ETH zürich

Networked Systems
ETH Zürich — seit 2015

Analyzing ML-Based Video Streaming Across Real-World Environments

Semester Thesis

Author: Benjamin Hoffman

Tutor: Alexander Dietmüller

Supervisor: Prof. Dr. Laurent Vanbever

September 2023 to January 2024

Abstract

Machine Learning (ML) algorithms in video streaming suffer from a reproducibility crisis. Proposed ML-based algorithms often fail to generalize to deployments outside of their training environment. Properly evaluating the performance of an algorithm prior to its deployment is also hard, as we do not fully understand what makes an environment challenging for a streaming algorithm. Further, the results an algorithm achieves in simulation or in a single real-world environment are often not representative of its performance across other real-world deployments. This hampers the design of new algorithms that truly offer improvements and raises the question: *How does an algorithm's performance differ across real-world environments?*

We address this challenge and present an infrastructure for the evaluation of ML-based video streaming algorithms in real-world environments. We compare the performance of different state-of-the-art ABR algorithms and collect streaming, transport and network data across environments in Europe, South America and Asia. Our results provide insight into how well an algorithm generalizes to real-world deployments outside of its training environment. With this, we hope to facilitate the development of new variants and meaningful improvements to existing schemes.

Contents

Chapter 1

Introduction

1.1 Motivation

Video streaming is currently the most prevalent internet application, accounting for more than 65% of the internet traffic in 2023 [13]. In the pursuit of optimizing a user's Quality-of-Experience (QoE), video streaming providers employ Adaptive Bitrate (ABR) Algorithms. These algorithms attempt to find a balance between the different factors impacting the QoE by dynamically adjusting the quality of a video stream based on the current network conditions. This is inherently a hard problem due to the complex nature of networks and conflicting QoE factors, such as maximizing video quality while minimizing rebuffering.

Previous research has focused on using buffer-occupancy-based or throughput-based ABR schemes in the pursuit of maximizing QoE [7, 17]. However, the complexity of conditions an ABR algorithm needs to perform under increases with the diversity of the device and networking landscape. Combined with the inherent complexity of network dynamics, this leads to a problem that requires rapid interpretation of patterns to make an ABR decision. With this premise, more recent research on ABR schemes proposes using a Machine Learning (ML) based approach [16, 18, 11, 12].

While these algorithms have shown promising results when tested within their original training environment, they often fail to generalize to new environments [4, 16]. The performance of learning-based algorithms is dependent on their training data or training environment. However, providing a sufficiently representative training environment or set of training data is challenging in the context of video streaming. This is due to the complex nature of networks and the heavy-tailed behavior they exhibit, which is hard to capture in a limited dataset[16]. For the same reason, an algorithm's performance in a single real-world environment is not representative of its performance across other environments. This restricts the ability to evaluate and compare different algorithms or variants.

The Puffer project, an ongoing study comparing different ABR schemes [16], approaches these issues by providing a platform for training, testing and data collection. This is done by allowing users to stream live television while collecting QoE data for the ABR schemes used. However, Puffer is only deployed to a single server which limits its ability to capture a representative set of network conditions. The platform's viewership is also restricted to North America. Further, the data suggests the presence of a survivorship bias. The study's proposed algorithm Fugu showed a performance improvement of roughly 50% in terms of time spent stalling when comparing February of 2023 to its initial performance in February of 2019. While the study initially attracted a diverse set of users at its launch, Fugu's performance improvement over time indicates that users experiencing higher QoE are more likely to continue using the platform, biasing the results. These limitations in turn negatively impact the performance of algorithms trained using Puffer when deployed outside of their training environment, leaving the pursuit of generalization still unresolved.

1.2 Tasks and Goals

To address the challenges of algorithm evaluation, we propose an infrastructure for the evaluation and comparison of ML-based ABR algorithm performance across real-world environments. We use this infrastructure to evaluate four state-of-the-art ABR algorithms in cloud deployments in Europe, South America and India. We compare their performance across the different environments and to data collected on the Puffer platform. We show that the performance ABR algorithms trained using Puffer data exhibit when tested in the Puffer environment is not necessarily representative of their performance in other environments. With this, we further highlight the necessity of diverse real-world data collection and training environments to allow for the generalization of learning-based schemes.

We achieve this by designing a platform that enables simultaneously evaluating multiple ABR algorithms across different target infrastructures. The platform allows for parallel deployment of multiple streaming clients and servers across different cloud infrastructures or virtual machines. Further, we enable the design of custom experiments and environments by including different streaming-specific and environment-specific design parameters. We include a data collection pipeline that collects QoE data and streaming measurements in the same format as the Puffer project for each streaming session. This allows us to evaluate the performance of different ABR algorithms across different environments and compare them to the Puffer data. Further, we include the collection of packet traces for both clients and servers.

1.3 Overview

The remainder of this thesis is structured as follows:

- In Section 2, we provide relevant background knowledge on video streaming, ABR algorithms and present relevant research in this field. Further, we introduce the Puffer project and netUnicorn, two infrastructures we use in our implementation.

- We present the design of our real-world testing infrastructure in Section 3 and expand on the implementation of the video streaming infrastructure, experiment platform and experiment design.

- In Section 4 we present the results of our evaluation. We analyze the real-world testing environments used, present the experiment setup and compare the algorithm performances.

- Section 5 summarizes our work and offers an overview of its potential expansions, both in terms of using the infrastructure, as well as using the results it provides.

Chapter 2

Background and Related Work

In this section, we provide the background knowledge necessary to follow the work presented throughout this thesis. We introduce the topic of video streaming, as well as ABR algorithms and their evaluation in real-world environments. Subsequently, related work relevant to this thesis is introduced. Most notably, we present the different ABR schemes BBA, Fugu and Gelato, as well as the Puffer project and the data collection platform netUnicorn [2].

2.1 Background

2.1.1 Video Streaming

Video streaming refers to the *streaming* of video from a media server to a client throughout a network such as the internet [8]. *Streaming* is the technique of playing out video while simultaneously receiving video from the server, avoiding the need to download the entire file before playing. In essence, HTTP-streaming uses *client buffering* and *prefetching* of video to mitigate changing end-to-end delays, as well as varying bandwidth availability[8]. *Client buffering* here refers to briefly delaying initial video playback to accumulate a reserve of buffered video in the client application buffer. *Prefetching* refers to the client downloading video into the TCP receive buffer at a higher rate than the rate at which the content is being consumed. A basic illustration of this process is shown in Figure 2.1.

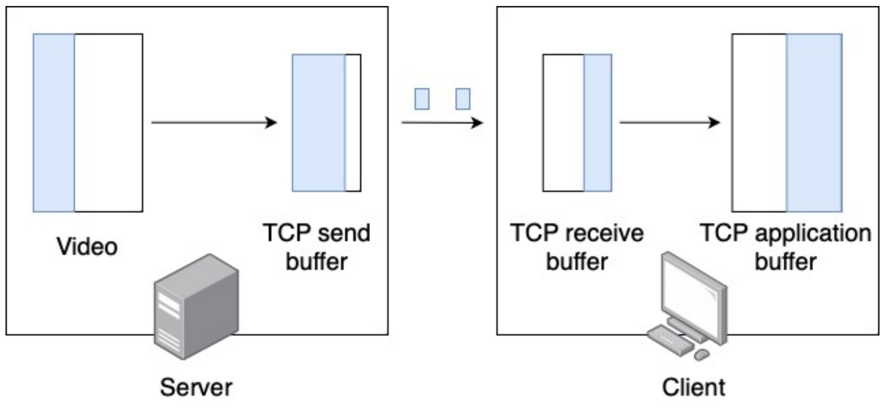


Figure 2.1: HTTP-streaming stored video using TCP [8].

Adaptive Bitrate Streaming

While several different approaches to video streaming exist, ABR streaming is the defacto default method used today. ABR streaming involves segmenting a video into equal-length chunks and encoding each chunk at multiple bitrates [14, 17]. This allows for the server to switch between different bitrates during streaming, depending on the current network conditions. Adapting to changing network conditions in turn allows for improving user-perceived QoE, a critical objective in video streaming applications that correlates with both user engagement and thus revenue for content providers [1]. To achieve this, ABR streaming employs ABR algorithms to interpret network conditions and choose an appropriate bitrate to maximize QoE.

2.1.2 Adaptive Bitrate Algorithms

Adaptive BitRate (ABR) algorithms dynamically adapt the bitrate of the video being streamed to the current network conditions. The bitrate can be adapted by choosing from the differently encoded chunks at every boundary, ie. every time a new chunk is sent to the client. The goal of ABR algorithms is simple: Optimize the user-perceived QoE. However, that is where the simplicity ends. The two main factors of QoE, minimal *time spent rebuffering* and maximal *average bitrate* formulate two contradictory objectives [14, 5]. If only the average bitrate is maximized, the bitrate chosen might surpass the available bandwidth, leading to a depletion of the client buffer and in turn stalling. Yet if only rebuffering is minimized, the bitrate chosen might be too low, leading to suboptimal video quality. Incorporating other QoE factors, such as minimizing bitrate oscillation (variations of video quality), promoting rapid reaction to network and user events or reducing startup latency, adds even further complexity.

The Landscape of ABR Schemes

Previous research on how to tackle ABR streaming has produced various different approaches to ABR schemes. To classify the different ABR algorithms, traditionally the required input is used as the defining feature [18]. However, more recent approaches have evolved to also include ML and control-based techniques that combine different input sources [3, 18, 4, 16, 11]. This allows for the ABR landscape to be roughly divided into *buffer-based*, *throughput-based*, *hybrid-control-based*, as well as *ml-based* algorithms [18, 10].

ML-Based ABR Algorithms

The complexity of networks, their dynamics and the diverse landscape of streaming devices render a problem that requires interpreting a large amount of data rapidly. With this premise, ML-based algorithms provide a seemingly natural solution. Instead of basing their ABR decisions on heuristic observations or fixed configurations tuned to a specific network context, ML-based algorithms can learn from past observations, predict future events and adapt their decision-making process to changing network conditions [3, 18, 1]. Ideally, they also adapt to conditions that were not present in training, that is they should generalize well and robustly provide good performance across the vastness of the internet.

