Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Master‘s Thesis

HASBRAIN: A Machine Learning-based Adaptation Algorithm for HTTP Adaptive Streaming

Author:	Hagn, Korbinian
Address:	Kloib 112
	83646 Wackersberg
Matriculation Number:	03668367
Supervisor:	Christian Sieber M.Sc.
Begin:	08. February 2017
End:	16. August 2017

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, 16.08.2017	
Place, Date	Signature

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of the license, visit http://creativecommons.org/licenses/by/3.0/de

Or

Send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

München, 16.08.2017	
Place, Date	Signature

Abstract

This thesis presents a new Quality of Experience (QoE) aware adaptation algorithm for HTTP Adaptive Streaming (HAS) based on supervised Machine Learning (ML) techniques named "**H**TTP **A**daptive **S**treaming with **B**it-**R**ate change aware **A**rtificial **IN**telligence" or just **HASBRAIN**. With use of the optimal adaptation path, the optimal video quality picking strategy, a ML algorithm is trained and learns to behave in similar optimal fashion. The first contribution of this work is the modification of an existing optimization formulation with the goal of determining the Pareto frontier for QoE aware adaptation algorithms. This formulation allows to evaluate with an aggressiveness switching parameter α the trade-off between average quality and quality switching in the video playout. The results of this evaluation lead to a modified version of a two-step optimization problem formulation with an allowable maximum quality level degradation parameter ϵ while minimizing the number of quality switches.
Given a video and a data-rate histogram, the optimization outputs the optimal adaptation path. The optimal adaptation path is the optimal quality level picking strategy at each decision instance in time. The optimal decisions are used as input for training a ML algorithm. These supervised trained ML algorithms consist of an Artificial Neural Network (ANN), a Support Vector Machine (SVM) and a k-Nearest-Neighbours approach.
The best performing learning technique, in terms of learning accuracy, is then used as an adaptation logic in a HAS scenario simulated through a Discrete Event Simulation (DES). The performance is evaluated through user-centric evaluation metrics gathered from the DES. These metrics are the average playout quality, the quality switching frequency, the stalling-frequency and -ratio and the buffer level throughout the video playout. The **HASRBAIN** algorithm's performance is furthermore compared with the performance of well-known threshold-based adaptation algorithms TRDA and KLUDCP within the same evaluation scenario. The ML's adaptation performance behavior allows it to reach a similar average playout quality than the other algorithms while reducing the number of quality switches. But due to wrong adaptation decisions the ML tool is in some scenarios not able to prevent stalling to occur.
This work is concluded by the pros and cons of this algorithm compared to the threshold-based algorithms. Furthermore, an outlook for future work is given to further improve the performance through adjustments in the ML algorithms learning.

Acknowledgement

I would like to thank all the people that gave insightful thoughts and inspiration throughout this master's thesis. Especially, i want to thank:

- Christian Sieber, for making this thesis possible. Thanks for the comprehensive explanations, for the guidance into elaborate academic research and the thoughtful ideas that helped for the advancement in the work when facing an allegedly dead end.
- Christian Moldovan, for his detailed counsel regarding the provision of the mathematical background of the optimization, as well as for his profound ideas for further improvement on several points in this work.
- The Chair of Communication Networks, for allowing me to use their computation facilities.
- My family and especially my girlfriend, for their support and cheering me up when something did not come out as planned.

Thanks to all of you for your contribution to this work.

Contents

Chapter 1

Introduction

The platforms for media consumption experience a great shift in popularity, where it was traditional linear television broadcasting in the past it is nowadays video streaming over the Internet. YouTube, Netflix and Amazon Prime Video, just to name a few, are increasingly popular platforms for internet video streaming. In Figure 1.1 an overview over the most popular online video streaming hoster at the time of writing is given.

Figure 1.1: An overview of the most popular adaptive streaming providers.

The most widespread technique to deliver this video content is HTTP Adaptive Streaming (HAS). By dividing a video into segments with each segment being available in different quality levels, i.e. bit-rates, an adaptation logic determines the quality level of a segment to download. The algorithms used for the adaptation on variable influence factors, such as volatile connection characteristics, allow for a high viewer experience expressed as the Quality of Experience (QoE). But these adaptation algorithms differ in their aggressiveness of quality adaptation, i.e. some are designed to maximize the video playout quality whereas

others are designed for a smooth playout with less switches of quality. It is not known which strategy or which level of adaptation aggressiveness is best. Hence, as part of the contribution of this work an extension of an existing optimization problem formulation to evaluate the trade-off between quality switching and average playout quality is evaluated. This optimization problem formulation is comprised of a weighting parameter α that allows for a trade-off between quality switching and average playout quality. By increasing the value of α the optimization maximizes the average playout quality. On the other hand does a lower value of α allow for minimizing the number of quality level switches but the optimal solution might not achieve the same average quality as with a high α value. Also another extension of an optimization formulation is discussed. This formulation allows with a quality degradation parameter ϵ, that specifies the maximal allowed degradation of optimal average quality during playout, to minimize the number of quality switches.

Furthermore, the rise of popularity among Machine Learning (ML) techniques in various fields of application fuels the development of new disruptive technologies. As the most applications of Machine Learning algorithms are hidden from the user, for example in video recommendation systems, their popularity gained traction, not only in research and development, but also in the understanding of the general public through increased coverage in the media.

Therefore it was just a matter of time until the field of Machine Learning was introduced into the research field of Adaptive Streaming. With these new techniques at hand, new powerful adaptation strategies can be implemented. Especially compared to simple threshold-based adaptation algorithms more complex adaptation strategies in terms of QoE awareness can be implemented.

The main contribution of this work concentrates on implementation and evaluation of a novel approach to use the resulting optimal adaptation path from the optimization, i.e. the optimal picking strategy of quality levels, and train a Machine Learning classifier through supervised learning, for example an Artificial Neural Network (ANN), to act as an optimal adaptation logic. This adaptation logic should then reflect in its behaviour the optimal adaptation strategy. The whole process of optimization, training and finally use of the algorithm as adaptation logic is named **HASBRAIN** and defined as **H**TTP **A**daptive **S**treaming with **B**it-**R**ate change aware **A**rtificial **IN**telligence.

The performance of the **HASBRAIN** adaptation logic is evaluated through Discrete Event Simulation (DES) of an adaptive streaming scenario with real goodput pattern and real YouTube videos. To put the results in perspective, two threshold based adaptation algorithms, TRDA and KLUDCP, have been implemented and their adaptation performance in the same adaptive streaming scenario is compared to the performance of the Machine Learning technique.

Finally the results are summarized and discussed and an outlook to future research in both the area of optimal adaptive streaming as well as the area of supervised learning techniques for adaptive streaming is given.

The first chapter gives the background on HTTP Adaptive Streaming and the different used ML techniques. Furthermore the related work on existing adaptation algorithms as well as Quality of Experience and Optimal Adaptive Streaming is given. Chapter 3 presents the methodology of this work and defines the used nomenclature. Also the **HASBRAIN** algorithm and the optimal adaptation path are described as well as the adaptation performance evaluation through the Discrete Event Simulation. Following in chapter 4 the **HASBRAIN** algorithm and it's implementation are discussed in detail. Thereafter the ML training results and the adaptation performance are evaluated and summarized in chapter 5. Finally, the thesis is concluded and an outlook on future work is given.

Chapter 2

Background & Related Work

In this chapter the necessary basics of HTTP Adaptive streaming (HAS) are given as well as two threshold-based adaptation algorithms from the literature are explained. A first implementation of HAS with a Machine Learning (ML) approach named Q-learning is shown. Furthermore an introduction into Quality of Experience (QoE) models is given and the previous work on Optimal Adaptive Streaming is presented. This chapter is concluded by an short introduction of the used Machine Learning classification algorithms that are used in this work.

2.1 HTTP Adaptive Streaming

HTTP Adaptive Streaming is the state of the art method for multimedia streaming. In the following the process of a streaming session is briefly explained on example of the MPEG-DASH technique.

MPEG-DASH was first specified in 2011 by [Sod11] and is a widely used HAS technique. DASH stands for Dynamic Adaptive Streaming over HTTP.

Figure 2.1 illustrates the schematic of a DASH session. In the beginning of a DASH sessions there are three components to be named. The client that wants to stream the multimedia content, the server which provides the multimedia content and the network as connection between those two with variable throughput conditions. The client consists of a buffer, a player and an adaptation logic to select the quality level.

When starting a new streaming session the client requests a Media Presentation Description (MPD) file from the server. This MPD holds information about the multimedia content that is to be streamed. For simplification we assume that the multimedia content is a video without any audio track. A video is in context of HAS split up into individual sub parts, called segments, therefore the MPD gives insight into the length of a video, its individual

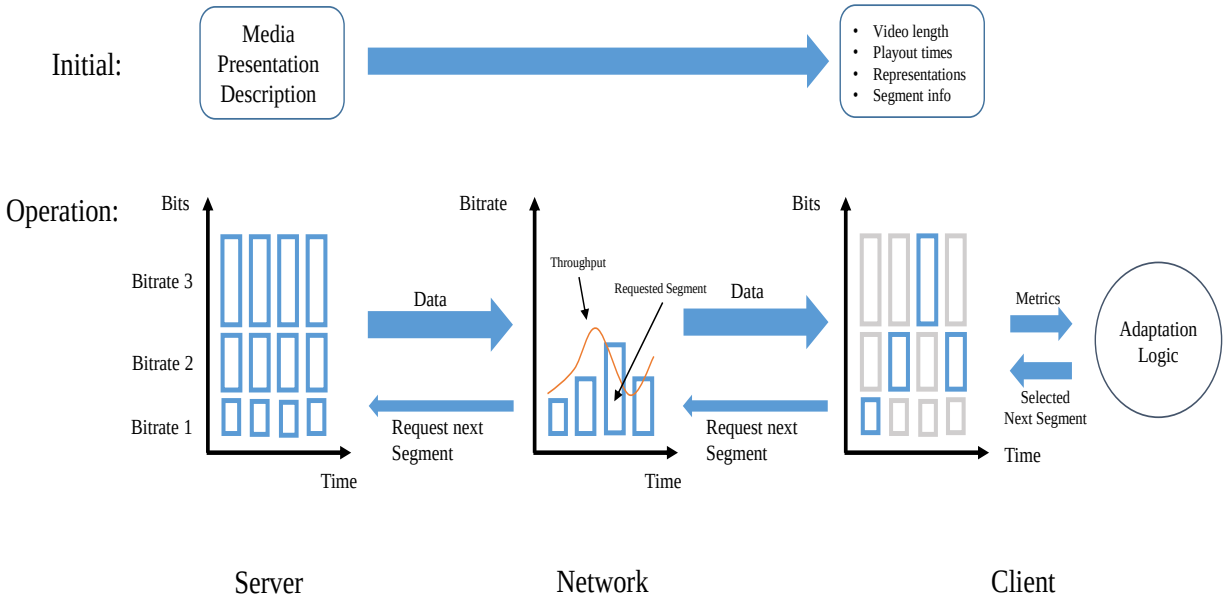


Figure 2.1: The schematic of an adaptive streaming session.

segment sizes and segment lengths, as well as playout times of each segment and available quality representations or bit-rates.

With this information the client now requests the first video segment in a quality level determined by its adaptation logic. The request is forwarded over the network and received by the server. The server now answers the request with the segment in the requested bit-rate. After fully receiving the segment at the client the playout of the same starts, while the adaptation logic again determines the next quality level of the following segment to download considering the current state of the client, for example the average goodput or the current buffer level. Then the next request to the Server is emitted. This process continues until the whole video has been fully received at the client.

The adaptation logic in this cases can vastly differ in the way how it selects the next quality level of a segment. While there are simpler implementations, i.e. a threshold-based adaptation that selects a quality level solely on the current measured throughput, there are also more complex adaptation techniques that consider a lot more conditions in a scenario as going so far, as to implement a QoE optimization strategy.

2.2 Adaptation Algorithms

To get an understanding on adaptation logics, two common adaptation algorithms are introduced and explained. These two algorithms were implemented and later in this work used to compare the performance of **HASBRAIN** against traditional HAS techniques.

Also another approach of an adaptation algorithm based on the ML tool of reinforcement learning is discussed.

2.2.1 Threshold-Based Adaptation Algorithms

Threshold-based algorithms are termed threshold-based because the choice of the next quality representation solely depends on input values that have to reach a certain defined threshold to cause a change of quality selection. A simple example would be to measure the average throughput and switch to a higher quality bit-rate if a sufficient value for the throughput has been measured.

The first algorithm henceforth referred to as KLUDCP was introduced by [MLT12] and the second one henceforth referred to as TRDA was published by [MQGW12]. Both algorithms were designed for single layer content, i.e. for example the Advanced Video Coding (AVC) or MPEG-4 standards. Whereas the KLUDCP algorithm is quite simple in its functionality, the TRDA algorithm is a more sophisticated and complex threshold-based algorithm.

KLUDCP

The KLUDCP algorithm needs in total three input parameters to determine a quality level for the next segment s_i to download. First, the average throughput measured by the download of the proceeding segment $i-1$. Next, the current buffer level as ratio of a configured maximum buffer level. Last, the average bit-rate of each available quality representation. Then an estimation of the current available bandwidth is calculated as a function of the buffer level and compared to the average bit-rates of each quality representation. Segment s_i is chosen to be the highest representation whose bit-rate is lesser than or equal to the estimated available bandwidth.

The estimation of the available bandwidth as a function of the buffer level decreases the estimation by 30% if the buffer level is below 15% of the maximum and decreases it by 15% if the buffer level is between 15% and 35%. But the estimation is also increased if the buffer reaches a fullness of over 50% then the available bandwidth is increased by $(1 + 0.5 * bl_i)$.

Where bl_i depicts the buffer level for the decision of segment s_i normalized by the maximum buffer. The following formulas show the available bandwidth estimation calculation where $maxbw(s_i - 1)$ is the maximum estimated bandwidth and $bw(s_i)$ is a function that returns the measured bandwidth through the download of segment s_i:

$$maxbw(s_i) = \begin{cases} bw(s_{i-1}) * 0.3 & \text{if } 0.00 \leq bl_i < 0.15 \\ bw(s_{i-1}) * 0.5 & \text{if } 0.15 \leq bl_i < 0.35 \\ bw(s_{i-1}) & \text{if } 0.35 \leq bl_i < 0.50 \\ bw(s_{i-1}) * (1 + 0.5 * bl_i) & \text{if } 0.50 \leq bl_i \leq 1.00 \end{cases} \tag{2.1}$$

This algorithm allows for higher bit-rates to be downloaded even if the actual current goodput is not that high, provided that the buffer reaches over half of its maximum fullness. On the other hand there are fast and also many quality switches to expect when the goodput pattern is highly volatile.

TRDA

As the first algorithm TRDA also takes three input parameters. The average bit-rates of each quality representation, the current buffer level in seconds of playback time and the average throughput measured during the download of proceeding segments as in comparison to KLUDCP where it is only the last segment.
The TRDA algorithm offers a so called fast start phase to reduce the initial playout delay. While the fast start is active the algorithm increases the selected next quality representation beginning from the lowest quality level until the highest quality level is reached or the measured bandwidth is decreasing.

After the start phase, like in the KLUDCP algorithm, the decision of the next representation is a function of the current buffer level. In the beginning there have to be defined several different buffer levels as shown in table 2.2.1.

Variable Name	Definition
B_{min} [2]	Minimum buffer level in [s]
B_{max} [20]	Maximum buffer level in [s]
B_{low} [5]	Low buffer level in [s]
B_{high} [15]	High buffer level in [s]
B_{opt} [10] $= 0.5(B_{low} + B_{high})$	Optimal buffer level in [s]

Table 2.1: Buffer level definition for the TRDA algorithm.

The values in the brackets correspond to the default values that were also used in the implementation of this algorithm.

If the buffer level is lesser than or equal to the threshold of B_{min} then the algorithm will always select the lowest representation as next segment. Furthermore if the buffer level is between B_{min} and a configured low buffer level B_{low} then the next lower representation to the current one is chosen for the next segment. If the buffer level is in between the B_{low} and B_{high} level then the next representation stays the same as the current, additionally the download can be delayed if the buffer level is above B_{opt}. Is the buffer level between B_{high} and B_{max} then the next segments representation is chosen to be the next higher bit-rate if

the measured bandwidth is sufficient, i.e. if the goodput bit-rate is greater than or equal to the next higher quality bit-rate. Otherwise the next selected segments representation stays the same and the download is delayed by the playout time of one segment.

The delaying of downloads for the next segment leads to a depletion of the current buffer level. This depletion is wanted as to reach the optimal buffer level B_{opt}.

A full implementation in pseudo code of the algorithm can be found in [MQGW12].

The goal of TRDA is to not only maximize the average quality of the video playout and to minimize the initial playout delay but also to minimize the quality switches between the different representation levels of the video. This is achieved by trying to hold the buffer level at the B_{opt} level by delaying subsequent segment downloads and only allow a switch to the next higher quality if the buffer is above B_{high} and the estimated throughput is sufficiently high or only switching to a lower quality level if the buffer is almost depleted.

2.2.2 Reinforcement Learning

The Reinforcement ML approach was not considered in this work but for the sake of completeness the result of various researchers is presented that used a reinforcement learning approach, namely Q-learning, to implement a QoE aware adaptive streaming client.

Q-learning is a model free approach to find the best set of actions for a given high level goal and was most thoroughly described and discussed by [WD92]. A client learns from a finite set of states and action to optimally fulfill a given policy. In the area of adaptive streaming it is the process of finding the best choices of segments for a decision situation in time following the given policy, for example the optimization of QoE requirements.

This approach was used for adaptive multimedia as early as 2006 by [FWL06]. In this work they laid their focus on the optimal Quality of Service (QoS) provisioning in mobile networks with the use of Q-learning.

Further application of Q-learning was by [CLF+13] with a first approach on HTTP adaptive streaming considering a QoE model that punishes extensive switching of the playout quality and on the other hand rewards a high average playout quality.

Maxim Clayes et al. refine their approach regarding QoE in [CLF+14] by simplifying the reward functions and paying more attention towards a healthy buffer filling. This refined approach was thoroughly evaluated by [CLFDT14] and it was shown that their approach could outperform the Microsoft IIS Smooth Streaming (MSS) algorithm by 13%.

Further improvements, i.e. in terms of bandwidth awareness, reduced buffer filling and multi-user fairness and evaluations where presented by [vdHPC+15]. This Q-learning approach was more recently evaluated for its adaptiveness by [MCG16].

