk boundary what bitrate the next chunk should have. This allows for adaptive video streaming, this means for example if the bandwidth is too small to watch a video on 4k without stalling all the time, the ABR algorithm might switch to a lower bitrate.

The goal of ABR algorithms is to choose the bitrates to maximuze QoE, i.e. maximize video quality while minimizing stalls or sudden jumps in quality.

2.2 Adaptive Bitrate Algorithm

The goal of an ABR algorithm is simple, optimize the QoE. For this the ABR chooses at every chunk boundary the bitrate for the next chunk. The real world network can be very dynamic which makes the bitrate selection tricky.

An other reason why choosing the right bitrate is not simple, is because an ABR algorithm has to consider different objectives that lead to the QoE.

Concretely an ABR algorithm wants to maximize the average bitrate, minimize the rebuffering time, minimize the change in bitrate and minimize the start up delay. However, these objectives are conflicting. A higher bitrate increases the risk of rebuffering, so for example we have a small bandwidth, to maximize the average bitrate the algorithm could choose always the highest possible bitrate, this would lead to frequent rebuffering. Another example, if we have a dynamic network and it's possible throughput is varying, choosing the highest bitrate that can still be provided by the network would lead to many changes in the quality, so the smoothness suffers from this.

There are buffer-based approaches, they choose the bitrate solely based on the playback buffer occupancy. There are rate-based approaches, they choose based on the throughput estimates from previously downloaded chunks. And there is the combination, where the decision depends on both buffer and previous bitrates.

2.3 Emulation and Simulation

Network emulation and simulation are techniques used to replicate the behaviour of a real network.

An emulation runs real applications in a virtualized network, a simulation virtualizes the application itself and computes what it would do.

With emulation one can connect real applications and/or network devices with virtual links that mimic user-defined network characteristics and dynamics. These characteristics and dynamics are defined as parameters, for example a constant propagation delay or a stochastic loss rate.

For a simulation the network model and the application have to be defined. This means the whole network topology, including network elements (routers, switches, hosts) and the connections between them (bandwidth, delay, loss, etc).

The advantages of both simulation and emulation over a real network are, they are fast and simpler to set up, they are reproducible and can be controlled, they are useful for testing and learning as it doesn't affect the real network.

But there are also limitations with the biggest one being the credibility. This will always be one, since it is not possible to guarantee perfect real world behaviour [8].

With that being said, we had to choose whether we use an emulation or simulation. We went for the emulation, because it allows us to connect two end systems, in our case server and client. We felt that this would make the experiments more authentic and needs less effort, as we do not have to re-implement an application for a simulator.

Chapter 3

Related Work

We use the the server and client setup from Puffer [11] and emulation of real world network paths provided by Pantheon [12]. In the following we introduce them and describe how we use them.

3.1 Puffer

Puffer is an ongoing research study, where they try to understand the challenge of video streaming and measure the performance of ABR algorithms over real networks.

To make this possible, they have built a streaming website and provide six over-the-air television channels. User's will be assigned a random ABR algorithm at the start of every session without them knowing which one. With this setup they can record important data about the performance of the algorithms.

Puffer records client telemetry on video quality and stalling. The concept behind this approach is to gather data from a sufficient number of participants and network paths, to derive reliable insights about the performance of algorithms.

As a metric for quality, they use SSIM, which stands for "structural similarity index measure". This is a method to predict the perceived quality.

Puffer also created an own algorithm called Fugu, we will also use a version of Fugu for our experiments and we will describe it in more detail in section 5.1.1. They use a control-theoretic algorithm and replace the throughput prediction with a transmission time predictor (TTP) which uses machine learning. This TTP is supposed to be trained on the clients data, so it can optimally perform and is adaptable to everywhere.

Surprisingly what they have found, was that sophisticated algorithms, which combine buffer-based and rate-based approach or use machine learning, did not perform better than a simple buffer-based approach. They have found that the best way for an algorithm to perform well over the wild Internet, is to learn from the environment where it is deployed. With this, the algorithm can adapt more to the current environment and does not need to be as generalized as if it would have to function on every different environment at the same time. The algorithms performed all about the same with the exception of Fugu which outperformed them.

We use the open-source Puffer client and server to run ABR algorithms in our network emulation.

3.2 Pantheon

The research of Pantheon is about congestion control and transport protocols. They provide a "training ground" for researchers to test their transport protocols and congestion control schemes.

They have created real world paths from and to places all over the world, where they did experiments multiple times a week between these places.

They also recreated these real world paths in an emulation. For the emulation they use Mahimahi. They have found that a 5-parameter network model is sufficient to create different emulators that can reproduce the throughput and delay within 17% on average.

The five mahimahi parameters they used are a constant propagation delay, a bottleneck link rate, a stochastic loss rate, a droptail threshold for the sender's queue and a bit that decides whether the interarrival times are equal or follow a memoryless poisson process, which can be often observed on cellular networks [10]. In our experiment, we determine the interarrival times with trace files, which will be later explained in section 5.1.2.

Chapter 4

Emulation setup

4.1 Creation of the emulation environment

We use a docker framework to run a client and a server container. The client container and server container are independently running, but can communicate with each other over a link.

The server container contains the Puffer interface and already prepared streaming data. This Puffer interface includes a webpage, where clients can log in and decide what to stream, the algorithms, where we can choose and configure them the way we want, and more to make this all work properly. There are around 7GB of pre-recorded TV-clips for test purposes, that are already encoded the right way, video chunks with length 2.002 seconds and audio chunks with length 4.8 seconds.

The client container watches a video stream. The client is implemented to directly go to the server's webpage, create an account, log in and start the stream. While watching, the client logs statistics about the video stream, for example the SSIM of the current chunk, the resolution, the buffer occupancy and a timestamp. It can be configured what should be outputted and stored.