Challenges of Learned Algorithms

This requirement of generalization leads to an enduring challenge for learning-based algorithms. The performance of the learned algorithms is dependent on the data used in training or, especially in the case of Reinforcement Learning (RL) schemes, the environments they were trained in [16]. Training in *simulated* or *emulated* network environments is often more feasible to perform at scale (*in situ* learning is all but impossible to perform at scale for RL-based schemes) and offers a controlled environment. However, due to the complexity of networks and their dynamics in the real world, it is hard to provide a representative set of network data. This can lead to the learned ABR algorithms overfitting to their training environments and lacking true generalization [16].

2.1.3 Real-World Evaluation

The challenge of reproducibility also impacts the evaluation of ML-based ABR algorithms. While many proposed algorithms perform well in their training environment or individual real-world environments, they often do not deliver when deployed outside of them [4]. The lack of transparency of what truly makes a network environment difficult for a streaming algorithm makes it challenging to accurately evaluate their performance before deployment. It prevents us from properly reasoning on whether an environment could benefit from an ML-based algorithm. Further, this also makes comparing different algorithms or variants rather challenging, as they often cannot be evaluated in the same environment due to incompatibilities such as conflicting input sources or different feature sets. These challenges call for the development of an infrastructure that allows for analysing different ABR algorithms across multiple real-world environments, allowing for a better evaluation of their generalization.

2.2 Related Work

This thesis builds on prior work and research in the fields of ABR schemes, video streaming infrastructure, as well as data collection infrastructure in network environments. More precisely, four different ABR algorithms are tested and their performance is compared. The video streaming infrastructure used for testing these schemes is built on different elements provided by Puffer [16]. Lastly, the data collection platform *netUnicorn* [2] is used to build the infrastructure that allows for deploying and testing the different experiments across the worldwide internet.

2.2.1 Selected ABR Algorithms

BBA

Buffer-Based Algorithms (BBA) are a class of algorithms that perform their ABR decisions as a function of the current buffer occupancy. Originally introduced in [7], the proposed algorithms often act as a research benchline. The approach is simple enough: employing a mapping function that maps buffer occupancy to the video bitrate, while avoiding unnecessary rebuffering and maximizing the average bitrate. This creates a continuous rate map upon which the ABR decisions are made. On this premise, [7] further include a simple capacity estimation scheme to be used during startup, ie., when the buffer is filling from an empty state and the rate map would underestimate the appropriate bitrate.

Several BBA variants are tested in [7] on the streaming platform Netflix, yielding good results and performing better than the previously employed control-based algorithm. It should be noted, however, that this study was released in 2014 and is thus rather dated. Nevertheless, more recent experiments conducted within Puffer show BBA performing “surprisingly well” [16], on par with more complex algorithms such as Pensieve, an RL-based algorithm [11].

Fugu

Fugu, as proposed in [16], is a composite control algorithm for bitrate selection. It is based on a classical MPC controller that is informed via a Transmission Time Predictor (TTP), a trained Deep Neural Network (DNN) that predicts transmission times. The MPC controller maximizes the expected cumulative QoE over a fixable horizon by querying the TTP and outputting a plan of chunks for transmission. After receiving new input, the controller replans based on the new TTP query. Fugu optimizes an objective function based on quantifying QoE as a linear combination of video quality (measured using SSIM [6]), video quality variation and stall time. Fugu is designed to be feasibly trained *in situ* and is introduced in combination with Puffer as a real-world training and deployment environment. The proposed symbiosis is illustrated in Figure 2.2.

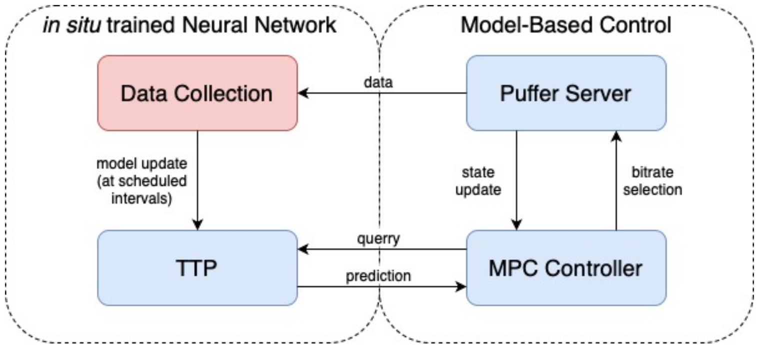


Figure 2.2: The data collected in Puffer is used to (re-)train the Neural Network *in situ.*

The results achieved in [16] show a promising approach, with Fugu outperforming other ABR algorithms such as MPC-HM and RobustMPC-HM [17], BBA [7] and Pensieve [11] during testing in Puffer. Much of this performance improvement is explained to be due to the training of Fugu *in situ* as opposed to being trained in simulation or emulation environments as is the case for Pensieve. Initially, Fugu was to be retrained daily. This approach was however abandoned, due to a lack of substantial performance improvement over non-retrained variants. A static variant of Fugu, Fugu_{Feb} trained in February of 2019, is used throughout this thesis.

Gelato

Proposed in [12] as an improvement over Pensieve [11], Gelato is a Deep Reinforcement Learning (DRL) based ABR algorithm. Gelato uses a similar rewards scheme to Fugu (+SSIM, −stalls, −ΔSSIM), as well as transmit times instead of throughput. Rich application-level features are used, omitting low-level TCP data. Gelato is presented together with a sampling and training scheme named *Priotitized Trace Selection* (PTS), a "generalizable solution for training high-performance DRL controller in simulators" [12]. PTS is proposed as a solution to the sample inefficiency of DRL and the skewness of real-world data used for simulation. To this end, PTS uses critical feature detection in trace behavior, followed by the classification of traces into clusters with shared characteristics. Lastly, the clusters that induce low performance and high potential learning are dynamically prioritized by continuous agent monitoring during the training process.

In [12], PTS is claimed to (i) offer higher performance than randomly sampling traces, (ii) produce more robust controllers due to wider exposure during training, (iii) reduce the need for retraining. Two variants of Gelato are trained, one using PTS (Maguro) and one using random sampling (Unagi) on data from the Puffer environment. The results achieved by Maguro when tested on the

Puffer platform show substantial improvements over other state-of-the-art algorithms running on the platform, including a 59% stall reduction and SSIM improvement of 0.14 dB when compared to Fugu. Unagi performs on par with Fugu in terms of stall ratio and improves SSIM by 0.09 dB. Still, the question of wider generalization remains, ie. if this performance can be replicated outside of the Puffer environment using which Gelato was essentially both trained and tested.

2.2.2 Puffer

Puffer, introduced in [16], is an ongoing project comparing different ABR algorithms. Puffer offers a platform that streams live TV to public users, with ABR algorithms being randomly assigned to each session. The per-session streaming data and QoE measurements is continuously collected and published. Its real-world deployment renders Puffer an interesting platform for comparing different algorithms and collecting training data. By gathering session data from sufficiently many participants and network paths, it is attempted to derive more general insights into algorithm performance. However, as explained in Section 1.1, the Puffer platform has limitations in terms of the variation in network conditions it captures. The Puffer source code and data are publicly available, with results being reported daily. This thesis uses both in its implementation and evaluation.

2.2.3 netUnicorn

NetUnicorn is a platform designed to simplify data collection for learning problems in network environments [2]. It facilitates deploying different data pipelines on target infrastructures across the internet for data collection experiments. Along with the ability to centrally orchestrate experiments and deployments, netUnicorn offers different connectors to virtual and physical infrastructures including SaltStack, Mininet, AWS Fargate, Azure Container Instances (ACI), Docker, Kubernetes, as well as a general SSH connector for Virtual Machines (VM). As for Puffer, the netUnicorn source code is openly available. Its modules, along with the diversity of its connectors, make netUnicorn a fitting tool to be used in the infrastructure we design in this thesis.

Chapter 3

Design

In this section, we introduce the design and implementation of our infrastructure. To allow for deploying, testing and evaluating different ABR algorithms in real-world environments, three elements are necessary: (i) the video streaming infrastructure, (ii) the experiment platform and (iii) experiment design. A high-level overview of the infrastructure design is shown in Figure 3.1.

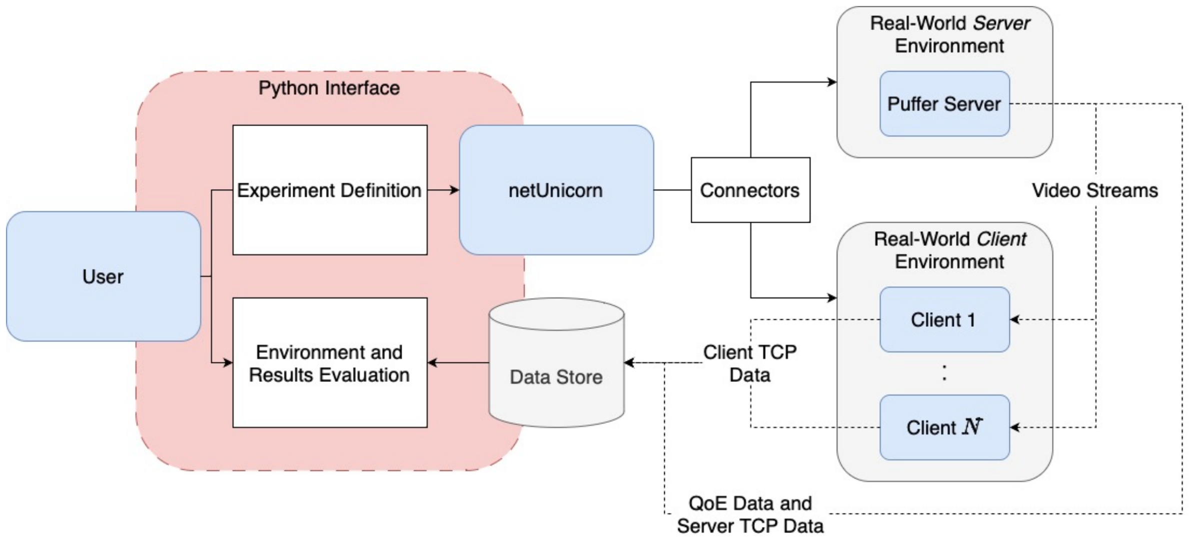


Figure 3.1: High-level overview of the infrastructure design

The *video streaming infrastructure* allows us to perform streaming experiments from a dedicated server to an arbitrary amount of clients, while also logging data on both the client and server side. The *experiment platform* allows for interpreting experiment design, deploying the video streaming infrastructure to the desired environments, regulating the experiment flow and performing data collection. Lastly, through *experiment design* we can define the streaming and environment configuration of an experiment: which algorithms to use, how many clients to stream to, the location of the target infrastructures, how long to stream for and so on.

