Self-learning Optimization of Adaptive Video Streaming
Parameters

Jeroen van der Hooft

Supervisors: Prof. dr. ir. Filip De Turck, Dr. Jeroen Famaey
Counsellors: Ir. Maxim Claeys, Stefano Petrangeli

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de ingenieurswetenschappen: computerwetenschappen
Self-learning Optimization of Adaptive Video Streaming
Parameters

Jeroen van der Hooft

Supervisors: Prof. dr. ir. Filip De Turck, Dr. Jeroen Famaey
Counsellors: Ir. Maxim Claeys, Stefano Petrangeli

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de ingenieurswetenschappen: computerwetenschappen

Department of Information Technology
Chairman: Prof. dr. ir. Daniël De Zutter
Faculty of Engineering and Architecture
Academic year 2013-2014
Foreword

Upon completing this thesis dissertation, I cannot help but feel a sense of relief. It symbolizes the end of an era, a chapter in my life that now has finally been closed. I want to seize this opportunity to thank a number of people who helped me along the way. First of all I want to express my gratitude towards my supervisor Filip De Turck. It is thanks to his unceasing support and belief in me that this dissertation has ever seen the light of day. A great deal of thanks is owed to my counsellors Jeroen Famaey, Maxim Claeys and Stefano Petrangeli as well, for their indispensable advice and feedback during the past couple of months. Grazie mille!

On a more personal level I want to thank my mother, stepfather and sister for their words of encouragement. They have been pushing me to reach goals I had never thought possible. A big thanks goes to my roommates Thomas and Lawrence as well, for helping me to enjoy my life as a student. Without them, things simply would not have been the same. I also want to express my gratitude towards Geert, Nancy, Jolien and Jona, for their motivation and encouragement over the last couple of years. Their unconditional acceptance and support are a gift for life. A final word of thanks goes to my friends in Maldegem’s table tennis club, for the camaraderie I look forward to every single week. You rock, guys!

Jeroen van der Hooft, June 2014
Permission for Usage

"The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the limitations of the copyright have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation."

Jeroen van der Hooft, June 2014
Self-learning Optimization of Adaptive Video Streaming Parameters

by

Jeroen van der Hooft

Master’s dissertation submitted in order to obtain the academic degree of Master of Science in de ingenieurswetenschappen: computerwetenschappen

Academic year 2013–2014

Supervisors: Prof. dr. ir. Filip De Turck, Dr. Jeroen Famaey
Counsellors: Ir. Maxim Claeys, Stefano Petrangeli

Faculty of Engineering and Architecture
Ghent University

Department of Information Technology
Chairman: Prof. Dr. Ir. D. De Zutter

Abstract

HTTP Adaptive Streaming (HAS) is becoming the de-facto standard for Over-The-Top video streaming. Even though today’s results are promising, one drawback is that current implementations are generally hard coded. Fixed parameter values are used to provide a decent Quality of Experience (QoE) under all circumstances, resulting in suboptimal solutions. By adaptively changing parameters however, results can be significantly improved. In this master dissertation, we show how the concept of reinforcement learning can be applied in HAS solutions to provide the user both with an acceptable QoE and a low play-out delay. A self-learning client is proposed, adaptively changing the parameter configuration of two existing rate adaptation algorithms: the Microsoft IIS Smooth Streaming algorithm and the Fair In-Network Enhanced Adaptive Streaming algorithm by Petrangeli et al. [1][2]. Results indicate that this approach is indeed useful when video is streamed under changing network conditions.

Key Words

HTTP Adaptive Streaming, Rate Adaptation, Reinforcement Learning, Microsoft IIS Smooth Streaming, Fair Adaptive Streaming, Quality of Experience
I. INTRODUCTION

Over the last years, delivery of multimedia content has become more prominent than ever. To enable video streaming over the best-effort Internet, the concept of HTTP Adaptive Streaming (HAS) has recently been introduced. Even though results are promising, current implementations are not capable of dealing with a highly variable network environment. One reason for this is the fact that implementations are generally hard coded, optimized to deal with specific network conditions. This leads to unsatisfactory results, since true adaptation to a changing environment is not possible.

In this paper, we address this issue for two existing rate adaptation algorithms: the Microsoft IIS Smooth Streaming (MSS) algorithm and the Fair In-Network Enhanced Adaptive Streaming algorithm by Petrangeli et al. [1][2]. We propose an HAS client that incorporates the concept of reinforcement learning (RL) in order to decide upon the optimal parameter configuration based on network conditions.

The remainder of this paper is structured as follows. The concept of HAS is discussed in Section II, along with key features of the MSS and FINEAS algorithms. The RL-based approach is presented in Section III, discussing all components required. Results are discussed in Section IV, before coming to final conclusions in Section V.

II. HTTP ADAPTIVE STREAMING

In HAS, video content is temporally segmented and encoded at different quality levels. A manifest file is maintained by the HAS server, which contains information concerning the segments and the available quality levels. Based on this information, the client can request the next segment to the server, after the previous segment has been completely downloaded. The client decodes all segments and plays back the sequence of chunks in linear order. The main advantage of this approach is that the client can decide the quality level of the next segment to download. A quality selection heuristic is used for this purpose, basing its decision on criteria such as the available bandwidth and buffer filling. This allows the client to adapt to network conditions and provide the user with a high QoE.

An extensive analysis of the two algorithms revealed that performance is different for a fixed and a variable bandwidth scenario. When the available bandwidth is fixed, a small buffer size is sufficient to provide the user with an acceptable QoE. When the available bandwidth is variable, a large buffer size is required to prevent play-out freezes and a decrease in the selected quality level. This however comes at the cost of a higher average buffer filling, leading to an increased play-out delay in a live TV scenario. We propose a self-learning HAS client that dynamically adapts its parameter configuration based on network conditions, to provide both an acceptable QoE and the lowest possible buffer filling level.

III. REINFORCEMENT LEARNING

RL is an area of machine learning in which an agent can only interact with its environment through a set of specified actions. The agent, which can be anything ranging from a robot to an elevator scheduler, does not
need any a priori knowledge of this environment, and evaluates its actions based on an assigned, numerical reward. The agent’s goal is to learn the optimal action to take in a given environmental state, in order to maximize a cumulative numerical reward [3].

We propose a RL-based HAS client, which uses a Q-learning algorithm to learn the optimal parameter configuration for different network conditions. The state of the agent is defined using two metrics: the average available bandwidth and the average difference between bandwidth samples. The former is required because it has a significant impact on the perceived QoE, while the latter is required to discriminate between a variable and a fixed bandwidth scenario. The action set consists of several parameter configurations, selected based on an extensive analysis. The reward is defined as a linear combination of two components: the estimated Mean Opinion Score (MOS, a metric for the QoE [4][5]) and the buffer filling level, evaluated over a time window. Using an appropriate trade-off weight between these two components, we attempt to provide the user with an acceptable QoE at all times, with a low buffer filling when the perceived bandwidth is fixed.

IV. Evaluation and Discussion

To evaluate the performance of the MSS and FINEAS quality selection heuristics and our RL-based approach, a simple network topology was modelled using the NS-3 network simulator2. It consists of a single HAS client, streaming the Big Buck Bunny video trace from a dedicated HAS server. The video trace consists of 299 segments, each 2 seconds of length and encoded at seven different quality levels. The learning phase of the agent consists of a large number of randomly generated bandwidth traces, simulating either a highly variable or a fixed bandwidth pattern. Afterwards, results were evaluated over 50 bandwidth traces, both in a variable and a fixed bandwidth scenario.

Preliminary results indicated that the FINEAS algorithm is more suitable to provide a high average MOS and a low average buffer filling than the MSS algorithm. Using this algorithm, four actions were defined in the action set of the Q-learning agent, each with a different buffer size and parameters optimized to maximize the average MOS for this buffer size. Results for each of these actions individually are presented in Table I, along with results for the proposed learning-based approach.

Using a fixed parameter configuration, the optimal configuration in the fixed bandwidth scenario comes with a buffer size of 4 seconds. In this case, the lowest average buffer filling is observed, while the QoE is acceptable. In a highly variable bandwidth scenario however, the average MOS is significantly lower because of a large number of freezes and quality drops. Maximizing the QoE, the best configuration for this scenario comes with a buffer size of 10 seconds. The average buffer filling is however significantly higher in a fixed bandwidth scenario, which is not necessary in order to provide an acceptable QoE.

Using our approach, the average buffer filling in the fixed bandwidth scenario is only 2.880 seconds, close to the average buffer filling level of 1.981 seconds for a buffer size of 4 seconds. In contrast to this fixed parameter configuration however, our approach still provides an acceptable QoE when the perceived bandwidth is highly variable, with an average MOS of 2.863 (+32.06%). Compared to fixed parameter configurations with a larger buffer size, the most important advantage of our approach is a significantly lower average buffer filling.

Even though results are promising, there is room for improvement. When the perceived bandwidth is highly variable, the observed MOS is 14.69% lower than for a fixed buffer size of 10 seconds, while the average buffer filling is 45.38% higher than for a buffer size of 4 seconds when the perceived bandwidth is fixed. In general however, we conclude that our approach is able to find a good trade-off between a high QoE and a low play-out delay.

V. Conclusions

In this master’s thesis, the concept of RL was successfully introduced in traditional HAS implementations to adaptively change the parameter configuration according to bandwidth conditions. Using the presented Q-learning algorithm, it is possible to achieve both an acceptable average MOS when the available bandwidth is highly variable, as a significantly lower play-out delay when the available bandwidth is fixed. Future work will focus on further improving these results and extending the proposed approach to a multi-client scenario.

REFERENCES


TABLE I

<table>
<thead>
<tr>
<th>BS [s]</th>
<th>PT [s]</th>
<th>BT [s]</th>
<th>Variable MOS BF [s]</th>
<th>Fixed MOS BF [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2.168 1.954</td>
<td>3.731 1.981</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>3.019 3.950</td>
<td>3.895 3.968</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>3.281 5.397</td>
<td>4.034 5.543</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>8</td>
<td>3.356 6.397</td>
<td>4.032 6.748</td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
<td></td>
<td>2.863 4.380</td>
<td>3.719 2.880</td>
</tr>
</tbody>
</table>

2http://www.nsnam.org
Zelflerende Optimalisatie van Adaptieve Video Streaming Parameters

Jeroen van der Hooft

Begeleider(s): prof. dr. ir. Filip De Turck, dr. Jeroen Famaey, ir. Maxim Claeys, Stefano Petrangeli

Abstract—HTTP Adaptive Streaming (HAS) wordt langzaam maar zeker de de-facto standaard voor Over-The-Top video streaming. Resultaten zijn reeds veelbelovend, maar een groot deel blijft dat huidige implementaties over het algemeen hardgecodeerd zijn. Vastgelegde parameterwaarden worden gebruikt om een aanvaardbare Quality of Experience (QoE) aan te bieden onder alle omstandigheden, wat tot suboptimale resultaten leidt. Door waarden echter dynamisch aan te passen, kunnen resultaten significant verbeterd worden. In dit artikel wordt besproken hoe reinforcement learning kan worden toegepast in reeds bestaande HAS implementaties, om zowel een hoge QoE als een lage afspelvertraging te voorzien. Een zelflerende client wordt voorgesteld, die de parameterconfiguratie van twee bestaande rate adaptation algoritmen dynamisch aanpast: het Microsoft IIS Smooth Streaming algoritme en het Fair In-Network Enhanced Adaptive Streaming algoritme van Petrangeli et al. Resultaten tonen aan dat deze aanpak inderdaad voordeel is wanneer video wordt gestreamd bij veranderende omstandigheden in het netwerk.

Sleutelwoorden—HTTP Adaptive Streaming, Microsoft Smooth Streaming, Fair In-Network Enhanced Adaptive Streaming, Reinforcement Learning, Quality of Experience

I. INLEIDING

E levering van multimedia-inhoud werd de afgelopen jaren belangrijker dan ooit. Om video streaming via het best-effort Internet mogelijk te maken, werd het concept van HTTP Adaptive Streaming (HAS) recent geïntroduceerd. Hoewel resultaten reeds veelbelovend zijn, kunnen huidige implementaties niet omgaan met veranderende netwerkcondities. Een reden hiervoor is dat de meeste implementaties hardgecodeerd zijn, waarbij parameters geoptimaliseerd zijn voor specifieke netwerkomstandigheden. Dit leidt tot onvoldoende resultaten, aangezien de algoritmen zich niet kunnen aanpassen aan veranderende omstandigheden.

In dit artikel bieden we een oplossing voor dit probleem, voor twee reeds bestaande bestaande algoritmen: het Microsoft IIS Smooth Streaming (MSS) algoritme en het Fair In-Network Enhanced Adaptive Streaming (FINEAS) algoritme van Petrangeli et al. Een HAS-client wordt voorgesteld die het concept van reinforcement learning (RL) gebruikt om de optimale parameterconfiguratie aan te leren onder verschillende omstandigheden voor de beschikbare bandbreedte.

Het artikel is ingedeeld als volgt. HAS wordt besproken in Sectie II, samen met de belangrijkste kenmerken van de MSS en FINEAS algoritmen. De RL-gebaseerde aanpak wordt vervolgens gepresenteerd in Sectie III, waarbij in detail ingegaan wordt op alle nodige elementen. Resultaten worden besproken in Sectie IV, alvorens tot conclusies te komen in Sectie V.