We use the emulation tool Mahimahi [6] to create different network environments. Mahimahi allows us to build a shell around our client, that means each traffic incoming or leaving first has to pass through this shell. With that we can configure an environment in the shell that emulates a certain behaviour. For example a constant delay or a packet loss rate.

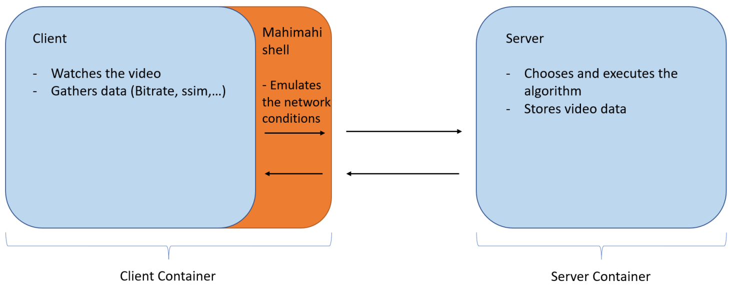

Client
- Watches the video
- Gathers data (Bitrate, ssim,...)
Mahimahi shell
- Emulates the network conditions
Server
- Chooses and executes the algorithm
- Stores video data
Client Container
Server Container

Figure 4.1: Our emulation setup with the client container and the server container.

4.2 How to use the setup

The creation of an environment is simple, in the following we explain what is given, what needs to be adjusted and what can be done with the setup.

The setup contains a file that can automatically generate the environments with the different algorithms (it is called *generate.py*). After a generation each environment will than have a separate folder for itself.

The generate.py file already contains three algorithms (Linear BBA, MPC and Fugu), the 6 Pantheon environments and the environments we created during our testing.

To add a new environment, one simply has to follow the example in the *generate.py* file, define a name and the Mahimahi parameters.

It is important to say, that he configuration of the algorithm itself has to be done manually in the server folder after the generation, in a file called, *make_config.sh*.

There are already trace files, but we encourage to add more for more variety in testing.

Puffer offers more algorithms which can be tested, they can be found in their git repository [2]. It is possible to add a new algorithm, there is an instruction on how to do that.

The streaming data has to be downloaded, there are around 7 GB of video data. They can be downloaded with the download_media.sh file and need to be placed in the same directory as the original docker-compose file.

An other important point is, that in some files absolute paths are used, they have to be adjusted to ones own paths. This is the case for the graphs and the files: *generate.py* and *docker-compose.yaml*.

If none of the steps above have been disregarded, then we are almost ready to run an experiment. The last thing that needs to be done is to choose a channel to watch and for how long the experiment should run. This can be done in the client folder in the *main.py* file.

After that the experiment should run. The results can be plotted in the graphs folder.

Chapter 5

Evaluation

In this chapter evaluate the algorithms performance in the environments and try to better understand what makes an environment challenging. We further investigate the insights that we gathered to verify them.

5.1 General setup

5.1.1 The Algorithms

For the algorithms we are choosing Linear BBA, MPC and Fugu. These three algorithms have a quite different approach.

1. Linear BBA: Linear BBA is a buffer-based control approach, it makes its decisions only based on the buffer occupancy. Which means if the buffer has low occupancy the bitrate is low and if the occupancy increases the chosen bitrate will increase.
   For our experiment we use the provided algorithm from Puffer with the ABR configuration "upper reservoir" set to 0.9.

2. MPC: MPC stands for model predictive control and has its origin in control theory. This algorithm combines the rate-based and the buffer-based approach. They provide a flexible QoE model, where every factor (Total bitrate, rebuffering, change in bitrate and start up delay) has a weight and this weight can be changed according to one's preferences.
   If we had perfect knowledge about the future, we would only have to solve an optimization problem for the QoE, but in reality such perfect knowledge is not obtainable. But even if we don't have this perfect knowledge, it is possible to reasonably predict for a subsequent period. This algorithm works in two steps, the first one is the throughput prediction. MPC looks at the download speed (throughput) of the previous chunks and predicts the throughput for the next N chunks. After that it solves a maximization problem with the throughput prediction, buffer occupancy and previous downloaded bitrate.
   Again we use the model provided by Puffer, where the objective function is to maximize the SSIM.

3. Fugu: Fugu is similar to MPC, but it uses machine learning for its transmission time predictor (TTP) instead of the normal throughput predictor.
   For our experiments we use the algorithm with the already pretrained TTP model provided by puffer. Puffer suggest the TTP model trained on 2/2/2019 from their website [1].

5.1.2 Network traces

We want to build on already existing real world traces, so we can compare them to better understand why one is easy and the other more challenging. Pantheon has already created mahimahi emulation environments for their network paths. Table 5.1 describes the six environments with the following parameters:

1. Constant delay: Constant delay is the time a packet needs to travel from one container to the other. This is the minimum time it takes when not considering queues or similar.

2. AVG throughput: The average throughput determines how much data can be sent from one container to another in a time frame. More specifically, there are trace files which indicate how much data you can send at each timestamp, which ultimately results in the throughput. This can be defined for uplink and downlink direction, uplink is for the packets leaving the container and downlink for the incoming packets. The trace file format used by Mahimahi [6] is the following. Each line in the trace is a number and represents a packet delivery opportunity, where a total of 1500 bytes can be sent. This can be one 1500 bytes packet or multiple that sum up to 1500 bytes. To increase the throughput, multiple lines with the same number can be added.

3. Queue size: For all our environments we use a droptail queues, if the queue is full they drop packets at the tail. The queue size just indicates how many packets can be in the queue at the same time.