In the following, we present the video streaming infrastructure in Section 3.1, describe how the experiment platform is implemented in Section 3.2 and explain how the experiments can be designed in Section 3.3.

3.1 Video Streaming Infrastructure

For our video streaming infrastructure, we extend the streaming server provided by the Puffer project and combine it with a Chromium-based web client to receive the video stream. Figure 3.2 provides an overview of the video streaming infrastructure used. In the following, we provide an overview of the different services that form the overall architecture and discuss the server and client implementation.

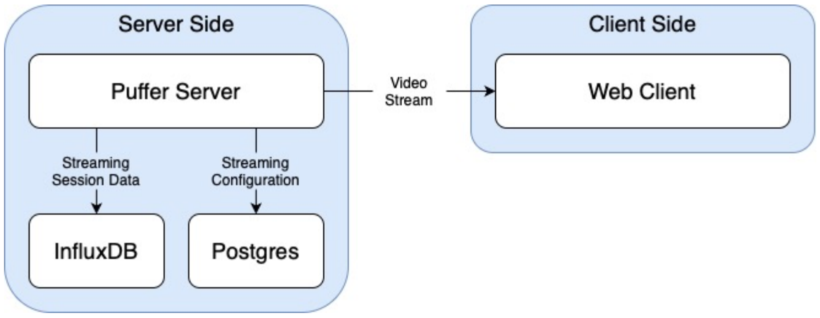


Figure 3.2: Overview of the video streaming infrastructure.

3.1.1 Containerized Services

As the goal of this project is to evaluate ABR algorithms in a diverse set of real-world environments, the server and client services should be self-contained and readily deployable. To this end, we containerize the services using the Docker framework. This enables the experiment platform to simply pull and deploy the individual containers to the desired environment.

Ideally, one container that handles all server-side services and one container for client-side services would be used. While we realize this for the client side, the implementation on the server side raises some issues. Puffer not only acts as a streaming server but simultaneously logs QoE and experiment data. To this end, we additionally require InfluxDB, a time series database, and Postgres, a relational database management system (RDBMS). Due to the inherent nature of containerization, ie. running services in isolation, as well as several version conflicts of the software packages required, we split the server side up into three different containers, one running the Puffer server, one for InfluxDB and one for Postgres. This also allows us to better regulate the experiment flow through the experiment platform, as we can ensure that both InfluxDB and Postgres are running before the Puffer server is launched.

Lastly, to enable compatibility with the netUnicorn-based experiment platform, we adapt the original Puffer, InlfuxDB and Postgres containers to include netUnicorn-specific packages.

3.1.2 Server

On the server side, Puffer uses a web server that authenticates users and hosts the website, which acts as the user interface and includes a JavaScript video client. Simultaneously, different Puffer media servers, each running a different ABR algorithm, serve the video content via a WebSocket connection. A streaming client can connect to the service via HTTP by visiting the website and playing the selected video. The allocation of a media server — which in turn determines which ABR algorithm is used — to each session is random. As mentioned above, we require InfluxDB and Postgres alongside the web server to collect QoE and configuration data for later evaluation. We provide an overview of the data being collected in Section 3.2.3.

The Puffer setup accommodates for a range of possible adaptations, enabling the addition of both custom video sources and custom ABR algorithms. In this project, we use a video clip from the CBS TV network, provided and encoded by Puffer, to be streamed during experiments. In Table 3.1 we provide an overview of the encodings used. The audio chunks are encoded at bitrates 32, 64 and 128 Kbps, with each chunk lasting 4.8 s. Further, the state-of-the-art ABR algorithms we use in this project, introduced in Section 2.2.1, are also readily available in Puffer. We provide an overview of the employed algorithms in Table 3.2.

For our streaming server, we can configure: (i) the video source and encodings, (ii) the media servers and ABR algorithms available, (iii) the specific configurations for each ABR algorithm, (iv) the congestion control algorithm used and (v) the provider and location used for the deployment. What we cannot configure are: (i) the IP address, which is dynamically assigned by the target infrastructure and fetched by the experiment platform and (ii) the server's resources, which we can only adapt in the netUnicorn settings.

Resolution [pixels]	**CRFs**
1920x1080	22, 24
1280x720	20, 22, 24, 26
854x480	24, 26
640x360	24
426x240	26

Table 3.1: An overview of the video-chunk encodings used throughout all experiments. Each chunk has a length of 2.002 s.

Algorithm	**Variant**	**Code Name**
BBA	Original	linear_bba
Fugu	Fugu$_{\text{Feb}}$	puffer_ttp
Gelato	Gelato-Random	unagi
Gelato	Gelato-PTS	maguro

Table 3.2: An overview of algorithms used throughout all experiments. The code name indicates the name in the source code.

3.1.3 Client

We implement our streaming client using a Chrome browser and a Selenium server which we use to interact with the browser headlessly via a Python script. Once deployed, the client connects to the Puffer server via the Chrome browser and streams the dedicated video clip. This browser-based implementation allows for deploying multiple clients efficiently, without the need to drastically change the Puffer source code.

From a client side, we can configure: (i) which channel or video to stream (given it is available on the server), (ii) the watch time and (iii) which target infrastructure and location to deploy to. The IP of the server to which we want the client to connect to is fetched and passed automatically by the experiment platform. We cannot define which ABR algorithm is used for the streaming session. This decision is made and assigned at random by the server.

3.2 Experiment Platform

We require our experiment platform to allow for:

1. an efficient deployment of containerized applications to a multitude of different infrastructures, physical or virtual.
2. defining and regulating the experiment flow.
3. communication with, as well as between, the deployed experiments.
4. an interface to orchestrate experiments and integrate the infrastructure into other services.
5. collecting data from the experiments and storing it in a data store for further evaluation.

With these capabilities, we describe a platform that can act as a link between a user and remote environments. While we considered creating an entire experiment platform from scratch at the onset of this project, this approach was discarded in favor of using netUnicorn, an existing framework that provides a very viable foundation for the required system. In Figure 3.3 we provide an overview of our implemented experiment platform.

In the following, we provide an overview of how the experiment platform is implemented and how the experiment data is collected.

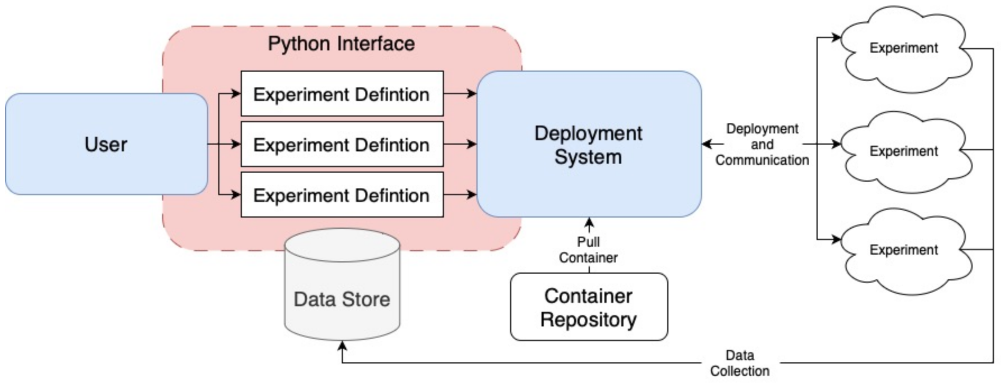


Figure 3.3: An overview of the experiment platform. The user provides the experiment definitions to the platform, these are then interpreted and passed on to be deployed to the remote environments.

3.2.1 Integration

On a system level, we integrated the experiment platform into the setup such that a user can feed the system an experiment definition describing what containers to deploy to which environments and what processes to perform within these containers. This includes all configuration information for the Puffer server and streaming client, as well as which environments to deploy to. Simultaneously, we provide the user with live information regarding the status of the deployed experiments. After an experiment has ended, we allow for automated storage of the collected data in the defined storage location.

On a technical level, we implement this interaction between the user and the experiment platform, as well as its integration into the entire setup, in Python. We provide a main script that interprets the defined experiment configurations and executes the experiment accordingly while providing information and logs to the user via the terminal. After an experiment has terminated, we have included an RSYNC task to transfer the data from the remote environment to a user-defined location.

3.2.2 NetUnicorn

Since we use different building blocks available in the netUnicorn python package in our implementation, we introduce netUnicorn in more detail in this section. Further, we present the different connectors for target infrastructures that are available.

How does netUnicorn work?

The general design of netUnicorn is based on three different parts: (i) a user-side *frontend*, (ii) several *core services* and (iii) an *executor*. The *frontend*, realized via providing a netUnicorn python package, allows a user to build a custom infrastructure and pipelines of experiments. The *core services*, running in a docker network on the user's local machine, prepare, deploy and maintain the experiments, as well as communicate with the *executors* via a gateway service. Lastly, the *executor*, running within the remote environment on the target infrastructure, receives and executes the experiments, as well as communicates any events and results back to the user. We show an overview of this general design in Figure 3.4.

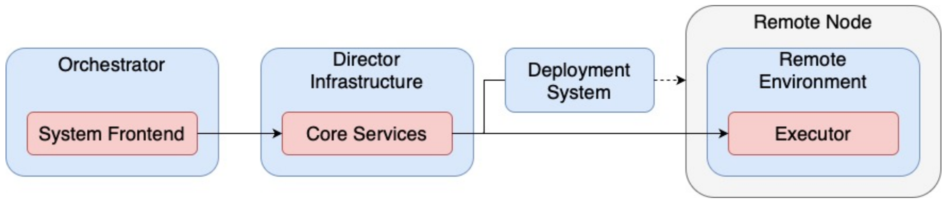


Figure 3.4: NetUnicorn's general design consists of a *frontend*, *core services* and an *executor*.