But this extensive study of the field of reinforcement learning in combination with adaptive streaming are still left for improvement as the simultaneously optimization of resource utilization and QoE are non optimal because the optimal adaptation path, i.e. the optimal quality level picking strategy according to certain QoE criteria, stays unknown to the Q-learning agent.

Therefore this work approaches the task of optimal adaptive streaming with the use of supervised learning techniques to learn from the optimal adaptation and maximizing the user perceived QoE.

2.3 Quality of Experience

The Quality of Experience (QoE) of a multimedia system is a subjective model for evaluating the users experience while watching multimedia contents. Emerged from the communications systems Quality of Service (QoS) that mainly targets the network or application centric service fulfillment's, the QoE gained more and more traction in recent years because it is of high interest for the content providers to adapt their services to the needs of its users.

The concepts of QoE were first published in 2013 in a white paper by [BBDM+13]. The QoE model has since then expanded in a more and more broad field of multimedia systems. Most recently the QoE models are refined to meet the needs of adaptive streaming as proposed in [SES+15].

The impact of different occurrences in an adaptive streaming scenario, e.g. quality switching or playout stalling are evaluated and brought into the context of the highly subjective QoE as in [HSSZ14]. Figure 2.2 gives an overview on all the influence factors of QoE in adaptive streaming.

In the further approach in this work the two subbranches of the perceptual influences namely *Waiting Times* and *Video Adaptation* as well as the subbranch *Client Side* and *Adaptation Logic* of the technical influences are used to meet the needs for a high QoE.

Considered very important in the perceptual influences is that the playout of the video should never stall and the initial delay it takes from first segment request to playout is kept very short. Also important is the impact of the average playout quality and the number of quality switches on the QoE. A fast quality switching video playback will lead to an unsatisfying user experience as a low quality video playback would, too.

With the knowledge about the impairments of perceived QoE there have been several studies and publications on how to optimize an adaptive streaming scenario with regards to a QoE model. This optimization formulations lead to the term of Optimal Adaptive Streaming.

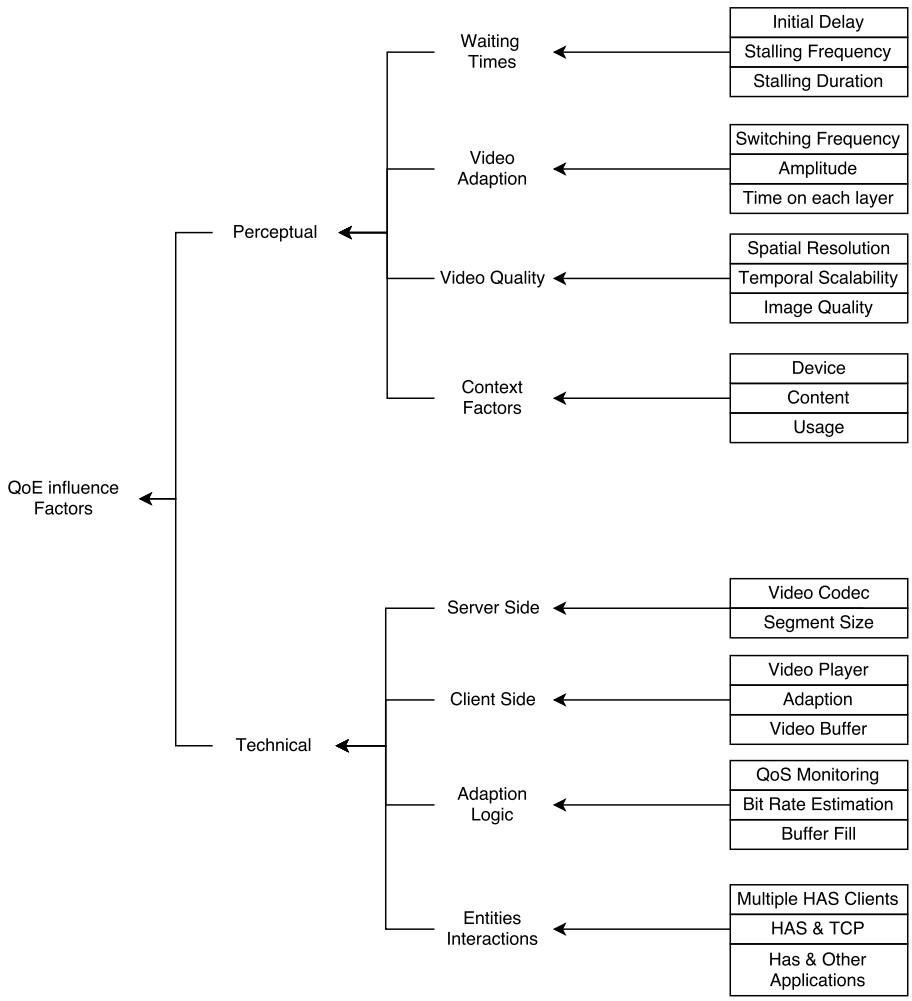


Figure 2.2: An overview of the influence factors on the Quality of Experience by [SES+15].

2.4 Optimal Adaptive Streaming

Optimal Adaptive Streaming is meant to be the optimization of an adaptive streaming scenario regarding certain criteria as an underlying QoE model. Therefore an optimization problem formulation has to be solved and the existing algorithms have to be compared to this optimal solution to be further improved.

There are several objectives that have been studied with optimal adaptive streaming. While in the past there was an interest on maximizing the utilization of the available goodput for example of a mobile connection as in [DVVB+13]. Other more recent attempts are not only utilizing the available connection resource but also considering a QoE model that regards the average quality, the switching behaviour and the initial delay [HSS+15]. It could also be shown that even major HTTP adaptive video streaming service providers like YouTube do not fully utilize its resources while also leaving a chance of improvement to the QoE as

shown in [MSH+16].

In this work the focus lies on the optimization of the QoE with regards to average playout quality and quality switching frequency, while utilizing the available resources, i.e. the goodput traffic pattern.

As one of the corner stones of this writing the trade off between the quality switching behaviour and the average playout quality is studied in while optimally utilizing the connection and avoid stalling of the play back at all cost. This evaluation is done in the Chapter 3 "Methodology".

As a result from such an optimization of an adaptive streaming scenario one receives the so called optimal adaptation path. This optimal adaptation path is the optimal choice of a segment quality representation for a given point in time. An example for an optimal adaptation path can be seen in Figure 2.3.

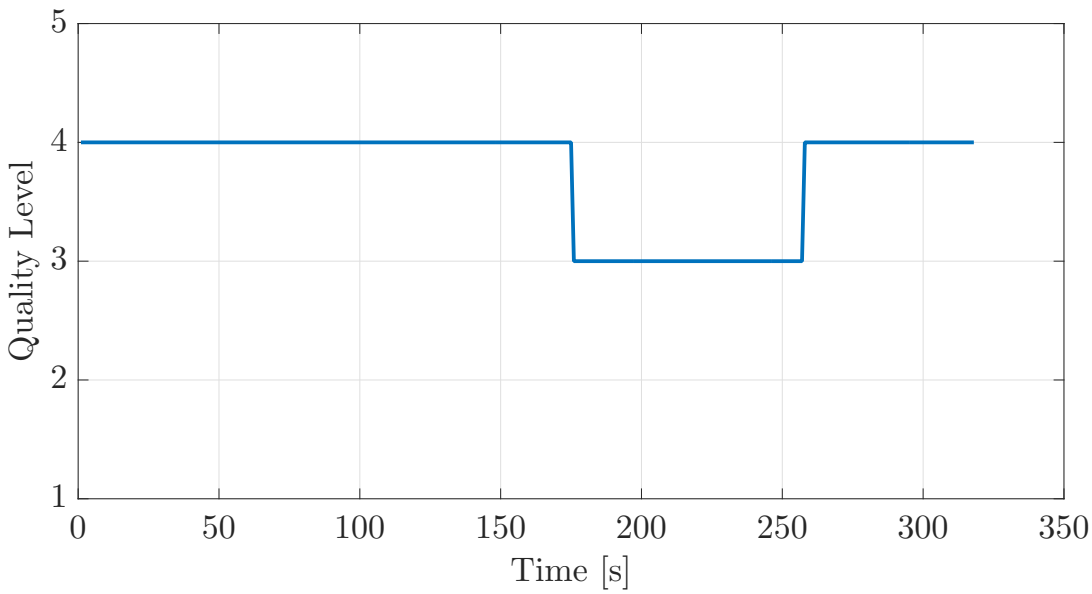


Figure 2.3: Example of an optimal adaptation path from a HAS scenario

The optimal choice of quality level is shown to be the quality level 4 from the beginning of the playout to 175s where it is then necessary to switch the quality down to the quality level of 3 until 255s and then allowing to switch back to the higher quality level 4. Would a clients adaptation logic follow this exact adaptation path then its QoE as well as its connection resource utilization would be maximized. The existence of such an optimal adaptation path in combination with the recent ubiquitous use of Machine Learning techniques where algorithms are trained to learn from a given set of input data to build a model of the underlying dependencies of the data, was a major contributor for the opportunity of combining those two.

With the choice of a broad field of available Machine Learning algorithms the most suitable have to be found that would allow to train from the optimal adaptation path and adopt this optimal behaviour in a streaming scenario.

2.5 Machine Learning Techniques

In the field of Machine Learning (ML) techniques one has to differentiate between three major groups of learning approaches:

- Unsupervised learning
- Supervised learning
- Reinforcement learning

In unsupervised learning the task of the algorithm is to find hidden structures and pattern in the given input data for example through grouping the input samples into clusters. These can be useful on discovering new relations in the structure of certain input data. This ML approach was discarded in this work as the adaptation path is already the clustered or labeled data that would otherwise be the solution of such an technique.

In Supervised learning on the other hand a algorithm is trained to map certain input features, i.e. specifically chosen input variables, to labeled output data. These kind of algorithms are used for the task of classification. Meaning to classify a set of input variables into a certain output class. The algorithms that have been used in this work for classification purposes in supervised learning are:

- Artificial Neural Networks (ANN) or Neural Networks (NN)
- Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel
- k-Nearest Neighbors

Last, the reinforcement learning approach introduced in [SB98] is a technique where an algorithm is given a certain high level goal, as in optimal adaptive streaming it could be for example be fulfillment of an optimal QoE, and is left to discover its own interactions with the environment and its impact on the given goal. One very prominent representative of this technique is named Q-learning and was brought up by [WD92].

The algorithm learns through its actions by receiving positive rewards if the actions taken do help to fulfill the high level goal or penalties if they don't. In a streaming scenario a good choice of a representation level while utilizing the available throughput will lead to a positive reward and therefore strengthen the algorithms behaviour towards such actions. On the other side a stalling occurrence through wrong behaviour will lead to a negative feedback of the algorithm and will prevent it from reoccurring such actions.

Regarding the task on implementing optimal adaptive streaming with machine learning techniques, supervised learning is chosen to be the most promising as the optimal adaptation path can be seen as labeled output data where each quality choice in the streaming scenario is a mapping of an input situation to an optimal choice of quality representation. Therefore the focus in this work is laid upon supervised learning techniques.

2.5.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is up to this time the state of the art machine learning algorithm for classification. Through discovery of the efficient back-propagation algorithm of error weight calculation their use has been increased significantly over the course of the last 5 to 10 years.

The general principle of Neural Network Classification (NNC) is to find for a certain input space, the so called input features, the non-linear relation to an limited output space, named the classes or labels. In this work the Neural Networks where solely used for the task of supervised learning classification and therefore the introduction will focus on this topic only.

To begin training a neural network a set of input features has to be created. These features represent the input variables that have to be mapped onto the different classes. For every set of input features there is a definite correct class. These corresponding input and output set is called the training set and is used to train the Neural Network (NN).

The next step is to determine the size and shape of an ANN. For this purpose many different design choices can be made. These choices are picked up later in this work in the chapter 5.2 "Machine Learning Model Definitions".

The ANN consists of layers of artificial neurons where each layer is interconnected the next following layer. A ANN has an input layer with the same number of neurons as there are input features. Followed by at least one hidden layer with variable size. Last, the output layer consist of as many neurons as there are classes. On the interconnections from neuron to neuron are the so called weights. These weights are in beginning randomly initialized to have an arbitrary value. An example for a simple neural network is given by figure 2.4.

Every neuron has a bias that has always the value 1 but through a variable weight this value can be changed.

The training of the ANN works now as follows. A set of input features, that are numerical values, is multiplied with the weights of the input layer and fed into the first layer of neurons. The neuron itself is a mathematical function that maps the input values to values in a certain output range and is named activation function. A variety of different activation functions can be used to choose from. To the most common used activation functions count the logistic function, the hyperbolic tangent or TanH and the rectified

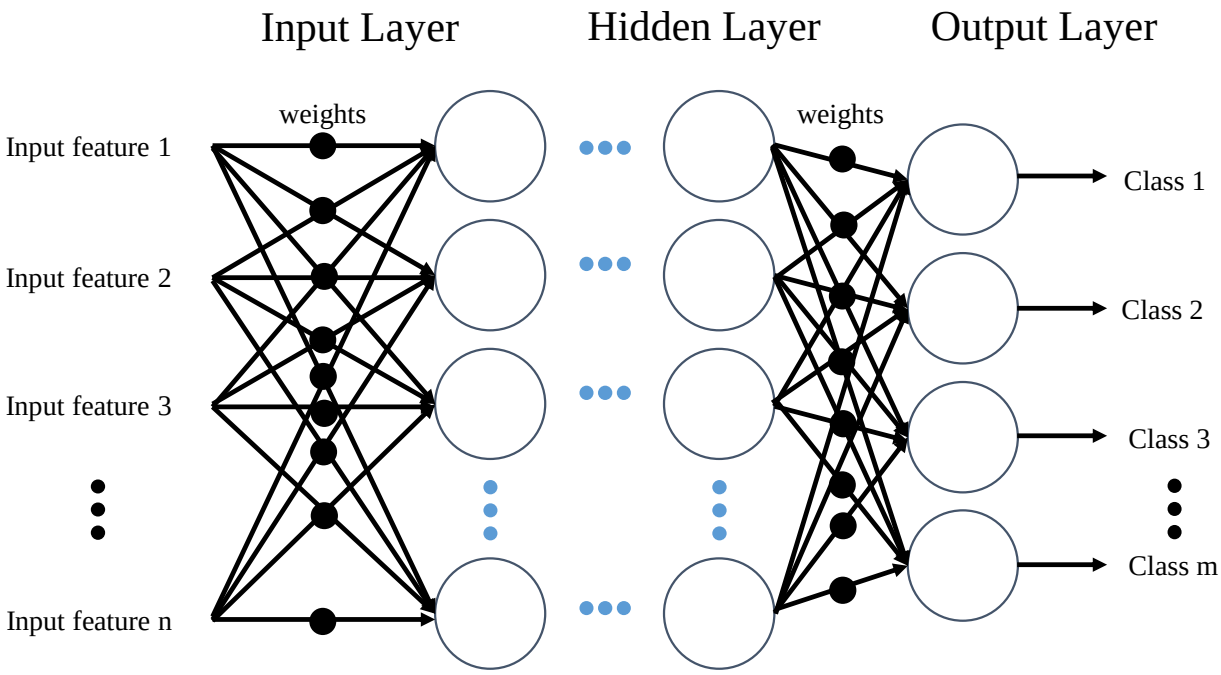


Figure 2.4: Example of an Artificial Neural Network structure

Function Name	Equation	Range
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
TanH	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$(-1, 1)$
ReLU	$f(x) = \frac{x}{1+\lvert x\rvert}$	$[0, \infty)$

Table 2.2: The activation functions

linear unit or ReLU. The mapping ranges and function definitions of these functions can be seen in Table 2.5.1.

After this mapping the values pass the next stage of weights and neurons until the last layer is reached. In the last layer the class is determined by the neuron which has the highest output value. Because the definite correct label is already given in supervised learning a deviation from the correct class to the chosen class can be calculated. With the use of the back-propagation algorithm the weights of the individual artificial neurons is now adapted in a way that with this input set the correct label will be classified.

After repeating this process of classification and error back-propagation long enough over a necessarily huge set of training samples then the neural network will eventually reach a state of maximum accuracy. Accuracy is the measure of correct classifications. The accuracy should not reach a value of 100% because this can cause overfitting. Overfitting is then the case when a classifier has not learned the hidden relations of a data through

generalization but rather just memorized the input to output values for each training sample. An overfitted ANN is therefore not able to classify correctly on new input data that was previously unknown to it. If the ANN was trained correctly it can then be tested on a new set of input features and the correctness of each classification is determined. These new input set is called the validation set and is used to show that the neural network has learned to generalize from the training set. Is the accuracy of the validation set sufficient, optimally as high or even higher as in the training set, then the neural network can be used for its further purpose.

This short and simplified introduction should give the reader a slight understanding of the topic of neural network classification while it should not be regarded as complete by no means. Any interested reader is recommended to see [HN04].

2.5.2 Support Vector Machines

The second used ML classification algorithm is the Support Vector Machine (SVM). This ML algorithm defines every input sample with n features as a point in a n-dimensional space and finds a linear hyperplane that separates the individual points of one class to another by the largest margin, therefore it is also called a large margin classifier. For the simple example of a 2 dimensional input the hyperplane is a linear function as can be seen in figure 2.5.

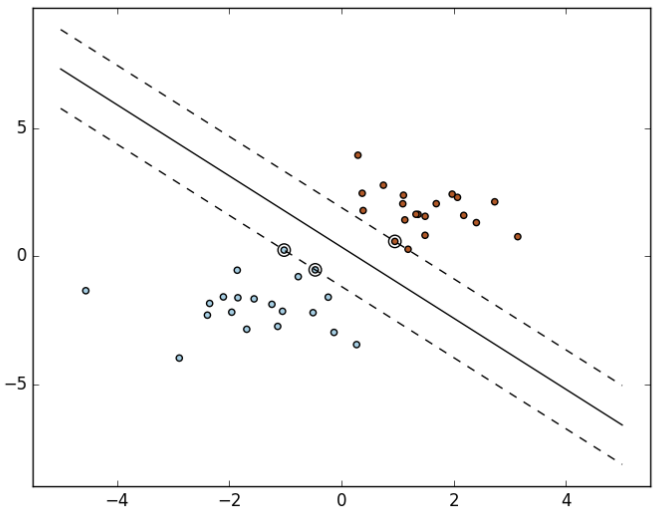


Figure 2.5: Example of a simple SVM linear classification [SVM]

Similarly to the neural network it is necessary to train with a large set of input and output values. While the learning of lower dimensional input sets is quite fast regarding

the training time, the duration it takes to train a SVM gets exceeding long for higher dimensional inputs especially compared the learning time of a neural network while not reaching the same accuracy. This is also the main reason in this work was laid the focus upon ANNs.