II. HTTP ADAPTIVE STREAMING

In HAS worden videofragmenten temporeel gesegmenteerd en gecodeerd op verschillende kwaliteitsniveaus. Een manifest wordt beheerd door een HAS server, die informatie bevat over de segmenten en de beschikbare kwaliteitsniveaus. Op basis van deze informatie kan de client het volgende segment verzoeken aan de server, nadat het vorige segment volledig werd gedownload. De client decodeert alle segmenten en speelt deze in lineaire volgorde af. Het belangrijkste voordeel van deze aanpak is dat de client het kwaliteits niveau van het volgende segment zelf kan beslissen. Hierdoor kan de client zich aanpassen aan netwerkomstandigheden en de gebruiker voorzien van een hoge QoE.

In de MSS heuristiek zijn de belangrijkste parameters de buffergrootte en de panic, lower en upper thresholds, die de bufferinhoud actief controleren. De kwaliteitsselectie is gebaseerd op de huidige beschikbare bandbreedte en de bufferinhoud. In de FINEAS heuristiek zijn de belangrijkste parameters de buffergrootte, de panic threshold en een streefdool voor de bufferinhoud. De kwaliteitsselectie is gebaseerd op een utility functie, een maatstaf voor de QoE.

Een uitgebreide analyse van de twee algoritmen besloot dat in verschillende experimenten voor een scenario met een variabele en een scenario met een vaste bandbreedte. Wanneer de beschikbare bandbreedte vast is, volstaat een kleine buffer om de gebruiker te voorzien van een aanvaardbare QoE. Wanneer de beschikbare bandbreedte variabel is, is een grote buffer nodig om haperingen en en een daling van het geselecteerde kwaliteits niveau te voorkomen. Dit gaat echter ten koste van een hogere bufferinhoud, wat leidt tot een grotere afspelvertraging in een live TV scenario. Wij stellen een zelflerende HAS-client voor die de parameterconfiguratie dynamisch aanpast op basis van netwerkcondities, om zowel een aanvaardbare QoE als de laagste mogelijke bufferinhoud te behalen.
III. REINFORCEMENT LEARNING

RL is a form of machine learning, where an agent alone can communicate with its environment through actions. The agent has no a priori knowledge of this environment, and evaluates its actions based on a reward signal. The agent learns to optimize performance by choosing the best actions to maximize its expected reward.

We define a reinforcement learning (RL) function as

\[ Q(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t, a_t] \]

where \( s_t \) and \( a_t \) are the current state and action, and \( R_t \) is the immediate reward. The goal of the agent is to learn a policy \( \pi \) that maximizes the expected cumulative reward.

We evaluate the performance of our RL-based HAS client over various scenarios with variable and fixed bandwidth. The results show that our approach outperforms traditional HAS implementations in terms of MOS.

IV. EVALUATION AND DISCUSSION

We evaluate our approach using two traditional HAS implementations, FINEAS and MSS. The results show that our approach outperforms both in terms of MOS and computational overhead.

We conclude that our RL-based approach is a promising method for improving HAS performance.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
BS [s] & PT [s] & BT [s] & BS MOS BF [s] & Vast MOS BF [s] \\
\hline
4 & 0 & 2 & 2.168 & 1.954 \\
6 & 2 & 4 & 3.019 & 3.950 \\
8 & 2 & 6 & 3.281 & 5.397 \\
10 & 2 & 8 & 3.536 & 6.397 \\
\hline
Zelflerend & & & 2.863 & 4.380 \\
\hline
\end{tabular}
\caption{TABEL I}
\end{table}

\section{Conclusions}

In this thesis, we introduced a novel approach to improving HAS performance using RL. Our results show that this approach outperforms traditional implementations in terms of MOS and computational overhead.

Wanneer een vaste parameterconfiguratie gebruikt wordt, is de optimale configuratie voor een vaste bandbreedte deze met een buffergrootte van 4 seconden. In dit geval wordt de laagste gemiddelde bufferinhoud waargenomen, terwijl de QoE aanvaardbaar is. Door het optreden van haperingen en kwaliteitsdalingen is de gemiddelde MOS echter aanzienlijk lager wanneer de bandbreedde variabel is, zodat een buffergrootte van 10 seconden hier meer aangewezen is. Bij deze configuratie is de gemiddelde bufferinhoud echter aanzienlijk hoger wanneer de bandbreedde vast is, wat niet noodzakelijk is om een aanvaardbare QoE aan te bieden.

Gebruik makend van onze oplossing, is de gemiddelde bufferinhoud bij vaste bandbreedte slechts 2,880 seconden, wat dicht bij het gemiddelde van 1,981 seconden voor een buffergrootte van 4 seconden ligt. In tegenstelling tot deze vaste parameterconfiguratie, biedt onze aanpak nog steeds een aanvaardbare QoE aan wanneer de beschikbare bandbreedte variabel is, met een gemiddelde MOS van 2,863 (+32, 06%). In vergelijking met een vaste configuratie met een grotere buffer, is het belangrijkst voordeel van onze aanpak dat de gemiddelde bufferinhoud significant lager is.

Hoewel resultaten veelbelovend zijn, is er nog steeds ruimte voor verbetering. Wanneer de beschikbare bandbreedte variabel is, is de waargenomen MOS 14,69% lager dan bij een vaste buffergrootte van 10 seconden. Wanneer de beschikbare bandbreedte vast is, is de gemiddelde bufferinhoud 45,38% hoger dan bij een buffergrootte van 4 seconden. In het algemeen kunnen we echter wel stellen dat onze aanpak in staat is om een goede afweging te vinden tussen een hoge QoE en een lage afspeelvertraging.
REFERENZES

Contents

1 Introduction ........................................ 1
   1.1 Research Objectives ..................................... 2
   1.2 Thesis Outline ........................................ 3

2 Related Work ........................................ 5
   2.1 Reinforcement Learning ................................ 5
      2.1.1 Basic Model ........................................ 5
      2.1.2 Q-learning ........................................ 6
      2.1.3 Exploration Policy ................................ 7
      2.1.4 Applications ...................................... 9
   2.2 HTTP Adaptive Streaming ............................... 9
      2.2.1 Traditional Streaming ............................. 9
      2.2.2 Progressive Download ............................ 10
      2.2.3 HTTP Adaptive Streaming .......................... 10
      2.2.4 HAS Quality Selection Heuristics .................. 11
      2.2.5 Reinforcement Learning in HTTP Adaptive Streaming .................. 13
   2.3 Conclusion .......................................... 14

3 Learning-based HAS Parameter Optimization .............. 15
   3.1 Experimental Setup .................................. 15
   3.2 Quality of Experience ................................ 17
   3.3 Microsoft IIS Smooth Streaming ...................... 18
      3.3.1 Quality Selection Heuristic ...................... 18
      3.3.2 Impact of Heuristic Parameters .................. 20
         3.3.2.1 Impact of the Buffer Size .................. 20
# CONTENTS

3.3.2.2 Impact of the Thresholds .................................. 22
3.3.3 Adaptively Changing the Parameter Configuration ......... 24

3.4 Fair In-Network Enhanced Adaptive Streaming .................. 27
3.4.1 Quality Selection Heuristic .................................. 27
3.4.2 Impact of Heuristic Parameters ............................... 28
3.4.3 Adaptively Changing the Parameter Configuration ........... 32

4 Evaluation and Discussion ........................................... 33
4.1 Bandwidth Discrimination ........................................ 33
4.2 Microsoft IIS Smooth Streaming ................................. 37
4.2.1 Parameter Analysis ........................................... 37
4.2.1.1 Decision Interval and Reward Window ................. 38
4.2.1.2 Trade-off Parameter .................................... 42
4.2.1.3 Q-learning Parameters ................................ 43
4.2.2 Results ....................................................... 45
4.3 Fair In-Network Enhanced Adaptive Streaming ................. 48
4.3.1 Parameter Analysis ........................................... 48
4.3.2 Results ....................................................... 49
4.4 Comparison ...................................................... 51

5 Conclusion .......................................................... 53
5.1 General Conclusions ............................................... 53
5.2 Future Work ....................................................... 54
List of Acronyms

ADB  Advanced Digital Broadcast
BF   Buffer Filling
BS   Buffer Size
BT   Buffer Target
DASH Dynamic Adaptive Streaming over HTTP
FINEAS Fair In-Network Enhanced Adaptive Streaming
HAS  HTTP Adaptive Streaming
HDS  HTTP Dynamic Streaming
HLS  HTTP Live Streaming
HTTP Hypertext Transfer Protocol
IIS  Internet Information Services
IPTV Internet Protocol Television
MDP  Markov Decision Process
MOS  Mean Opinion Score
MPEG Motion Picture Expert Group
MS-WMSP Microsoft Windows Media HTTP Streaming Protocol
MSS  Microsoft IIS Smooth Streaming
OTT  Over-The-Top
List of Acronyms

QoE  Quality of Experience

RL   Reinforcement Learning

RTSP Real-Time Streaming Protocol

VDBE Value-Difference Based Exploration
Chapter 1

Introduction

Over the last years, delivery of multimedia content has become more prominent than ever. Particularly, video streaming applications are responsible for more than half of the internet traffic \(^3\). In this context, two types of services can be distinguished. On the one hand, video streaming services such as Internet Protocol Television (IPTV), are offered and managed by a network provider over a dedicated and managed network. This allows the optimization of services to suit network and end-device capabilities, so that the Quality of Services (QoS) can be guaranteed. On the other hand, video streaming services can be offered over best effort networks by Over-The-Top (OTT) service providers. Even though the QoS is not guaranteed, this approach is gaining in popularity because of lower costs and cross-device flexibility. Furthermore, OTT services are highly stimulated by important content deliverers such as YouTube\(^1\) Netflix\(^2\) and MySpace\(^3\).

For OTT video streaming, the concept of HTTP Adaptive Streaming (HAS) is becoming the de-facto standard. In HAS, video content is temporally divided into segments with a typical length of 1 to 10 seconds, each encoded at different quality levels. Segments are dynamically requested by the HAS client, equipped with a rate adaptation algorithm to select the best quality level based on criteria such as the current available bandwidth and the video player’s buffer filling level. This allows the client to adapt to perceived network conditions and provide the user with a higher Quality of Experience (QoE).

\(^1\)http://www.youtube.com
\(^2\)https://www.netflix.com
\(^3\)https://www.myspace.com
HAS comes with several advantages. As for the provider, video content delivery is cheaper because no dedicated network elements are required. Better scalability is guaranteed, since quality selection is performed by clients in a distributed way. As for the end user, a smoother playback experience is generally perceived, as the client can adapt the requested bit rate to the available bandwidth. However, HAS solutions are susceptible to network congestion and high variations in the available bandwidth, since the video content is delivered over the best-effort Internet. This has a significant impact on the QoE, which depends among others on the average quality level, switches in the selected quality level and the occurrence of play-out freezes.

A large number of rate adaptation algorithms have recently been proposed, attempting to provide the user with a high QoE whilst taking into account current network conditions. These heuristics are generally hard coded, using parameters and threshold values that are optimized for specific network conditions. This prevents true adaptation to changing network environments, so that better solutions are required. Furthermore, many commercial HAS implementations focus on a video-on-demand scenario, in which case a large buffer size is used to avoid play-out freezes. When the focus is on a live TV scenario however, a low buffer size is used as the video play-out delay should be as low as possible. Current implementations generally use a fixed buffer size, and are therefore not capable of dealing with both scenarios.

1.1 Research Objectives

In this thesis dissertation, we attempt to address the issues described above for two existing rate adaptation algorithms: the Microsoft IIS Smooth Streaming algorithm (MSS) and the Fair In-Network Enhanced Adaptive Streaming (FINEAS) algorithm by Petrangeli et al. We propose the use of an HAS client that incorporates the concept of Reinforcement Learning (RL), in order to decide upon the optimal parameter configuration to use under different network conditions. In RL, the only way for an agent to learn about its environment is by interacting with it. The agent does this by performing actions, which are evaluated through a numerical reward. The goal of the agent is to learn the optimal action in a given environmental state, in order to maximize the cumulative numerical reward. In our case, actions are defined by possible parameter configurations for the quality selection heuristic, while the state of the environment

\footnote{FINEAS is an algorithm designed to provide fairness among clients in a multi-client scenario, while attempting to maximize the perceived QoE. The focus in this dissertation is however on a single-client scenario, so we are only interested in the quality selection heuristic at client side.}
is defined by certain properties of the perceived bandwidth. In contrast to traditional rate adaptation algorithms, the agent should learn the optimal parameter configuration to use under different network conditions. Furthermore, by assigning the right reward function, the client should be driven to achieve both a high QoE and a low buffer filling, which is important in a live TV scenario.

1.2 Thesis Outline

The remainder of this dissertation is structured as follows. Related work is presented in Chapter 2 discussing the concept of RL with a focus on the Q-learning algorithm. The HAS principle is presented as well, discussing a number of rate adaptation algorithms and solutions proposed in literature. In Chapter 3, the MSS and FINEAS algorithms are discussed in detail. Using a specific experimental setup and a model for the user’s QoE, performance of both algorithms is evaluated, showing the possible advantage of adaptively changing the parameter configuration. A number of optimizations are proposed, for which results are thoroughly discussed in Chapter 4. Finally, Chapter 5 presents final conclusions and future work.
1.2 Thesis Outline
Chapter 2

Related Work

In this chapter, an overview of related work is presented. The concept of RL is discussed in Section 2.1 with a focus on the Q-learning algorithm. The principle of HAS is elaborated upon in Section 2.2 discussing previous generations of video streaming and a number of HAS rate adaptation algorithms. The use of RL in the context of video streaming is shown as well, with a focus on HAS quality selection heuristics. Finally, a conclusion is drawn in Section 2.3.