Building the Experiment Platform

We implement the the experiment platform to execute an experiment in three steps:

1. Fetch and interpret the experiment definitions.
2. Prepare, deploy, execute and maintain the experiments in the defined remote environments.
3. Fetch and store the collected results.

Step one:, We handle the fetching and interpreting of experiment definitions using Python. Our script parses the user-defined configurations, verifies them and then passes them to the next step.

Step two: This part of the pipeline is mainly based on the functions provided by netUnicorn. We use the experiment definitions to initialize a netUnicorn-pipeline-object formulated as a Directed Acyclic Graph (DAG) that defines a netUnicorn-experiment, as well as the environment in which it should be executed, ie. which remote target infrastructure it should be deployed to. An experiment in netUnicorn terms is essentially a sequence of different tasks that are executed in series. Once the experiment is started by the user, netUnicorn's deployment system connects to the target infrastructure via a dedicated connector, deploys the defined services (ie. Puffer server or streaming client) and executes all tasks within the experiment pipeline via the netUnicorn-executor. Once all clients have finished streaming their defined watch time, we use the netUnicorn-flag-task to communicate with the InfluxDB container. This allows us to execute a python script with which we post process and export all QoE and streaming data from the database.

Step three: In this step we wrap up the experiment. We execute our RSYNC transfer command that extracts the data from the remote infrastructure and stores it in the dedicated location. Subsequently, we terminate the experiment and the deployment system removes all containers from the target infrastructure.

NetUnicorn Connectors and Target Infrastructure

The *connectors* are a collection of Python packages, based on a common netUnicorn-interface, that allow for a deployment to different virtual and physical target infrastructures. These packages integrate target-infrastructure-specific Python System Development Kits (SDKs), such as the AWS or Azure SDK, into the netUnicorn-interface. This enables netUnicorn to connect and deploy experiments to these infrastructures, as is shown in Figure 3.5.

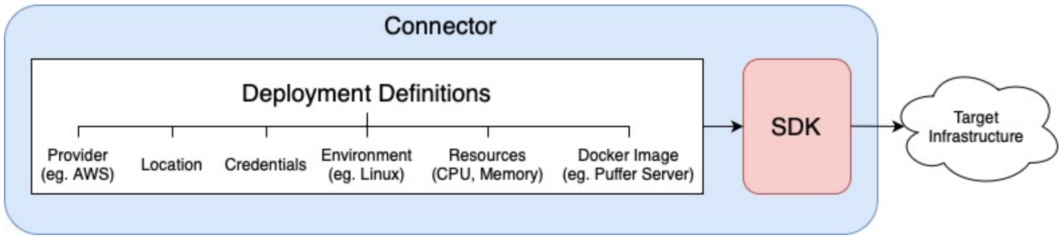


Figure 3.5: An overview of the deployment via netUnicorn.

Currently, netUnicorn natively offers connectors for SaltStack, Mininet, AWS Fargate, Azure Container Instances (ACI), Docker, Kubernetes, as well as a general SSH connector. As netUnicorn is an open-source platform, the library of connectors can be augmented with new connectors and custom implementations. Being the two largest cloud providers by market share [9], this project uses AWS and ACI as target infrastructures to run the experiments. We customize the native connectors available for AWS and ACI to fit our resources (additional cores and memory) and networking needs (exposed ports and public IP address assignment). Using the general SSH connector, as well as custom set-up Virtual Machines (VMs), was also considered as a viable approach that might allow for creating especially challenging experiment environments. However, the use of clouds was prioritized due to the flexibility in terms of locations and scalability in terms of resources they offer.

3.2.3 Data Collection

The underlying goal of the experiment platform, as well as the entire infrastructure, is to collect data from the experiments for evaluation. The setup involves two different data collection pipelines that focus on (i) collecting QoE and streaming measurements and (ii) collecting TCP data. While the data collected by (i) serves this project's main purpose of evaluating the performance of different ABR algorithms, the data collected by (ii) offers an additional data source for training ML-based

ABR algorithms, as we explain below.

1. **The collection of QoE and streaming measurements** in the first pipeline is relatively simple. As explained in Section 3.1.2, Puffer already allows for logging all QoE and measurement data from the video streaming sessions. We then post-process this data within the container and transfer it to a user-defined location via RSYNC. The data-collection scheme used in the Puffer project provides a rich selection of measurements, including SSIM and buffer occupancy, as well as measurements describing the environment including the round-trip time (RTT), minimal RRT and delivery rate. We provide an overview of the collected data in the appendix A.1.
2. **The collection of packet traces** is based on TCPDUMP, a command-line packet analyzer, that is used to capture and log the TCP packets on both the server and client side. We execute TCPDUMP within the containers via netUnicorn, which provides this task natively. This data is also included in the RSYNC transfer. This packet-level data provides more fine-grained information than the chunk-level data collected via Puffer. We perform this collection to enable for future evaluation if training ML-based algorithms on more fine-grained data could lead to improved performance or improved generalization. The evaluation itself is outside the scope of this thesis.

3.3 Experiment Design

In this section, we first introduce the definition of an "experiment" in the context of this thesis. Subsequently, we describe the different design parameters available for building custom experiments.

3.3.1 What is an Experiment?

As this project evaluates the performance of *streaming* algorithms in different *environments*, it follows naturally that we define an experiment via two factors: (i) the streaming definitions and (ii) the environment definitions. An "experiment" is the combination of both:

1. **The streaming definitions** encode the different streaming schemes and the streaming setup that should be tested. This includes the number of clients that should be deployed, the watch time, the ABR algorithms that should be used, as well as the configuration of these algorithms. This allows us to test different ABR algorithms in the same environments, as well as comparing different ABR configurations of the same algorithm.
2. **The environment definitions** describe the different environments in which the experiments should be performed, ie. what "cornerstones" should define the desired network conditions. Here we select the provider, region and location of the server and client target infrastructures. This allows for varying the multiple networking variables in an attempt to create a diverse set of network conditions and enables a more comprehensive evaluation of the performance of different ABR algorithms.

Design Parameters

On a higher level, we allow for different experiments to be designed by pairing a set of multiple streaming definitions with a set of multiple environment definitions and vice versa. This allows for testing the same streaming setup in different environments, as well as testing different streaming setups in the same environment. On a lower level, to create these sets, we define the individual streaming and environment definitions. To this end, we offer several design parameters in our implemented infrastructure. Table 3.3 and Table 3.4 provide an overview of the parameters available for streaming setup and environment design, respectively.

Design Parameter	**Description**	**Code Name**
Amount of Clients	The amount of clients that should be deployed during an experiment.	amount_of_clients
Watch Time	The amount of video in seconds that should be streamed by a single client.	watch_time
Media Servers	The total amount of media servers that should be used.	amount_of_servers
ABR Algorithm	Which ABR algorithm should be used during the experiment. One ABR algorithm can be defined per media server.	abr
ABR Configuration	Specifies the configuration of the ABR algorithm. The options available are dependent on the specific ABR algorithm.	abr_config
Congestion Control	Which Congestion Control algorithm should be used. This can also be defined per media server.	cc

Table 3.3: An overview of the parameters available for streaming setup design. The code name here refers to how the parameter is identified in the source code. Note that *media* server is not the same as *streaming* server. We only use one streaming server for each experiment.

Design Parameter	**Description**	**Code Name**
Server Provider	Which provider to use for the Puffer server deployment (eg. AWS)	server_provider
Server Region	Which region to use for the Puffer server deployment (eg. eu-west-2). The regions follow a specific naming convention defined by the selected provider.	region
Server Location	The specific deployment location within the region (eg. London) for the Puffer server. The available locations are determined by the provider and the region.	location
Client Provider	Which provider to use for the streaming client deployment (eg. AWS)	server_provider
Client Region	Which region to use for the streaming client deployment (eg. eu-west-2). The regions follow a specific naming convention defined by the selected provider.	region
Client Location	The specific deployment location within the region (eg. London) for the streaming client. The available locations are determined by the provider and the region.	location

Table 3.4: An overview of the parameters available for environment design. The code name here refers to how the parameter is identified in the source code.

Chapter 4

Evaluation

In this chapter, we present and discuss the results of the experiments conducted using the infrastructure defined in Chapter 3. In Section 4.1, we provide an overview of the different real-world environments across which the ABR algorithms were tested, as well as a comparison of these environments. In Section 4.2, we introduce the different experiments conducted and present the achieved results.

4.1 Comparing Real-World Environments

Three different environments are created to test the ABR algorithms within real-world settings. We label them *Europe-Europe*, *Europe-Abroad* and *Abroad-Abroad*, based on the location of server and client. Further, we also evaluate the *Puffer environment* in order to compare its network conditions with the selected environments. The data from the Puffer environment contains one week of measurements (30.12.2023 - 05.01.2024) and we filter it to only include the data from the sessions that use the ABR algorithms BBA, Fugu, Unagi and Maguro.

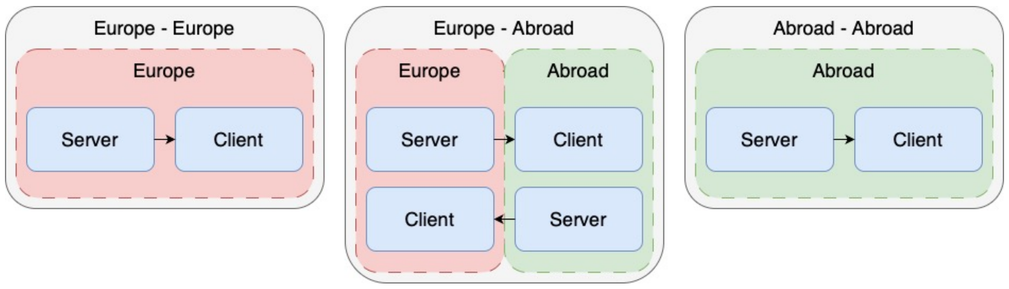


Figure 4.1: An overview of the different environments used for evaluation.