For a more in depth introduction to SVMs for classification see [HCL+03].

2.5.3 k-Nearest Neighbours

The last used ML algorithm is named k-Nearest Neighbours and can, as the preceding introduced algorithms, also be used for supervised learning classification.

In figure 2.6 one can see an example for a k-Nearest Neighbours classification with k = 15 and 3 classes.

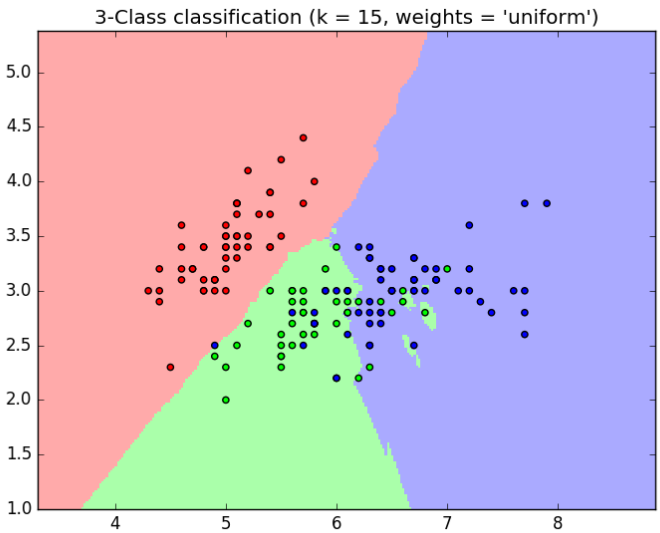


Figure 2.6: Example of an k-Nearest Neighbours classification [kNe]

Each individual class is pictured as a colour and each point corresponds to a 2-dimensional input, i.e. a sample with 2 features.

The algorithm uses pairs of input features and output labels as common to all supervised learning algorithms. This ML algorithm does not adapt any weights or hyperplanes to separate the set of input values with their corresponding classes from one another. While training, the set of input features to corresponding output features is saved in memory. Should after learning a new input be classified the following is done.

As with the SVM a input sample with n features is considered a point in a n-dimensional space. So each input sample has an individual point in this space and an corresponding

class. A sample that has to be classified is then set into this n-dimensional space according to its features and the distance to all other samples in memory is calculated. For the distance calculation most commonly the euclidean distance is used. So after calculating the distance to the all other samples only the k-Nearest Neighbours are used for the further classification. It is just counted which class appears most often in the for the k-Nearest Neighbours and this class is then chosen for the new input sample. If a draw occurs then the class is chosen randomly from the classes appearing the most. This algorithm can also be modified by adding a weight to the distance calculation. This weight can be used to for example devalue neighbours at a further distance, so that nearer samples are considered more important.

Due to the fact that all training samples are saved into memory one has to consider that a k-Nearest Neighbours model will be disproportionately huge in memory usage. Further information about this ML technique can be found in [CH67].

Chapter 3

Methodology

In this chapter the methodology and nomenclature of the work are explained. In the first section the nomenclature of the optimal adaptation path as well as the one from the ML classification is defined. In the next section an overview of the system model, also termed **HASBRAIN** algorithm, is given and explained step by step. Figure 3.1 shows an overview on the algorithms system model and methodology.

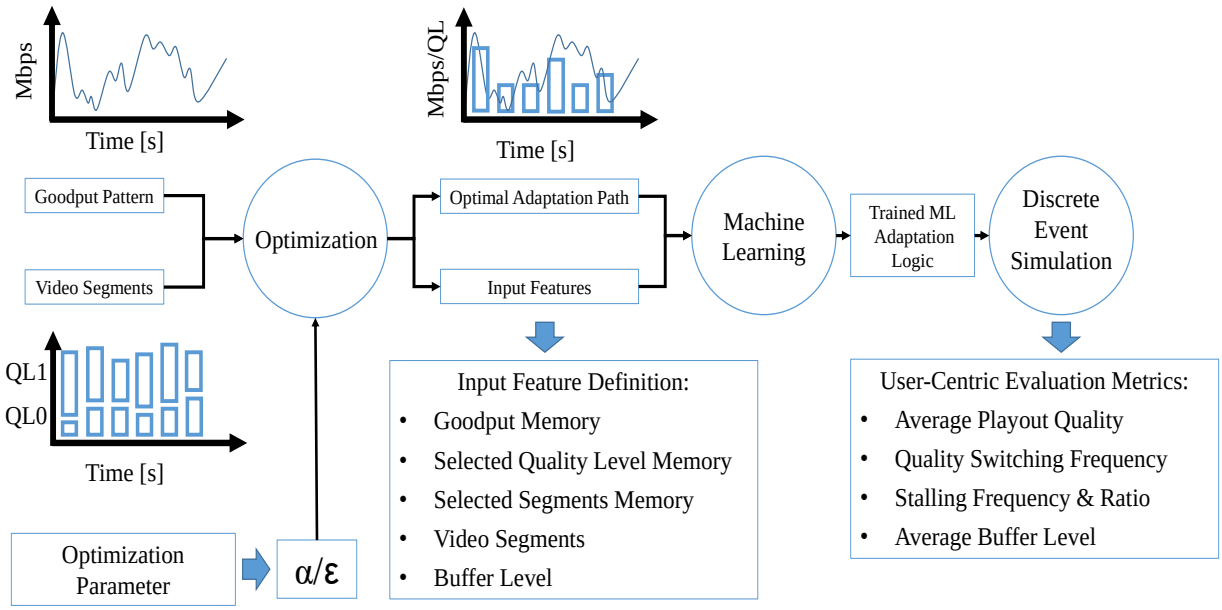


Figure 3.1: The system model or **HASBRAIN** algorithm

With a given input goodput pattern and video segments the optimal adaptation path is generated. The optimization itself is dependent on the optimization parameter α or ϵ. With the optimal adaptation path a set of input features are created from this streaming scenario. These features represent the metrics that a common adaptation algorithm would

use when deciding the selection of the next quality level. The memory features are the past recordings of certain metrics in a defined time span, e.g. the recorded goodput of the last 30s. The optimal adaptation path as well as the created features are then used to train a supervised ML technique. The trained ML algorithm is used in a Discrete Event Simulation (DES) as adaptation logic in a real streaming scenario and user-centric evaluation metrics are obtained. These metrics allow then to evaluate the overall adaptation performance.

Following, the derivation of the optimal adaptation path is explained in detail. The influence of an optimization parameter α on the trade-off between average playout quality and quality level switching is evaluated. Thereafter an introduction to the DES is given that is used for the concluding evaluation of the performance of the ML algorithms and the threshold-based algorithms in a realistic adaptive streaming scenario. The performance metrics used for the evaluation are also explained in the this section. For this evaluation a new set of validation videos is defined and the characteristics of the validation video set are described. Concluding with the description of the resulting evaluation scenarios as combination of goodput patterns evaluation videos.

3.1 Nomenclature

In Table 3.1 the terminology of the optimization process is defined. Variables with a default value are set into brackets. If not mentioned otherwise these default values were used throughout this work.

Variable	Definition
r [5]	Available representations
n [200]	Number of segments
τ [1]	Duration of a segment in [s]
S_{ij}	Size of a segment i from representation j
D_i	Playback deadline for segment i
T_0 [5]	Start-up (or initial) delay in [s]
$V(t)$	Total amount of data V(t) received by a client during the time [0,t]
$T(v)$	Time $T(v)$ required by a client to download volume v. $T(v)$ is the inverse function of $V(t)$, i.e. $T(V(t)) = t$
$x_{ij} \in \{0, 1\}$	Target variable indicating if client downloads segment i from representation j ($x_{ij} = 1$) or not ($x_{ij} = 0$)

Table 3.1: Optimization variable definition

In an adaptive streaming scenario a video consists of n segments where each segment is available in r quality layers or levels. Every segment i that is downloaded is in one of the mentioned r quality levels. To prevent stalling in the video playout, the segments i must be downloaded and therefore be available for playout before the deadline D_i. The data volume at a point in time t that is available since the initial point in time $t = 0$, the video request, is defined as $V(t)$.

As stated as default value, the startup delay, i.e. the time before the playout starts, is fixed to 5 s.

A given segment size of segment i on the quality layer j is denoted by S_{ij} in the unit of Bytes. A switch between qualities is defined as a change between the layers of two consecutive segments.

Is the segment i downloaded on a certain quality layer j, then it is defined as $x_{ij} := 1$, if not then $x_{ij} := 0$.

The same variable definition are used in the second part of the work, the ML. Furthermore some additional variable definitions are needed and therefore listed in table 3.2.

Variable	Explanation
t_{d_i}	Current time at point of download decision for segment i
$Tp_{d_i} = \frac{S_{i-1}}{t_{d_i} - t_{d_{i-1}}}$	Goodput at time t_{d_i} calculated by the download of the proceeding segment
Bl_{d_i}	Buffer level at time t_{d_i}

Table 3.2: Machine Learning variable definition

The time t_{d_i} is defined as the current time in the adaptive streaming scenario when the decision logic decides the next quality representation of the next segment and the Player thereafter schedules the download. For example t_{d-1} would be the time instance at the point of decision from the proceeding segment. Tp_{d_i} is the current throughput or goodput measured immediately before the time t_{d_i}. Last, Bl_{d_i} corresponds to the buffer level also at the download decision time of segment i, t_{d_i}.

3.2 HASBRAIN Algorithm Description

The system model or **HASBRAIN** algorithm is presented step by step in the beginning of this chapter in figure 3.1. The two main parts of this work can be separated by step 1, the optimization, and step 2 and 3 the Machine Learning part as well as the Evaluation through the DES.

In step 1 a beforehand defined adaptive streaming scenario is optimized through solving a specific optimization problem formulation for the optimal quality picking strategy at every decision point in time, the optimal adaptation path. The optimally criteria can be varied by means of QoE demands through the optimization parameter α and ϵ and is furthermore always maximizing the optimal goodput usage for streaming a video while not allowing stalling to occur. As one main contribution of this works methodology a new optimization problem formulation with a parameter α that allows for a trade off between average playback quality and quality switching is in its entirety evaluated.

Thereafter, the optimal adaptation path and input features which were created from the adaptive streaming scenario are used for the second step. Each input feature for the Machine Learning algorithms corresponds to the state information of the output player right before the decision of the next quality representation, i.e. the information can for example consist of memory of the past goodput pattern, the memory of selected quality levels and segments, the current buffer level and also information on the video segment sizes to come.

In step 2, the ML, the created input features and the optimal adaptation path are used to train a ML algorithm through supervised learning. By repeatedly iterating over the set of training values the ML algorithm learns the hidden structure and relations from the given data. The learning success is continuously validated by a set of input and output values previously unknown to the ML algorithm to monitor the learning success.

With a sufficiently trained ML algorithm, the last step is then to evaluate its performance in a real adaptive streaming scenario. This is done by Discrete Event Simulation of such a scenario and gathering of corresponding performance metrics, for example the average playout quality or the average buffer level. With these metrics a conclusion for the performance of a supervised trained ML algorithm for optimal adaptive streaming can be drawn.

3.3 Optimal Adaptation Path

In this section the first step of the **HASBRAIN** algorithm, the optimal adaptation path, is explained and evaluated. Parts of the results regarding the extension of the existing optimization problem formulation with a new weighting parameter α and its evaluation are available in [MHH+].

The modification of the optimization formulation allows to identify the trade-off between the average video quality and switches in the quality in an adaptive streaming scenario during the video playout. These findings are then used for the further steps in the work, the generation of training and validation samples for the ML algorithms.

The further organization of this chapter is as follows. Beginning with the optimization problem definition and the formulation of the optimization formulation. Afterwards the α parameters influence on the trade of between quality switching and average quality of the played out video is evaluated. Followed by the conclusions that can be drawn from the impact of this α parameter influence study. Last, the transition from the optimal adaptation path with creation of the corresponding features to training and validation samples used for the ML training is explained. Therefore another optimization problem formulation is presented that resulted from the findings of the proceeding formulation.

3.3.1 Optimization Problem Definition

The optimization formulation can be found in [MHH+] and is explained as follows.

To increase and optimally maximize the QoE of the user while streaming, it is necessary to increase the mean quality while reducing the number of switches in the playout quality. To weigh the importance of the two main QoE the parameter $\alpha \in [0, 1]$ is defined. While higher α values indicate that it is better to avoid vivid quality switching than to increase the average quality. Correspondingly the lower values of α weigh the optimization to increase the average quality and to allow more quality switches. The point is to adapt the optimization to users with different preferences of the QoE while video streaming.

The general goal of the optimization is now to optimize the weighted sum of mean quality and the number of quality switches while avoid stalling through optimally utilizing the available goodput volume. In other words, it has to be calculated on which quality level each segment i has to be downloaded to achieve this optimal result. As defined in the paper the formulation of the optimization is a quadratic program formulated as follows.

$$\text{maximize} \quad \sum_{j=1}^{r} \left(\frac{\alpha}{nr} \sum_{i=1}^{n} j x_{ij} - \frac{1-\alpha}{2(n-1)} \sum_{i=1}^{n-1} (x_{ij} - x_{i+1,j})^2 \right) \tag{3.1}$$

$$\text{subject to} \quad \sum_{j=1}^{r} x_{ij} = 1 \;\; \forall i \in \{1, \ldots, n\} \tag{3.2}$$

$$x_{ij} \in \{0, 1\} \;\; \forall i \in \{1, \ldots, n\}, \; j \in \{1, \ldots, r\} \tag{3.3}$$

$$\sum_{i=1}^{k} \sum_{j=1}^{r} S_{ij} x_{ij} \leq V(D_k) \;\; \forall k \in \{1, \ldots, n\} \tag{3.4}$$

Equation 3.1 is the objective function that is maximizing the with α and $1 - \alpha$ weighted sum of the mean quality layer and the number of quality switches. To obtain values in range of 0 to 1 the two subterms are normalized. The mean quality is normalized by the number of segments n times the number of quality layers r. The switches are normalized by the maximum number of possible switches $n - 1$ with an additional factor $\frac{1}{2}$ because the quadratic term increases by 2 with every switch.

The two constraints 3.2 and 3.3 make sure that a segment is only downloaded in exactly one quality level. While 3.4 ensures that every segment k is downloaded before the corresponding deadline D_k and the downloaded data does not exceed the available data $V(D_k)$.

3.3.2 Evaluation Scenario Definition

To evaluate a realistic and challenging adaptive streaming scenario a mobile goodput trace was used which has been recorded while travelling by car through a city.

While the videos, which consists of the individual segments, are selected YouTube videos that have been collected with certain properties, i.e. their playout time, number of quality layers and popularity among its category.

The next short sections discuss the properties of the mobile goodput trace in detail, followed by the properties of the selected videos.

The Goodput Pattern

The trace represents the downlink throughput of a mobile client's network connection on the application-layer. Originally this trace can be found in the supplements of the work by [MLT12]. It was collected by driving in a car around the city of Klagenfurt in Austria.

To collect the trace, a HTTP Get request was sent via a UMTS stick and the goodput of the download was continuously recorded in a time span of 750 s. To allow a more diverse evaluation of the adaptive streaming scenario, the trace is re-scaled to three goodput traces with three different means. 0.33 Mbps, 0.67 Mbps and 1.34 Mbps are the resulting average goodputs of the rescaled traces. While re-scaling the coefficient of variation (CV) was kept the same at 0.38.

These three patterns are further referred to as *low*, *medium* and *high* and have an autocorrelation of 0.80 for a lag of 1. In addition to the re-scaling, the three traces are shifted with seven different time shifts. This circular shift moves the starting point of a trace to a certain shifted point of the trace and appends the skipped goodput trace to the end. The time shifts are $\{0\,\text{s}, 120\,\text{s}, ..., 720\,\text{s}\}$ and are used to further enhance the number of available versions of the goodput pattern, i.e. a total of $7 \cdot 3 = 21$ goodput traces are used for the evaluation. These three patterns are depicted in Figure 3.2.

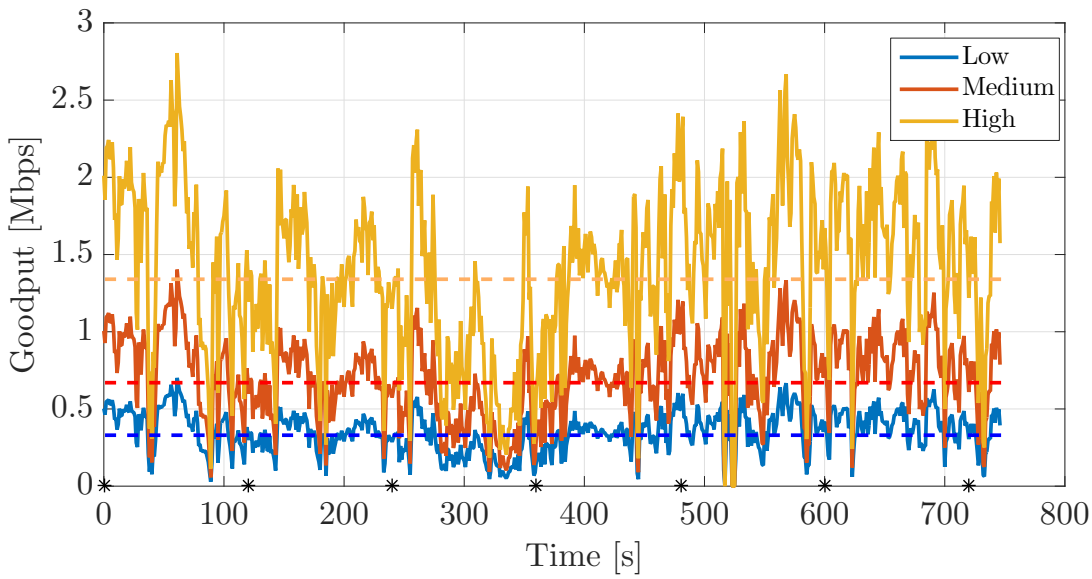


Figure 3.2: The three goodput patterns used for the evaluation, low, medium and high.