2.1 Reinforcement Learning

RL is an area of machine learning in which an agent learns the optimal actions to take in a certain environment, in order to maximize a given numerical reward [4]. The agent, which can be anything ranging from a robot to an elevator scheduler, has no need for any a priori knowledge of the environment. In fact, the agent can only interact with its environment through a set of specified actions. By evaluating every taken actions and assigning an appropriate numerical reward, the agent is capable of learning the optimal behaviour without the need for supervision or complete models of the environment.

2.1.1 Basic Model

The basic RL model is typically formulated as a Markov decision process (MDP), which is a discrete time stochastic control process. Such a process is formally described as a 4-tuple \((S, A, P, R)\), where:
2.1 Reinforcement Learning

- $S$ is a finite set of states
- $A$ is a finite set of actions
- $P$ is a state transition probability matrix
- $R$ is a reward function

The RL scheme is shown in Figure 2.1. At each time step, the process is in an environmental state $s \in S$. A decision maker then chooses to perform an action $a \in A$, which causes the environment to move into a new state $s'$. The probability of a transition from state $s$ to state $s'$ by performing an action $a$ is given by:

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$ (2.1)

Note that the Markov property applies: the new state $s'$ is conditionally independent of any state that occurred before the previous state $s$. The state transition comes with a reward $r = R_a(s, s')$, which is used by the decision maker to evaluate its decision and update its knowledge about the quality of its decisions. By repeating these steps in an iterative way, the agent is able to learn how to behave in an optimal way without explicit knowledge about the environment. Moreover, by introducing the concept of future rewards, the agent can reason about long term consequences.

### 2.1.2 Q-learning

Q-learning is a model-free RL technique that was first introduced by Watkins in 1989 [6]. Typically a Q-table is used, where rows correspond to the state set $S$ and columns correspond to the action set $A$. For each state-action combination $(s, a)$ a Q-value $Q(s, a)$ is stored, which
reflects the quality of performing action $a$ when the environment is in state $s$. The Q-values are updated every time an action $a$ is taken in a state $s$, resulting in a reward $r$ and a new state $s'$:

$$Q(s, a) = Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$ \hspace{1cm} (2.2)

In this equation, $\alpha \in [0; 1]$ and $\gamma \in [0; 1]$ are the learning rate and the discount factor respectively. The former determines to what extent the agent learns from newly acquired information, the latter determines the importance of future rewards. Note that for $\alpha$ equal to 0, the agent does not learn anything, while for $\alpha$ equal to 1, the agent only considers the most recent information.

In this formulation, rewards are only accounted to the last action performed. To account rewards to actions taken further in the past, eligibility traces can be applied. These traces record which states have recently been visited and indicate the degree to which each state-action combination is eligible for undergoing learning changes when a new reward is perceived. A variable $e(s, a)$ is introduced for each state-action combination, to which the following update rule is applied:

$$e(x, y) = \begin{cases} 
1 + \gamma \lambda 
& \text{if } (x, y) = (s, a) \land Q(s, a) = \max_{a'} Q(s, a') \\
\gamma \lambda 
& \text{if } (x, y) \neq (s, a) \land Q(s, a) = \max_{a'} Q(s, a') \\
0 & \text{otherwise} 
\end{cases}$$ \hspace{1cm} (2.3)

Using this definition of eligibility traces, the updating rule for the Q-values can be defined as follows:

$$Q(s, a) = Q(s, a) + \alpha e(s, a) \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$ \hspace{1cm} (2.4)

In this equation, $\lambda \in [0; 1]$ is the eligibility trace-decay parameter. Higher values for this parameter lead to longer traces, so that more credit is given to actions taken further in the past. Note that for $\lambda$ equal to 0, Equation 2.4 is reduced to Equation 2.2.

### 2.1.3 Exploration Policy

One of the challenges during the learning process is finding the right balance between looking for new information (exploration) and using the current model to maximize the reward (exploitation). Although complex exploration methods have successfully been applied in the past, simpler methods are used in practice. One of these is the $\epsilon$-greedy approach, which was also
proposed by Watkins [6]. With a probability of $1 - \epsilon$ the agent chooses the action with the highest current Q-value, while a random action is selected with probability $\epsilon$. The parameter $\epsilon$ is application dependent, so it has to be fine-tuned to find a near-optimal value. Its value can also be changed over time: a high value can be used when learning has just started and the focus is on exploration, while a lower value can be used when the algorithm is more or less converged and the focus is on exploitation.

One drawback of the $\epsilon$-greedy approach is that all actions have an equal chance to be selected when the agent is exploring. One might however expect the estimated next-to-best action to have a higher chance of being chosen than the worst action. To overcome this issue, the Softmax exploration method was proposed by Sutton and Barto [4]. In this method, a Boltzmann distribution is most commonly used to rank the Q-values for a specific state. Based on this distribution, the selection probability $P(s,a)$ is calculated as follows:

$$P(s,a) = \frac{e^{Q(s,a)/\beta}}{\sum_{a'} e^{Q(s,a')/\beta}}$$

(2.5)

In this equation, $\beta$ is a strictly positive parameter called the Softmax inverse temperature. Low values for this parameter cause the actions to be nearly equiprobable, while high values lead to a higher selection probability for actions with a higher estimated Q-value. For $\beta \to +\infty$ the Softmax selection method is equal to the $\epsilon$-greedy approach.

Another approach that is often used is the Value-Difference Based Exploration (VDBE) by Tokic et al. [7] In this approach, the exploration probability is state dependent and can be written as $\epsilon(s)$. At the beginning of the learning phase, knowledge about the environment is still uncertain. This is indicated by strong fluctuations of the Q-value changes. Consequently, higher values are assigned to $\epsilon(s)$, so that exploration is encouraged. When changes in the Q-values decrease over time, meaning that the learning phase is almost finished, the values of $\epsilon(s)$ can be decreased so that exploitation is preferred. The VDBE approach was further extended by Tokic et al., using a Softmax action selection policy instead of $\epsilon$-greedy. Results show that under certain conditions, this VDBE-Softmax policy can outperform the $\epsilon$-greedy, Softmax and VDBE approach [8].

The initialization of the Q-table has a significant impact on the balance between exploration and exploitation. In some approaches the table is initialized at a default value, usually equal to 0. This makes sense, since the agent typically has no prior knowledge of the environment or the quality of possible actions. Using the same Q-values, all state-action combinations initially have
an equal chance of being selected. However, there is a disadvantage to this approach. When the reward function produces strictly negative values, unexplored state-action combinations typically have the highest Q-values, and therefore are more likely to be selected by the exploration policy. On the contrary, when the reward function produces strictly positive values, previously used actions are favoured. To overcome this issue, some approaches use domain knowledge to initialize the Q-tables more appropriately. Careful consideration is however required, since the initialization has an impact on the learning phase: when the magnitude of the Q-values is too high, exploration is limited and can prevent the algorithm from finding the optimal solution.

2.1.4 Applications

Many real-life applications of RL exist, most of them found in the area of automation and robot control. One such example is an elevator dispatching task, in which a number of elevators are servicing a number of floors. Using the concept of RL, Crites and Barto were able to outperform any existing elevator control system in terms of the average passenger waiting time [9]. Other examples are found in the industry, where robots learn how to move, how to manufacture and package fabric in a production process etc. In research by Muelling et al., robots were even able to learn how to play table tennis [10]. Even in aviation the concept of RL has been successfully applied. For instance, Abbeel et al. made it possible for a helicopter to fly autonomously [11]. Another popular research domain is the area of game-playing. Board games offer a fixed environment, yet often come with a high complexity. Well-known examples of RL-based game engines are TD-Gammon and KnightCap, designed to play backgammon and chess respectively [12, 13]. In general, we conclude that there are myriad possible applications of RL.

2.2 HTTP Adaptive Streaming

In the context of video streaming, three generations of media delivery methods are generally distinguished: traditional streaming, progressive download and HTTP adaptive streaming (HAS).

2.2.1 Traditional Streaming

In traditional streaming methods, the server sends data packets to a client at real-time rate only. This means that a video encoded at a certain bit rate will be streamed at approximately
2.2 HTTP Adaptive Streaming

Figure 2.2: HTTP Adaptive Streaming concept.

the same bit rate as well. The server only sends enough data to fill the buffer at client-side, so that no extra video is sent when the video is paused or stopped. This way, no bandwidth goes to waste. Using this approach is however not always recommended: if the bit rate exceeds the available bandwidth, play-out freezes will inevitably occur. Pausing the video offers no solution, since only a limited amount of video data will be available afterwards. Examples of traditional streaming protocols are the stateful Real-Time Streaming Protocol (RTSP) and the stateless HTTP-based Windows Media HTTP Streaming Protocol (MS-WMSP).

2.2.2 Progressive Download

A progressive download is a simple file download from a web server to a client, typically using the HTTP protocol. The download is progressive, meaning that the client can play the video while the download is still in progress. Web servers keep sending the video data until the download is complete, in contrast to streaming servers that generally stop sending data when a certain amount of video is available to the client. A progressive download allows the client to play the video more smoothly, yet is not always bandwidth-effective: when discarding a video after one minute, ten more minutes might have been downloaded already.

2.2.3 HTTP Adaptive Streaming

HAS represents the third generation of HTTP-based streaming solutions. An overview of the general concept is shown in Figure 2.2. The video content is temporally segmented and encoded at different quality levels. The segment duration generally varies between 1 to 10 seconds, depen-
2.2 HTTP Adaptive Streaming

A manifest file is maintained by the HAS server, which contains information concerning the segments and the available quality levels. Based on this information, the client can request the next segment to the HAS server after the previous segment has been completely downloaded. The client decodes all segments and plays back the sequence of chunks in linear order. The main advantage of HAS is that the client can decide at which quality level the next segment is requested. A quality selection heuristic is used for this purpose, basing its decision on criteria such as the perceived bandwidth and buffer filling. In this way, the client can adapt to network conditions and provide the user with a better video streaming experience. The latter is usually referred to as the user’s QoE, which is further discussed in Section 3.2.

Many rate adaptation algorithms have recently been proposed, some of which are discussed in the next section.

2.2.4 HAS Quality Selection Heuristics

A large number of HAS quality selection heuristics exist. Among commercial implementations, Microsoft’s IIS Smooth Streaming (MSS) is a well-known rate adaptation heuristic that uses two different states and a number of thresholds on the buffer filling to decide on the next quality level. It is provided as a feature in Microsoft’s IIS Media Services, an HTTP-based media delivery platform. In Apple’s HTTP Live Streaming (HLS), segments are loaded simultaneously at variable bit rates. A commercial implementation is provided, allowing video streaming to iPhone, iPod, iPad and Apple TV. Similar to Microsoft’s and Apple’s HAS implementations, Adobe’s HTTP Dynamic Streaming (HDS) provides streaming services in the latest versions of Flash Player and Flash Media Server. As most of these implementations tend to use the same architecture, the Motion Picture Expert Group (MPEG) proposed a single standard for the interfaces and protocols of rate adaptation algorithms, now known as Dynamic Adaptive Streaming over HTTP (DASH). Heuristics are however still implementation specific. Recently, the first commercial implementation was brought to the market by Advanced Digital Broadcast (ADB), a service provider in Switzerland.

The performance of these commercial implementations was evaluated in multiple studies. Akhshabi et al. showed that heuristics are not capable of dealing with highly variable bandwidth conditions, reacting too slow on peaks in the available bandwidth. This causes drops in the

\[^1\]Note that some approaches allow parallel downloading as well, such as the Parallel Adaptive HTTP Media Streaming approach by Liu et al. \[^{[14]}\]
buffer filling and unnecessary bit rate reductions. Müller et al. evaluated the performance of several commercial players on real mobile networks [20]. The authors pointed out that current commercial implementations are not yet capable of dealing with dynamic network conditions.

A large number of other quality selection heuristics have recently been proposed in literature. In order to tackle the issue of bandwidth estimations and other network-related conditions, Huang et al. proposed to select the appropriate video quality level based solely on the current buffer filling [21]. Even though results seem promising, the approach is only evaluated for a single client under certain bandwidth conditions, so that performance in a highly dynamic bandwidth environment should be questioned. Jarnikov et al. proposed the use of a MDP to train the quality selection heuristic based on network conditions and user preferences [22]. Parameters of the HAS heuristic are however fine-tuned in an off-line training process, preventing true adaptation to new network environments. The same is true for the research by Xiang et al., in which the optimal MDP solution is found through the use of dynamic programming [23]. Bandwidth transition probabilities are used in the learning process, so that an off-line training policy is required. Based on control-theory, De Cicco et al. proposed the use of a quality adaptation controller, in which feedback control is used to maximize the user’s QoE based on the available bandwidth and buffer filling [24]. A centralized approach is used, as feedback control is carried out at server side. This relieves the client from its quality selection task and provides both simplicity and modifiability. Zhou et al. proposed the use of a controller as well, yet selecting the quality level for a block of video segments which are downloaded in parallel from multiple servers [25]. A similar approach is used by Tian et al., in which decisions are based on the available bandwidth, the current video rate and buffer filling [26]. Even though results for these approaches seem promising, results are only evaluated in scenarios with low bandwidth variability. Furthermore, a centralized approach comes with two disadvantages: scalability issues arise when the number of clients increases and a modification to the classical HAS architecture is required, since clients no longer autonomously decide which quality level to select next.