4. Stochastic loss rate: Packets are lost at the given rate.

Environment name	Constant delay (ms)	AVG throughput (mbps)	Queue size	Stochastic loss rate
Brazil-Colombia	130	3.04	426	0
California-Mexico	45	114.68	450	0
India-India	27	100.42	173	0
Korea-China	51	77.72	94	0.0006
Mexico-California	88	2.64	130	0
Nepal-India	28	0.57	14	0.0477

Table 5.1: Pantheon environments

The network traces are one way paths, we do not know the network behaviour in the other direction. In Mahimahi, there can be defined an uplink queue and a downlink queue and Pantheon specified the queue only in the uplink direction, the direction the data is sent. As we have implemented the emulation environment in the client container and the main traffic goes from server to client, we had to change it to a downlink queue. Because we want the traffic coming from the server to pass through the queue.

5.2 Experiment 1: Algorithm's performances

In this experiment we want to see how each algorithm performs in each environment. We measure the SSIM for each chunk and the rebuffer time. We display the average SSIM and rebuffer time from each test to compare the overall result. The goal is to understand how challenging the environments are and if we can already see differences in performance. For all ABRs, we stream the same video for 45 seconds (channel 2 from the puffer-supplied videos).

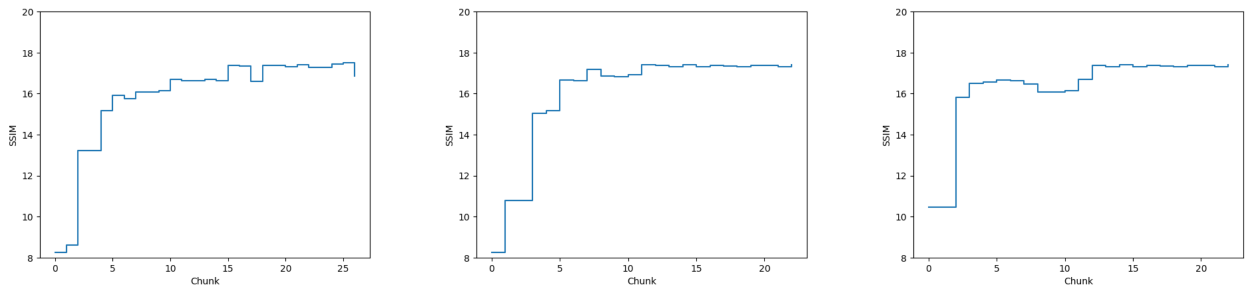
5.2.1 Results

We can see that in the environments Brazil-Colombia (figure 5.1), California-Mexico (figure 5.2), India-India (figure 5.3), Korea-China (figure 5.4) and Mexico-California (figure 5.5) all the ABR algorithms perform well. In the Nepal-India environment (figure 5.6) all the ABR algorithms have difficulties.

Fugu seems to perform a bit better than the other two algorithms exept in the Nepal-India environment where MPC has a higher average SSIM but also 0.5 seconds more rebuffer time.

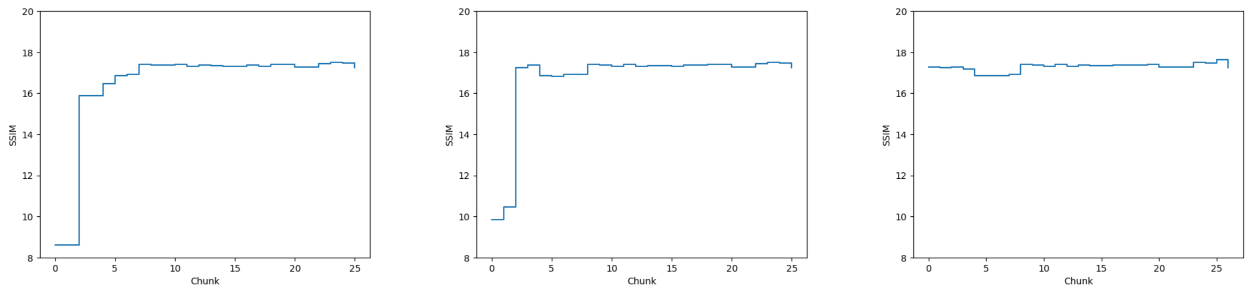
In table 5.2 are the algorithms performances in terms of average SSIM and rebuffer time.

There are five easy environments and one challenging, possibly already too challenging environment, which is the Nepal-India.



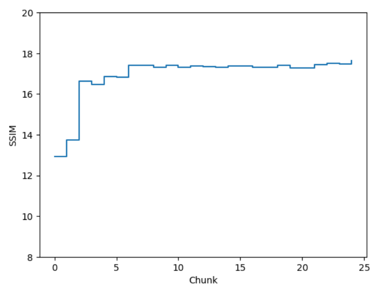
(a) BBA, AVG SSIM: 16.06 (b) MPC, AVG SSIM: 16.14 (c) Fugu, AVG SSIM: 16.52

Figure 5.1: SSIM in the Brazil-Colombia environment

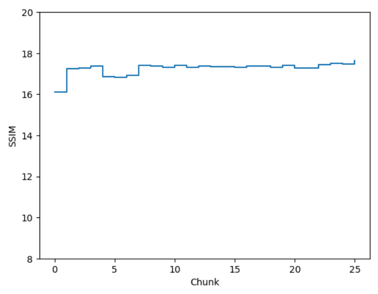


(a) BBA, AVG SSIM: 16.83 (b) MPC, AVG SSIM: 17.0 (c) Fugu, AVG SSIM: 17.28

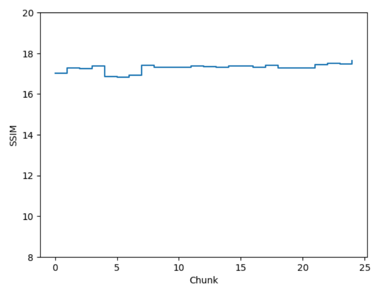
Figure 5.2: SSIM in the California-Mexico environment