A dashed horizontal line indicates the mean of each trace. Whereas the stars on the horizontal axis indicate the starting point of the individual time shifts, in summary seven stars.

The Figure allows to draw conclusions on the challenges of this vehicular mobile pattern, because as one can see is that the trace experiences frequent drops to below 0.1 Mbps. Furthermore, at around 510 s the throughput falls even to zero. There is not a time span longer than 60 s where no significant drops appear. The corresponding maxima of each trace *low*, *medium* and *high* are 0.67 Mbps, Mbps 1.34 and Mbps 2.68. These modified traces can be found in the supplemental material to this work or here [sup].

The Video Characteristics

In this work a total of 41 different YouTube videos was used. These videos were randomly chosen by popularity and represent a variation of different contents from different categories. On average these videos have a duration of 5.3 minutes and range in length from 1 up to 10 minutes. The videos exist in five quality levels, namely $\{144p, 240p, 360p, 480p, 720p\}$. The mean bitrates range from 0.1 Mbps for $144p$ up to 1.3 Mbps for the highest quality level of $720p$. To have consistent segment sizes, the videos have been split up into segments of 5 seconds length and are then used as input for the optimization. Table 3.3 summarizes the important video characteristics.

For further details about the videos and the selection process of each video, refer to [SHHK16].

Avg. bit-rates	140p	240p	360p	480p	720p
Min (Mbps)	0.08	0.16	0.04	0.08	0.14
Mean (Mbps)	0.10	0.23	0.36	0.68	1.33
Max (Mbps)	0.11	0.24	0.56	1.05	2.08
Std	0.00	0.01	0.13	0.24	0.48
Segmentsize	5s				
Duration	{1, 2, ..., 10} minutes, avg 5.3 m				

Table 3.3: Video characteristics

As an example for an used test video Figure 3.3 shows the 5 quality levels bit-rates throughout the video named *CRZbG73SX3s*.

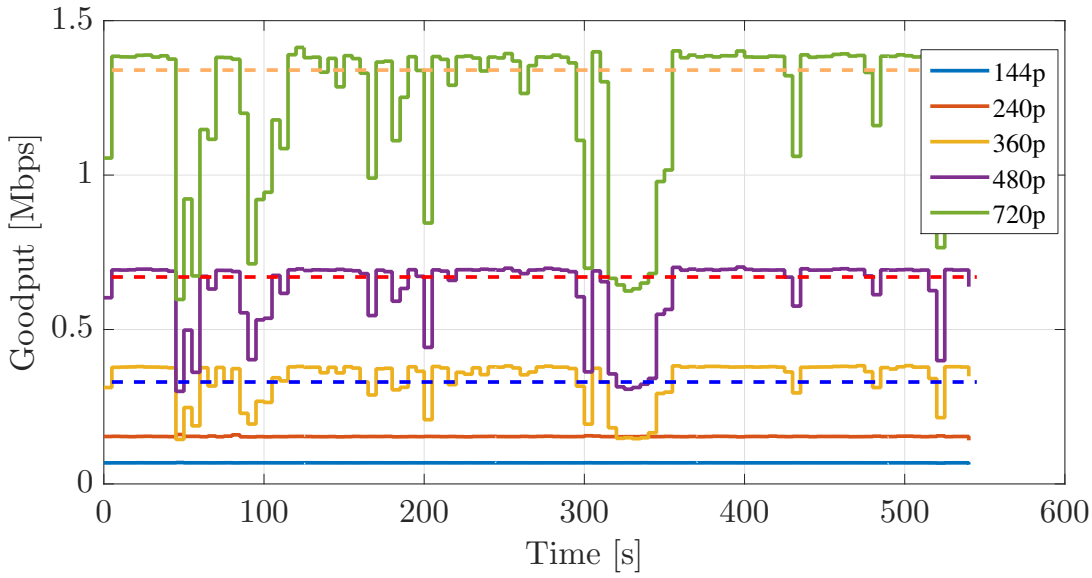


Figure 3.3: Example of the video segments for video CRZbG73SX3s and the quality levels $\{144p, 240p, 360p, 480p, 720p\}$.

To get an understanding of the proportions of the goodput patterns, the average of the three traces are shown as dashed horizontal lines. It is to notice that each quality layer is limited by the maximum bit-rate allowed by the decoder. Also the drops of the bit-rate of the three higher quality layers are due to more efficient encoding of low detail and low

motion parts of the video. The generated video segment sizes from the videos are available in the supplemental material of this work [sup].

3.3.3 Alpha Parameter Influence Evaluation

The focus on the evaluation as the section title suggests is set on the influence of the parameter α to the quality switching behaviour, the average playback quality and also the average buffer level.

In the following evaluation these assumptions hold true. The video player does not halt during the segment downloads, as well as exhibits an unlimited buffer. If these criteria are fulfilled then the following can be observed.

A high average quality is achieved if the quality switching behaviour is very aggressive, but this results in a overall low average buffer level and a high number of quality switches. A more conservative switching behaviour is found to not only decrease the average quality but also to decrease the number of quality switches and increase the average buffer level.

Therefore, when considering the objective of maximizing the QoE, the challenge to increase the average quality, to avoid unnecessary switching and avoid stalling in the playout at any means, i.e. maintaining a considerate buffer level, are mutual exclusive.

To overcome this problem a trade off between an acceptable average quality and an acceptable amount of quality switches has to be found.

To understand the influence of the α parameter on this behaviour, observations are first analyzed on an example video and then over all videos.

Results for an example video

For further analyzation purposes the before mentioned video CRZbG73SX3s is used. In Figure 3.4 the adaptation path for three different α parameter values are shown. The used traffic pattern was the medium traffic pattern with a starting time shift of 0 s.

As one can see a α parameter value of zero leads to a very conservative optimal adaptation path, because there are no quality switches and the quality level of 3 is chosen in the beginning and kept until the end.

If $\alpha = 0.5$, then the number of quality switches increases to one switch and also the average playback quality increases to 3.29. This is due to the switching in three periods to the next higher quality level.

When α is further increased to the maximum of 1 then the adaptation path shows a very aggressive behaviour. A total of 30 quality switches are observed but also a single period of reaching the quality level 5 can be seen.

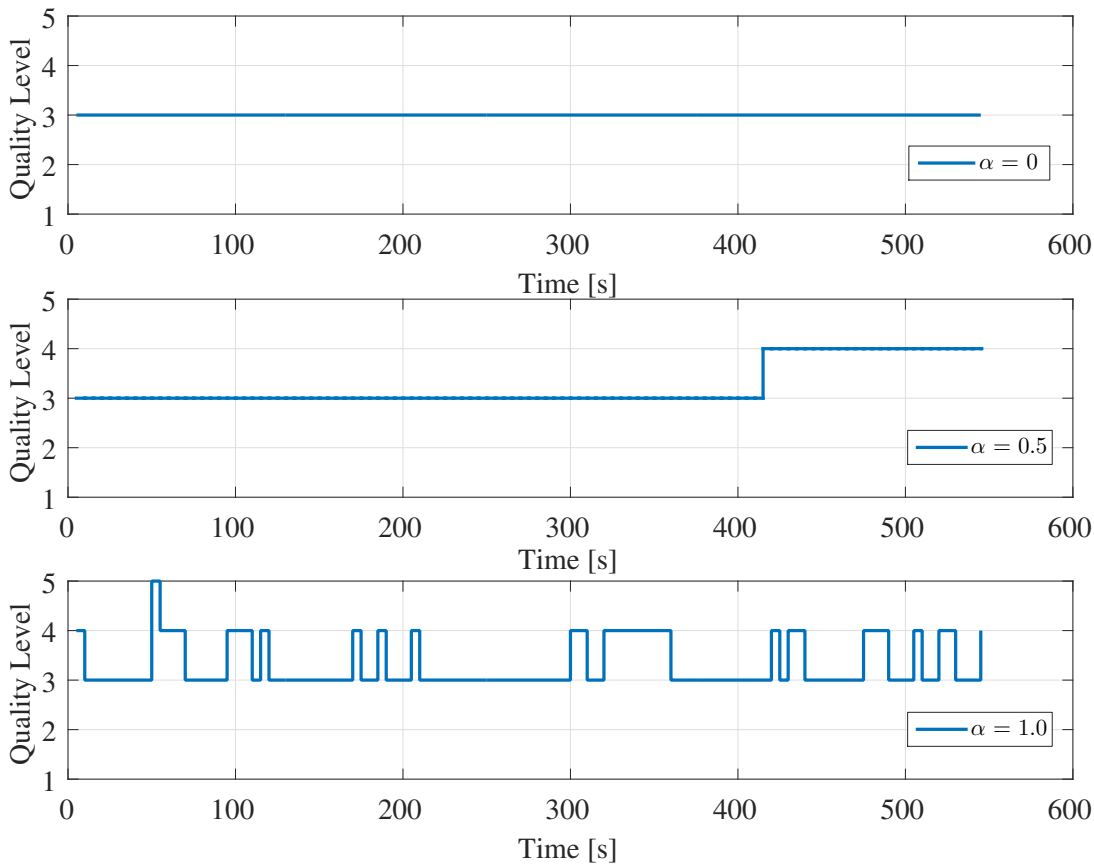


Figure 3.4: Adaptation path for three different values of α for the example video CRZbG73SX3s. The average playback quality from top to bottom are 3.0, 3.29 and 3.30.

But, if one compares the average quality increase from $\alpha = 0.5$ to $\alpha = 1$ the difference is only 0.01, whereas the switching occasions increase by 29. It is therefore found that a high increase in quality switches is not necessarily related to a high increase in the average quality.

To illustrate the relation between α and the buffer levels of the player, Figure 3.5 depicts the maximum and average buffer level in seconds for the example video CRZbG73SX3s. The maximum buffer level corresponds to the maximum occurred buffer level value in the playback of the video. The average buffer level is the average observed buffer level over the course of the playout.

As one can see, for high, i.e. aggressive, choices of α the buffer level continuously decreases with higher values. If the α parameter is low, e.g. $\alpha = 0.05$, an value of 55 s for the average buffer level and 110 s for the maximum buffer level are observed. While for high α parameter values, e.g. $\alpha = 1.0$, the value of the average buffer is just 18 s and for the maximum buffer level 40 s.

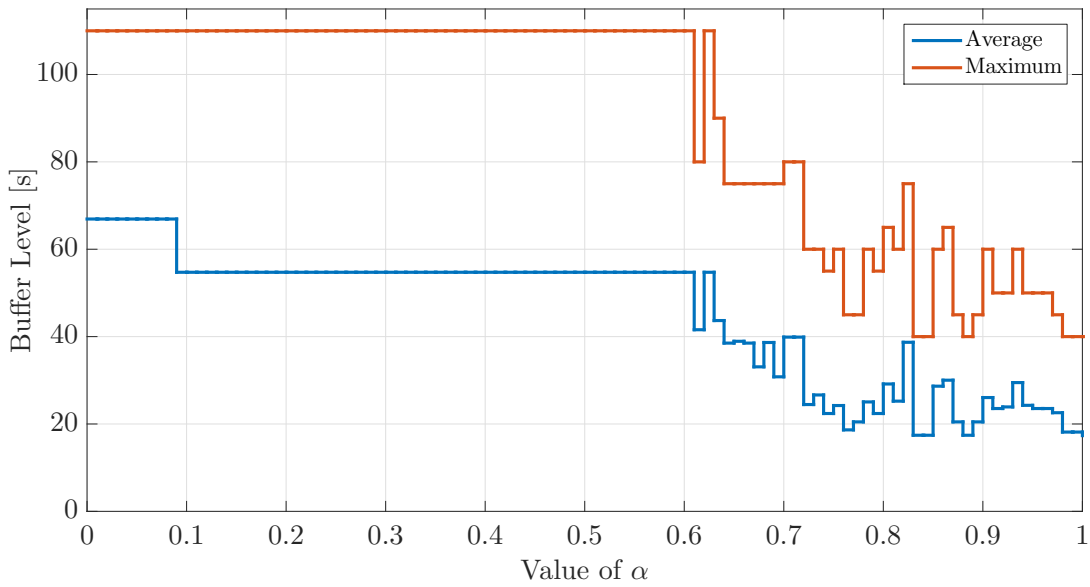


Figure 3.5: Average and maximum buffer level for video CRZbG73SX3s for the medium goodput pattern and 0s time shift. Higher values of α decrease the observed average buffer level compared to lower values.

Also to note is, that an average of 55 s for the average buffer level is achieved until even higher levels of α, i.e. $\alpha = 0.6$.

Due to the prevention of stalling occurrences from wrong quality adaptation decisions it is better to maintain a higher buffer level. Therefore an optimal adaptation logic can still maintain high buffer levels while preferring higher average quality.

Therefore the trade off between average quality and the switching frequency has to be evaluated.

In Figure 3.6 the average quality, i.e. the left axis, and the switching frequency, i.e. the right axis, are depicted over α parameter values ranging from 0 to 1.

Until α is below 0.1 the number of switches is exactly zero and the average quality stays at 3.0. Increasing α to values between 0.1 and 0.6 influences the switching frequency to an increase of 0.1 switches per minute, as well as increasing the average quality to 3.25.

A further increase of the α parameter to values between 0.6 and 0.90 leads to an increase in the switching frequency until 0.5 switches per minute, while on the other hand the average playback quality only increases by small amounts to 3.28. Further increasing the α parameter value leads to a very high increase in the switching frequency while the average quality only increases by a small margin to 3.29.

As it still holds true, that a aggressive, i.e. high, value of the α parameter which leads to a high switching frequency is not necessarily beneficial for an high increase in the average

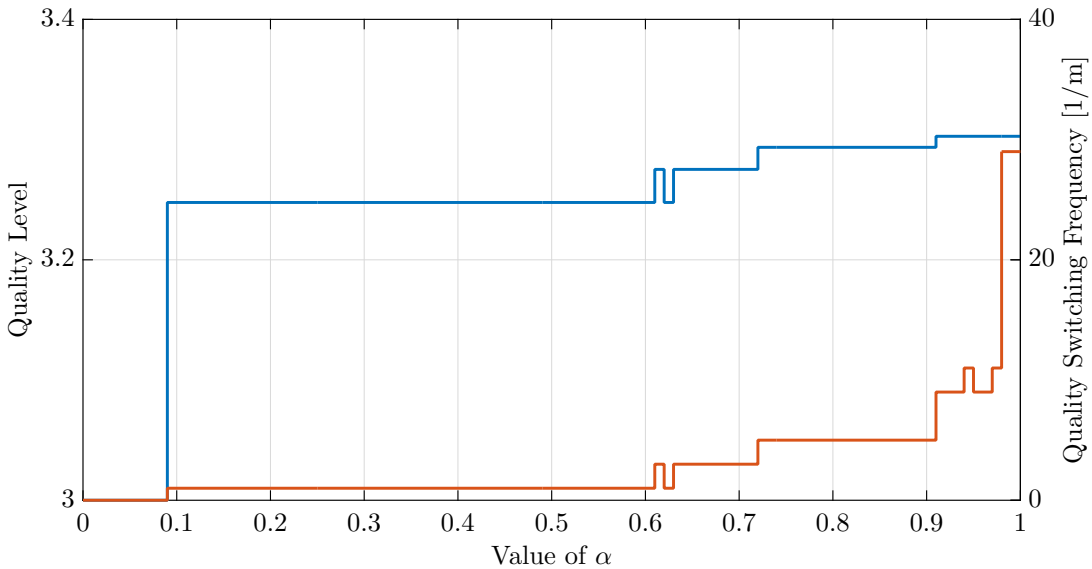


Figure 3.6: Average playback quality (blue) versus switching frequency (red) progression over α for video CRZbG73SX3s with the medium goodput pattern and 0s time shift.

quality. Keeping the QoE in mind, the user will also experience major deterioration of its viewing experience by many quality switches while not experiencing a huge quality difference compared to conservative quality picking strategies.

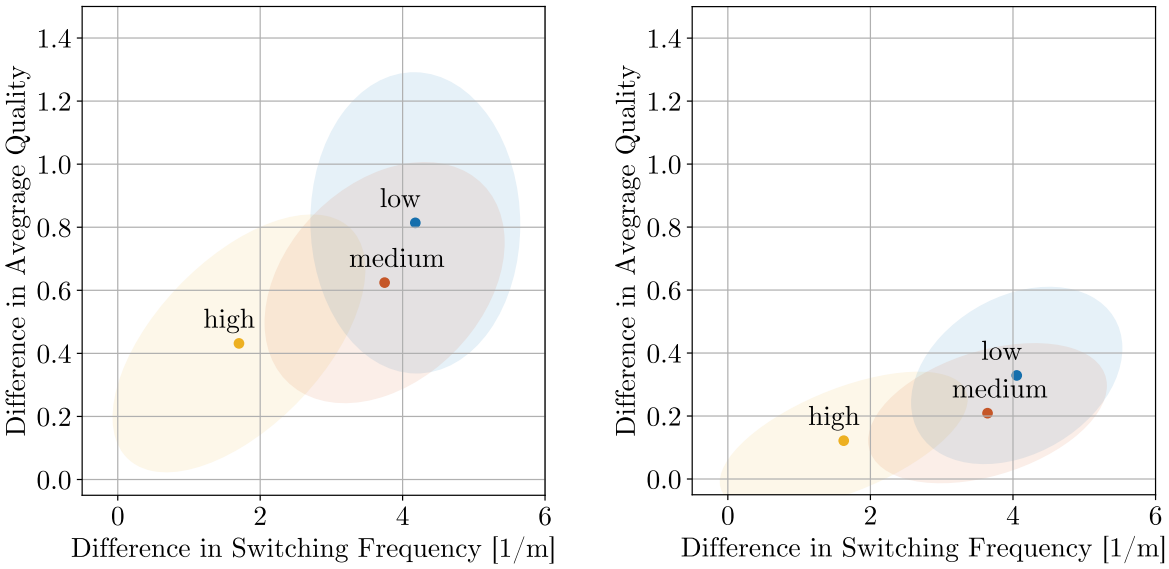
Results for all videos

When drawing conclusions on the observation over the whole set of videos one major answer has to be determined. When comparing a conservative to an aggressive switching strategy then how big is the maximal achievable gain in average quality.