We choose to create the environments to range from Europe-Europe, Europe-Abroad and Abroad-Abroad to create a progression in “difficulty” in terms of network conditions. Based on our comparison, Europe-Europe provides the easiest network conditions, with both the lowest minimum Round Trip Time (RTT) and the highest average throughput. Europe-Abroad and Abroad-Abroad show progressively higher minimum RTTs and lower average throughput. When we compare the cloud environments to the Puffer environment, we can observe that both Europe-Abroad and Abroad-Abroad exhibit more challenging conditions in terms of average and minimum RTT, as well as average throughput. This indicates that only training on the conditions perceived in the Puffer environment might not provide the range of conditions needed for algorithm generalization. However, we can observe that the RTT difference in the Puffer environment is significantly higher than in the cloud environments. This is due to the clients deployed on the cloud platforms all experiencing similar network conditions, which is not the case for the users of the Puffer platform. In Table 4.1 we provide an overview of the environment comparison. Note that the experiment environments (Europe-Europe, Europe-Abroad, Abroad-Abroad) have similar amounts of watchtime (total time of streaming), while Puffer has significantly more, as it is a larger scale study.

In the following, we define the different environments, introduce the metrics for comparison and present the results of the environment evaluation.

	Watchtime [h]	**RTT** [µs]	**Min. RTT** [µs]	**RTT Diff.** [µs]	**Throughput** [B/s]
Europe-Europe	63.56	**12 954.95**	**10 333.98**	2541.89	**6 924 814.59**
Europe-Abroad	60.27	93 265.83	90 551.74	2655.11	1 953 702.69
Abroad-Abroad	63.71	106 003.92	103 807.18	**2013.68**	1 521 074.85
Puffer	15 660.63	82 189.05	51 193.57	29 335.06	5 959 003.16

Table 4.1: Europe-Europe shows the easiest average network conditions.

4.1.1 Environment Classification

We classify the environments based on the locations to which we deploy the streaming server and the streaming client. For each target infrastructure we can vary (i) the *provider* and (ii) the *location*[1]. For this evaluation, we select the following providers and locations:

The infrastructure providers: As explained in Section 3.2.2, we choose to use the two largest cloud providers, AWS and ACI. Both offer a wide range of infrastructure locations, which allows us to create different environments.

The infrastructure locations: To achieve a certain diversity within the network environments, we spread the infrastructure locations across the globe. In total, we use four different locations, ie. two per infrastructure provider.

This selection gives us the following four target infrastructures: (i) *AWS-London*, AWS infrastructure in London, United Kingdom (i) *AWS-Sao-Paulo*, AWS infrastructure in Sao Paulo, Brazil (iii) *ACI-Norway*, ACI infrastructure in Oslo, Norway, (iv) *ACI-India*, ACI infrastructure in Pune, India. Each is available for server or for client deployment, leading to the classification of the environments as shown in Table 4.2.

[1]Note that while the parameter *region* is also presented in Section 3.3.1, this only defines which locations are available and is thus not used here.

Europe - Europe		**Europe - Abroad**		**Abroad - Abroad**	
Server	Client	Server	Client	Server	Client
AWS-London	ACI-Norway	AWS-London	AWS-Sao-Paulo	AWS-Sao-Paulo	ACI-India
ACI-Norway	AWS-London	AWS-London	ACI-India	ACI-India	AWS-Sao-Paulo
		ACI-Norway	AWS-Sao-Paulo		
		ACI-Norway	ACI-India		
		AWS-Sao-Paulo	AWS-London		
		ACI-India	AWS-London		
		AWS-Sao-Paulo	ACI-Norway		
		ACI-India	ACI-Norway		

Table 4.2: An overview of the different environments. Note that no streaming experiments within the same target infrastructure were conducted.

4.1.2 Environment Evaluation

Evaluation Metrics

We compare the different environments based on the metrics below. First, we calculated each metric on a per-session basis, ie. for each video streaming session, and subsequently take a weighted average across all sessions within an environment. We set the weight for each session to the watchtime of the session. All equations are shown in Table 4.3.

Average RTT: RTT is defined as the time it takes for a chunk to travel from the sender (the streaming server) to the receiver (the streaming client) and back [8]. In that sense, the average RTT is a metric for network latency and includes propagation delays, queuing delays and processing delays. The average RTT per session is calculated as the average over the RTTs for all video chunks sent during a streaming session.

Average Minimum RTT: The minimum RTT is defined as the lowest RTT measured during a streaming session. It can be understood as a metric for indicating the best-case network latency per session. The minimum RTT per session is calculated as the minimum over the RTTs for all video chunks sent during a streaming session.

Average RTT Difference: The RTT difference is defined as the difference between the minimum RTT and the average RTT. This can be interpreted as a metric for the variation of the network latency across a session. The RTT difference per session is calculated as the difference between the minimum RTT and the average RTT for all video chunks sent during a streaming session.

Average Throughput: Throughput is defined as the rate at which a host receives a chunk from a sender [8]. In that sense, average throughput is a metric for how much data is transmitted through a network in a certain timeframe. As the throughput is not available in the data collected by the Puffer platform, it is calculated using the size of the video chunks sent during a streaming session and their transmit time. The average throughput per session is calculated as the average over all video chunks sent during a session.

Metric	Equation	Unit
$\text{watchtime}_{\text{session}}$	$= n \cdot 2.002$	[s]
$\text{metric}_{\text{environment}}$	$= \frac{\sum_{i=1}^{m} \text{metric}_{\text{session}\,i} \cdot \text{watchtime}_i}{\sum_{i=1}^{m} \text{watchtime}_i}$	
$\text{RTT}_{\text{session}}$	$= \frac{\sum_{i=1}^{n} \text{RTT}_{\text{chunk}\,i}}{n}$	[µs]
$\text{RTT}_{\text{min,session}}$	$= \min_{i=1}^{n} \text{RTT}_{\text{chunk}\,i}$	[µs]
$\text{RTT}_{\text{difference,session}}$	$= \text{RTT}_{\text{session}} - \text{RTT}_{\text{min,session}}$	[µs]
$\text{throughput}_{\text{session}}$	$= \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\text{size}_{\text{chunk}\,i}}{\text{time\_recevied}_{\text{chunk}\,i} - \text{time\_sent}_{\text{chunk}\,i}}$	[B/s]

Table 4.3: The equations used to calculate the environment evaluation metrics. Here n is the number of chunks (2.002 s each) in a session and m is the number of sessions in an environment.

Evaluation Results

We show an overview of the environment evaluation in Table 4.1. By comparing the different environments we can observe the following:

Cloud environments: Our evaluation shows a stepwise increase in difficulty in terms of network conditions across the different environments. For Europe-Europe, we can observe the lowest average RTT (86.1% decrease versus Europe-Abroad), as well as the lowest average minimum RTT (88.6% decrease versus Europe-Abroad). This trend continues when we compare the throughput, Europe-Europe shows a 71.8% increase over the average throughout of Europe-Abroad. Europe-Abroad and Abroad-Abroad exhibit similar conditions. Europe-Abroad has a slightly lower average and minimum RTT (12.0% and 12.8% decrease), as well as a lower average throughput (22.1% decrease).

Clouds versus Puffer: Puffer exhibits results inbetween Europe-Europe and Europe-Abroad. We can see that both in average RTT and minimum RTT there is a larger gap between Puffer and Europe-Europe, Puffer showing a 84.2% and a 79.8% increase over the conditions in Europe-Europe. The difference between Puffer and Europe-Abroad (11.9% and 43.46% decrease) is smaller. This indicates that the network conditions in sessions streamed via Puffer experience similar latency as in the Europe-Abroad and Abroad-Abroad environments. However, when we compare the average throughput, the conditions in the Puffer network are closer to the throughput observed in Europe-Europe and significantly higher than in Europe-Abroad and Abroad-Abroad (205.0% and 291.8% increase). In terms of average RTT difference, we can observe higher values in the Puffer environment versus the cloud-based environments. We interpret this as reflecting the variability in the range of different network conditions present in the sessions on Puffer. In other words, Puffer streams to clients across a variety of different devices and network connections. In contrast, our clients in the cloud-based environments only stream from one cloud infrastructure to the other, with each client in the same environment experiencing similar network conditions. However, this does not translate into the cloud-based environments exhibiting better network conditions overall.

Conclusion: From these results, we conclude that (i) the network conditions become progressively more challenging from one experiment environment to the next. This is the ideal behavior we aimed to achieve, as this allows us to test the ABR algorithms in increasingly difficult environments and evaluate their generalization to these environments. Further, (ii) while Europe-Europe shows better network conditions in comparison to all environments, the conditions in the Puffer environment are less challenging in terms of latency and throughput when compared to Europe-Abroad and Abroad-Abroad. This does not necessarily mean that the Puffer environment has no variation in network conditions across the sessions, as is visible in the higher average RTT difference measured. However, this does indicate that algorithms trained only in the Puffer environment are exposed to less challenging latency and throughput conditions on average, as compared to other real-world environments such as Europe-Abroad and Abroad-Abroad. This in turn could impact their performance outside of the Puffer environment.