(a) BBA, AVG SSIM: 17.07

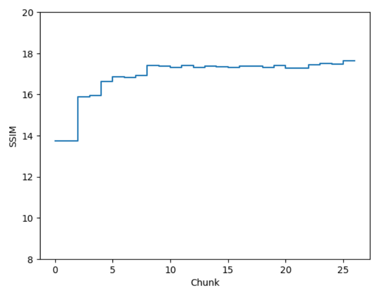


(b) MPC, AVG SSIM: 17.26

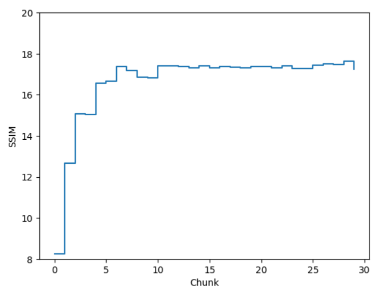


(c) Fugu, AVG SSIM: 17.29

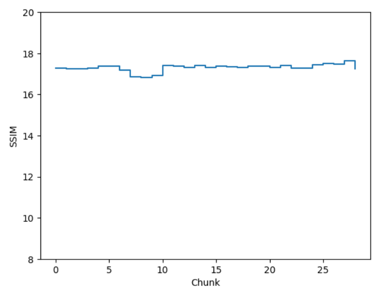
Figure 5.3: SSIM in the India-India environment



(a) BBA, AVG SSIM: 17.01

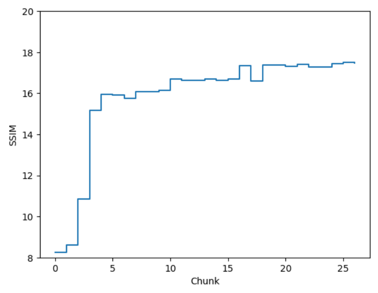


(b) MPC, AVG SSIM: 16.71

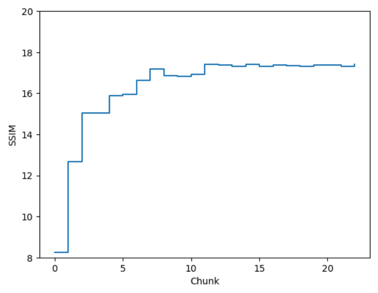


(c) Fugu, AVG SSIM: 17.31

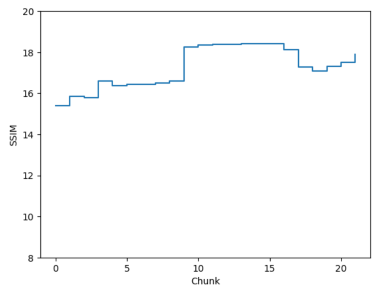
Figure 5.4: SSIM in the Korea-China environment



(a) BBA, AVG SSIM: 16.09

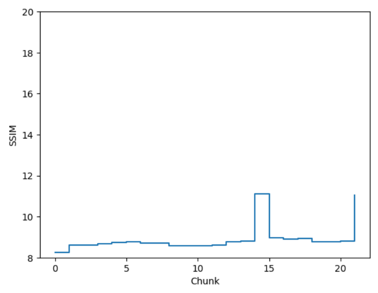


(b) MPC, AVG SSIM: 16.33

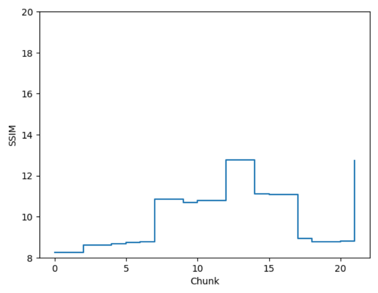


(c) Fugu, AVG SSIM: 17.26

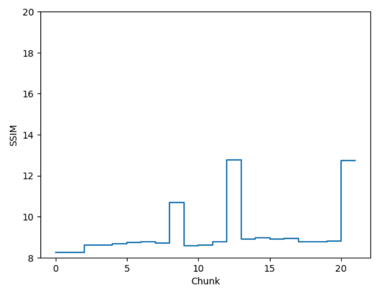
Figure 5.5: SSIM in the Mexico-California environment



(a) BBA, AVG SSIM: 8.93



(b) MPC, AVG SSIM: 10.08



(c) Fugu, AVG SSIM: 9.81

Figure 5.6: SSIM in the Nepal-India environment

Measurement	Br-Co	Ca-Me	Ind-Ind	Ko-Ch	Me-Ca	Ne-Ind
SSIM Linear BBA	16.06	16.83	17.07	17.01	16.09	8.93
SSIM MPC	16.14	17.0	17.26	16.71	16.33	10.08
SSIM Fugu	16.52	17.28	17.29	17.31	17.26	9.18
Rebuffer (s) Linear BBA	0.5	0	0	0.5	0	3.5
Rebuffer (s) MPC	0.5	0	0	0	0.5	2
Rebuffer (s) Fugu	0	0	0	0	0	1.5

Table 5.2: Average SSIM and rebuffer time of the algorithms in each environment

5.3 Experiment 2: Comparing the environments

The algorithms perform good in the environments Brazil-Colombia, California-Mexico, India-India, Korea-China and Mexico-California, but Nepal-India is challenging. Now we investigate the environments in detail to better understand, what parameters had more or less influence on the outcome of the experiments. First, we compare the easy environments with each other and after that we compare them with the challenging Nepal-India environment.

For that, we analyse the trace files and the mahimahi parameters used in the emulation. For the trace files, we look at the rate, at which packets can be sent and how this varies. An other factor we look at is the bandwidth over time and how this varies.

5.3.1 Comparison of the easy environments

The environments California-Mexico, India-India and Korea-China have much more bandwidth over time compared to the environments Brazil-Colombia and Mexico-California. But in all environments the bandwidth over time does not change.

There is only one environment which has a stochastic loss rate and it is very small.

When it comes to the Mahimahi parameter queue size, there is a wide range of different values, with the smallest queue size being 94 and the biggest 450.