In Figure 3.7 the difference in switching frequency and the difference in average quality are depicted for α parameter values of $\alpha = 0.01$ as in Figure 3.7(a) and $\alpha = 0.1$ in Figure 3.7(b) compared to a $\alpha = 1.0$. These plots are over the whole video test set and for the three traffic patterns *low*, *medium* and *high*.

The mean over a traffic pattern is represented as a dot, whereas the overall coloured area is representing the 2 dimensional standard deviation.

Now, several observations can be made. Beginning with the small observable gain in average quality from the high goodput pattern. As one can see in Figure 3.7(a) the high traffic pattern is on average at most times, considering the bit-rate, above the highest quality representations bit-rate. This means that for most videos the high traffic pattern is sufficient to download the whole video in the highest representation without any switching. The next observation that can be made is, the low goodput pattern reaches on average a



(a) $\alpha = 0.01$

(b) $\alpha = 0.1$

Figure 3.7: Difference in average quality and switching frequency between $\alpha = 1.0$ and $\alpha = \{0.01, 0.1\}$ for the three goodput patterns low, medium and high. The shaded areas denote the 2 dimensional standard deviation of the samples.

switching frequency of $4.2\,m^{-1}$ and a corresponding change of quality of 0.8 with $\alpha = 0.01$. Concluding that with a more aggressive switching behaviour, in a scenario with a low goodput, the average quality can be improved by 0.8 through switching every 14 s. At last, as one can see in Figure 3.7(b) with $\alpha = 0.1$ compared to $\alpha = 1.0$, the achievable gains in quality are decreasing to values of about 0.4. This suggests that from the user's perspective the gain with an aggressive switching behaviour is for higher values of α only marginal.

Concluding this evaluation. With slowly changing this α parameter from 0 to 1 it could be observed that the aggressiveness of the quality switching behaviour, meaning the increase of switching occurrences, increased. While higher values of α until $\alpha = 1.0$ result in an adaptation path that optimizes the average quality without regarding the accumulation of quality switches.

The main findings can therefore be drawn as the following. While high values of α guarantee an optimal maximum video quality it is not necessary to use such high values because the gain in average quality is diminishing compared to the loss of QoE due to more switches.

3.3.4 Towards Machine Learning Sample Generation

With the findings from the alpha parameter influence evaluation and a contribution of it to future work the last steps toward the generation training and validation samples for optimal adaptive streaming with ML are linked.

In Figure 3.8 it is illustrated that there is a suggested Pareto frontier for HAS adaptive streaming every algorithm is bound to. There has to be considered a trade off between maximum average quality and minimal number of quality switches.

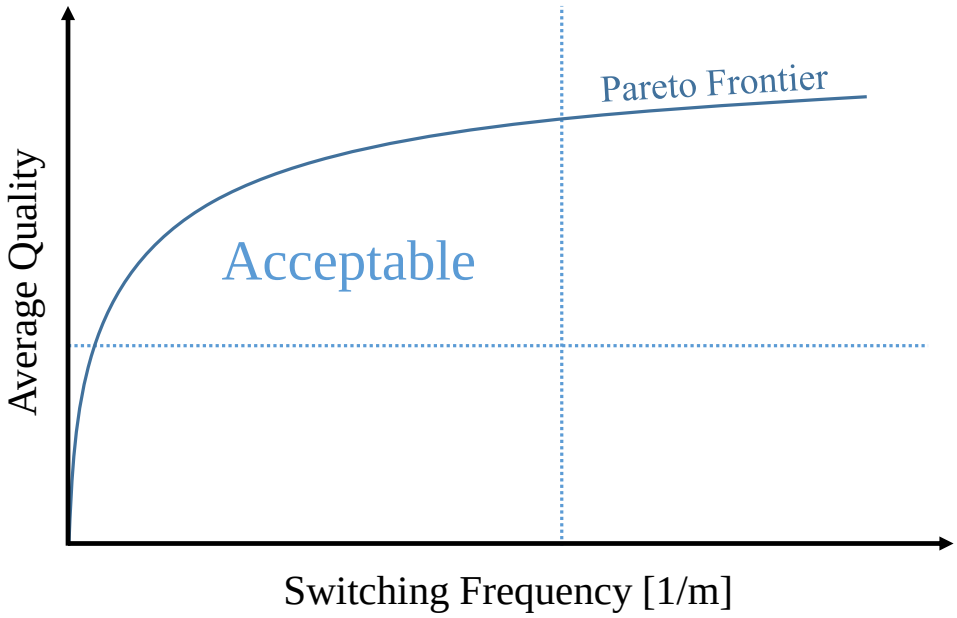


Figure 3.8: The Pareto frontier can be defined as the optimal trade-off between average quality and switching frequency.

No algorithm can surpass this frontier and if it is optimal then it has to be on the line of the graph, the Pareto Frontier. The definition of new QoE models that define an acceptable area of average quality and quality switching frequency is still up for future research.

These findings led to the consideration of a modified version of the optimization problem formulation for the further use in the application of the optimal adaptation path for ML.

With the approach from [HSS+15], that was basis for the modified optimization formulation, a two step optimization approach was suggested. This formulation maximizes the average playback quality with regard of the download-able data volume in a first optimization step and then uses this optimal solution as a constraint for a second optimization step where the number of switches are to be minimized.

This could also be seen as an optimization with $\alpha = 1$ in the first step and then a subsequent optimization with a $\alpha = 0$ but with an additional constraint that does not allow for lower

average quality results than the first optimization. The optimal adaptation path is then a solution that has the same maximum average quality but uses less switches to achieve it.

But as it was shown with the suggestion of the Pareto frontier and the area of acceptance for QoE optimal algorithms even with minimizing the switches after maximizing the average quality the QoE could still be in the non acceptable area because the number of switches is to high and reduces the QoE. With a small decrease of average quality there can be a huge decrease in the number of quality switches, as was shown in the previous results, and therefore a higher QoE can be achieved while still be optimal considering the Pareto frontier.

This small decrease in quality is set in the modified optimization formulation by the new parameter ϵ and defaults to 0.1 in allowed average quality degradation compared to the maximal achievable average quality.

The formulation of the optimization for the first step is then as follows.

$$\text{maximize} \quad W_{opt} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{r} j x_{ij} \tag{3.5}$$

$$\text{subject to} \quad \sum_{j=1}^{r} x_{ij} = 1 \;\; \forall i \in \{1,\dots,n\} \tag{3.6}$$

$$x_{ij} \in \{0,1\} \;\; \forall i \in \{1,\dots,n\},\; j \in \{1,\dots,r\} \tag{3.7}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{r} S_{ij} x_{ij} \leq V(D_k) \;\; \forall k \in \{1,\dots,n\} \tag{3.8}$$

The only change in this formulation compared to the one from the alpha parameter study is in the objective function 3.5 that maximizes the average quality layer, normalized by the number of segments, and assigns this value to the variable W_{opt}.

In the next optimization step the W_{opt} is set as constraint as minimal achievable target quality.

$$\text{minimize} \quad \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j=1}^{r}(x_{ij} - x_{i+1,j})^2 \tag{3.9}$$

$$\text{subject to} \quad \sum_{j=1}^{r} x_{ij} = 1 \;\; \forall i \in \{1,\dots,n\} \tag{3.10}$$

$$x_{ij} \in \{0,1\} \;\; \forall i \in \{1,\dots,n\},\; j \in \{1,\dots,r\} \tag{3.11}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{r} S_{ij}x_{ij} \leq V(D_k) \;\; \forall k \in \{1,\dots,n\} \tag{3.12}$$

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{r} jx_{ij} \geq W_{opt} - \epsilon \;\; \text{with } \epsilon \; \in \{0,\dots,r\} \tag{3.13}$$

The objective function 3.9 as one can see is now minimizing the number of switches as the second term of the α parameter bound optimization did.

The constraint 3.13 constrains the optimization to minimize the switches while still achieving at least the average quality of $W_{opt} - \epsilon$. The value of ϵ is defined to be in the range of $[0,\dots,r]$.

With the default value of $\epsilon = 0.1$ an optimal adaptation path is generated that is at maximum in a value of 0.1 worse in average quality than the maximal achievable average quality but is diminishing the number of switches manifold. A comparison of the adaptation path, calculated with the medium goodput pattern and the video DXU360o_w64, from step one and the further optimized adaptation path with $\epsilon = 0.1$ can be seen in Figure 3.9.

The number of quality switches from the quality optimization step could be drastically reduced from 32 quality switches to only 2. On the other hand the average quality did only reduce from 3.372 to 3.275, so it is in the range of the allowed $\epsilon = 0.1$ of quality degradation. The gain in QoE through reduction of quality switches outweighs the small loss of average quality by far.

This modified optimization problem formulation, with an ϵ of 0.1, is therefore used further throughout the process of training and validation sample generation for the ML algorithms. Therefore the resulting optimal adaptation path, the optimal quality picking strategy, is used as the correct choice of a class, i.e. quality level, for certain environmental input features, e.g. the average goodput.

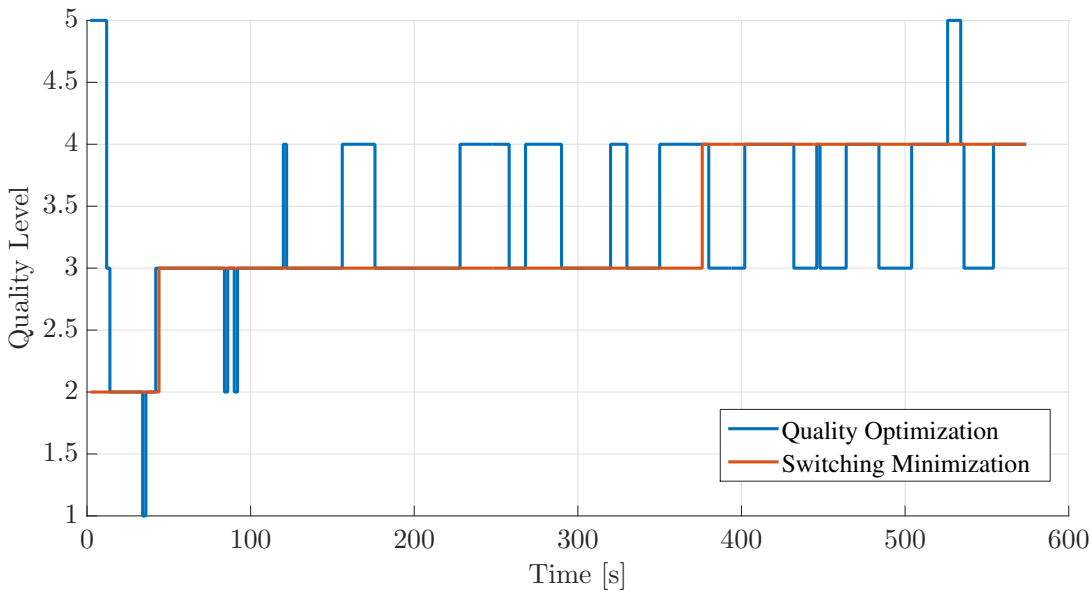


Figure 3.9: Comparison of the optimal adaptation paths after the first and second step in the modified optimization formulation.

3.4 Discrete Event Simulation based Performance Evaluation

The Discrete Event Simulation (DES) is used to evaluate the performance of the ML classifier, mainly to see if the algorithm could learn to behave similar to an optimal adaptive streaming adaptation logic. A DES is a kind of simulation that, as the name suggests, discretesizes each event in time and schedules them in a heap memory. The details on the DES implementation can be found in chapter 4.4.4 "Discrete Event Simulation implementation".

Figure 3.10 depicts the evaluation methodology for the generation of performance metrics.

A goodput pattern and video segments are passed into the DES. In the DES one can choose between several adaptation logics, the HASBRAIN algorithm, KLUDCP and TRDA threshold-based algorithms or none. After the simulation of the adaptive streaming scenario with these given inputs an adaption path is created, as well as several simulation performance metrics are gathered.

The next step is to compare the simulation adaption path with the optimal adaptation path given from the optimization, as well as comparison between the simulation metrics and the optimal metrics also from the optimization. These comparisons are done through calculating the difference between optimal solution and simulation results. When comparing then several adaption logics performances in the DES to one another the point of

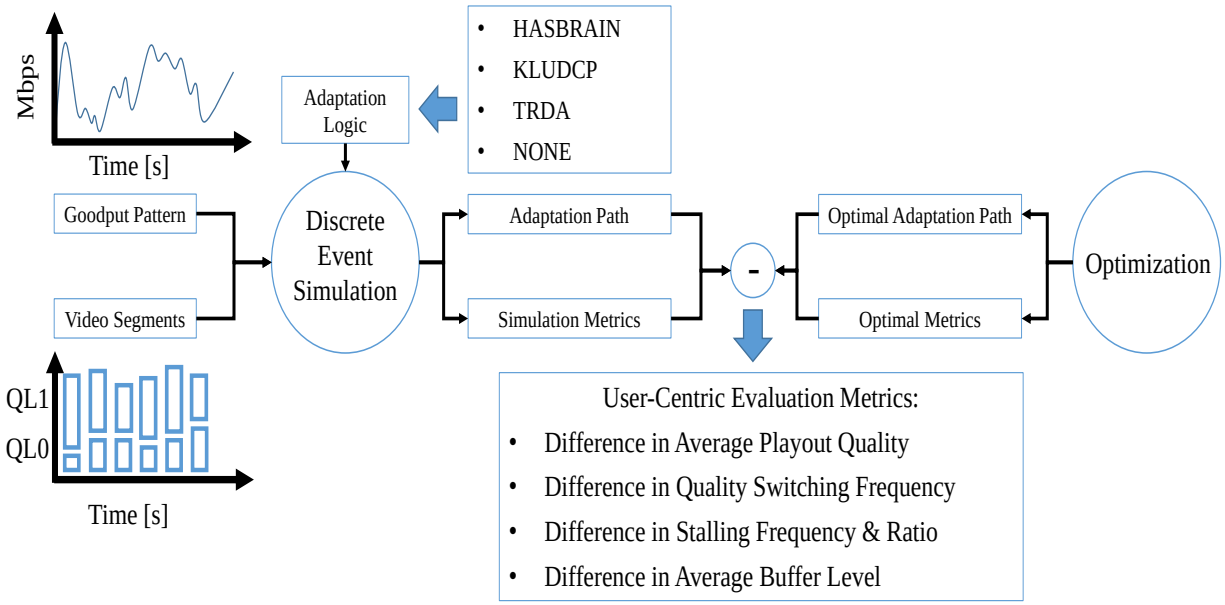


Figure 3.10: The evaluation methodology with the Discrete Event Simulation

reference of the performance metrics is always the optimal solution.

3.4.1 Evaluation Metrics

In an adaptive streaming scenario there are several metrics one has to consider to evaluate the performance of the adaptation algorithm that are derived during the playback. These metrics can be differentiated into two sub sets of metrics, the resource-centric and the user-centric metrics. where the resource-centric metrics allow to evaluate the adaptation logic from a technical perspective, e.g. bandwidth utilization or bandwidth fairness, the user-centric metrics allow to evaluate the performance from the resulting output perceived by the user, e.g. average quality or quality switching behaviour. Because the focus of the performance evaluation was laid on user-centric evaluation metrics, all the chosen evaluation metrics are part of this subset.

As the performance is relative to a derived optimum, i.e. the optimal adaptation path from the optimization, the described metrics are defined as differences from the optimal result. An exception is the evaluation of the stalling behavior because per definition there are no stalling occurrences in the optimal adaptation path.

The chosen **user-centric** metrics are then described as follows.

Difference of Average Buffer Level

The average buffer level is defined as the content data that is buffered during the playout in the player in seconds, averaged over the whole session duration. This average is then subtracted by the average buffer level from the optimal adaptation path over the whole session to receive the difference of the average buffer level.

Difference of Average Quality

Difference of Average Quality is quantified by the average quality level over all received segments achieved by the adaptation logic subtracted by the optimal achievable average quality level from the optimal adaptation path.

Difference in Switching Frequency

The Difference in Switching Frequency is the number of occurring quality level switches from the evaluated algorithm in the playout session divided by the session duration, subtracted by the quality switching frequency from the optimal adaptation path. A quality switch is defined as a change of the quality level during playout neither differentiating if the quality switch was lowering or increasing the quality level nor if the switch was over more than on quality level.

Stalling Frequency

The Stalling Frequency is the sum of all stalling occurrences in a simulation run divided by the video duration in minutes. This Stalling Frequency describes therefore the number of stalling occurrences that arise on average during this adaptive streaming session.

Stalling Ratio

The Stalling Ratio is defined as the overall session duration, that is the video length plus the cumulative stalling duration, divided by the video length. With this metric it can be evaluated how long the stalling occurrences expand the overall playout.

3.4.2 Evaluation Videos

The videos chosen for the evaluation are a set of 20 randomly selected videos from YouTube. These videos, as the one used for the training samples, are available in five quality levels. The overall characteristics of these videos are summarized in table 3.4.

Avg. bit-rates	140p	240p	360p	480p	720p
Min (Mbps)	0.03	0.05	0.08	0.15	0.30
Mean (Mbps)	0.06	0.14	0.26	0.51	0.99
Max (Mbps)	0.07	0.16	0.39	0.72	1.44
Std	0.02	0.03	0.09	0.17	0.33
Segmentsize	5s				
Duration	{4, 5, 6, ..., 8} minutes, avg 5.79 m				

Table 3.4: Validation video characteristics

These characteristics are similar to the one from the training videos described in 3.3.2 "Evaluation Scenario Definition" as to not exacerbate the already complex adaptation scenario, especially for the ML algorithm as it learns from the video characteristics of the training videos.

3.4.3 Evaluation Scenarios

With the validation videos and the described medium goodput pattern with 107 time shifts as for the training samples generation it is possible to create $107 * 20 = 2140$ different streaming scenarios and the corresponding optimal adaptation paths. Due to the occurrence of a non solvable optimization combination, i.e. there could not been found an optimal solution or due to the optimization's implementations tolerance a stalling event occurred an this run got removed, the total number of generated evaluation streaming scenarios is 1986. These streaming scenarios or runs are used in the DES to create the evaluation metrics for the performance analysis.

Chapter 4

HASBRAIN Algorithm

In this chapter, the Machine Learning algorithm feature definition, model parameter definition and the implementation of the **HASBRAIN** algorithm are explained in detail.The most in depth description of the parameterisation is done on the ML tool of ANNs.