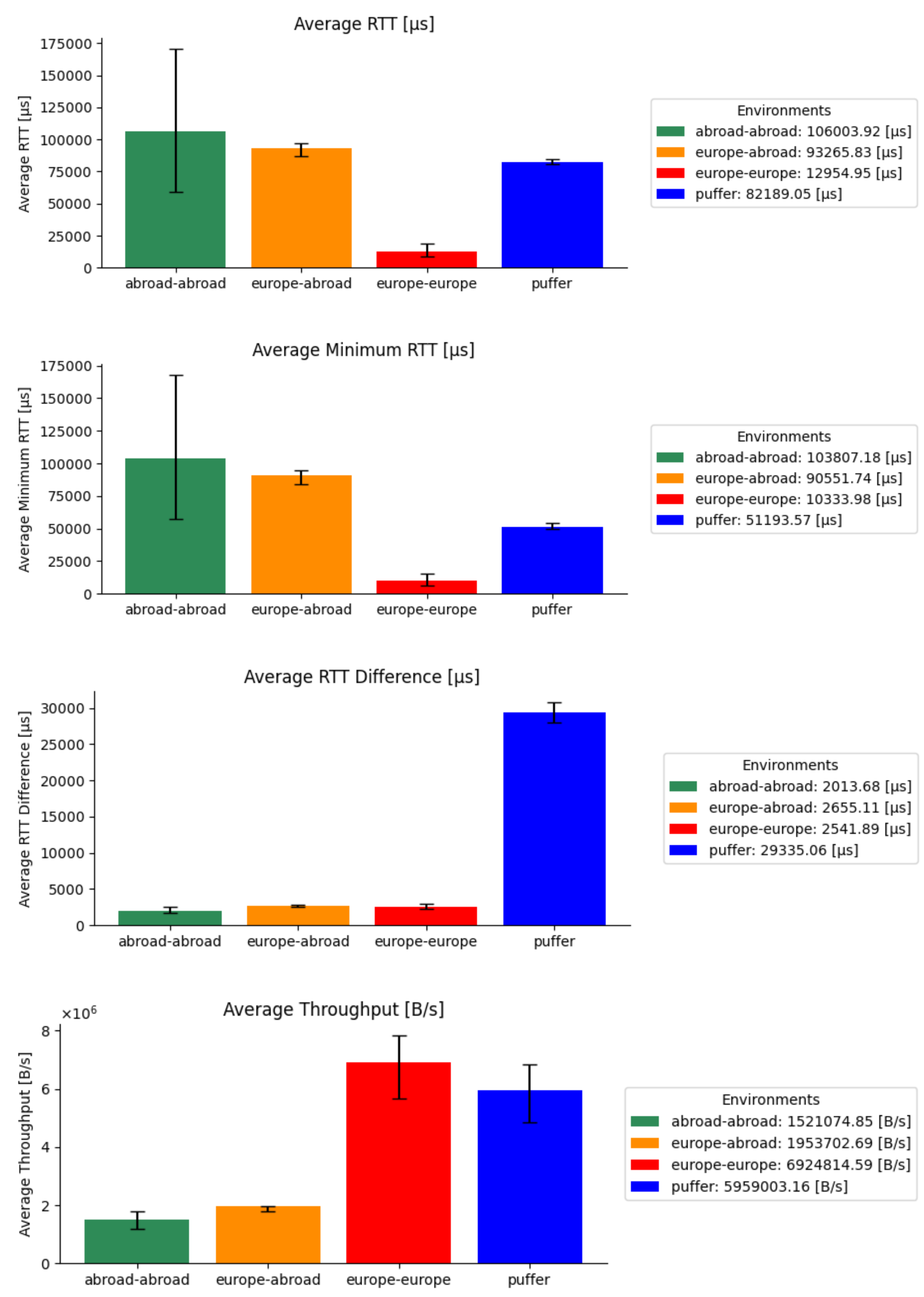


Figure 4.2: Average values and 90% confidence intervals of all metrics. Europe-Europe provides the easiest networking conditions out of all environments.

4.2 Experiments and Results

We perform eight identical experiment sequences, each consisting of twelve individual experiments. The experiment sequences are spread across different days and different start times to capture varying network conditions. In every individual experiment, we test the algorithms BBA, Fugu$_{\text{Feb}}$, Maguro and Unagi within one environment. Each experiment sequence roughly consists of an equal amount of streaming sessions within all environments. This allows us to evaluate the performance of all algorithms in progressively more challenging conditions. In total, we collect the data from 187.54 hours of streaming sessions. An overview of the experiment sequences is given in Table 4.4.

	Timestamp	**Watchtime**[h]	**Amount of Clients**	**ABR Algorithms**
Sequence 1	28.12.23 13:47	23.87	382	BBA, Fugu, Unagi, Maguro
Sequence 2	29.12.23 14:24	25.09	383	BBA, Fugu, Unagi, Maguro
Sequence 3	30.12.23 13:59	23.62	376	BBA, Fugu, Unagi, Maguro
Sequence 4	31.12.23 11:48	24.45	382	BBA, Fugu, Unagi, Maguro
Sequence 5	02.01.24 19:48	24.04	383	BBA, Fugu, Unagi, Maguro
Sequence 6	04.01.24 14:17	21.26	352	BBA, Fugu, Unagi, Maguro
Sequence 7	06.01.24 20:38	22.35	376	BBA, Fugu, Unagi, Maguro
Sequence 8	07.01.24 12:19	22.86	380	BBA, Fugu, Unagi, Maguro
Total		187.54	3014	

Table 4.4: An overview of all experiment sequences. In total, 3014 streaming clients were deployed and 187.54 hours of video were streamed.

We compare the performance of the ABR algorithms across the different environments, as well as to their performance in the Puffer environment. The Puffer data used contains one week of measurements (30.12.2023 - 05.01.2024) with 15 660.63 h of watchtime. In the results, we can observe that:

1. The performance of the algorithms across the custom environments aligns with the results from the environment evaluation. All algorithms show both lower stall time and higher SSIM when progressing from Abroad-Abroad to Europe-Abroad and Europe-Europe.

2. All algorithms achieve high average SSIM values across all environments, with the average values per environment being within 0.22 dB of each other. This is 1.24 % of the total average SSIM. This indicates that no large difference in video quality is apparent between the ABR schemes.

3. While BBA and Fugu show similar performance in the Europe-Europe and Puffer environments, Maguro and Unagi spend more time stalling in the custom environments as compared to the Puffer environment. Further, across all custom environments, both Maguro and Unagi spend more time stalling than BBA and Fugu. They also exhibit a less consistent performance when comparing the results across the custom environments. This is in contrast with the performance they show in recent studies conducted in the Puffer framework [12] and indicates that the algorithms do not generalize well to the environments used in this evaluation.

We present the experiment setup and evaluation metrics in Sections 4.2.1 and 4.2.2. The results of our experiments are presented in Section 4.2.3.

Clients	**Watchtime [s]**	**Media Servers**	**ABR**	**Congestion Control (CC)**
16*	360	4	BBA, Fugu, Maguro, Unagi	cubic

*For experiments 1,2,11,12 we used 64 clients.

Table 4.5: An overview of the streaming parameters used in every experiment. For all ABRs, we use the same configurations as Puffer.

		Europe		**Abroad**	
	server / client	AWS-Norway	ACI-London	AWS-Sao-Paulo	AWS-India
Europe	AWS-Norway	⟍	Expt. 2	Expt. 7	Expt. 9
	ACI-London	Expt. 1	⟍	Expt. 8	Expt. 10
Abroad	AWS-Sao-Paulo	Expt. 3	Expt. 5	⟍	Expt. 11
	AWS-India	Expt. 4	Expt. 6	Expt. 12	⟍

Table 4.6: An overview of the environment parameters used in every experiment.

4.2.1 Experiment Setup

Within each experiment sequence, we conduct twelve individual experiments, ie. twelve different combinations of streaming definitions and environment definitions. The streaming definitions remain widely the same throughout all experiments, we only change the amount of clients to ensure an equal distribution across the different environments. For each client, we set the watchtime to 360 s, which covers over 90% of all session lengths present in the Puffer data. The environment definitions are varied to ensure we are testing across all environments, as well as all locations and providers. Tables 4.5 and 4.6 provide an overview of the streaming and environment design parameters we use for each experiment.

4.2.2 Evaluation Metrics

To compare the performance of an ABR algorithm, we use the two most prominent QoE factors: Video quality and stall time [14]. To measure these factors, we use the average SSIM (Structural Similarity Index) and the average percentage of time spent stalling. Following the same procedure as for the environment evaluation, each metric is first averaged on a per-session basis and subsequently, a weighted average is taken over all sessions, using the watchtime as weights. The equations we use for our calculations are provided in Table 4.7.

SSIM: The SSIM is a metric developed to measure the *perceived* quality of an image or video [15]. The Puffer project suggests converting the standard SSIM metric from its range between 0 and 1 to a decibel (dB) scale, which is the format we follow here. The SSIM [db] per session is then calculated as an average over each chunk.

Percentage of time spent stalling: The percentage of time spent stalling (TSS) is defined as the ratio between total time spent buffering and total session time, ie. the watchtime plus the time spent rebuffering.

Metric	**Equation**	**Unit**
$\text{SSIM}_{\text{chunk}}$	$-10 \cdot \log_{10}(1 - \text{SSIM}_{\text{chunk}})$	[dB]
$\text{SSIM}_{\text{session}}$	$= \frac{\sum_{i=1}^{n} \text{SSIM}_{\text{chunk}\,i}}{n}$	[dB]
$\text{TSS}_{\text{session}}$ [%]	$\frac{\text{total\_time\_buffering}_{\text{session}}}{\text{watchtime}_{\text{session}} + \text{total\_time\_buffering}_{\text{session}}} \cdot 100$	[dB]

Table 4.7: The equations we use to calculate the performance evaluation metrics. Here n is the number of chunks (2.002 s each) in a session.

	Abroad-Abroad		**Europe-Abroad**		**Europe-Europe**		**Puffer**	
	TSS [%]	SSIM [dB]	TSS [%]	SSIM [dB]	TSS [%]	SSIM [dB]	TSS [%]	SSIM [dB]
BBA	**0.43**	17.55	**0.24**	17.58	**0.07**	17.61	0.14	17.51
Fugu	0.59	17.66	0.33	17.73	0.10	17.78	0.13	17.45
Maguro	1.97	17.74	1.68	17.78	0.95	**17.83**	**0.10**	**17.66**
Unagi	1.74	**17.77**	1.60	**17.80**	1.02	**17.83**	0.14	**17.66**

Table 4.8: An overview of the average time spent stalling (TTS) [%] and the average SSIM [dB].

4.2.3 Results

In Figures 4.4 and 4.5, we show the average SSIM over the average TSS for each algorithm across all environments (both rounded to two digits), including the Puffer environment. We provide an overview of the average values in Table 4.8.

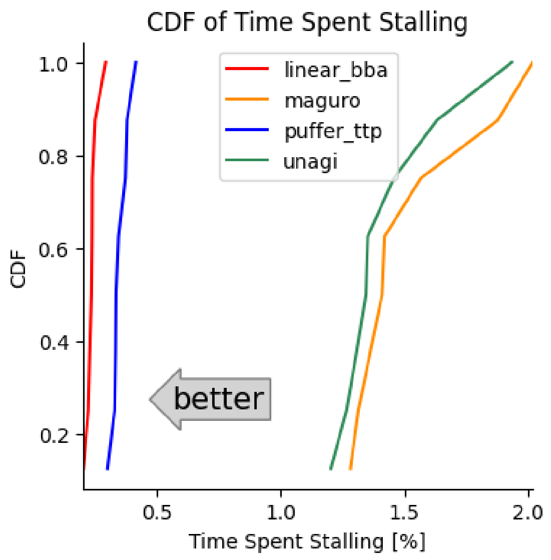
Performance across the environments: When we examine the plots showing the SSIM over TSS in Figure 4.4, a progression in terms of performance is visible when going from Abroad-Abroad to Europe-Abroad and Europe-Europe. We can observe that the time spent stalling decreases for all algorithms when we go from a more challenging environment to an easier one. The same trend is recognizable in SSIM, however at a much smaller scale. This aligns with our observations from the environment analysis and validates our approach by showing that we are able to create custom environments that exhibit different network conditions and impact an algorithm's performance.