While the goal in a single-client scenario is only to achieve the best QoE at client side, fairness should also be taken into account in a multi-client scenario. Research by Akhshabi et al. showed that competing players generally lead to three performance problems: (i) instability, meaning that bit rate levels are requested which cannot be sustained for long, (ii) unfairness, in which case one client observes a higher throughput than another and (iii) bandwidth underutilization, in which case the available bandwidth is not fully used [27]. According to the authors, the main
cause for unfairness is the synchronization among clients when downloading segments, which can lead to wrong bandwidth estimations. A first solution would be a centralized approach, where decisions regarding the most appropriate bit rate levels are made at server side. A decentralized approach is however more appropriate, since scalability issues arise when the number of clients increases. Bouten et al. proposed the use of an intermediate network proxy, solving a local optimization problem to decide upon the maximum bit rate level each client should request [28]. Mok et al. proposed a similar approach, in which the proxy estimates the available bandwidth within its respective network [29]. Each assigned client then selects the most appropriate quality level based on this measurement and its current buffer filling. Petrangeli et al. suggested the FINEAS approach, in which the system consists of network proxies that do not directly control the quality selection process at client side [2]. Instead, proxies simply estimate the fair bandwidth allocation for each video streaming client. Using the FINEAS rate adaptation heuristic, each client selects the next quality level in an attempt to optimize the user’s QoE and to achieve fairness among clients. Results indicate that fairness can indeed be significantly increased when compared to other state-of-the-art solutions, even when the available bandwidth is highly variable.

2.2.5 Reinforcement Learning in HTTP Adaptive Streaming

In order to dynamically adapt to network conditions, the concept of RL has recently been introduced in the HAS quality selection process. Menkovski et al. proposed the use of the linear gradient-descent SARSA(\(\lambda\)) technique to adaptively select the most appropriate quality level, based on the estimated bandwidth, the buffer filling and the position in the video stream [30]. Even though convergence is shown with respect to the QoE, performance is not compared with respect to other existing HAS implementations. Claeys et al. proposed the use of Q-learning to select the next quality level, based on the estimated bandwidth and the buffer filling [31]. Results show that the client is able to outperform deterministic algorithms such as MSS in several network environments. In a multi-client scenario, Petrangeli et al. suggested an approach in which each client learns to adaptively select the most appropriate quality level, maximizing a reward based both on its own QoE and on the QoE perceived by other clients [32]. To this end, a coordination proxy estimates all perceived rewards and generates a global signal that is sent periodically to all clients. Without explicit communication among agents, the algorithm is able to outperform both MSS and the algorithm proposed by Claeys et al. in a multi-client scenario.
2.3 Conclusion

Related work in the domain of HAS suggests that even though today’s results are promising, there still is room for improvement. Current implementations are generally hard coded, with parameter values optimized to perform decently in different network environments. While Menkovski, Claey, Petrangeli et al. introduced the concept of RL to adaptively select the next quality level, we propose a Q-learning algorithm that adaptively changes the parameter configuration for existing quality selection heuristics. In contrast to traditional rate adaptation algorithms, the agent can learn the optimal parameter configuration for different network environments. Focus in this dissertation will be on two scenarios for the available bandwidth. When high variations in the available bandwidth occur, a parameter configuration should be selected that leads to a high buffer filling. In this way, buffer starvation is actively prevented at the cost of a larger play-out delay. When the available bandwidth is more or less fixed, a parameter configuration should be selected that leads to a low buffer filling yet still provides the user with a high QoE. Using the right environmental model and reward function, we aim to provide both an acceptable QoE and an appropriate play-out delay at all times. In Chapter 3 we first discuss the MSS and FINEAS algorithms in detail, before presenting a learning-based optimization approach for both algorithms.
Chapter 3

Learning-based HAS Parameter Optimization

In this chapter, the performance of the MSS and FINEAS rate adaptation algorithms is evaluated and a number of optimizations are proposed. First, the experimental setup is discussed in Section 3.1. Next, a model for the QoE is presented in Section 3.2. In Sections 3.3 and 3.4, the MSS and FINEAS algorithms are discussed in detail and the impact of several parameters on the QoE and buffer filling is evaluated. Based on this evaluation, a number of optimizations are proposed and discussed in detail.

3.1 Experimental Setup

To evaluate the performance of the traditional quality selection heuristics and our RL-based solution, the network topology in Figure 3.1 was modelled the NS-3 network simulator\(^1\). It consists of an HAS server and a single client, connected through a link with a maximum capacity of 4Mbps. The performance of MSS, FINEAS and our learning-based optimizations are evaluated both in a highly variable and a fixed bandwidth scenario. To this end, several bandwidth traces were synthetically generated. In a variable bandwidth scenario, traces were constructed by simulating cross traffic over a 3Mbps link and measuring the available throughput at client side. The generated cross traffic consists of a sequence of bandwidth bursts, normally distributed between 0kbps and 2640kbps with a granularity of 264kbps. Each burst persists for a uniformly

\(^1\)http://www.nsnam.org
Table 3.1: Bit rates for the *Big Buck Bunny* video trace.

<table>
<thead>
<tr>
<th>Quality level</th>
<th>Bitrate [kbps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>427</td>
</tr>
<tr>
<td>3</td>
<td>608</td>
</tr>
<tr>
<td>4</td>
<td>866</td>
</tr>
<tr>
<td>5</td>
<td>1233</td>
</tr>
<tr>
<td>6</td>
<td>1636</td>
</tr>
<tr>
<td>7</td>
<td>2436</td>
</tr>
</tbody>
</table>

distributed amount of time, ranging from 1 to 300 seconds. This approach resulted in an available bandwidth ranging from 28kbps to 2812kbps, with an average of 1550kbps and a standard deviation of 463kbps. In a fixed bandwidth scenario, traces simply consist of a uniformly selected value for the available bandwidth, ranging from 350kbps to 3277kbps.

The video trace streamed by the client is *Big Buck Bunny*. The duration of a single episode is 598 seconds, corresponding to 299 segments with a fixed segment length of 2 seconds. Each of these segments has been encoded at seven different quality levels, for which the corresponding bit rates are presented in Table 3.1. To evaluate the performance of the traditional MSS and FINEAS algorithms, 50 episodes of this video are streamed using 50 different bandwidth traces, both for a variable and a fixed bandwidth scenario. When the concept of RL is introduced, even up to 3000 episodes are streamed in the learning phase of the agent. As this requires a lot of computing power, especially when a large number of experiments are performed at the same time, the High Performance Computing (HPC) infrastructure\(^2\) hosted at Ghent University was used.

\(^2\)http://www.ugent.be/hpc/
3.2 Quality of Experience

In the context of video streaming, several metrics for the Quality of Service (QoS) exist. Focus generally is on the network QoS (in terms of the average bandwidth, packet loss etc.) and the application QoS (defined using application specific metrics). These metrics however only indicate the technical performance and are not capable of evaluating the quality as perceived by the end user. The latter is usually referred to as the Quality of Experience (QoE) and will be the main criterion for the evaluation of the traditional algorithms and their optimizations.

Several metrics for the QoE exist. One of these is the Mean Opinion Score (MOS), which is an average score ranging from 1 (bad QoE) to 5 (excellent QoE) that was first introduced in the domain of telephony [33]. It can be obtained from subjective measurements, in which users are asked to assign a score for the quality of a video fragment. The MOS is then retrieved averaging all these scores. The main problem with subjective measurements is the need for human test subjects. For this reason, several studies have been conducted to estimate the MOS based on objective measurements.

De Vriendt et al. proposed a metric which depends on two factors: the average requested quality level and its standard deviation [34]. The authors showed that frequent quality switches have a negative impact on the user’s QoE, and thus should be kept limited. To maximize the MOS, the average segment quality level should be as high as possible, while the standard deviation should be as low as possible. The estimated MOS is computed as a linear combination of these two factors, for which parameters were fine-tuned based on results from subjective measurements.

Another important factor having an impact on the QoE, is the occurrence of video freezes. Mok et al. showed that this influence depends both on the number and the average length of freezes [35]. In their research, estimations are calculated using three discrete levels of freeze frequency and length. Using interpolation on these levels, Claeys et al. proposed the following continuous function to measure the impact of freezes [31]:

\[ \phi = \frac{7}{8} \max \left( \frac{\ln(F_{freq})}{6} + 1, 0 \right) + \frac{1}{8} \left( \min(FT_{avg}, 15) \right) \]  

(3.1)

In this equation, \( F_{freq} \) and \( FT_{avg} \) represent the frequency of freezes and the average length of freezes respectively. An estimation of the MOS is now possible through the combination of the average quality level, its standard deviation and the impact of freezes. Consider a video
fragment which consists of $K$ segments, $N$ quality levels and a played quality level $QL_k$ for any segment $k$. The average quality level $\mu$ and its standard deviation $\sigma$ are then given by:

$$
\mu = \frac{\sum_{k=1}^{K} QL_k}{KN}, \quad \sigma = \sqrt{\frac{\sum_{k=1}^{K} (\frac{QL_k}{N} - \mu)^2}{K - 1}}
$$

Equation 3.3 shows an estimation of the MOS. Its theoretical range is $[0.00;5.84]$, though in practice a range of $[0.00;5.06]$ was observed by Claeys et al.

$$
MOS_{est} = \max(5.67\mu - 6.72\sigma - 4.95\phi + 0.17, 0)
$$

Besides the QoE, other aspects of video streaming should be taken into account as well when evaluating the performance of rate adaptation heuristics. For instance, in a live TV scenario, the play-out delay and thus the buffer filling should be as low as possible. This generally leads to a lower QoE, because buffer starvations are more likely to occur and the lowest quality is selected more often. In the following sections, results are reported considering both the average MOS and the average buffer filling.

3.3 Microsoft IIS Smooth Streaming

In this section the MSS algorithm is discussed in more detail, based on an open source version included in the MSS video player\cite{original_code}. The impact of parameters on the average MOS and buffer filling is analyzed, and a learning-based optimization is proposed.

3.3.1 Quality Selection Heuristic

The goal of the quality-selection heuristic is to select the next quality level $q$ at which to download the next segment, based on the current buffer filling and the available bandwidth. The pseudocode of this algorithm is presented in Algorithm 1. Two states are defined: a buffering state, in which the buffer filling is increasing, and a steady state, in which the buffer filling is kept constant. With respect to the buffer filling, three thresholds are defined: the panic threshold $PT$, the lower threshold $LT$ and the upper threshold $UT$.

\cite{original_code} Original source code available from https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming/
Algorithm 1 The Microsoft IIS Smooth Streaming quality selection heuristic.

```plaintext
1: state ← BUFFERING
2: bufferFilling ← getBufferFilling()
3: currentQL ← getCurrentQualityLevel()
4: while segmentsAvailable() do
5:   if state ≡ BUFFERING then
6:     if bufferDecreasing() ∨ BufferSlowlyChanging() then
7:       ensure currentQL − 1 ≤ nextQL ≤ currentQL + 1
8:     nextQL ← getCalculatedQualityLevel()
9:   if bufferFilling ≥ LT + UT \frac{2}{2} then
10:      state ← STEADY
11: else if state ≡ STEADY then
12:   if bufferFilling = 0 then
13:     nextQL ← 1
14:   state ← BUFFERING
15: else if lastDownloadLate() then
16:     nextQL ← currentQL − 1
17:   else if bufferFilling < PT then
18:     nextQL ← 1
19:     state ← BUFFERING
20: else if bufferSlowlyChanging() then
21:   if bufferFilling < LT then
22:     nextQL ← currentQL − 1
23:   else if bufferFilling > UT then
24:     attemptQualityIncrease()
25:   else if bufferDecreasing() ∧ bufferFilling < LT then
26:     nextQL ← 1
27:   state ← BUFFERING
28: else
29:   attemptQualityIncrease()
30: return nextQL
31: end while
```

The quality selection is based on the state of the client. At the beginning of the video streaming, the algorithm is in the buffer state. When buffering, the next quality level is identified based on the current estimated bandwidth. The highest possible quality level is selected (line 8), unless the buffer filling is decreasing or slowly changing. In this case, possible choices are restricted based on the current quality level `currentQL` (lines 6-7). The heuristic remains in the buffering state as long as the buffer filling is lower than \( \frac{LT+UT}{2} \) (lines 9 and 10). When in the steady state, several scenarios are considered. When the video freezes because of buffer starvation, the lowest quality level is selected and the heuristic returns to the buffering state (lines 12-14).
3.3 Microsoft IIS Smooth Streaming

3.3.2 Impact of Heuristic Parameters

3.3.2.1 Impact of the Buffer Size

We first evaluate the performance of MSS for different buffer sizes. Famaey et al. empirically showed that values of 25%, 40% and 80% of the buffer size for the panic threshold, the lower threshold and the upper threshold respectively generally lead to good results [36]. Figure 3.2 shows the average MOS and buffer filling in function of the buffer size, when these threshold values are used. The figure illustrates the impact of the buffer size on the average MOS and buffer filling for thresholds defined by Famaey et al. [36]. When the download time is higher than expected, the quality level is simply decreased by one (lines 15-16). When the buffer filling drops below the panic threshold, the lowest quality level is selected and the heuristic returns to the buffering state (lines 17-19). This way, the client actively tries to prevent buffer starvation. When the buffer filling is slowly changing and is below the lower threshold, the quality level is decreased by one in order to increase the buffer again (lines 20-22). On the contrary, if the buffer filling is above the upper threshold, an attempt is made to increase the quality level (lines 23-24). If the buffer filling is not slowly changing, decreasing and below the lower threshold, measures are again taken to prevent buffer starvation: the lowest quality level is selected and the heuristic goes back to the buffering state (lines 25-27). In any other case, an attempt is made to increase the quality level (lines 28-29). At last, the next segment is downloaded at the selected quality level nextQL and the process is repeated when the new segment is completely downloaded.
3.3 Microsoft IIS Smooth Streaming

Figure 3.3: Impact of the buffer size on the average MOS and buffer filling, for thresholds optimized with respect to the MOS.

values are used. When the buffer size is only one segment, the algorithm always stays in the buffering state. The buffer filling level is zero, as segments are played out completely before requesting the next one. In this case, the lowest quality level is always selected. When a buffer size of two segments is used, the buffer filling reaches a maximum of one video segment when downloading the next segment, never exceeding the value of $\frac{LT + UT}{2} = 60\%$. The algorithm again stays in the buffering state, although a higher value for the quality level can be selected. We conclude that a buffer size of at least three segments is required. Surprisingly however, a lower MOS is perceived when this buffer size is used. This is explained by the fact that the thresholds that are defined using the ratios of 25, 40 and 80%, do not make sense: the algorithm constantly switches between the buffering and the steady state, and the upper threshold of 4.8 seconds for a buffer size of 6 seconds can never be exceeded. For this reason, thresholds should first be optimized with respect to the buffer size.