The following subsections are structured as follows. Beginning with the definition of the ML features that have to be created for training the optimal adaptation path. Followed by the design and parameter choices of the ML algorithms, i.e. shape and size of a ANN or kernel of a Support Vector Machine (SVM). Then a short overview of the implementation of the whole **HASBRAIN** algorithm is given.

4.1 Machine Learning Feature Definition

To learn certain characteristics and relations from data in supervised learning, the data has to be prepared beforehand. This means that if a certain behaviour of a machine learning algorithm is wished for, then the fed data to the algorithm has to have the information given in a form that a common adaptation algorithm could draw conclusions from it.

Meaning that the input data should comprise of all the necessary information that is needed to understand the hidden relations in the streaming scenario. Therefore redundant data should be left out and important data, for example the buffer level, has to be included. For further in-depth information about feature definition refer to [BL97].

4.1.1 Input Features Definition

To begin with the feature definition one has to think about what information would a common threshold algorithm need to decide which quality segment to select. Because the future is unknown, i.e. regarding the future goodput, a thoughtful decision could only be

drawn by inspecting the past and current decisions, as well as memory on the goodput behaviour. With these observations a conclusion can be drawn for the choice of the next quality segment.

To represent the choices and behaviour of the past, certain memory constants are introduced. These constants are listed in table 4.1.

Constant	Definition	Explanation
ν	$\max_{\forall i,j}(Tp_{d_i}, S_{ij})$	Normalization Constant
Bl_{max} [20]	$\in \mathbb{N}$	Maximum Buffer level in [s]
c_{OT} [30]	$\in \mathbb{N}$	Memory of Observed Throughput
c_M [30]	$\in \mathbb{N}$	Memory of proceeding and succeeding segments

Table 4.1: Machine Learning constant definition

The value of ν is used to normalize the goodput pattern and the segment sizes to values in the range of $\{0, \ldots, 1\}$ and is defined as the maximum bit-rate of the goodput pattern and the segmentsizes. Bl_{max} defines the buffer level that is used to normalize the buffer level input to a range of $\{0, \ldots, 1\}$. The remaining constants are the sizes of the memory input features from the observed throughput as well as the size of the memory input features of proceeding and succeeding segments.

Next, the input features of the ML algorithm have to be defined. In table 4.3 the definition of the individual features is given.

The first feature is the average of the throughput, or goodput, over a certain period defined by the constant c_{AT}. The goodput is defined as the application level throughput. Followed by the memory of the observed throughput over c_{OT} seconds. The observed throughput is calculated by the download time of a proceeding segment. With these information a ML algorithm will learn the fluctuations of the goodput pattern and will learn to decide more conservative if the goodput was decreasing in the last seconds or more aggressive if the goodput was increasing. The next features are the selected quality levels normalized by the number of available quality levels over the course of the last c_M seconds, followed by the also last c_M seconds of the corresponding selected segment sizes normalized by the normalization constant ν. These input features are also named Memory Inputs as they correspond to the past information that is available to a common adaptation algorithm. Especially the memory of the choice of previous segments and quality levels are needed to learn a QoE optimal behaviour as this information corresponds to the average playout

Input	Definition	Explanation
$x_0 ... x_x$		Set of all input features
	Memory Inputs	
x_0	$\frac{1}{\nu} \sum_{i=1}^{c_{OT}} \frac{x_i}{c_{OT}}$	Average throughput of the last c_{OT} *seconds*
$x_{1...c_{OT}}$	$\in \{Tp_{d_{i-c_{OT}}}, \ldots, Tp_{d_i}\} \cdot \frac{1}{\nu}$	Observed throughput for the download of the last c_{OT} *segments*
$x_{c_{OT}+1...c_{OT}+1+c_M}$	$\frac{1}{r} \cdot x_{ij} \cdot j \ , \forall x_{ij} = 1, \forall i \in \{1, ..., c_M\}$	Memory of selected quality levels
$x_{c_{OT}+2+c_M...c_{OT}+2+2c_M}$	$\frac{1}{\nu} \cdot S_{ij} \ , \forall x_{ij} = 1, \forall i \in \{1, ..., c_M\}$	Memory of selected segments
	Video Inputs	
$x_{c_{OT}+3+2c_M...c_{OT}+2+2c_M+c_M \cdot R}$	$\frac{1}{\nu} \cdot S_{(i+k)j} \ , \forall k \in \{0, ..., c_M\}, \forall j \in \{1, ..., r\}$	Future representations
	Player State Inputs	
x_x	$\frac{1}{Bl_{max}} \cdot Bl_{d_i}$	Buffer level at time $t = t_{d_i}$ normalized by the maximum Buffer level

Table 4.3: Machine Learning input feature definition

quality and the switching frequency.

Next, the video inputs are defined. These inputs represent the information that can be drawn from the given video. Because in a DASH scenario the information of the whole video, in terms of available bit-rates or quality levels as well as segment sizes, is given through the MPD file the next input features are the segment sizes of all quality representations r over the last c_M seconds. This inputs help to draw conclusions over the video itself to adapt for possible changes through increasing segment sizes in the future.

Last, the Player State Information inputs are defined. This information is solely the buffer level of the player at the decision time normalized with the previously defined maximum buffer level. This feature is used to understand the behaviour of the player according to its buffer fullness. Meaning that conclusions can be drawn over a more aggressive switching behaviour with a high buffer level as well as a more conservative behaviour with a low buffer level.

If there is no data available for feature creation, for example in the beginning of an adaptive streaming scenario there are no memory inputs available as no choices have been made to this point, then these features are set to the value 0.

All these input features would allow an common threshold-based adaptive streaming algorithm to adapt to the necessity of the current situation in the streaming process. With the default values of the constant definition the number of input features totals to 242, when not stated otherwise this is the count of features used throughout the further work.

4.1.2 Output Features Definition

Because the learning method used in this work is supervised learning, the machine learning algorithm needs not only input features to learn from but also output labels that correspond to the right choice of quality level given these input features, the optimal adaptation path. The output labels are therefore defined in table 4.4.

Variable	Definition	Explanation
$y_{0...r-1}$	$\in \{0, 1\}$	Optimal quality level

Table 4.4: Machine Learning output feature definition

With the default value of $r = 5$ the number of output labels is also 5. The process of classification is now to learn from the great number of input features the hidden relations given from the optimal adaptation path to determine the right quality level at a point of decision in the adaptive streaming scenario.

4.2 Machine Learning Model Definitions

The design and in this regard the parameterization of a ML algorithm, especially ANN, are a topic under current research and defined by a great amount of rule of thumbs. The design choices allow for a greater adaptation on hidden relations in the data as well as improving the ability to generalize from the training samples.

As one of the most important metrics in the topic of ML is the so called accuracy. The accuracy is defined in the performance evaluation of a ML technique as follows. The sum of all correct classified labels over the sum of all classified labels. This metric is used to determine the classification performance of the used ML algorithms.

4.2.1 Artificial Neural Network Classifier

Artificial Neural Networks (ANN) allow for a great amount of variations, design choices and shapes. As mentioned in the introduction each input feature is fed into the first layer of a neural network, the input layer. The input layer consists of a neuron for every input feature. The choice for the activation function was for the logistic or also known as the sigmoid activation function for all input and hidden layer neurons because no beneficial change of training speed was observed by using the TanH or ReLU activation function as well as the mapping of the sigmoid activation function corresponds to the normalization of all the input features in the range $(0, 1)$.

Further information regarding the choice of the activation function as well as practical advice for the design of a ANN can be found in [LBOM12].

After clarifying the choice of activation functions, one has to consider the number of hidden layers. Hidden layers are the neurons that are between the input and output layer and are therefore named hidden as neither their in- nor their output can directly be observed. Training a artificial neural network with more than 1 hidden layer is commonly referred to as deep-learning.

As data would be linear separable there is no need to add any hidden layers at all but this could not be guaranteed for the given input features. Therfore in the case of the **HASBRAIN** ANN a common rule of thumb was used. Exactly one hidden layer with lesser than or equal numbers of neurons than the input layer is used. This results to an ANN with 110 hidden layer neurons.

The size of the output layer is defined by the number of classes that are available and is defaulted to 5 because of the 5 quality levels. The output layer does not have a sigmoid activation function but uses a softmax function to determine the classified label. The softmax function takes the values that would be inputted into an neuron of this layer sums them up and maps the corresponding result to a range of $(0, 1)$. This mapping is a special mapping of the input to a probability distribution of K different possible outcomes, where

K is the number of output labels. In other words, with the softmax function shown in table 4.5 it is possible to map the inputs into a probability value for a certain class.

Function Name	Equation	Range
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \; for \; i = 1, \ldots, J$	$(0, 1)$

Table 4.5: The Softmax function

The label that is then chosen to be the classified one is the one with the highest probability.

To conclude the design choices one has to consider that this ANN is fully connected, meaning that each neuron of a layer is connected to every other neuron of the next following layer. With the default 242 input features and therefore 242 input neurons, the ANN contains in total 357 neurons.

The implementation, design and shaping was done with the Machine Learning library for ANNs from Google, named *TensorFlow*. For further information refer to [AAB+15]. To ease the use of this library a wrapping library was used, named *KERAS*. *KERAS* allows the abstraction of the design of an ANN with simple to use methods. Further information can be found at [C+15].

4.2.2 Other Machine Learning Classifiers

For the remaining used ML algorithms, namely SVM and k-nearest-neighbours the design options are rather limited compared to the ones of an ANN. Beginning with the SVM for the classification purpose that is referred to as Support Vector Blassification (SVC).

Support Vector Classification

As the SVM tries so separate the input data through linear hyper planes, then a single classification input feature set of size n can be seen as single data point in a n-dimensional space corresponding to a certain class. As thoroughly explained in the background of this work it is sometimes not possible to linearly separate these data points with said hyper planes. Therefore a Kernel function has to be used. This Kernel function maps the non linear separable data points to a higher dimensional space where a linear hyper plane can be used to separate the data. Further information can be obtained [SS01]. The Kernel functions used in this work was the Gaussian radial basis function or RBF.

Because SVC can in its definition only be used as binary classifier, i.e. it can be used to classify two labels, a special multi-class classification approach is used. This approach was defined by [KPD90] and is called one-versus-one approach. A binary SVC classifier is trained for each pair of available classes. The formula 4.1 is used to calculate the number of resulting trained binary classifiers if there are in total r classes.

$$Number\ of\ Binary\ Classifiers\ =\ \frac{r*(r-1)}{2} \tag{4.1}$$

With the default value of $r = 5$ classes there have to be in total 10 binary classifiers learned. If a label has to be classified each binary classifier is used to determine the corresponding class. The label that has been classified the most is the one that is ultimately classified. If there is a draw, then the output class is chosen randomly between the most classified labels.

k-Nearest-Neighbours Classification

As mentioned in the Background, the k-nearest-neighbours approach memorizes each training data point and calculates the corresponding class of a new sample by the euclidean distance to the memorized training data points and selects the class that has been found the most in the set of the k-nearest data points. For the evaluation the k parameter was set to 10.

One of the other few parameters that can be changed in this approach is the weighting of distances. This is used to weigh data points more important if the distance is for example very low to the sample data point. In this work only the uniform weighting was used, i.e. that each neighbours distance was weighted the same.

Both the SVC and the k-Nearest-Neighbour classifiers have been implemented with the use of the scikit-learn Machine Learning python library by [PVG+11].

4.3 Implementation

The Implementation is, as the **HASBARIN** algorithm suggests, split up into 3 steps. First, the feature generation with the implementation of the optimization followed by the implementation of the ML algorithms training and validation. Last, the implementation of the DES and the generation of validation metrics.

4.3.1 Feature Generation

The feature generation is done by iterating over the available goodput patterns and the available training videos and then generating an optimal adaptation path for each combination of goodput pattern and video. Simultaneously to the resulting optimal adaptation path as output features, the input features are calculated.

4.3.2 Machine Learning Implementation

The defined ML model is trained with the samples created from the feature generation. These samples are shuffled before training and are split up into a specific portion of a training and a validation set. After training the model is saved and can further be used as adaptation logic.

4.3.3 Discrete Event Simulation Implementation

Last, the Discrete Event Simulation is used to determine the overall performance of the machine learning techniques. The DES is a simulation of an adaptive streaming scenario, therefore the adaptation logic has to be chosen in the beginning.

The DES uses the video segments and the available goodput pattern at simulation time as adaptive streaming scenario inputs. To gain further insight in the DES implementation, figure 4.1 illustrates the corresponding system model.

In the beginning of the simulation the Player loads the adaptation logic. Thereafter the Player uses a method from the Adaptation Logic that calculates the according features for its current situation in the simulation, i.e. with the according Memory, Video and Player State Inputs. After this feature calculation the adaptation logic is used to determine the next quality representation.

This quality selection is then forwarded to a Connection that schedules the download of the next segment. Therefore it puts an arrival event into the event heap and the buffer level is increased by the segment length in seconds.

When the simulation time has reached the initial delay threshold then the player starts to playout its segments from the buffer. In parallel it iterates over the feature calculation method, followed by the decision method and then downloads the selected quality representation until the last segment has been downloaded.

If it is the case that the buffer is empty while the player wants to play out the next segment a stalling event occurs. To prevent further stalling occurrences a certain buffer strategy has been implemented namely the D-policy. It was derived by [MM16] and states that the player has to remain idle until the sum of the service times of the units in the queue

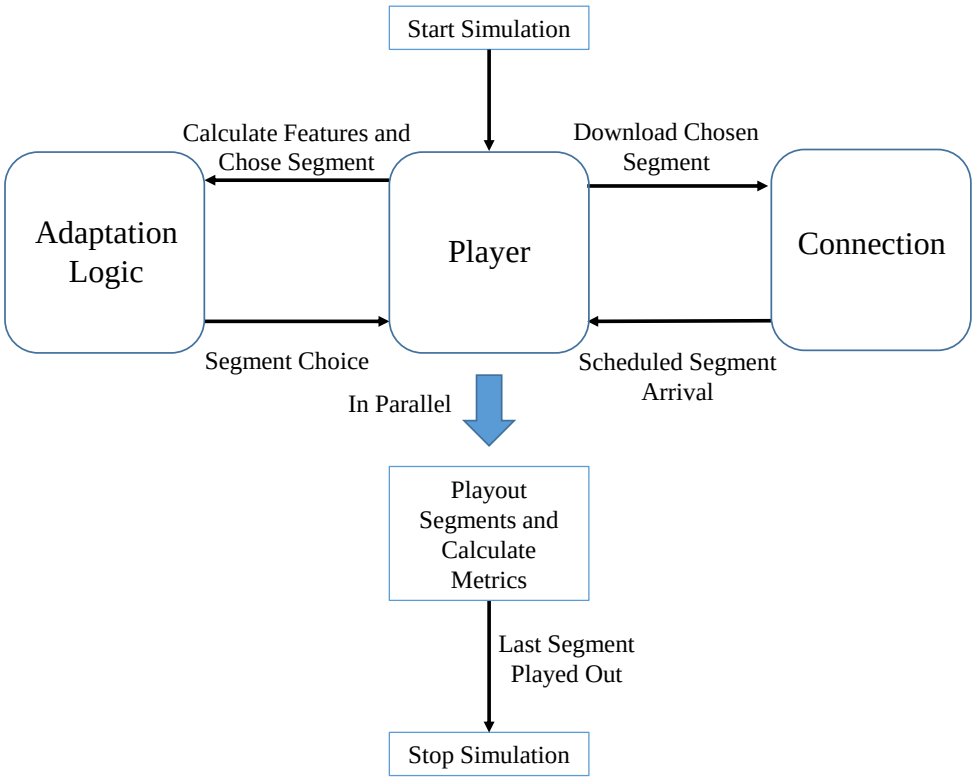


Figure 4.1: System model of the Discrete Event Simulation of an adaptive streaming scenario.

amounts to at least D seconds. Where in this implementation D was set to be 10s. After this threshold has been reached or the last segment from the video has been downloaded then the play out continues. If the last segment of the video has been played out then the simulation ends and the DES logs all performance metrics for this simulation run.

The metrics gathered over the whole set of videos and goodput patterns are saved into a result database. The definition of this result database is available in Appendix A. This result database is then further used for the performance evaluation.

Chapter 5

Evaluation

In this chapter the evaluation of the ML algorithms in terms of classifying performance is presented. Next, the evaluation of the performance on the DES for the ANN is done using the obtained metrics from the result database. The results are summarized at the end of this chapter and further implications of these results are discussed.

5.1 Machine Learning Training

The first technique that is evaluated is the ANN. For the training sample generation the videos and goodput pattern described in 3.3.2 "Evaluation Scenario Definition" where used. Deviating from the described 7 time shifts, for the training sample generation where used time shifts of 7s steps, i.e. $\{0, 7, 14, \ldots, 747\}$. Furthermore only the medium goodput pattern was used, so that the resulting number of available goodput pattern is 107. The training samples where then calculated by creation of optimal adaptation paths of random combinations of goodput pattern and videos. The resulting number of training samples is 908,132.

Artificial Neural Network

When learning an ANN there has to be done a split between the samples into a training and a validation set. The validation set of samples is used to validate the performance of the NN with samples that where not used in the training and therefore unknown or unseen by the algorithm, so that conclusions about the generalization error of the ANN can be drawn. This split was set to be 90% test samples and 10% validation samples from the whole sample set. There is a common rule of thumb about this split ratio that recommends either 90% to 10% or 80% to 20% as optimal. In this case the number of training samples is 817,319 samples and 90,814 validation samples.

Another parameter that has to be set prior to learning is the number of training epochs and batch sizes. A training epoch is a full iteration over the training set whereas the batch size is the number of samples that are used for a single update step in the NN weights. E.g. if the batch size is 8 then 8 samples are classified the loss function, that describes the classification error, is calculated for each of it and then averaged and further used in the back-propagation algorithm to update the weights.

Last, there has to be set a callback that ends the learning process if ANN begins to overfit. Overfitting can be observed if the accuracy on the validation set is decreasing or the loss is increasing. The callback is set to end the training if after 10 succeeding epochs the loss did not decrease by at least 0.001 in the validation set.

Before training the samples are randomly shuffled so that the ANN has to deal with single decision situations rather than with consecutive adaption paths.

In Figure 5.1 the progression of the accuracy over the course of the training can be seen.