Video quality: In terms of SSIM, the algorithms perform similarly across all environments, as is visible in both Table 4.9 and the scaled plots in Figure 4.5. The average SSIM values per environment are within 1.26% of each other, with the lowest SSIM overall being 17.45 dB (Fugu in the Puffer environment) and the highest being 17.83 dB (Maguro and Unagi in the Europe-Europe environment). This indicates that there is no large variation in video quality.

Time spent stalling: The relative performance of the algorithms in time spent stalling we observe in the custom environments is significantly different than in the Puffer environment, as shown in Figure 4.5. While all algorithms perform similarly well in the Puffer environment, with Maguro achieving the lowest TSS (0.10%), this changes when we progress to the custom environments.

	Abroad-Abroad	**Europe-Abroad**	**Europe-Europe**	**Puffer**
SSIM Range [db]	0.22	0.22	0.22	0.21
SSIM Range [%]	1.23	1.26	1.23	1.19

Table 4.9: The range of SSIM values in dB and in percent of the lowest value.



	Average TSS [%]	**Relative Max. Difference to BBA**
BBA	0.239	
Fugu	0.351	32.00%
Maguro	1.533	84.40%
Unagi	1.438	83.38%

Figure 4.3: Here we show the CDF and the average of percentage of time spent stalling. BBA (linear_bba) and Fugu (puffer_ttp) exhibit better performance in the experiments.

We can observe: (i) Maguro and Unagi stall more than both BBA and Fugu, with BBA spending 84.40% and 83.38% less time stalling on average across all experiments, as we show in Figure 4.3. BBA shows the lowest stall times in the custom environments with Fugu performing in a similar range (+32.00% to BBA on average). (iii) Both Maguro and Unagi show more variation in stall time across the environments than BBA and Fugu. The TSS [%] values across the custom environments for BBA are in a range of 0.36 and in a range of 0.49 for Fugu. In contrast, Maguro and Unagi show a range of 1.02 and 0.73, respectively. This indicates that Maguro and Unagi are more sensitive to the network conditions in the custom environments than BBA and Fugu.

Conclusion: While all algorithms show similar QoE results in terms of video quality, the main performance difference that we observe is the percentage of time spent stalling. In the custom environments, Maguro and Unagi spend more time stalling than BBA and Fugu. This is especially interesting, as both algorithms were shown to outperform Fugu and BBA in prior studies conducted in the Puffer environment [12]. The shift in performance from the Puffer environment to the custom environments indicates that Maguro and Unagi are overfitted to perform well in one environment, ie. on the Puffer platform. We thus conclude that Maguro and Unagi do not generalize well to the real-world environments we use in this evaluation. One possible explanation for this observation is that the data collected from the Puffer platform, which was used to train Maguro and Unagi, exhibits less challenging network conditions on average than environments such as Europe-Abroad and Abroad-Abroad, as we show in the environment evaluation. However, Fugu, also a learning-based algorithm that was trained in the Puffer environment, shows more stable performance across the custom environments. This discrepancy might be explained by the different types of ML approaches used in these ABR schemes [16]. While Fugu uses a DNN-based TTP, both Unagi and Maguro are RL-based algorithms. As they are trained in a training environment, as opposed to *in situ* as Fugu is, they tend to perform worse when it comes to generalization [16]. We interpret this result as further underlining the importance of training and validating ML-based algorithms in a diverse set of network conditions and real-world environments.

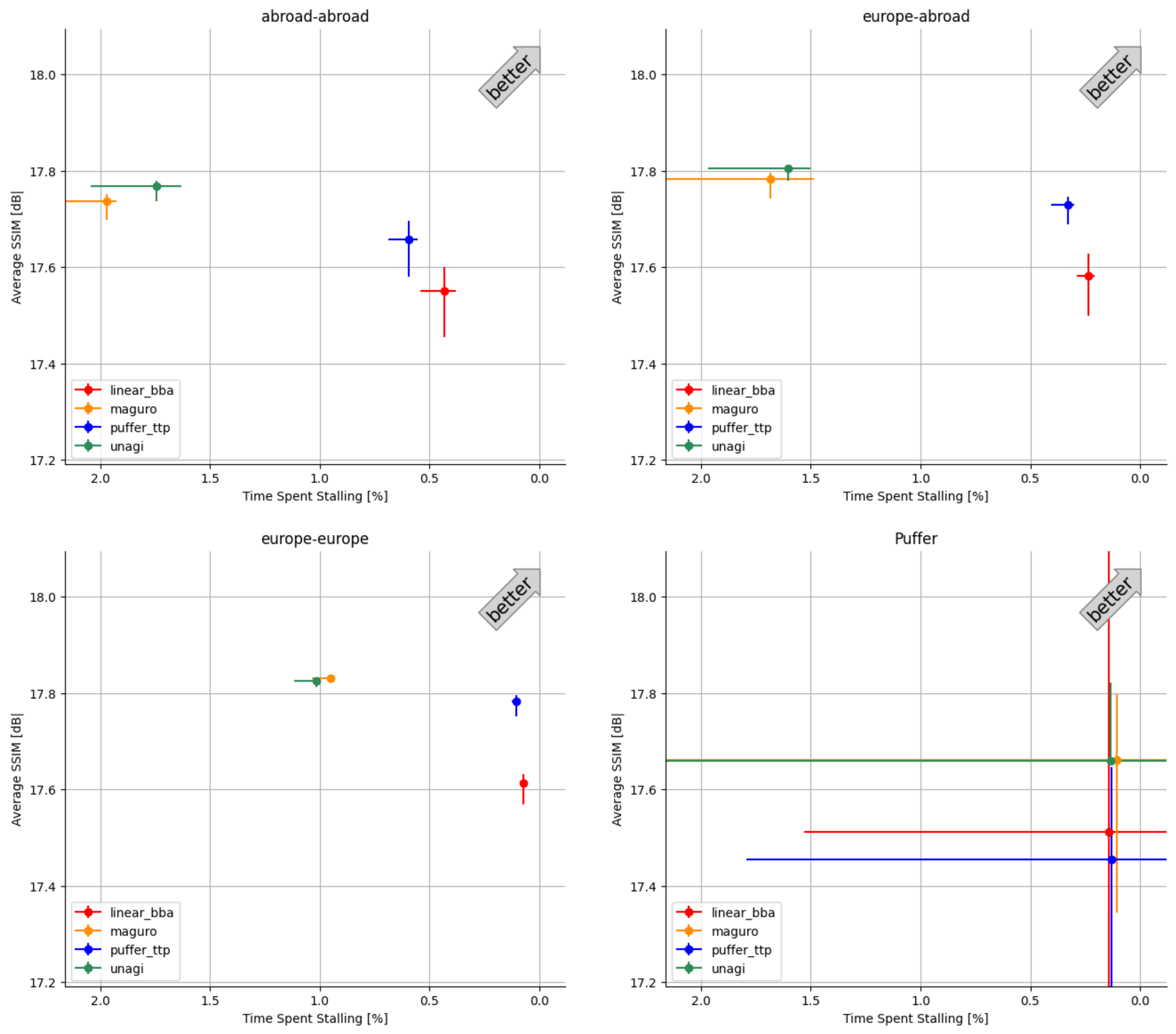


Figure 4.4: The plots show the average for SSIM [dB] over the time spent stalling [%], as well as the 90% confidence intervals. Note that the y-axis does not start at zero and is zoomed in to make the differences in SSIM visible.