An extensive sweep of possible threshold combinations was explored. Note that thresholds can easily be discretized, as the buffer filling is always an integer times the segment size. Furthermore, it must hold that $0 < PT < LT \leq UT < BS - 2s$. When considering a buffer size of 20 seconds and a panic threshold of 2 seconds, for instance, this leads to 28 possible combinations. Figure 3.3 shows the MOS and buffer filling in function of the buffer size, when optimizing thresholds with respect to the MOS. We observe that by using the most appropriate threshold values, the average MOS is now - approximately - strictly increasing. Note that in some cases, the optimal threshold combination slightly differs for the variable and the fixed bandwidth scenario.
### 3.3 Microsoft IIS Smooth Streaming

#### 3.3.2 Impact of the Thresholds

Results indicate that the MOS hardly increases when a buffer size larger than 12 seconds is used. For this reason, we consider this buffer size to evaluate the impact of threshold parameters on the MOS and buffer filling. Note that, when a panic threshold of 2 seconds is considered, only six threshold combinations for the lower and upper threshold are valid, with values ranging from 4 to 8 seconds. As the heuristic aims for a buffer filling between this lower and upper threshold, we expect to find a range for the resulting average buffer filling from 4 to 8 seconds.

<table>
<thead>
<tr>
<th>PT [s]</th>
<th>LT [s]</th>
<th>UT [s]</th>
<th>Variable MOS</th>
<th>Variable BF [s]</th>
<th>Fixed MOS</th>
<th>Fixed BF [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2.722 0.645 7.537 0.430</td>
<td>3.565 1.029 8.233 0.974</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>2.828 0.743 7.607 0.424</td>
<td>3.711 1.016 8.146 0.976</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2.900 0.749 7.820 0.457</td>
<td>3.731 1.005 8.325 0.885</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2.955 0.596 8.386 0.225</td>
<td>3.694 1.005 8.45 0.577</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td>3.074 0.627 8.613 0.263</td>
<td>3.728 0.986 8.953 0.545</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>2.957 0.551 9.170 0.136</td>
<td>3.640 1.023 9.407 0.263</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Impact of thresholds on the MOS and buffer filling, for a buffer size of 12 seconds and a panic threshold of 2 seconds.

A significant difference is observed between the results for the two bandwidth scenarios. In a fixed bandwidth scenario, the average MOS hardly improves when a buffer size larger than 6 seconds is used. In fact, a maximum increase of 10.56% is observed. The average buffer filling however increases in a linear fashion, with a relative increase of 130.27% when a buffer size of 12 seconds is used. In a variable bandwidth scenario, the MOS is significantly increased when a larger buffer size is used. This behaviour is mainly explained by the fact that bandwidth variability often causes the download to take longer than expected, leading to buffer starvations when the buffer size is small. Furthermore, the lowest quality level is more likely to be selected, since the panic threshold is exceeded more often. This causes both a lower average quality level and a higher standard deviation. Using a buffer size larger than 6 seconds, an increase of up to 26.27% is observed. We conclude that in a fixed bandwidth scenario, a buffer size of 6 seconds is sufficient to provide an acceptable QoE with a low average buffer filling, while in a variable bandwidth scenario, a buffer size of 12 seconds is required to provide a high QoE at the cost of a higher average buffer filling.
3.3 Microsoft IIS Smooth Streaming

Figure 3.4: Quality bit rate and buffer filling in a fixed bandwidth scenario, for a buffer size of 12 seconds and a panic threshold of 2 seconds.

as well. In Table 3.2, the average MOS and buffer filling level ($\mu_{MOS}$, $\mu_{BF}$) and the standard deviation ($\sigma_{MOS}$, $\sigma_{BF}$) are presented. Results show that the range for the average buffer filling is $[7.537; 9.407]$ seconds, so that closer examination is required.

Consider the lowest possible threshold combination, with a value of 4 seconds for the lower and upper threshold. In this case, the lowest average buffer filling is expected. Figure 3.4 shows the selected quality bit rate and buffer filling over time, when streaming the video in a fixed bandwidth scenario. We observe how the buffer filling level is increased at first, while the available bandwidth still exceeds the download bit rate. When the buffer filling exceeds the upper threshold, the heuristic attempts to improve the quality bit rate. This increase is however not always immediately performed, because the implementation requires certain conditions for the perceived bandwidth to be met. As a consequence, it takes a while before the buffer filling level starts to slowly decrease again, when the quality bit rate is finally higher than the available
bandwidth. The heuristic only intervenes when the buffer filling drops below the lower threshold, thus explaining the observed oscillations. Because of this behaviour, the average buffer filling for the fixed bandwidth scenario is 8.233 seconds, which is significantly higher than the 4 seconds we initially expected. When considering the highest possible threshold combination, the highest average buffer filling is expected. Indeed, using a value of 8 seconds for the lower and upper threshold, peaks in the buffer filling are considerably reduced. The reason for this is the higher lower threshold, which actively averts lower levels for the buffer filling. The selected quality level is lowered more often, which results in a higher average buffer filling of 9.407 seconds for the fixed bandwidth scenario. Based on the observed results, we conclude that the average buffer filling cannot be strictly limited by only changing the threshold parameters.

Results in Table 3.2 indicate that the threshold combination has an impact on the MOS when the available bandwidth is variable: using the lowest value of 4 seconds for the lower threshold tends to lead to a lower MOS. This is explained by the fact that the requested quality level is decreased too late, i.e. when the buffer filling has already dropped to 2 seconds. As the available bandwidth is highly variable, play-out freezes are more likely to occur, negatively influencing the average MOS. When the available bandwidth is fixed however, results are more or less comparable for all configurations. This is explained by the fact that no variations in the available bandwidth occur, making play-out freezes and quality drops less likely. We conclude that, even though small differences are observed in terms of the MOS, one threshold combination can be selected that performs well in both bandwidth scenarios. A panic threshold of 2 seconds, a lower threshold of 6 seconds and an upper threshold of 8 seconds, for instance, is a good choice when a buffer size of 12 seconds is used.

3.3.3 Adaptively Changing the Parameter Configuration

In general, a trade-off between the average MOS and buffer filling exists: when the average buffer filling is higher, the same is true for the average MOS. This is explained by a lower amount of play-out freezes and a higher average quality level, since the panic and lower thresholds are exceeded less often. The balance of this trade-off however depends on whether or not the available bandwidth is variable or fixed. Were the algorithm to use the same parameter configuration in both scenarios, the following behaviour would be observed. If a configuration is used that leads to a low average buffer filling, performance will generally be good when the available bandwidth is fixed. When the available bandwidth is highly variable however, the QoE
Table 3.3: Defined actions with corresponding parameter configuration.

<table>
<thead>
<tr>
<th>Action</th>
<th>BS [s]</th>
<th>PT [s]</th>
<th>LT [s]</th>
<th>UT [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

will be severely affected: a large number of play-out freezes will occur and the average quality level will be significantly lower. On the contrary, if a configuration is used that leads to a high average buffer filling, the QoE will be good in both scenarios. This however comes with a higher play-out delay, which could have been avoided when the available bandwidth is fixed.

In our approach, we propose a Q-learning-based algorithm to adaptively change the parameter configuration according to perceived network conditions. The initial idea was to use a fixed buffer size and only change the panic, lower and upper thresholds, in order to reduce the average buffer filling whenever appropriate. The analysis above however revealed that the impact of these thresholds on the average MOS and buffer filling is confined. For this reason, we propose to adaptively change both the buffer size and the according lower, panic and upper thresholds. Based on the analysis in the previous section, the action set is defined as in Table 3.3. Using a buffer size below 6 seconds is not recommended, as the algorithm is not capable of dealing with a buffer size of one or two video segments. A large buffer size is not required as well, as the MOS does not further increase when a buffer size larger than 12 seconds is used. As for the respective thresholds, the same optimal threshold combinations as in Figure 3.3 are used.

The state of the agent is defined using two properties of the perceived bandwidth. First, we incorporate the average available bandwidth, as it strongly affects the QoE. Boundaries are based on the available quality bit rates in Table 3.1, leading to eight considered intervals. Second, a metric that is capable of discriminating between a variable and a fixed bandwidth scenario is added to the state definition. One possible metric is the standard deviation, which theoretically reaches a value of zero if the available bandwidth is perfectly fixed. In Section 4.1 possible metrics are evaluated and the one that is most suitable to discriminate between the two bandwidth scenarios is selected.

As for the reward, a trade-off is made between the perceived QoE and the buffer filling. We propose a reward function that is the linear combination of two factors. The first factor is
defined as the MOS obtained over a certain time window, computed using the model introduced in Section 3.2. The second factor represents the average buffer filling level over the same time window. The reward is defined as follows:

$$\text{Reward} = \phi \times \text{MOS}_\text{window} + (1 - \phi) \times (-\text{bufferFilling}_\text{window}) - 5.84 \quad (3.4)$$

The trade-off between a high MOS and low buffer filling are reflected by the parameter $\phi$. A value close to 1 means that the only goal of the client is to pursue a high QoE, while a value close to 0 means that the client is driven to select the parameter configuration leading to the lowest buffer filling. Using an appropriate value, we hope to achieve a high buffer filling when the available bandwidth is variable, and a low buffer filling when the available bandwidth is fixed. The rationale behind this reward is that, in a variable bandwidth scenario, a significant increase is in terms of the MOS is obtained when increasing the average buffer filling. Using a value $\phi = \phi'$, the agent should learn to use a parameter configuration leading to a higher average buffer filling. In a fixed bandwidth scenario however, the possible increase in terms of the MOS is relatively small. Using the same value $\phi'$, we expect the agent to learn to use a parameter configuration with a low buffer filling. Note that the reward component $\text{MOS}_\text{window}$ will differ significantly for different levels of the available bandwidth, thus illustrating the importance of incorporating the average available bandwidth in the environmental state. In our design we make sure rewards are negative at all times, by subtracting a value of 5.84. This is because the learning phase starts using an all-zero Q-table, and a positive reward in the first iteration would most likely lead to the repeated selection of the same action in following iterations.

The following parameters are important in the proposed approach. First, the decision interval $D$, indicating when a new parameter configuration will be selected. This interval is expressed as a number of segments that is downloaded between two succeeding configuration changes. Second, the reward window $W$, indicating how many segments are taken into account when evaluating the average MOS, buffer filling and average available bandwidth. Note that $W$ should be equal to or lower than $D$, as the reward needs to reflect the performance for the current parameter configuration only. Third, the number of required bandwidth samples to discriminate between a variable and a fixed bandwidth scenario. Ideally this number is equal to $W$, but as will be discussed in Section 4.1, an extensive analysis revealed that at least 50 samples are required. As the client should be responsive (i.e. not having to wait about 100 seconds before changing the parameter configuration), we decided to separate the two concerns. Finally, the Q-learning
parameters, which should be optimized with respect to the algorithm performance. An extensive analysis of all these parameters is conducted in Chapter 4.

3.4 Fair In-Network Enhanced Adaptive Streaming

In this section, the FINEAS quality selection heuristic presented by Petrangeli et al. is discussed. As was stated in Chapter 2, this algorithm attempts to maximize the user’s QoE while providing fairness within the network. The former is achieved by intelligently selecting the next quality level, pursuing a high average quality level while limiting quality switches and play-out freezes. The latter is achieved by the use of coordination proxies in the network, calculating the fair share of available bandwidth for every client. In this approach, two components are of the essence: the quality selection heuristic at client side and the fair bandwidth share computation at proxy side. As our focus is on a single-client scenario, we discuss a slightly adapted version of the quality selection heuristic that works without proxy components.