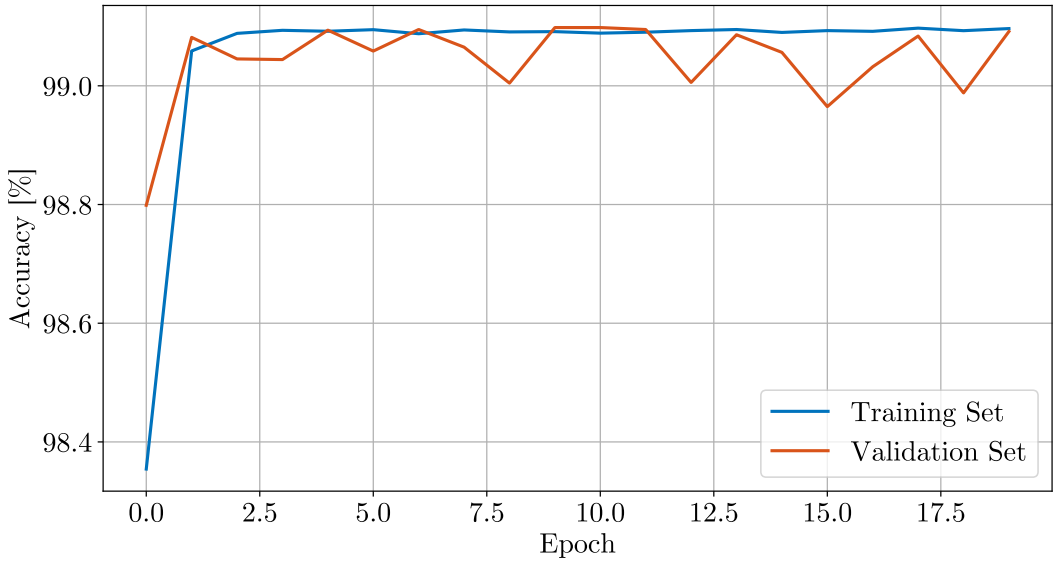


Figure 5.1: Progression of the accuracy of a Artificial Neural Network during training.

The first observation that an be made is, that after the first epoch the accuracy over the training and validation set is at around 99.05%. This value progresses to 99.1% accuracy in the training set after 3 epochs. This does not increase any further only decrease for a small margin. But the accuracy on the validation set fluctuates more visible. This is due to the fact that the validation set is randomly shuffled at each epoch, so that always different samples form a batch are classified and these can on average perform slightly different.

To conclude the training results of an ANN, in Figure 5.2 is the progression of the training and validation loss depicted.

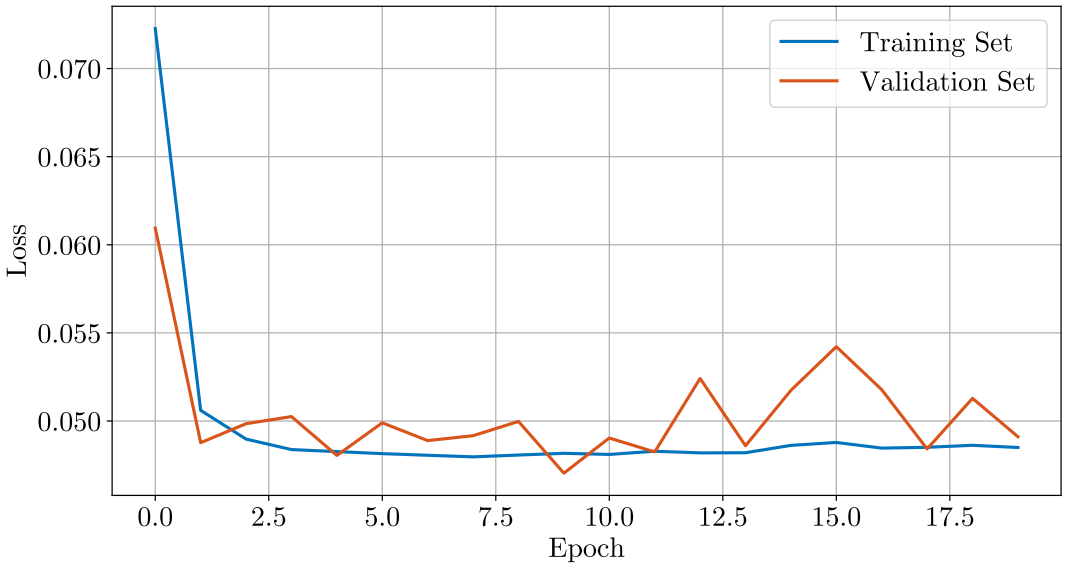


Figure 5.2: Progression of the loss of a Artificial Neural Network during training.

Correspondingly to the observations with the accuracy one can see that after the first epoch the loss is down to a value of slightly above 0.050. The training loss further decreases till reaching its minimum after 6 epochs and then starts slowly to increase. This increase is a indicator for the beginning of overfitting. As mentioned the training ends after the loss does not further decrease by at least 0.001 in the next 10 epochs. This point of minimal loss is reached at epoch 9 and the training continues until epoch 19 and stops there.

Because the effects of slight overfitting could have happened until the stop of training the resulting ANN that is used for the further simulation is set to be the one with the smallest validation loss. Table 5.1 shows the resulting performance of the used ANN.

Parameter	Value
Training Accuracy	99.0912972781%
Validation Accuracy	99.0981566717%
Training Loss	0.0481660584243
Validation Loss	0.0470399902968

Table 5.1: Performance parameter values of the trained Artificial Neural Network.

A validation accuracy of around 99.1% is a very well trained ANN and considered satisfactory for the cause, therefore no further actions to improve the training performance were done. It is to mention that the time it takes to train this ANN is around 1 hour.

Support Vector Machine

Next, the training performance of the Support Vector Machine with radial basis function kernel is evaluated.

The training accuracy results cannot be calculated while learning. Only the validation accuracy can be determined after fitting of this ML algorithm. Fitted with the same training set as the ANN the validation accuracy of the SVC results to 98.74%. Despite being in close range to the ANN in terms of accuracy, the training time of this classifier is 10 times longer, around 10 hours.

k-Nearest-Neighbours

Last, the performance of the k-Nearest-Neighbours ML technique is evaluated. Due to the definition of this algorithm, inherently no training accuracy can be determined. For the validation accuracy the k-Nearest-Neighbours ML algorithm reaches after memorizing the training samples a value of 97.69%. In comparison to the ANN, the training is depended on the size of available memory equally fast but the calculation of a even a single classified sample increases with the number of trained samples and is beyond the time for a useful operation as adaptation logic.

Concluding with the result that the overall classification performance of the Artificial Neural Network outperforms the ones of the SVM and k-Nearest-Neighbours technique it was further considered to evaluate only the ANN with the Discrete Event Simulation of the adaptive streaming scenario.

5.2 Evaluation Results

With the introduced evaluation metrics in 3.4.1 "Evaluation Metrics" it is now possible to draw conclusions on the performance of the **HASBRAIN** algorithm compared to the optimal adaptation path given from the optimization in the sense of how well the ML technique could generalize from the training samples.

These performance metrics from the ML algorithm are then taken into comparison with the two introduced threshold-based algorithms, namely TRDA and KLUDCP. This comparison allows for a more realistic performance analysis because the threshold-based algorithms have to perform on the same set of validation videos as the ML algorithm.

Because the metrics where gathered over a set of different validation runs the resulting metrics are therefore figured as the Empirical Cumulative Distribution Function (ECDF). The ECDF allows to draw conclusions about the whole set of different playback scenarios as on the x-axes the respective metric is plotted and on the y-axis the cumulative probability

from 0 to 1 is drawn. In other words, for a certain x-value the corresponding y-value suggests the number of runs in percent that are lesser than or equal to this metric value. To ease the reading of this ECDFs the cumulative probability value of 0.5 is marked by a shaded grey horizontal line in each plot.

First, the difference of average buffer Level is analyzed. In Figure 5.3 the ECDF of this metric from the **HASBRAIN** algorithm, as well as the one from the KLUDCP and TRDA algorithms are illustrated.

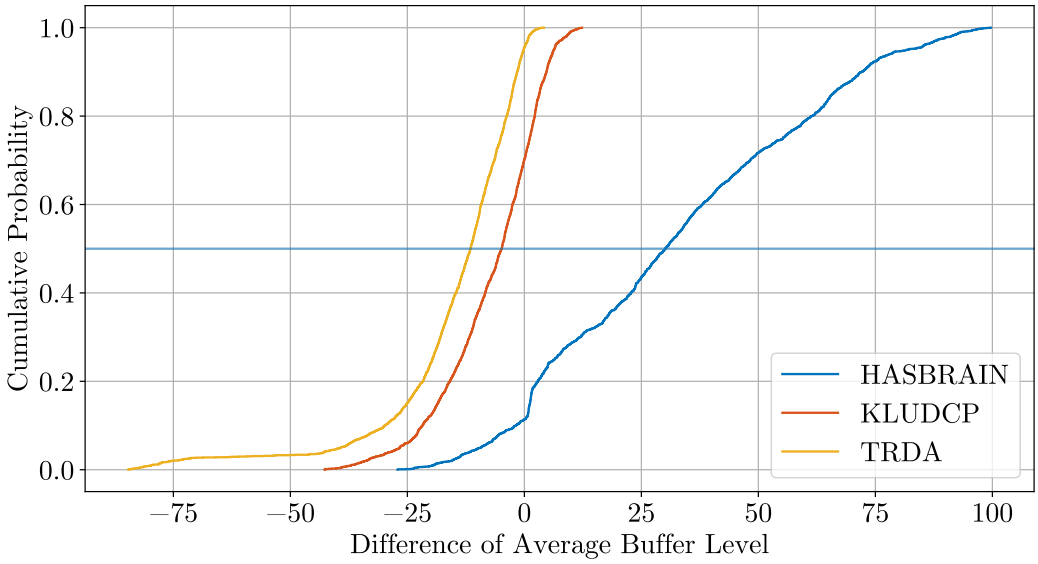


Figure 5.3: Difference of the average buffer level compared to the optimal adaptation path.

For 15% of all runs the **HASBRAIN** algorithms average buffer level in the simulation is below the one from the optimal adaptation path to a worst case of -25 seconds. On the other side the ML technique could achieve for the remaining 85% of the runs an increasing buffer level for more than 50% of the runs even to an average buffer aevel of over 25 seconds. Comparing this first result to the performance of the threshold-based algorithms it is to see that both, the KLUDCP and the TRDA algorithm are worse in building up a high buffer level as their buffer levels remain for most of the runs below the average buffer level of the optimization. To see if this high buffer levels are due to a more conservative adaptation behaviour of the **HASBRAIN** algorithm the next metric that has to be considered is the Difference in the average quality.

The average quality of the adaptation algorithms in the DES versus the average quality in the optimization is compared and shown in Figure 5.4.

Where around for 85% of the runs the average quality of the simulation decreases down until a minimum of -1.2. The most runs lay in the range between a decrease of -1.0 and 0.

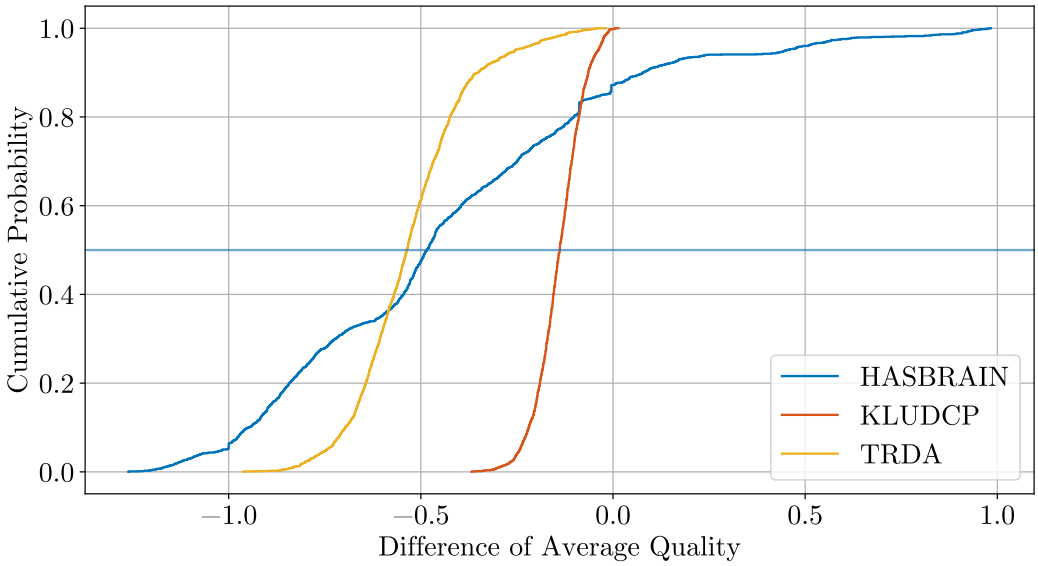


Figure 5.4: Average playback quality in the simulation versus the average playback quality from the optimization.

Only a small number of runs reaches an average quality on par with the optimal adaptation path. Even 15% of the runs show an increasing quality up to around 1 quality level above the optimal adaptation path. This must lead to stalling occurrences because it is not possible to adapt better than the optimal adaptation paths average quality plus $\epsilon = 0.1$. Taking the information gathered from the proceeding Figure 5.3 into this analysis, then a few first conclusions on the adaptation behaviour can be drawn. As the average buffer level increases and the average quality decreases for 85% of the runs it is due to a more conservative adaptation behaviour as the buffer level can only be increased when more lower and therefore smaller in size quality levels than the optimal adaptation path are downloaded. On the other hand for the remaining 15% of the runs the average quality can only be increased by downloading greater in size, higher quality levels that take longer to download and therefore decrease on average the buffer level during a session. This coincides with the observations on the difference of the average buffer level for the ML adaptation algorithm.

For the comparison in the difference of the average quality Level it is to see that both, the TRDA as well as the KLUDCP, algorithms average quality is less than the optimal adaptation path. The TRDA performs even worse than the KLUDCP. Whereas the KLUDCPS worst difference in average quality is -0.4 the worst case in the TRDAs is -0.95. The ML algorithm experiences a worst case of -1.25 and lies for 35% of all runs under the performance of the other algorithms. For 50% of all runs the performance of the **HASBRAIN** algorithm lies in between the performance of the TRDA algorithm, which it tops, and the

KLUDCP algorithm. For the next 5% of all runs the ML technique outperforms both of the threshold-based algorithms. The last 10% of runs are even above the average quality of the optimal adaptation path. It is possible to gain an average quality of $\epsilon = 0.1$ on cost of more switches but no more without any stalling. Therefore this positive difference in quality, especially the runs with > 0.1 are indicators for runs with stalling events.

Next, the ECDFs in Figure 5.5 depicts the quality switching frequency of the adaptation logics in the simulation versus the quality switching frequency of the optimization.

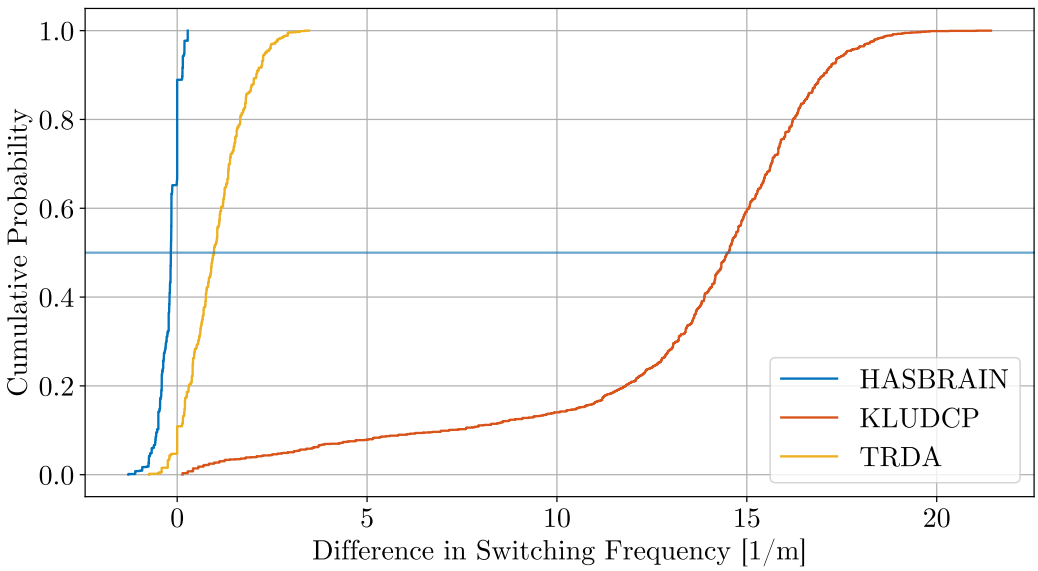


Figure 5.5: Difference of the quality switching frequency in the simulation from the quality switching frequency of the optimization.

For around 65% of the runs the **HASBRAIN** algorithm was more conservative than the optimization and switched less often the quality. For 25% the optimization and the ML tool switched as often, whereas for the remaining 10% it switched more often until to a maximum of 1 more switch per minute. With this observations an the observations about the average quality and the average buffer level it is suggested that the high average buffer level and the lower average playout qualities are due to a conservative switching behaviour.

For the threshold-based algorithms it is to see that the KLUDCP algorithm due to its simple threshold adaptation logic occupies a very large number of quality switches due to it's high switching frequency. Also the TRDA algorithm performs worse, with a higher switching frequency than the ML technique and in 90% of the runs even higher than the switching frequency of the optimal adaptation path. The **HASBRAIN** algorithm on the other hand has always a lower quality switching frequency than the other two algorithms. From a QoE user-centric point of view the ML algorithm will perform better than the compared algorithms.

Following the ECDF of the Stalling Frequency is depicted in Figure 5.6.