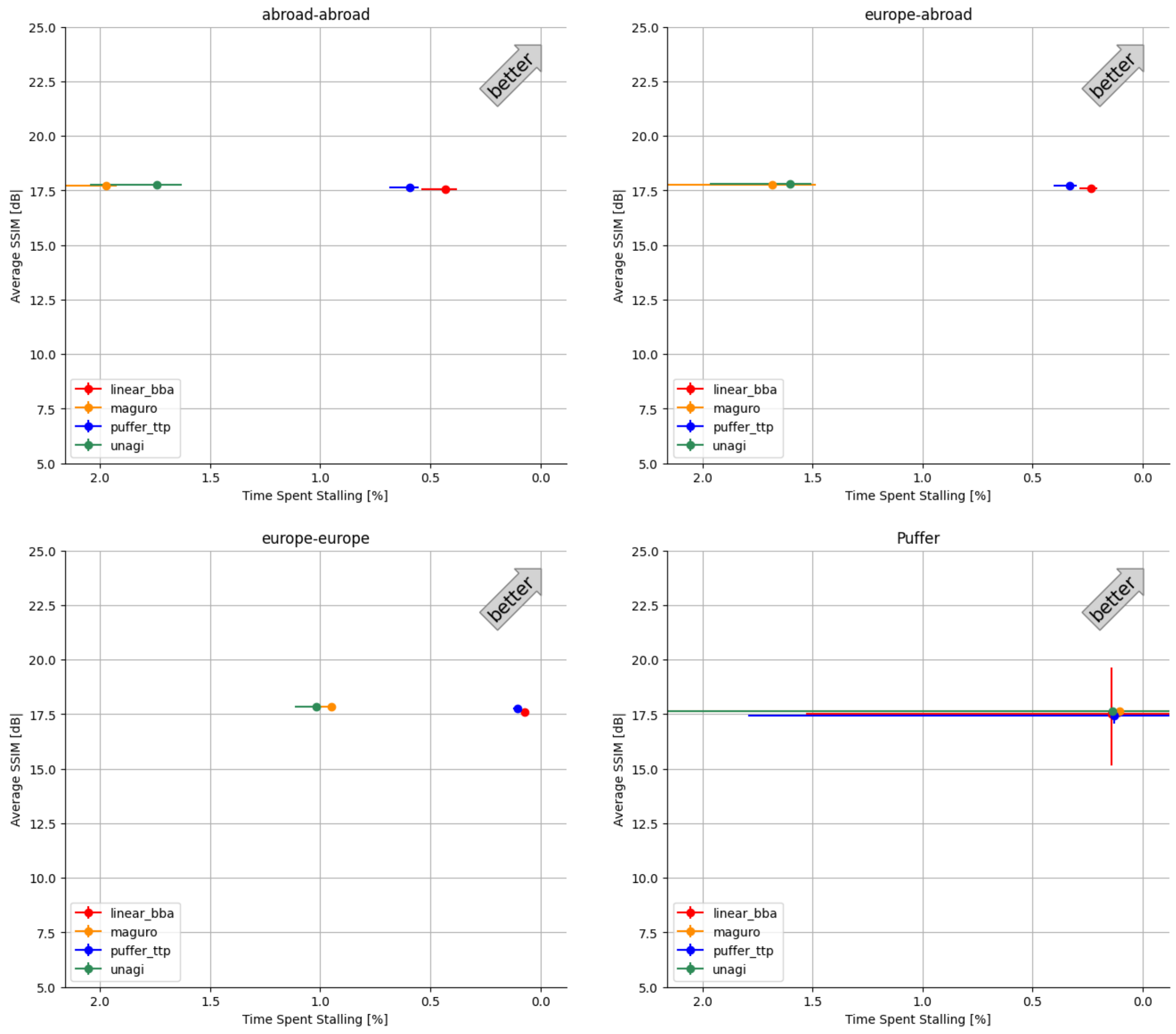


Figure 4.5: The plots show the average for SSIM [dB] over the time spent stalling [%], as well as the 90% confidence intervals. Here the differences in time spent stalling are more visible.

Chapter 5

Summary and Outlook

We presented an infrastructure for the evaluation of ML-based video streaming algorithms across real-world environments. In our implementation, we adapted the video streaming infrastructure offered by the Puffer project to be compatible with elements of netUnicorn to deploy and manage experiments. By doing so, we created a platform that allows for simultaneously testing different ABR schemes on streaming servers and clients deployed to different cloud infrastructures across the globe, as well as collecting the resulting QoE data, streaming measurements and packet traces. We designed the experiment pipeline to allow for the execution of custom experiments and environments that are defined by a set of streaming-specific and environment-specific parameters.

Via this infrastructure, we designed and conducted experiments across three different real-world environments. In our evaluation, we compared both the network conditions in the different environments, as well as the performance of the four ABR algorithms BBA, Fugu, Unagi and Maguro across all environments and their performance on the Puffer platform. All algorithms showed similar results in video quality, however, their performance in terms of time spent stalling differed significantly. While Unagi and Maguro, both trained using Puffer data, showed the best performance in the Puffer environment, they exhibited lower performance than BBA and Fugu when tested in the custom environments. We concluded that using only a single real-world environment for training and validating learning-based algorithms, does not guarantee their generalization to other real-world deployments. In doing so, we highlight the importance of comparing ABR algorithms in different settings outside of their training environments.

By providing this infrastructure, as well as the insights it generates, we hope to contribute towards the development of more robust and generalizable ABR algorithms. To approach this goal, we propose to expand the use of our framework for additional algorithm evaluation, as well as training and improving learning-based algorithms.

Variety of algorithms: Increasing the variety of algorithms allows for a more comprehensive comparison of different approaches. While we focus on four different algorithms in our evaluation, the proposed platform can scale to an arbitrary number of different schemes. This allows for gaining more insight into which schemes, or variations of an algorithm, perform well in which environments. This can be of particular use when testing different retrained iterations of learning-based algorithms to prevent overfitting to a specific environment. To this end, apart from supporting algorithm-based research, future implementation work could also include increasing the amount of different algorithms that are natively available.

The variety of environments: Increasing the variety of environments used for evaluation is key to truly capturing the underlying factors that impact the performance of an algorithm. By evaluating both the network conditions, as well as the QoE results achieved, for a large range of different environments, we can attempt to understand which conditions make a particular algorithm struggle and why. As network conditions vary between different target infrastructures, additional environments using other locations and cloud providers might provide additional insights not observed in the data we collected. Further, as we exclusively used cloud providers in our experiments, an expansion to include other arbitrary endpoints and VMs would improve the diversity even further. Since we use netUnicorn in our setup, this expansion is supported.

Amount of experiments: Increasing the amount of experiments could provide further insight into phenomena not present in the limited dataset we have collected. By increasing the amount of iterations per experiment and deploying the same experiments on a more regular basis, we would allow for capturing temporal patterns in network conditions, as well as their impact on algorithm performance. Further, the amount of streaming data collected in the Puffer project significantly exceeds the watchtime we have incurred in our experiments. Accumulating more streaming data, possibly even at a similar rate as the Puffer project, would allow for a more comprehensive comparison of the performance of different algorithms across the different platforms.

Algorithm training: In addition to algorithm evaluation, the data collected by our infrastructure can be used to improve the performance of learning-based algorithms. As the data format is identical to the one used by Puffer, additional data collected via our platform can be used in addition to the Puffer data when training or retraining algorithms. This would increase the different network conditions an algorithm experiences during training and generally adds diversity to the dataset. Future work could also include expanding the proposed infrastructure to be used for *in situ* training of algorithms. Inspired by the Prioritized Trace Selection framework proposed in prior work [12], dynamically prioritizing the training in environments that show low performance and high learning potential could be implemented.

Packet traces: In addition to the chunk-level data that is collected by including Puffer in our setup, we additionally collect packet traces on both the client and the server. Future work could include using this fine-grained information for more detailed analysis of experiment results, or as training data for algorithms.

Bibliography

[1] Akhtar, Z., Rao, S., Ribeiro, B., Nam, Y. S., Chen, J., Zhan, J., Govindan, R., Katz-Bassett, E., and Zhang, H. OBoe: Auto-tuning video ABR algorithms to network conditions. *SIGCOMM 2018 - Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication 1* (2018), 44–58.

[2] Beltiukov, R., Gupta, A., Guo, W., and Willinger, W. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. *CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (2023), 2217–2231.

[3] Claeys, M., Latre, S., Famaey, J., and De Turck, F. Design and evaluation of a self-learning http adaptive video streaming client. *IEEE Communications Letters 18*, 4 (2014), 716–719.

[4] Dietmüller, A., Ray, S., Jacob, R., and Vanbever, L. A new hope for network model generalization. *HotNets 2022 - Proceedings of the 2022 21st ACM Workshop on Hot Topics in Networks* (2022), 152–159.

[5] Dobrian, F., Sekar, V., Awan, A., Stoica, I., Joseph, D., Ganjam, A., Zhan, J., and Zhang, H. Understanding the impact of video quality on user engagement. *Computer Communication Review 41*, 4 (2011), 362–373.

[6] Duanmu, Z., Zeng, K., Ma, K., Rehman, A., and Wang, Z. A Quality-of-Experience Index for Streaming Video. *IEEE Journal on Selected Topics in Signal Processing 11*, 1 (2017), 154–166.

[7] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 187–198.

[8] James F. Kurose, K. W. R. *Computer Networing A Top-Down Approach 6th ed.* 2013.

[9] KINGSLEY, M. S. *Cloud Technologies and services: Theoretical Concepts and Practical Applications.* SPRINGER INTERNATIONAL PU, 2023.

[10] Kua, J., Armitage, G., and Branch, P. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP. *IEEE Communications Surveys and Tutorials 19*, 3 (2017), 1842–1866.

[11] Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. *SIGCOMM 2017 - Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication* (2017), 197–210.

[12] Patel, S., Zhang, J., Jyothi, S. A., and Narodytska, N. Plume: A Framework for High Performance Deep RL Network Controllers via Prioritized Trace Sampling.

[13] Sandvine Corporation. Video Permeates, Streaming Dominates. *The Global Internet Phenomena Report*, January (2023), 14–15.

[14] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking 28*, 4 (2020), 1698–1711.

[15] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[16] Yan, F. Y., Ayers, H., University, S., Zhu, C., Fouladi, S., Hong, J., Zhang, K., Levis, P., and Winstein, K. This paper is included in the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) Learning in situ: a randomized experiment in video streaming Learning in situ: a randomized experiment in video streaming.

[17] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *Computer Communication Review 45*, 4 (2015), 325–338.

[18] Yousef, H., Feuvre, J. L., and Storelli, A. ABR prediction using supervised learning algorithms. *IEEE 22nd International Workshop on Multimedia Signal Processing, MMSP 2020*, Cmcd (2020).

Appendix A

Appendix

A.1 Puffer Data Collection

Measurement	**Description**
time	timestamp [ns] when the chunk is sent
session_id	unique ID for the video session
index	non-unique ID for the video session
expt_id	unique ID of ABR-scheme
channel	TV channel name
video_ts	presentation timestamp of chunk (always 2.002[s])
format	encoding settings of the chunk, [WidthxHeight-CRF]
size	chunk size [B]
ssim_index	SSIM of chunk [unitless]
cwnd	congestion window size in packets
in_flight	number of unacknowledged packets in flight
min_rtt	minimum RTT [µs]
rtt	smoothed RTT estimate [µs]
delivery_rate	TCP's estimation of delivery rate [B/s]
buffer	playback buffer size [s]
cum_rebuf	total time [s] client has spent rebuffering in current stream

Table A.1: An overview of the data collected for each sent video chunk. This description is based on the Puffer project [16].

Measurement	Description
time	timestamp [ns] when chunk acknowledgement received
session_id	unique ID for the video session
index	non-unique ID for the video session
expt_id	unique ID of ABR-scheme
channel	TV channel name
video_ts	presentation timestamp of chunk (always 2.002[s])
buffer	playback buffer size [s]
cum_rebuf	total time [s] client has spent rebuffering in current stream

Table A.2: An overview of the data collected for each acknowledged video chunk. This description is based on the Puffer project [16].

Measurement	Description
time	timestamp [ns] when the server receives the client message
session_id	unique ID for the video session
index	non-unique ID for the video session
expt_id	unique ID of ABR-scheme
channel	TV channel name
event	event types include: init, startup rebuffer, play, timer (periodic event)
buffer	playback buffer size [s]
cum_rebuf	total time [s] client has spent rebuffering in current stream

Table A.3: An overview of the data collected for each event message from the client. This description is based on the Puffer project [16].