3.4.1 Quality Selection Heuristic

The pseudo-code of the quality selection heuristic is presented in Algorithm 2. First, the estimated bandwidth and buffer filling are retrieved (lines 2-3). When the buffer filling is lower than the panic threshold $PT$, the lowest quality level is simply selected (lines 4-5). In any other case, the client first calculates the average quality level over the last $qualityWindow$ seconds (lines 7-8). This value will be used to decide upon the next quality level, taking into account that the quality level should be maximized, while the number of switches should be minimized. Next, the highest possible quality level is determined for which the buffer filling, after downloading the new segment, will not have dropped below the panic threshold (lines 9-14). This way, play-out freezes are actively avoided. Finally, for all quality levels that meet this last requirement, a utility for the QoE is determined (lines 18-19). This utility consists of three components, each targeting a different objective. The first component targets the highest quality level, as a high average quality level should be pursued. The second component targets a quality level close to the average quality level over the last $qualityWindow$ seconds, as to reduce the number of quality switches. The third component, lastly, targets a quality level that most closely achieves a certain buffer filling level, specified by the $bufferTarget$ parameter. Eventually, the quality level is selected that provides the highest value for this utility (lines 20-22). In this way, the client is driven to select the quality level that most likely leads to the highest QoE.
3.4 Fair In-Network Enhanced Adaptive Streaming

**Algorithm 2** The Fair Adaptive Video Streaming quality selection heuristic.

```plaintext
1: while segmentsAvailable() do
2:   bandwidth ← getEstimatedBandwidth()
3:   bufferFilling ← getBufferFilling()
4:   if bufferFilling ≤ PT then
5:     nextQL ← 1
6:   else
7:     qVector ← updateQualityVector(currentTime, qualityWindow)
8:     qAvg ← getAverageQuality(qVector)
9:     ql ← 1, estimatedDownloadTime ← 0
10:    while ql < qMax ∧ estimatedDownloadTime ≤ bufferFilling − PT
11:      + segmentDuration do
12:      estimatedDownloadTime ← \frac{videoBitRate(ql) \times segmentDuration}{bandwidth}
13:      ql ← ql + 1
14:    end while
15:    nextQLMax ← ql, qoeUtilityMax ← −100
16:    for ql ← 1 to nextQLMax do
17:      estimatedDownloadTime ← \frac{videoBitRate(ql) \times segmentDuration}{bandwidth}
18:      qoeUtility ← |ql − qMax| + |ql − qAvg| + |bufferFilling − estimatedDownloadTime − segmentDuration − bufferTarget|
19:      if qoeUtility ≥ qoeUtilityMax then
20:        qoeUtilityMax ← qoeUtility
21:        nextQL ← ql
22:    end for
23:  end while
24:  return nextQL
25: end while
```

3.4.2 Impact of Heuristic Parameters

Relevant parameters for the quality selection heuristic are the buffer size, the buffer target, the panic threshold and the quality window. For the quality window a fixed value of 70 seconds will be used, as was suggested by Petrangeli et al. [2] For all other parameters, a closer evaluation has been performed.

As for the buffer target, only an integer amount of video segments are considered. Results show that a buffer target equal to the maximum buffer filling when the next segment is downloaded, i.e. the buffer size minus one video segment, typically leads to the highest MOS. This behaviour is explained as follows. If the buffer target is higher than the maximum buffer filling, the heuristic is more likely to select a lower quality level. Indeed, even when the buffer filling is
3.4 Fair In-Network Enhanced Adaptive Streaming

 maximal, an attempt is made to increase the buffer filling by 2 seconds. This is only possible when downloading the next segment at a lower quality level, so that the average MOS is unduly decreased. If the buffer target is lower than the maximum buffer filling, the buffer filling is more likely to reach the panic threshold. This results in a lower average quality level and even increases the chances of buffer starvation. As an illustration, the impact of the buffer target on the MOS and buffer filling is shown in Figure 3.5 for a buffer size of 10 seconds and a panic threshold of 2 seconds.

As for the panic threshold, a value of 0 seconds should be used when the buffer size is 4 seconds or lower, in order to avoid requesting the lowest quality level too often. For a buffer size of 6 seconds or higher, a value of 2 seconds should be preferred. In this way, the occurrence of freezes is avoided and a higher MOS is perceived. As an illustration, Figure 3.6 shows the impact of the panic threshold on the MOS, for different buffer sizes and a buffer target equal to the maximum buffer filling level.

To have a clear view on the algorithm performance, Table 3.4 shows an overview of the results for the average MOS and buffer filling, for different parameter configurations. It is clear that a buffer size of 2 seconds leads to an unacceptable QoE, and should therefore not be considered in future experiments. In a fixed bandwidth scenario, we observe that the increase in terms of MOS is much less significant when a higher buffer size is used. A maximum increase of 8.12% is observed when a buffer size of 8 seconds and a buffer target of 6 seconds is used, compared to
### 3.4 Fair In-Network Enhanced Adaptive Streaming

#### Figure 3.6: Impact of the panic threshold on the average MOS, using a buffer target equal to the maximum buffer filling.

<table>
<thead>
<tr>
<th>BS [s]</th>
<th>PT [s]</th>
<th>BT [s]</th>
<th>Variable bandwidth</th>
<th>Fixed bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(\mu_{MOS}) (\sigma_{MOS}) (\mu_{BF}) [s] (\sigma_{BF}) [s]</td>
<td>(\mu_{MOS}) (\sigma_{MOS}) (\mu_{BF}) [s] (\sigma_{BF}) [s]</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.708 (\pm) 0.360</td>
<td>0.000 (\pm) 0.000</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0.708 (\pm) 0.360</td>
<td>0.000 (\pm) 0.000</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2.168 (\pm) 0.808</td>
<td>1.954 (\pm) 0.018</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2.168 (\pm) 0.808</td>
<td>1.954 (\pm) 0.018</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1.881 (\pm) 0.445</td>
<td>1.986 (\pm) 0.003</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>3.019 (\pm) 0.616</td>
<td>3.950 (\pm) 0.016</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>3.019 (\pm) 0.616</td>
<td>3.950 (\pm) 0.016</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>2.006 (\pm) 0.490</td>
<td>3.970 (\pm) 0.009</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2.331 (\pm) 0.722</td>
<td>4.242 (\pm) 0.065</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4</td>
<td>2.331 (\pm) 0.722</td>
<td>4.242 (\pm) 0.065</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>3.281 (\pm) 0.614</td>
<td>5.397 (\pm) 0.145</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>2.146 (\pm) 0.458</td>
<td>5.918 (\pm) 0.071</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2.331 (\pm) 0.722</td>
<td>4.242 (\pm) 0.065</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>2.331 (\pm) 0.722</td>
<td>4.242 (\pm) 0.065</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>6</td>
<td>3.245 (\pm) 0.643</td>
<td>5.369 (\pm) 0.146</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>8</td>
<td>3.563 (\pm) 0.634</td>
<td>6.397 (\pm) 0.386</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
<td>2.127 (\pm) 0.474</td>
<td>7.840 (\pm) 0.151</td>
</tr>
</tbody>
</table>

Table 3.4: Performance for the traditional FINEAS algorithm, for different parameter configurations.
3.4 Fair In-Network Enhanced Adaptive Streaming

Figure 3.7: Impact of the buffer size on the average MOS and buffer filling, using the appropriate panic threshold and a buffer target equal to the maximum buffer filling.

(a) Variable bandwidth

(b) Fixed bandwidth

Figure 3.7: Impact of the buffer size on the average MOS and buffer filling, using the appropriate panic threshold and a buffer target equal to the maximum buffer filling.

a buffer size of 4 seconds and a buffer target of 2 seconds. The average buffer filling however, is increased by 179.81%. In a variable bandwidth scenario, we observe that the MOS is significantly increased when a larger buffer size is used. The same explanation as in Section 3.3.2 applies: play-out freezes are more likely to occur and the panic threshold is exceeded more often, leading to the selection of the lowest quality level. The highest average MOS of 3.356 is achieved for a buffer size of 10 seconds and a buffer target of 8 seconds, with an average buffer filling of 6.397 seconds. When a buffer size of 8 seconds and a buffer target of 6 seconds is used, a similar MOS is perceived (−2.23%), yet with a significantly lower buffer filling (−15.63%). Compared to a buffer size of 4 seconds and a buffer target of 2 seconds, this corresponds to an increase of 51.34% and 176.20% in terms of the MOS and the buffer filling respectively. To illustrate the difference between the variable and the fixed bandwidth scenario more clearly, Figure 3.7 shows the MOS and buffer filling as a function of the buffer size, using the parameter configuration that leads to the highest average MOS. Based on the analysis in Figures 3.5 and 3.6, this corresponds to a buffer target equal to the maximum buffer filling level and a panic threshold equal to 0 or 2 seconds, depending on the buffer size. A significant increase in terms of the MOS is observed when a larger buffer size is used in a variable bandwidth scenario, while the relative increase is rather limited in a fixed bandwidth scenario.
3.4 Fair In-Network Enhanced Adaptive Streaming

3.4.3 Adaptively Changing the Parameter Configuration

In the previous section, we showed that the trade-off between the MOS and buffer filling again depends on the characteristics of the available bandwidth: an increase of 179.81% in terms of the buffer filling is required to increase the MOS by only 8.12% when bandwidth is fixed, while a similar effort leads to an increase of 51.34% in terms of the MOS when bandwidth is highly variable. Based on the same reasoning as in Section 3.3.3, we propose to adaptively change the parameter configuration for the FINEAS quality selection heuristic.

The same Q-learning algorithm is proposed, yet possible actions now correspond to changing both the buffer size, the panic threshold and the buffer target. The action set is selected based on results in Table 3.4 and is shown in Table 3.5. Using a buffer size of 2 seconds is not recommended, as an acceptable QoE cannot be provided. A large buffer size is not required, since the MOS does not further improve when a buffer size larger than 10 seconds is used. Values for the panic threshold are selected based on the analysis in Figure 3.6. Since the best results in terms of the MOS are obtained when a buffer target equal to the maximum buffer filling level is used, this respective value is used for every buffer size. One exception is an extra action that comes with a buffer size of 10 seconds, a panic threshold of 2 seconds and a buffer target of 6 seconds, since both an acceptable average MOS and buffer filling were observed for this configuration.

As for the agent state and the assigned rewards, the same environmental model and reward function are used as in Section 3.3.3. This way, we again attempt to achieve a lower average buffer filling when the available bandwidth is fixed, providing an acceptable QoE at all times.

<table>
<thead>
<tr>
<th>Action</th>
<th>BS [s]</th>
<th>PT [s]</th>
<th>BT [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3.5: Defined actions with corresponding parameter configuration.
Chapter 4

Evaluation and Discussion

In this chapter, the proposed optimizations are thoroughly evaluated using the same experimental setup as in Section 3.1. First, we decide upon the most appropriate metric to discriminate between a variable and a fixed bandwidth scenario. Next, results for the optimizations for the MSS and the FINEAS algorithms are discussed in Sections 4.2 and 4.3. Finally, results for the two optimizations are compared in Section 4.4.

4.1 Bandwidth Discrimination

When defining the environmental state in Section 3.3.3, we proposed to use a metric that is capable of discriminating between a variable and a fixed bandwidth scenario. This metric depends on the available bandwidth, estimated by the HAS client whenever a segment is downloaded. This estimation is obtained as the ratio between the total throughput, the round trip time RTT and the download time DT:

\[
estimation = \frac{\text{throughput}}{RTT + DT}\quad (4.1)
\]

Consider a set \(B_K\) of \(K\) bandwidth samples \(b_k\), estimated when the \(k\)-th segment is downloaded. When the available bandwidth is fixed, following metrics should theoretically be equal to zero:

- The standard deviation \(\sqrt{\frac{\sum_{k=1}^{K-1} (b_k - \text{avg}(B_K))^2}{K-1}}\)
- The mean absolute deviation \(\text{avg}(|B_K - \text{avg}(B_K)|)\)
- The median absolute deviation \(\text{med}(|B_K - \text{med}(B_K)|)\)
- The mean absolute difference \(\frac{\sum_{k=1}^{K-1} |b_{k+1} - b_k|}{K-1}\)
4.1 Bandwidth Discrimination

On the contrary, if the available bandwidth is highly variable, non-zero values for these metrics should be observed. In this way, the agent should be able to discriminate between a fixed and a variable bandwidth scenario. In our case however, the bandwidth samples are only estimates of the available bandwidth, and consequently they are affected by noise. As an illustration, Figure 4.1 shows the estimated bandwidth for a variable and a fixed bandwidth scenario. Even when the available bandwidth is perfectly fixed, deviations are observed, leading to higher values for the metrics defined above. This prevents the client from correctly discriminating between a fixed and a variable bandwidth scenario, especially if the number of bandwidth samples $K$ is small. As an example, Figure 4.2 shows results for the standard deviation and the mean absolute difference when 15 bandwidth samples are used. A non-negligible overlap between the fixed and variable bandwidth scenarios is observed for both metrics. This issue is mitigated if a higher number of samples is used. However, when this number is too high, the client’s responsiveness to changing network conditions is reduced. For this reason, an analysis was performed to decide upon the optimal number of samples to use, evaluating the discrimination capability for a number of samples between 5 and 100. Furthermore, using a simple procedure, outliers are removed for every metric. For the deviation metrics, the mean absolute deviation is first calculated. Then, samples for which the absolute deviation exceeds a certain threshold (a constant times the average absolute deviation), are removed. A similar approach is applied for the mean absolute difference, where samples are removed when the absolute difference is too high. In this way, extreme outliers have a less significant impact on the considered metrics and more accurate discrimination is possible.
4.1 Bandwidth Discrimination

(a) Standard deviation

(b) Mean absolute difference

Figure 4.2: Results for the standard deviation and mean absolute difference, using 15 bandwidth samples.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Lower [kbps]</th>
<th>Upper [kbps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>280</td>
</tr>
<tr>
<td>6</td>
<td>280</td>
<td>400</td>
</tr>
<tr>
<td>7</td>
<td>400</td>
<td>4000</td>
</tr>
</tbody>
</table>

Table 4.1: Defined interval ranges.