The constant propagation delay in the environments is in a range between 27 ms and 130 ms.

5.3.2 Comparison between Nepal-India and the easy environments

Nepal-India has less bandwidth over time than all of the easy environments.

The overall throughput is almost five times lower than in the Mexico-California environment, which has the lowest amount of the easy environments.

The packet queue size is only 14, this means that packets are dropped much faster. There is a stochastic loss rate which increases the lost packets.

But the delay is only 28 ms, almost the same as the shortest from above.

5.3.3 Environment conclusions

We first notice that there is a threshold for the need of possible throughput, because even with massive difference in throughput, there seems to be relative little to none improvement in the average SSIM.

Although the Nepal-India environment has a short delay, it doesn't compensate the deficit in performance. There are two possibilities, either the throughput is simply not enough and it is not possible to perform better, even if there would be no delay or it could be that, since there are not as many opportunities to send packets, they pile up in the server until they are sent. The queue in the network can only accommodate for 14 packets, which is significantly less than in the other environments. That could lead to packets being dropped.

The bandwidth does not vary over time in any environment, that's why we can not say how it would influence the performance.

5.4 Experiment 3: Increasing the queue size

Now that we have seen the performances in the different environments and had a closer look at the environments, we first want to verify our claim from section 5.3.3 about the queue size. If the increased queue size has no impact, then the throughput was simply not enough to perform any better.

Since the Nepal-India environment was difficult, we build on this and increase the queue size.

The Nepal-India environment has a packet queue size of 14. We increase it in four tests to 34, 54, 74 and 94. The rest of the Nepal-India environment remains the same.

I will refer to the Nepal-India environment with its queue size as follows, Ne-size. For example the normal Nepal-India has queue size 14, I will address it as Ne-14.

5.4.1 Results

When we look at the results from the ABR Linear BBA in figure 5.7 and table A.1, first the increased queue size has a positive impact on the average SSIM, but with the largest queue size the average SSIM drops again and also results in a rebuffer time of 6.5 seconds.

We interpret this as follows, if the queues in the network are smaller, packets are dropped faster, but we also receive feedback faster and can therefore adapt quicker. With bigger queues in the network, it takes longer until we receive feedback from the network.

Linear BBA seems to struggle with this, since it's only choosing the bitrate based on the buffer occupancy. In figure 5.7e, we see that around chunk 6 the SSIM starts to go up and at around 10 back down. And if we look at the time the rebuffering occurs in figure 5.10, it is directly at the start of streaming and after the period where the SSIM is high. Which means that it takes some time in the beginning until the packets arrive, but when they start to arrive the buffer occupancy goes up, which at some point leads to an increase in SSIM. Since the buffer occupancy is up, higher bitrates are requested, but higher bitrates drain the buffer faster than the algorithm can adapt, which leads to the many rebufferings and the SSIM dropping again.

When we look at the results from Fugu in figure 5.9 and table A.1, we see that the increased queue size directly has a positive impact in all cases. The bigger packet queue has no impact in comparison with Linear BBA. But in the Ne-54 environment the rebuffer time over 2 seconds, which is significantly higher than in the other environments.

Fugu includes more factors and uses machine learning for its transmission time prediction, it seems that contributes to a better understanding and with that better performance in the network.

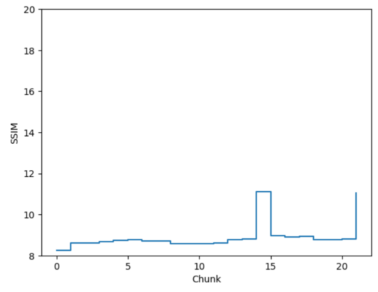
When we look at the results from the ABR MPC in figure 5.8 and table A.1, we see that with an increased queue size the average SSIM increases, but in the Ne-54 environment it even drops below the average SSIM of the original environment (Ne-14) and has a rebuffer time of over 3 seconds. If we increase the queue size further it goes up and then drops under the original SSIM again with a rebuffer time of 2 seconds.

Fugu's and MPC's rebuffer time in Ne-54 is rather high, but Fugu can still perform well in terms of average SSIM. There must be something in this environment with queue size 54 which makes it hard for them to perform as in the Ne34 and Ne-74. As both are quite similar the only

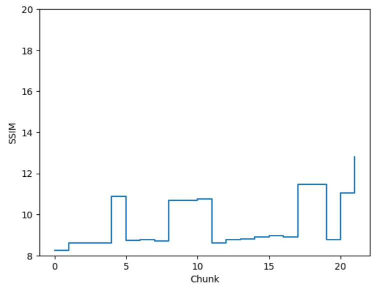
Measurement	Ne-14	Ne-34	Ne-54	Ne-74	Ne-94
SSIM Linear BBA	8.93	9.71	10.64	11.54	9.22
SSIM MPC	10.08	11.99	9.31	11.55	9.48
SSIM Fugu	9.18	12.0	11.98	12.06	11.99
Rebuffer (s) Linear BBA	3.5	2	1.5	0.87	6.5
Rebuffer (s) MPC	2	0.18	3.17	0.77	2
Rebuffer (s) Fugu	1.5	0	2.18	0	0

Table 5.3: Average SSIM and rebuffer time of the algorithms in each environment

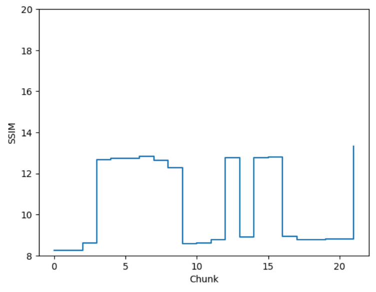
difference is that MPC has a simple throughput predictor and Fugu a transmission time predictor which uses machine learning. Which shows the advantage of Fugu's TTP.