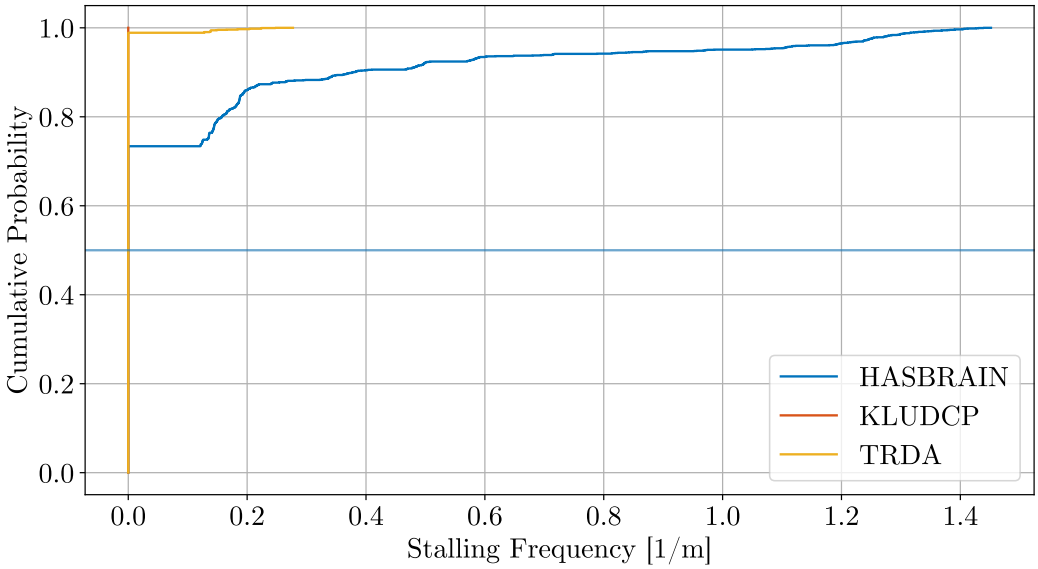


Figure 5.6: Stalling frequency in the simulation, defined as number of stalling occurrences per minute.

The first observation for the ML algorithm that can be made is that for 75% of all runs the stalling frequency is 0 and therefore there are no stalling occurrences at all. For the next 15% of the runs, the stalling frequency is lower than $1.0\frac{1}{m}$. The maximum stalling frequency reached is $2.5\frac{1}{m}$. This stalling occurrences are from a QoE optimization perspective degrading the user view-experience but are a consequence of the conservative switching behaviour and the therefore sometimes resulting higher average quality than the optimization that must per definition lead to stalling.

Comparing this results to the performance of the KLUDCP algorithm it is to see that there are none stalling events at all because the stalling frequency is exactly 0.0 over all runs. The TRDA algorithm in comparison exhibits only a very small number, about 2%, of runs with stalling. Directly compared the ML technique performs therefore worse for 25% of all runs.

The Figure 5.7 illustrates the stalling ratio as last considered evaluation metric.

Similar observations than to the Figure 5.6 for the stalling frequency can be made. For 75% of the runs the stalling ratio is 1.0 that coherently fits with the observation of no stalling events in those runs. The stalling ratio for the remaining 25% increases to a maximum of 1.75. Meaning that the whole streaming session of this worst case run is stretched, due to stalling, to a length of 1.75 times of the video duration. This behaviour is owed to the D-policy buffer strategy with D=10s that suggests to fill up the buffer to a level of 10s

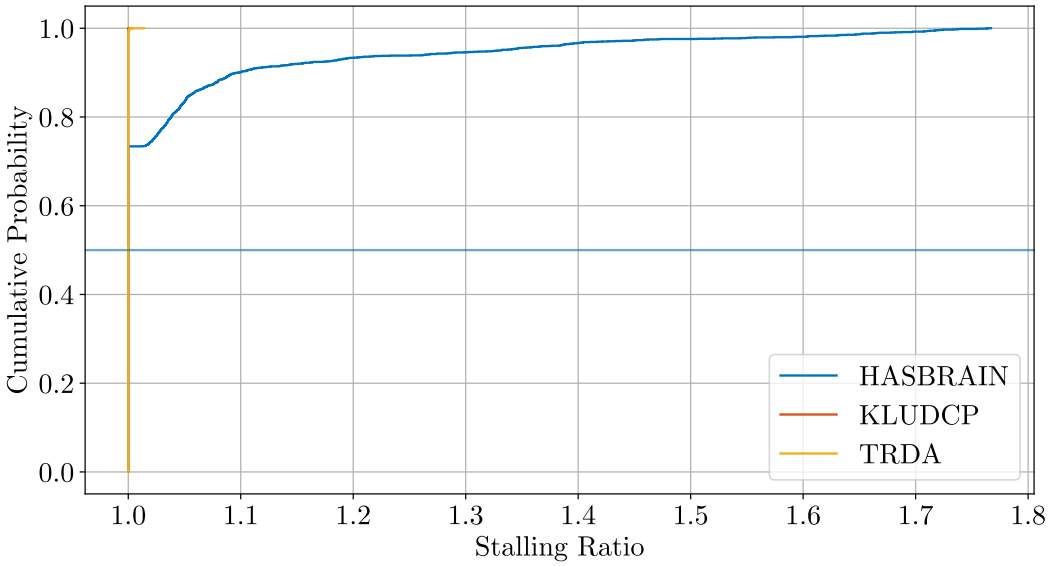


Figure 5.7: Stalling ratio in the simulation, defined as the ratio between session duration and video duration.

after a stalling event and then continue the playout. When due to the complexity of the scenario the **HASBRAIN** algorithm does not perform very well, many stalling events will happen and the session time will rise by the time it takes to fill the buffer up with segments until playout can continue for each such event.

Similar to the conclusion drawn on the stalling frequency comparison it is to see that the ML technique performs worse in 25% of the runs due to the mentioned stalling occurrences.

5.3 Summary & Discussion

In the following the pros and cons of the **HASBRAIN** algorithm are summarized and the further implications are discussed. The impact of this work is evaluated and future improvements and research work are proposed.

Regarding solely the performance of the **HASBRAIN** algorithm several conclusions could be drawn. The algorithm maintains a very healthy buffer level that is in general considered a good characteristic of an adaptation logic. On the other hand this healthy buffer level is obtained through a decrease in average playout quality. Due to a very conservative switching behaviour the ML technique fails in 25% of all validation runs to adapt to the challenging nature of the streaming scenario and allows for stalling events to occur.

Compared to the overall performance of the threshold-based algorithms in the same stream-

ing scenarios it is shown that the ML algorithm is at least on par with these. Considering the average quality that is achieved during the playout the **HASBRAIN** algorithm is comparable well performing as the other algorithms even though that 35% of the runs with the ML algorithm are worse in average quality than the threshold base algorithms there are 20% of the runs that are even better in average quality. In terms of the quality switching behaviour the ML technique could even outperform both of its threshold-based adversaries as the adapted behaviour was very conservative. Considering the metric with a high impact on the QoE, the stalling events, the **HASBRAIN** algorithm performs worse with as mentioned stalling events happening on 25% of the validation runs, whereas the other algorithms did experience little to any stalling occurrences.

While it is also to note that on worst case there occur only 2.5 stallings per minute which is still in an acceptable range, even when prolonging the session duration by the factor 1.8 due to influences of the D-policy.

The trained **HASBRAIN** algorithm in form of an ML technique allows for a lightweight but non optimal streaming client that suffers from slight performance loss due to stalling. Lightweight therefore, because for the case of an ANN only the obtained weights from the training phase have to be distributed to a remote client. A classification in terms as a decision logic is then done by simple vector multiplication of the calculated input features with the weights.

Chapter 6

Conclusion and Outlook

This thesis formulates a new Quality of Experience (QoE) aware adaptation algorithm for HTTP Adaptive Streaming (HAS) based on supervised Machine Learning (ML) techniques named **HASBRAIN**. The ML technique was prior trained with an optimal adaptation path, the optimal quality picking strategy gained through the optimization of an adaptive streaming scenario and was then used as an adaption logic in a real HAS scenario.

An important contribution of this work is the extension and evaluation of an existing adaptive streaming optimization formulation with a weighing parameter α that allows to set a trade-off in the optimization between maximizing the average quality or minimization of quality switches. Furthermore through the evaluation of the α parameter influence another existing two-step optimization formulation from literature was extended. This modification allows for a parameter ϵ to be defined. This parameter defines the maximal allowed degradation in optimal average quality from the first optimization step when minimizing the quality switches in the second optimization step.

In the methodology of this work the evaluation of the α parameter influence study resulted in the finding that it is not necessarily helpful to allow more quality switches while gaining a higher average quality, because the QoE is worsening when allowing more switches. It was shown that a considerate choice of α as 0.1 is sufficient to gain a high average quality while still allow only few quality switches to happen. In addition the contribution for future work with this optimization formulation was summarized.

With this optimization problem it is now possible to determine a Pareto frontier for any given HAS adaptation algorithm, in terms of trade-off between average quality and quality switching. A new more advanced QoE model, based on this finding, allows then for a new evaluation and classification of adaptation algorithms for HAS, including the one suggested in this work.

In the second half of the work the ML part of the **HASBRAIN** algorithm was explained in detail and its performance evaluated. The learning performance in terms of accuracy was

found to be highest with Artificial Neural Networks (ANN) and was therefore considered for the further use as optimal adaptation logic. It was found that the ANN could outperform even two other threshold-based adaptive streaming algorithms, KLUDCP and TRDA, in terms of the quality switching frequency and the average buffer throughout the video playout. But the ML adaptation logic performance suffers from stalling events that impair the QoE and are considered as starting point for further improvement of the **HASBRAIN** algorithm.

Therefore for future work some further improvements in the performance have to be done. First, for the optimization other values of ϵ can be used to change the adaptation behaviour, so that the trained ML algorithm will learn a less conservative switching behaviour. Second, another buffering strategy can be implemented, in this work the D-policy was used but [MM16] suggest two other buffer policies. Also the D value of 10s in this used policy can be changed and its influence on the performance evaluated. Last, because the **HASBRAIN** algorithm does not necessarily depend on supervised learning algorithms in general, it is merely the adaptation behaviour learning of a ML algorithm with the use of an optimal adaptation path. One could think of a reinforcement approach that learns from the given optimal adaptation path. Also in the Discrete Event Simulation (DES) there was only one client considered. It is also necessary to find out how this adaptation logic behaves in a scenario where more clients rival for the available goodput in terms of fairness.

Appendix A

Result Database

The result database from the Discrete Event Simulation is defined in the following table A.1.

Column Name	**Description**
Run identifier	
run_id	Unique run identifier
Video description	
video_name	Name of the YouTube video
video_length	Duration of the video in s
video_avg_bitrate_quality1	Average bit-rate of quality representation 1
video_avg_bitrate_quality2	Average bit-rate of quality representation 2
video_avg_bitrate_quality3	Average bit-rate of quality representation 3
video_avg_bitrate_quality4	Average bit-rate of quality representation 4
video_avg_bitrate_quality5	Average bit-rate of quality representation 5
Goodput pattern description	
pattern_avg_throughput	Average bit-rate of the goodput pattern

pattern_timeshift	Timeshift that was used to create this goodput pattern
Simulation results	
sim_avg_quality	Average playback quality
sim_nr_of_switches	Number of quality switches
sim_nr_of_stallings	Number of stalling events
sim_avg_stalling_time	Average stalling duration of a stalling event
sim_total_stalling_time	Cumulative stalling duration
sim_max_stalling_time	Maximum stalling duration
sim_avg_buffer_level	Average buffer level
sim_max_buffer_level	Maximum buffer level
sim_startup_nr_of_stallings	Number of stalling events that occurred in the first 60s of the scenario
sim_startup_nr_of_switches	Number of quality switches during the first 60s of the scenario
sim_startup_total_stalling_time	Cumulative stalling time in the first 60s of the scenario
sim_endphase_nr_of_stallings	Number of stalling event that occurred in the last 60s of the scenario
sim_endphase_nr_of_switches	Number of quality switches during the last 60s of the scenario
sim_endphase_total_stalling_time	Cumulative stalling time in the last 60s of the scenario
sim_critbuff_time	Total time the buffer level was lower than 5s of playback segments
sim_lowbuff_time	Total time the buffer level was lower than 10s of playback segments
sim_riskybuff_time	Total time the buffer level was lower than 15s of playback segments
sim_healthybuff_time	Total time the buffer level was greater than or equal to 15s of playback segments
sim_bloatedbuff_time	Total time the buffer level was greater than 60s of playback segments
sim_jumps_1_levels	Number of quality switches that switched over 1 quality level

sim_jumps_2_levels	Number of quality switches that switched over 2 quality levels
sim_jumps_3_levels	Number of quality switches that switched over 3 quality levels
sim_jumps_4_levels	Number of quality switches that switched over 4 quality levels
Optimization results	
opt_avg_quality	Average playback quality
opt_nr_of_switches	Number of quality switches
opt_avg_buffer_level	Average buffer level
opt_max_buffer_level	Maximum buffer level
opt_critbuff_time	Total time the buffer level was lower than 5s of playback segments
opt_lowbuff_time	Total time the buffer level was lower than 10s of playback segments
opt_riskybuff_time	Total time the buffer level was lower than 15s of playback segments
opt_healthybuff_time	Total time the buffer level was greater than or equal to 15s of playback segments
opt_bloatedbuff_time	Total time the buffer level was greater than 60s of playback segments
opt_jumps_1_levels	Number of quality switches that switched over 1 quality level
opt_jumps_2_levels	Number of quality switches that switched over 2 quality levels
opt_jumps_3_levels	Number of quality switches that switched over 3 quality levels
opt_jumps_4_levels	Number of quality switches that switched over 4 quality levels
Buffer parameter	
buff_D	D-policy buffer parameter in [s]
Meta information	
meta_alpha	Value of α if α-parameter optimization was used otherwise not applicable (na)

meta_nn_model	Description of the Artificial Neural Network Model

Table A.1: The result database generated from the Discrete Event Simulation.

List of Figures

List of Tables

Appendix B

Notation and Acronyms

Acronym	Definition
ANN	Artificial Neural Network
DASH	Dynamic Adaptive Streaming over HTTP
HAS	HTTP Adaptive Streaming
HASBRAIN	HTPP Adaptive Streaming for dynamic Bit-Rate change aware Artificial Intelligence
HTTP	Hyper Text Transfer Protocol
NN	Neural Network
NNC	Neural Network Classification
QoE	Quality of Experience
RBF	Radial basis function kernel
SVC	Support Vector Classification
SVM	Support Vector Machine

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[BBDM+13] Kjell Brunnström, Sergio Ariel Beker, Katrien De Moor, Ann Dooms, Sebastien Egger, Marie-Neige Garcia, Tobias Hossfeld, Satu Jumisko-Pyykkö, Christian Keimel, Mohamed-Chaker Larabi, et al. Qualinet white paper on definitions of quality of experience. 2013.

[BL97] Avrim L Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.

[C+15] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[CLF+13] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design of a q-learning-based client quality selection algorithm for http adaptive video streaming. In *Adaptive and Learning Agents Workshop, part of AAMAS2013 (ALA-2013)*, pages 30–37, 2013.

[CLF+14] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design and optimisation of a (fa) q-learning-based http adaptive streaming client. *Connection Science*, 26(1):25–43, 2014.

[CLFDT14] Maxim Claeys, Steven Latre, Jeroen Famaey, and Filip De Turck. Design and evaluation of a self-learning http adaptive video streaming client. *IEEE communications letters*, 18(4):716–719, 2014.

[DVVB+13] Danny De Vleeschauwer, Harish Viswanathan, Andre Beck, Steve Benno, Gang Li, and Ray Miller. Optimization of http adaptive streaming over mobile cellular networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 898–997. IEEE, 2013.

[FWL06] Yu Fei, Vincent WS Wong, and Victor Leung. Efficient qos provisioning for adaptive multimedia in mobile communication networks by reinforcement learning. *Mobile Networks and Applications*, 11(1):101–110, 2006.

[HCL+03] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[HN04] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004.

[HSS+15] Tobias Hoßfeld, Michael Seufert, Christian Sieber, Thomas Zinner, and Phuoc Tran-Gia. Identifying qoe optimal adaptation of http adaptive streaming based on subjective studies. *Computer Networks*, 81:320–332, 2015.

[HSSZ14] Tobias Hoßfeld, Michael Seufert, Christian Sieber, and Thomas Zinner. Assessing effect sizes of influence factors towards a qoe model for http adaptive streaming. In *Quality of Multimedia Experience (QoMEX), 2014 Sixth International Workshop on*, pages 111–116. IEEE, 2014.

[kNe] k-Nearest Neighbors Classification. `http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html`. Accessed: 2017-06-26.

[KPD90] Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. *Neurocomputing: algorithms, architectures and applications*, 68(41-50):71, 1990.

[LBOM12] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[MCG16] Virginia Martín, Julián Cabrera, and Narciso García. Evaluation of q-learning approach for http adaptive streaming. In *Consumer Electronics (ICCE), 2016 IEEE International Conference on*, pages 293–294. IEEE, 2016.

[MHH+] Christian Moldovan, Tobias Hoßfeld, Korbinian Hagn, Christian Sieber, and Wolfgang Kellerer. Keep calm and don't switch: About the relationship between switches and quality in has.

[MLT12] Christopher Müller, Stefan Lederer, and Christian Timmerer. An evaluation of dynamic adaptive streaming over http in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video*, pages 37–42. ACM, 2012.

[MM16] Christian Moldovan and Florian Metzger. Bridging the gap between qoe and user engagement in http video streaming. In *Teletraffic Congress (ITC 28), 2016 28th International*, volume 1, pages 103–111. IEEE, 2016.

[MQGW12] Konstantin Miller, Emanuele Quacchio, Gianluca Gennari, and Adam Wolisz. Adaptation algorithm for adaptive streaming over http. In *Packet Video Workshop (PV), 2012 19th International*, pages 173–178. IEEE, 2012.

[MSH+16] Christian Moldovan, Christian Sieber, Poul Heegaard, Wolfgang Kellerer, and Tobias HoßFeld. Youtube can do better: Getting the most out of video adaptation. In *Teletraffic Congress (ITC 28), 2016 28th International*, volume 3, pages 7–12. IEEE, 2016.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[SES+15] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hobfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2015.

[SHHK16] Christian Sieber, Poul E. Heegaard, Tobias Hoßfeld, and Wolfgang Kellerer. Sacrificing efficiency for quality of experience: YouTube's redundant traffic behavior. In *IFIP Networking 2016 Conference (Networking 2016)*, Vienna, Austria, May 2016.

[Sod11] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[SS01] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[sup] Supplemental material to the alpha parameter influence evaluation. `github.com/csieber/alpha-dataset`.

[SVM] SVM: Maximum margin separating hyperplane. `http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html`. Accessed: 2017-06-26.

[vdHPC+15] Jeroen van der Hooft, Stefano Petrangeli, Maxim Claeys, Jeroen Famaey, and Filip De Turck. A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients. In *Integrated Network Management*

(IM), 2015 IFIP/IEEE International Symposium on, pages 131–138. IEEE, 2015.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.