Figure 4.3: Relative interval occurrence for the mean absolute difference and the median absolute deviation, using 50 bandwidth samples.
4.1 Bandwidth Discrimination

Results for all metrics were analysed over a moving window of 5 to 100 bandwidth samples, for 50 variable and fixed bandwidth traces. To select the most appropriate discretization boundaries, the result space was first divided into small intervals of 10kbps. Then, based on the number of interval occurrences for each bandwidth scenario, intervals were further grouped together. The total number of intervals should be kept limited to not slow down the learning process, but should be large enough to effectively discriminate between a variable and fixed bandwidth scenario. In general, the interval ranges in Table 4.1 enable an accurate discrimination when 50 bandwidth samples are used in the moving window. Closer analysis revealed that the standard deviation, the mean absolute deviation and the mean absolute difference are all capable of almost perfect discrimination, while a significant overlap is observed for the median absolute deviation. This is illustrated in Figure 4.3 where the relative interval occurrence is shown for the mean absolute difference and the median absolute deviation.

Until now, we have not considered bandwidth transitions in a fixed bandwidth scenario: due to congestion or a network change, the available bandwidth might decrease significantly yet still remain fixed. If we want to effectively discriminate between a variable and a fixed bandwidth scenario, results for the selected metric should not increase when such a transition occurs. This is however inevitable when the metric is based on bandwidth deviations, even when significant outliers are removed. As an example, Figure 4.4 illustrates the resulting behaviour for the standard deviation, the mean absolute deviation and the mean absolute difference when a bandwidth transition occurs. Since the mean absolute difference is the only metric that is not based on...
bandwidth deviations, it is the most suitable to provide an accurate discrimination between a
variable and a fixed bandwidth scenario. For this reason, we decide to incorporate this metric
as a state element, using the ranges defined in Table 4.1.

4.2 Microsoft IIS Smooth Streaming

In this section, optimizations for the MSS algorithm are evaluated. First, the influence of the
algorithm parameters on the overall performance is analysed. Next, results for the proposed
optimizations are thoroughly discussed.

4.2.1 Parameter Analysis

As described in Section 3.3.3, the proposed optimization comes with the following parameters:
the decision interval $D$, the reward window $W$ and the trade-off parameter $\phi$. Furthermore, as
described in Section 2.1.2, four other parameters are considered when applying the Q-learning
algorithm: the learning rate $\alpha$, the discount factor $\gamma$, the eligibility trace-decay $\lambda$ and the
Softmax inverse temperature $\beta$. An analysis of the impact of these parameters is required to
select the optimal parameter configuration. As the number of experiments grows exponentially
with the number of considered parameters, we decided to evaluate the optimization and Q-
learning parameters separately. Based on preliminary results, a good configuration for the
Q-learning parameters was decided upon. In the following section, this configuration is used to
evaluate the impact of the optimization parameters. Next, using the optimal configuration for
these parameters, we show that the selected Q-learning parameter values are indeed optimal.

To carry out experiments, 3000 bandwidth traces were randomly generated. Each trace is
generated using the experimental setup and approach presented in Section 3.1, either defining a
variable or a fixed bandwidth scenario. Both scenarios have a 50% of being selected, so that a fair
mixture of variable and fixed bandwidth traces is generated. Using these traces, the agent was
trained on 3000 episodes of the Big Buck Bunny video in changing network environments. Note

\footnote{Even when only four values for every parameter had been evaluated, a total of $4^7 = 16384$ experiments would have had to be conducted. At the time however, the HPC infrastructure was overloaded due to a high number of experiments being conducted in the context of other research projects. For this reason, the total number of experiments had to be severely limited. The proposed approach is generally not recommended, since the optimal parameter values for the two types of experiments might influence one another. Preliminary results however indicated that this is not the case, justifying the approach.}
that these episodes are in fact provided as one long, single video, rather than as 3000 separate episodes. Afterwards, the same 50 variable and fixed bandwidth traces as used in Sections 3.3 and 3.4 were used to evaluate the resulting behaviour.

4.2.1.1 Decision Interval and Reward Window

Preliminary results indicated that a Q-learning parameter configuration of $\alpha = 0.1$, $\gamma = 0.1$, $\lambda = 0.5$ and $\beta = 5$ generally leads to good results. As the properties of the available bandwidth are changing constantly, a low learning rate is recommended to slowly converge to the optimal behaviour. This also explains why a low discount factor should be used: future rewards cannot be controlled, and thus should not be taken into account. Note that a low discount factor causes the system to be rather insensitive to the eligibility trace-decay, as the decay is strongly accelerated (cfr. Equation 2.3). For this reason, a value of 0.5 can certainly be used. For the Softmax inverse temperature, a large value is chosen so that the best configuration is more likely to be selected. This seems reasonable, as the learning phase is started with an all-zero Q-table (cfr. Section 2.1.3).

The set of evaluated parameter configurations is presented in Table 4.2. Remember that the decision window is the interval for the decisions of the client, and that the reward window is the number of samples considered in the reward function. Both are expressed as a number of segments. The largest considered number of segments is limited to 20, since using a larger number would severely decrease the algorithm’s responsiveness to changing network conditions. As for the trade-off parameter $\phi$, a number of values have initially been selected to evaluate the MOS and buffer filling trade-off. A high average MOS and buffer filling are expected for a value close to 1, while a low average MOS and buffer filling are expected for a value close to 0. Using an intermediate value, we attempt to achieve an acceptable QoE at all times, yet with a lower average buffer filling in the second scenario.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Evaluated values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision interval $D$</td>
<td>5, 10, 15, 20</td>
</tr>
<tr>
<td>Reward window $W$</td>
<td>5, 10, 15, 20</td>
</tr>
<tr>
<td>Trade-off parameter $\phi$</td>
<td>0, 0.2, 0.4, 0.6, 0.8, 1</td>
</tr>
</tbody>
</table>

Table 4.2: Evaluated parameter configurations.
4.2 Microsoft IIS Smooth Streaming

Figure 4.5 shows the impact of the decision interval and the reward window on the average MOS for $\phi = 1$ and on the average buffer filling for $\phi = 0$. Note that results for the decision interval are averaged out over results for all four values of the reward window and vice versa. The best results are obtained with a decision interval of 15 segments: the highest average MOS is observed for $\phi = 1$, while the lowest buffer filling is observed for $\phi = 0$. As for the reward window, lower values tend to lead to better results. When a reward window larger than the decision interval is used, results for the previous configuration are also taken into account. This causes a misleading feedback to the client, and consequently a lower performance. Using a reward window equal to the decision interval is not recommended as well, as the quality selection heuristic requires time.
4.2 Microsoft IIS Smooth Streaming

Figure 4.6: Convergence relative to a buffer size of 12 seconds, a panic threshold of 4 seconds, a lower threshold of 6 seconds and an upper threshold of 8 seconds, for $\phi = 1$.

to adapt to the new selected configuration: when changing the buffer size from 6 to 12 seconds for instance, the buffer filling level will slowly increase by lowering the selected quality level. This transient has an impact on the perceived reward, which is dependent both on the MOS and on the buffer filling. Using a reward window smaller than the decision window, the first few transition samples are not taken into account and thus no longer have an impact on the perceived reward. Based on this reasoning and on the obtained results, the configuration with a decision interval of 15 segments and a reward window of 10 segments is selected.

To illustrate that the agent is capable of finding an optimal action using this configuration, we first evaluated results for the extreme values of the trade-off parameter. To this end, Figures 4.6 and 4.7 show the MOS, the buffer filling, the perceived reward and the Q-value changes over time, averaged out over a moving window of 50 episodes. For $\phi = 1$, results are presented relative to results for a buffer size of 12 seconds and optimized thresholds, since the only goal
4.2 Microsoft IIS Smooth Streaming

Figure 4.7: Convergence relative to a buffer size of 6 seconds, a panic threshold of 2 seconds and a lower and upper threshold of 3 seconds, for $\phi = 0$.

of the client is to maximize the QoE. For $\phi = 0$, results are presented relative to results for a buffer size of 6 seconds and optimized thresholds, since the only goal of the client is to minimize the average buffer filling. Results from the learning phase were extracted according to which bandwidth scenario each trace appertains. Even though complete decoupling is not possible (the bandwidth discrimination metric is evaluated over 50 samples, so that samples from two consecutive bandwidth traces are considered whenever a new episode is started), this separation does allow a more thorough evaluation of results for the two scenarios separately.

Figure 4.6 shows that for $\phi = 1$, the relative MOS for both scenarios is converging towards a value of 1, indicating that the agent is learning to use the configuration with a buffer size of 12 seconds. Note that the increase in terms of the MOS is more significant in the variable bandwidth scenario, since results for the MOS are less affected when the perceived bandwidth is fixed. The average buffer filling is increasing as well, but merely as a consequence of the
4.2 Microsoft IIS Smooth Streaming

Figure 4.8: Impact of the trade-off parameter on the average MOS and buffer filling, for $D = 15$ and $W = 10$.

agent’s preference towards a larger buffer size. No strictly increasing convergence is expected for the average reward, since the perceived MOS is strongly dependent on the available bandwidth. Relative to the perceived rewards for the fixed parameter configuration however, the average reward clearly does converge towards a value of 1. Finally, convergence is also observed through the flattening out of the average Q-value change. Note that the observed changes are higher in a variable bandwidth scenario, because of the lower absolute rewards: while the buffer filling is comparable for both scenarios, the MOS is not.

Figure 4.7 shows that for $\phi = 0$, the relative buffer filling is converging towards a value of 1, indicating that the agent is learning to use the configuration with a buffer size of 6 seconds. As a consequence, the average MOS and the relative reward are converging towards a value of 1 as well. Again, convergence is also observed through the decreasing Q-value changes.

4.2.1.2 Trade-off Parameter

Using the defined decision interval and reward window, the agent can find the optimal configuration when extreme values for the trade-off parameter $\phi$ are used. We are however looking for an intermediate value, resulting in an acceptable QoE and the most appropriate buffer filling level. Figure 4.8 shows results for the evaluated values of $\phi$. Using a value of 0.6 or lower, the preference of the agent is towards the lowest buffer filling. Using higher values, both the average MOS and the average buffer filling are slowly increased. Because the QoE has to be acceptable for both scenarios, only values within the interval $[0.6; 1]$ should be taken into account. A finer
4.2 Microsoft IIS Smooth Streaming

(a) Average MOS

(b) Average buffer filling

Figure 4.9: Impact of the trade-off parameter on the average MOS and buffer filling, for $D = 15$, $W = 10$ and $0.6 \leq \phi \leq 1$.

evaluation was performed for this interval, with results presented in Figure 4.9. We observe that the average buffer filling levels for the variable and fixed bandwidth traces are closely related, even though a strictly larger average buffer filling was expected when the available bandwidth is highly variable. It does however seem that for $\phi \leq 0.75$, the average buffer filling is generally lower when the available bandwidth is fixed. For $\phi = 0.75$, the average buffer filling is 5.722 and 4.896 seconds for the variable and fixed bandwidth scenario respectively. As for the average MOS, values of 2.730 and 3.550 are observed. Compared to results for a fixed buffer size of 6 seconds and optimal threshold values, this corresponds to an increase of 12.07% and 5.28% respectively. Compared to results for a fixed buffer size of 12 seconds, this corresponds to a still acceptable decrease of 11.19% and 4.77% respectively. In the next section, results will be discussed using this parameter value. First however, it still needs to be shown that the selected Q-learning values are indeed optimal.

4.2.1.3 Q-learning Parameters

Since the Q-learning parameters are all continuous values, a discretized subset was selected to evaluate its impact on the algorithm performance. An overview of the evaluated parameter sweep is presented in Table 4.3. The focus is on lower values for the learning rate and discount factor, based on the aforementioned reasoning. A more uniform spread is used for the eligibility trace-decay, to show that the system is indeed rather insensitive to changing values for this parameter.
### 4.2 Microsoft IIS Smooth Streaming

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Evaluated values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate $\alpha$</td>
<td>0.1, 0.2, 0.3, 0.4</td>
</tr>
<tr>
<td>Discount factor $\gamma$</td>
<td>0.1, 0.3, 0.5</td>
</tr>
<tr>
<td>Eligibility trace-decay $\lambda$</td>
<td>0.1, 0.5, 0.9</td>
</tr>
<tr>
<td>Softmax inverse temperature $\beta$</td>
<td>0.1, 1.0, 5.0</td>
</tr>
</tbody>
</table>

Table 4.3: Evaluated Q-learning parameter configurations.

![Figure 4.10: Impact of the Q-learning parameters on the algorithm performance, for $D = 15$, $W = 10$ and $\phi = 0.75$.](image)

As for the Softmax inverse temperature, selected values are more wide-spread because of the exponential relation in the probability distribution. Values higher than 5 are not considered, as this would prevent exploration when the agent is still learning. To evaluate the algorithm performance, results for the metric $\phi \ast \text{MOS} - (1 - \phi) \ast \text{bufferFilling}$, with $\phi$ equal to 0.75, are
Table 4.4: Performance for the traditional MSS algorithm and the proposed optimization.