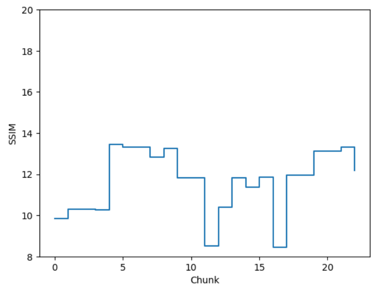
(a) Queue size: 14



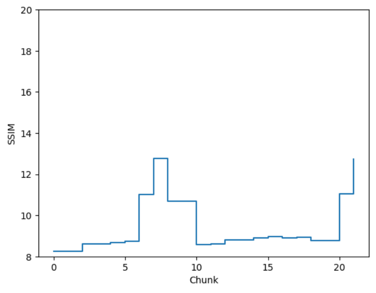
(b) Queue size: 34



(c) Queue size: 54

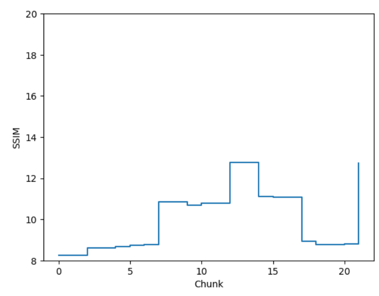


(d) Queue size: 74

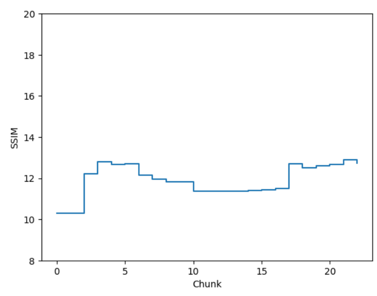


(e) Queue size: 94

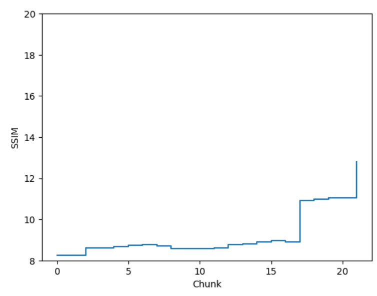
Figure 5.7: SSIM from Linear BBA in the Nepal-India environment with different queue sizes



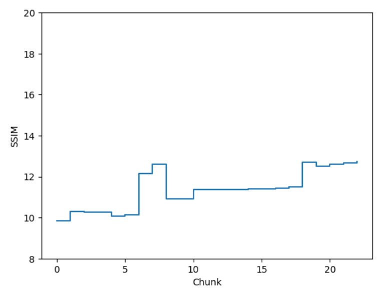
(a) Queue size: 14



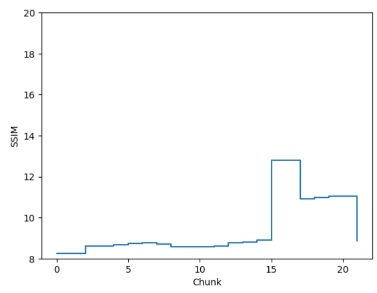
(b) Queue size: 34



(c) Queue size: 54

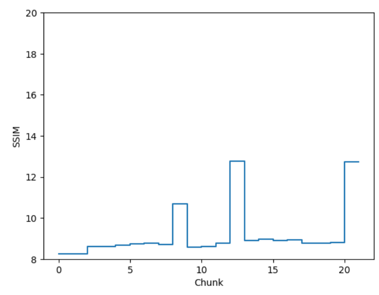


(d) Queue size: 74

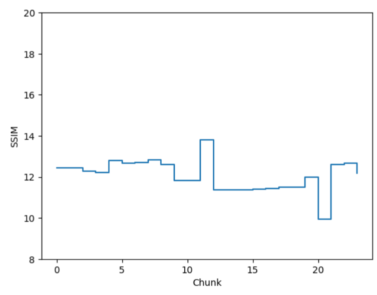


(e) Queue size: 94

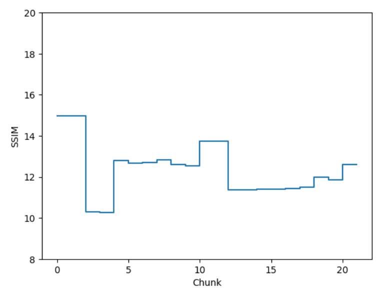
Figure 5.8: SSIM from MPC in the Nepal-India environment with different queue sizes



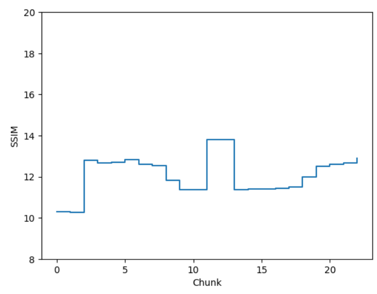
(a) Queue size: 14



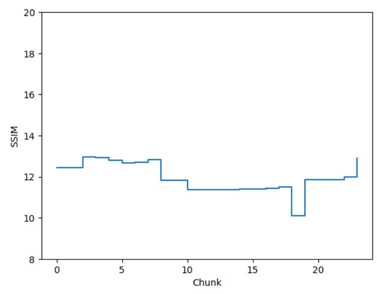
(b) Queue size: 34



(c) Queue size: 54



(d) Queue size: 74



(e) Queue size: 94

Figure 5.9: SSIM from Fugu in the Nepal-India environment with different queue sizes.

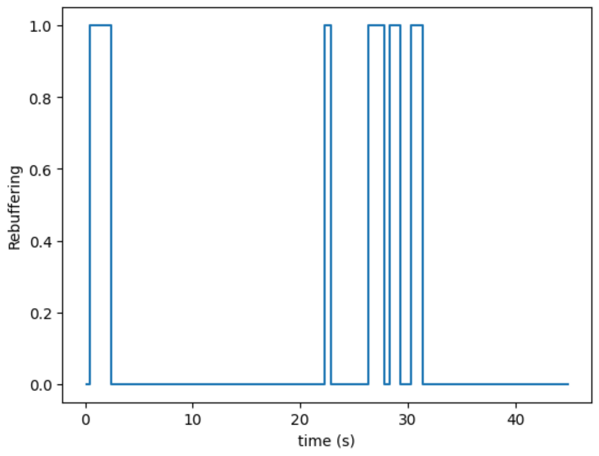


Figure 5.10: Rebuffering's of Linear BBA in the Nepal-94 environment.

5.5 Experiment 4: Bandwidth-delay product

We want to verify the claim made in section 5.4.1 about the behaviour of the increasing queue size on Linear BBA and Fugu and look at the bandwidth-delay product (BDP) and SSIM. For that we create five additional environments as in Experiment 3 (5.4). Now we have additional packet queue sizes of 24, 44,664, 84 and 104. We test Linear BBA and Fugu on Nepal-India environment with all the queue sizes from 14 to 104.

In this experiment we plot the bandwidth-delay product and we calculate it like this:

$$\text{BDP} = \text{Bandwidth} * (\text{Constant delay} + \text{Queue size} * 1500b * 8) \tag{5.1}$$

The Delay consists of the constant delay and number of bits that can be in a queue. We calculate it like this: packet queue size $*$ size of a packet in bits.
The BDP tells us the capacity of our environment in bits. Normally this is calculated as round-trip delay time, but since our environment only points in one direction, it just stands for this direction.

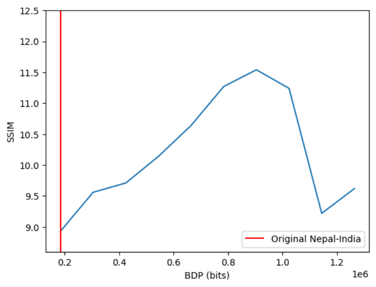
5.5.1 Results

In figure 5.11a, we observe that with an increased BDP the average SSIM increases, but when the possible bits capacity hits a certain size, the average SSIM starts to decrease.

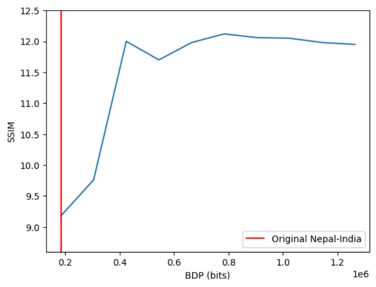
In figure 5.11b we can see that for Fugu the average SSIM doesn't decrease with a bigger BDP.

This supports our claim from section 5.4.1, that Fugu is able to understand the network better than a simple ABR algorithm like Linear BBA. If the queue size is small and packet arrive or drop faster, which results in in a faster reaction time for the algorithm. With a larger queue size it takes more time until information arrives at the algorithm which results in a slower reaction time.

We see that with a growing queue size the algorithms have a slower reaction time, which makes the environment harder.



(a) Linear BBA



(b) Fugu

Figure 5.11: The result of the two ABR algorithms when plotting SSIM against BDP. Here a bigger BDP implies bigger queue size.

Chapter 6

Conclusion

In this report we investigated different environments and ABR algorithms to better understand what parts of communication networks make video streaming challenging.

We first had to create a docker framework, which allowed us to easily implement all sorts of streaming environments. This docker framework is convenient for future testing, it can be used without having to go through the trouble of setting everything up.

In the first experiment we looked at the performance of the algorithms in the emulation environments of Pantheon with respect to average SSIM and rebuffer time. We found that the Nepal-India environment is difficult, whereby the other five environments are easy.

With that in mind, we compared the environments with each other to understand what makes the Nepal-India environment more challenging than the others. We found that the Nepal-India environment has lower throughput. We also found that the amount of packets that can be in the network queue is significantly less than in the easy environments. Which could lead to more packets being dropped. The stochastic loss rate in Nepal-India is higher than in any other environment. This again leads to more packets which do not arrive at the client.

What we found in the last experiment is that an increased packet queue size first has a positive impact on the average SSIM, but the larger the queue becomes the slower is the reaction time for the algorithms. The performance of Linear BBA dropped at a certain queue size while Fugu's performance did not decrease. This indicates that larger queue size leads to slower reaction times for the algorithms, which makes an environment more challenging.

As next steps it would be interesting to test the influence of the loss rate. With a lower loss rate more packets arrive at the client, Linear BBA would probably profit from that and can keep a good performance for an even larger queue size.

We recommend also using more dynamic trace files, our trace files have a pretty static behaviour. With a more dynamic trace file it would require the algorithms to adapt more and to different situations.

If we better understand what parts of the network make video streaming challenging, we could create better testing environments for emulation and simulation, which would help create and fine tune better streaming algorithms.

As the network is always changing it may bring new challenges and requirements for ABR's, so the knowledge on how to approach the comparison between environments is a key element.

Bibliography

[1] Puffer's experiment results. `https://puffer.stanford.edu/results/`. Accessed: 2023-06-13.

[2] Puffer's git repository. `https://github.com/StanfordSNR/puffer/tree/master/src/abr`. Accessed: 2023-06-13.

[3] Bartulovic, M., Jiang, J., Balakrishnan, S., Sekar, V., and Sinopoli, B. Biases in data-driven networking, and what to do about them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (2017), pp. 192–198.

[4] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM* (2014), pp. 187–198.

[5] Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication* (2017), pp. 197–210.

[6] Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., and Balakrishnan, H. Mahimahi: Accurate record-and-replay for http. In *Usenix annual technical conference* (2015), pp. 417–429.

[7] Paxson, V., and Floyd, S. Why we don't know how to simulate the internet. In *Proceedings of the 29th conference on Winter simulation* (1997), pp. 1037–1044.

[8] Rampfl, S. Network simulation and its limitations. In *Proceeding zum seminar future internet (FI), Innovative Internet Technologien und Mobilkommunikation (IITM) und autonomous communication networks (ACN)* (2013), vol. 57, Citeseer.

[9] Sandvine. 2023 global internet phenomena report. `https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2023/reports/Sandvine%20GIPR%202023.pdf`. Accessed: 2023-06-11.

[10] Winstein, K., Sivaraman, A., Balakrishnan, H., et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI* (2013), vol. 1, pp. 2–3.

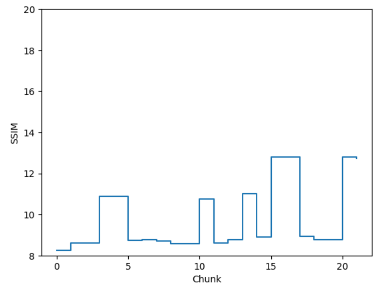
[11] Yan, F. Y., Ayers, H., Zhu, C., Fouladi, S., Hong, J., Zhang, K., Levis, P. A., and Winstein, K. Learning in situ: a randomized experiment in video streaming. In *NSDI* (2020), vol. 20, pp. 495–511.

[12] Yan, F. Y., Ma, J., Hill, G. D., Raghavan, D., Wahby, R. S., Levis, P., and Winstein, K. Pantheon: the training ground for internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 731–743.

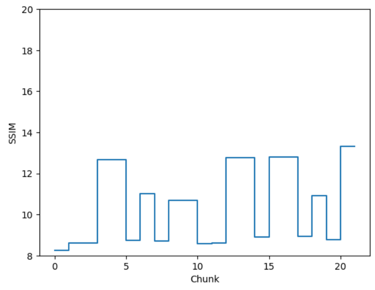
[13] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 325–338.

Appendix A

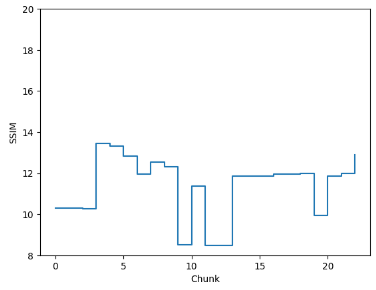
My Appendix



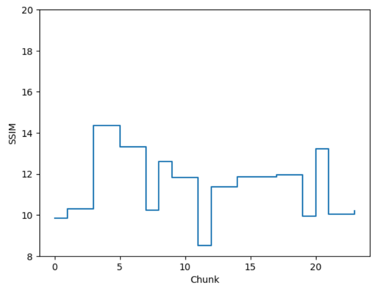
(a) Queue size: 24



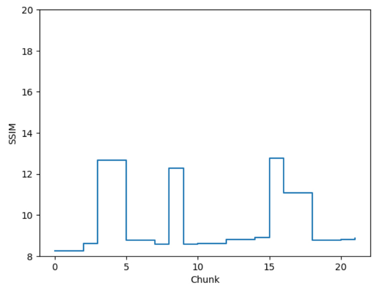
(b) Queue size: 44



(c) Queue size: 64



(d) Queue size: 84

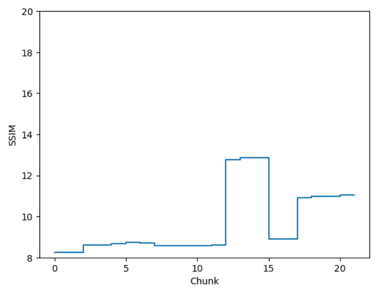


(e) Queue size: 104

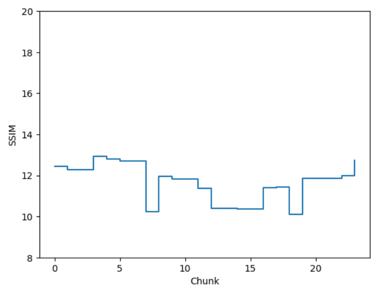
Figure A.1: SSIM from Linear BBA in the Nepal-India environment with different queue sizes

Measurement	Ne-24	Ne-44	Ne-64	Ne-84	Ne-104
SSIM Linear BBA	9.54	10.14	11.27	11.24	9.62
SSIM Fugu	9.76	11.70	12.12	12.05	11.95
Rebuffer (s) Linear BBA	2.5	2.5	0	0.5	1.6
Rebuffer (s) Fugu	1.8	0	0	0	0

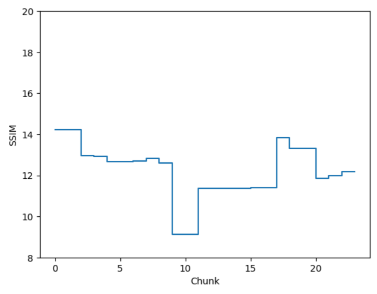
Table A.1: Average SSIM and rebuffer time of Linear BBA and Fugu in the additional environments from section 5.5.1



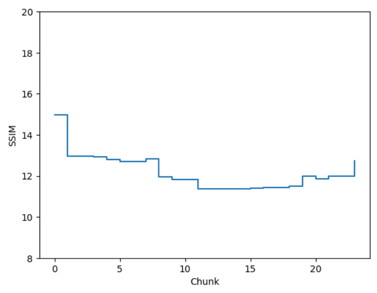
(a) Queue size: 24



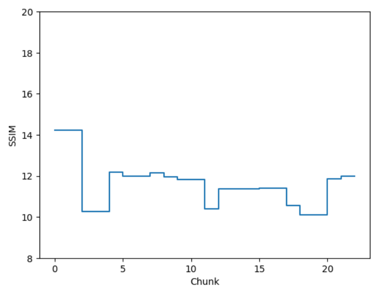
(b) Queue size: 44



(c) Queue size: 64



(d) Queue size: 84



(e) Queue size: 104

Figure A.2: SSIM from Fugu in the Nepal-India environment with different queue sizes.