<table>
<thead>
<tr>
<th>BS [s]</th>
<th>PT [s]</th>
<th>LT [s]</th>
<th>UT [s]</th>
<th>Variable bandwidth</th>
<th>Fixed bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\mu_{MOS}) (\sigma_{MOS}) (\mu_{BF}) [s] (\sigma_{BF}) [s]</td>
<td>(\mu_{MOS}) (\sigma_{MOS}) (\mu_{BF}) [s] (\sigma_{BF}) [s]</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2.436 0.692 3.840 0.028</td>
<td>3.372 1.091 3.888 0.037</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>2.709 0.749 5.361 0.137</td>
<td>3.634 1.050 5.529 0.259</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>2.934 0.554 7.239 0.141</td>
<td>3.637 1.016 7.478 0.270</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>3.076 0.628 8.611 0.266</td>
<td>3.728 0.986 8.953 0.545</td>
</tr>
<tr>
<td>Learning-based</td>
<td>2.730 0.645 5.722 0.430</td>
<td>3.550 1.020 4.896 0.565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard coded</td>
<td>3.004 0.668 8.218 0.353</td>
<td>3.356 1.106 3.939 0.112</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Section 3.3.2, thresholds for every buffer size were optimized with respect to the MOS. As a reference, Table 4.4 shows results for a number of fixed parameter configurations, both for the average MOS and buffer filling level. As was discussed before, the optimal configuration depends on the bandwidth scenario. When the perceived bandwidth is highly variable, the best configuration consists of a buffer size of 12 seconds and corresponding optimal thresholds. The average MOS is relatively high, with a value of 3.076 and 3.728 for the variable and fixed bandwidth scenarios respectively. The average buffer filling is however relatively high as well, at 8.611 and 8.953 seconds respectively. We already established that such a level for the buffer
filling is not required to provide an acceptable QoE when the perceived bandwidth is fixed, so this solution is not optimal. On the contrary, when the perceived bandwidth is fixed, the best configuration consists of a buffer size of 6 seconds and corresponding optimal thresholds. The average buffer filling is relatively low in this case, at 3.840 and 3.888 seconds respectively. However, while a MOS of 3.356 in the fixed bandwidth scenario is acceptable, a MOS of 2.436 in the variable bandwidth scenario is not.

Using the proposed approach, an average MOS of 2.730 is obtained in the variable bandwidth scenario, corresponding to an increase of 12.07% relative to a fixed buffer size of 6 seconds. This increase comes at the cost of a higher buffer filling level of 5.722 seconds (+49.01%). When compared to a buffer size of 12 seconds, the perceived loss in terms of the MOS is limited to 11.25%, in contrast to the loss of 20.81% when a fixed buffer size of 6 seconds would be used. In a fixed bandwidth scenario, an average MOS of 3.550 is obtained, corresponding to a decrease of 4.77% relative to a fixed buffer size of 12 seconds. This comes with a lower buffer filling of 4.896 seconds (−39.24%). When compared to a buffer size of 6 seconds, the increase in terms of the average buffer filling is limited to 25.93%, in contrast to 130.73% when a buffer size of 12 seconds would be used.

It is clear that the proposed approach leads to better results, taking into account both the average MOS and buffer filling. However, results should also be compared with other intermediate configurations, using a fixed buffer size of 8 or 10 seconds. Compared to results for a buffer size of 8 seconds, our approach achieves a higher average MOS (+0.23%) with a higher buffer filling (+6.73%) when the perceived bandwidth is variable. The average MOS is slightly lower when the perceived bandwidth is fixed (−2.31%), yet the average buffer filling is significantly lower (−11.45%) as well. Compared to results for a buffer size of 10 seconds, a significant decrease of 20.96% and 34.53% is observed for the average buffer filling, for the variable and fixed bandwidth scenario respectively. This comes at the cost of a lower average MOS, with a decrease of 6.95% and 2.39% respectively. Results for our approach are preferred, since the average buffer filling is reduced significantly while the average MOS is not.

Using other approaches, it is theoretically possible to achieve even better results. Table 4.4 also shows results for a hard coded “adaptive” approach, in which the following update rule is used. If results for the discriminating bandwidth metric are higher than 120kbps, indicating that the perceived bandwidth is highly variable, a buffer size of 12 seconds is selected, along with the optimal threshold values. If values lower than 120kbps are observed, indicating that
4.2 Microsoft IIS Smooth Streaming

(a) Variable bandwidth

(b) Fixed bandwidth

Figure 4.11: Average rewards for a variable and a fixed bandwidth scenario, relative to results for a fixed buffer size and optimal threshold values.

the available bandwidth is more or less fixed, a buffer size of 6 seconds is selected. In this way, a higher average MOS of 3.004 (+10.04%) is observed in the variable bandwidth scenario. On the contrary, when the perceived bandwidth is fixed, a lower average buffer filling of 3.939 seconds is obtained (−19.55%). These results are more in line with our original intent, yet are the consequence of a hard coded solution. This approach will fail when new bandwidth scenarios are introduced, while our solution can still adapt to changing bandwidth conditions. Furthermore, comparison is not truly fair: using intermediate actions with a buffer size of 8 and 10 seconds, the same extreme results can never be achieved.

Even though results are promising, one concern is raised. In Section 4.2.1.1, convergence was shown for values of 1 and 0 for the trade-off parameter $\phi$: the average MOS, buffer filling and reward all converged towards a value of 1, relative to a fixed parameter configuration with a buffer size of 12 and 6 seconds respectively. Using a value of 0.75 however, the following issue arises. Figure 4.11 shows the average reward over time, relative to the rewards perceived by the agent when a fixed buffer size and optimal threshold values are used. Convergence is observed relative to all fixed buffer size configurations, especially when the available bandwidth is fixed. However, the average reward never exceeds the reward for a fixed buffer size of 6 seconds. This means that, in order to obtain the highest average reward, it is better to simply use a fixed buffer size of 6 seconds. This behaviour is possibly caused by a number of reasons. For one, selection of the next action is based on the Boltzmann distribution. As was mentioned before, even though one action generally leads to the highest average reward, the observed Q-values are
relatively close to one another. Based on the acquired Q-table, the average probability of the best action being selected in every state is limited to 68.76%, discarding unseen states. Using even higher values for the Softmax inverse temperature might resolve this problem, yet will impede exploration during the learning phase. Especially when the agent is introduced to new network environments, this will be an issue. A second reason for the lower average reward is the occurrence of the previously mentioned transients. When a new configuration is selected, it takes time for the heuristic to adapt the buffer filling level to the newly selected thresholds. Even though the first samples are not taken into account when the reward is calculated, typically a lower average buffer filling will be observed when the buffer size is changed from 6 to 12 seconds, than when a fixed buffer size of 12 seconds would be used. This behaviour results in an increased reward for larger buffer sizes, thus increasing the likelihood of a larger buffer size being selected. The effects of transients could possibly be reduced when a larger decision interval is used, albeit at the cost of lower responsiveness to changing network conditions and a longer learning phase.

Despite this raised concern, we conclude that our approach generally leads to better results when both the average MOS and buffer filling level are taken into account. An acceptable QoE is provided both in a variable and a fixed bandwidth scenario, with a significantly lower average buffer filling when the perceived bandwidth is fixed.

4.3 Fair In-Network Enhanced Adaptive Streaming

In this section, the optimizations for the FINEAS algorithm are thoroughly evaluated. First the optimal parameter configuration is discussed, results are presented next. Note that the parameter analysis is not as extensive as in the previous section, since a large degree of similarity was observed between both optimizations.

4.3.1 Parameter Analysis

To analyse the impact of parameters on the average MOS and buffer filling, the same approach was used as in Section 4.2.1. Note that the quality selection heuristic has no true impact on the estimations of the available bandwidth, so that the same bandwidth discrimination metric can be used as in Section 4.2. Similar trends for the Q-learning parameters were observed, so the same configuration of $\alpha = 0.1$, $\gamma = 0.1$, $\lambda = 0.5$ and $\beta = 5$ was used to evaluate the parameter sweep mentioned in Table 4.2. Similar to the MSS approach, results indicated that the optimal
4.3 Fair In-Network Enhanced Adaptive Streaming

Figure 4.12: Impact of the trade-off parameter on the average MOS and buffer filling, for $D = 15$, $W = 10$ and $0.6 \leq \phi \leq 1$.

decision interval and reward window correspond to 15 and 10 segments respectively. As for the trade-off parameter $\phi$, results again indicated that for values lower than 0.6, the configuration with the lowest buffer size is always selected. For this reason, Figure 4.12 shows the average MOS and buffer filling for the interval $[0.6; 1]$ only. This time, separation between the average buffer filling in the two bandwidth scenarios is more clear: for $\phi < 0.85$, a strictly lower buffer filling is observed for the fixed bandwidth scenario. Maximal separation is observed for a value of 0.75, where the average buffer filling is 4.380 and 2.880 seconds for the variable and fixed bandwidth scenario respectively. The average MOS is acceptable, with values of 2.863 and 3.719 respectively. Consequently, this value for the trade-off parameter has been selected.

4.3.2 Results

Results for the FINEAS algorithm and the proposed optimization are presented in Table 3.4. Using the traditional algorithm, the optimal parameter configuration in the fixed bandwidth scenario comes with a buffer size of 4 seconds, a panic threshold of 0 seconds and a buffer target of 2 seconds. In this case, the lowest average buffer filling is observed, while the QoE is still acceptable. In a highly variable bandwidth scenario however, the average MOS is significantly

\footnote{Note that the average buffer filling is not strictly increasing for higher values of $\phi$. For $\phi = 0.7$, an average buffer filling level of 3.317 seconds is observed, even though a higher weight of 0.3 is assigned in the reward function. The average MOS is however higher as well, with a value of 3.823. Because, after all, a trade-off is used, a higher value for $\phi$ does not necessarily lead to a higher average MOS and buffer filling.}
lower, because of a large number of freezes and quality drops. In fact, the best configuration for this scenarios comes with a buffer size of 10 seconds, a panic threshold of 2 seconds and a buffer target of 8 seconds. The average buffer filling is however significantly higher, which is not at all required to provide an acceptable QoE when the perceived bandwidth is fixed.

Using the proposed approach, an average MOS of 2.863 is obtained in the variable bandwidth scenario, corresponding to an increase of 32.06% relative to a fixed buffer size of 4 seconds. This increase comes at the cost of a higher buffer filling of 4.380 seconds (+124.16%). Compared to a buffer size of 10 seconds, the perceived loss in terms of the MOS is limited to 14.69%, in contrast to a loss of 35.40% when a buffer size of 4 seconds would be used. In a fixed bandwidth scenario, an average MOS of 3.719 is obtained, corresponding to a decrease of 7.76% relative to a fixed buffer size of 10 seconds. This comes with a lower buffer filling of 2.880 seconds (−57.32%). Compared to a buffer size of 4 seconds, the increase in terms of the average buffer filling is limited to 45.38%, in contrast to 240.64% when a buffer size of 10 seconds would be used.

Compared to a buffer size of 6 seconds, a significantly lower average buffer filling is observed (−27.42%) with a slightly lower average MOS (−4.52%) when the perceived bandwidth is fixed. Results in a variable bandwidth scenario are however inferior, since both a lower average MOS (−5.17%) and a higher average buffer filling (+10.89%) are observed. Compared to a buffer size of 8 seconds, a significant decrease of 18.84% and 48.04% is observed for the average buffer filling, for the variable and fixed bandwidth scenario respectively. This comes at the cost of a lower average MOS, with a decrease of 12.74% and 7.81% respectively. Results for our approach are again preferred, since the average buffer filling is reduced significantly while the average MOS is not.

<table>
<thead>
<tr>
<th>BS [s]</th>
<th>PT [s]</th>
<th>BT [s]</th>
<th>Variable bandwidth</th>
<th>Fixed bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>µMOS</td>
<td>σMOS</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2.168</td>
<td>0.808</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>3.019</td>
<td>0.616</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>3.281</td>
<td>0.614</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>8</td>
<td>3.356</td>
<td>0.634</td>
</tr>
<tr>
<td>Learning-based</td>
<td>2.863</td>
<td>0.712</td>
<td>4.380</td>
<td>0.353</td>
</tr>
<tr>
<td>Hard coded</td>
<td>3.175</td>
<td>0.738</td>
<td>6.020</td>
<td>0.508</td>
</tr>
</tbody>
</table>

Table 4.5: Performance for the traditional FINEAS algorithm and the proposed optimization.
4.4 Comparison

Both the MSS and the FINEAS optimization approach are promising, when the average MOS and buffer filling are compared with results for a fixed parameter configuration. A comparison of
the two approaches is however still required. Results for the average MOS and buffer filling are presented in Table 4.6 both for a variable and a fixed bandwidth scenario. When the perceived bandwidth is highly variable, the average MOS is 4.87% higher when the FINEAS approach is used, while the average buffer filling is 23.45% lower. The same trend is observed for the fixed bandwidth scenario, where the average MOS is 4.76% higher and the average buffer filling is 41.18% lower. This indicates that the optimization for the FINEAS algorithm is most suitable to provide both a high average MOS and a low average buffer filling. Furthermore, it is worth noticing that results for the MSS optimization are outperformed by the traditional FINEAS algorithm, when a fixed buffer size of 6 seconds is used: both a higher MOS (+10.59% and +8.86) and a lower average buffer filling (−30.97% and −18.95) are observed in a variable and a fixed bandwidth scenario respectively.

There are three reasons for this significant difference between the two approaches, which are all related to the quality selection heuristic. First, the average MOS is higher because the FINEAS quality selection heuristic is based on the considered QoE model. Taking into account the requested quality level over a moving window allows the next quality level to be requested in a more reasoned way, thus providing a higher average MOS. Second, the heuristic is more capable of dealing with high variations in the available bandwidth. In contrast to MSS, the quality level is not necessarily increased one step at a time. This means that the heuristic can react faster to high variations in the perceived bandwidth, again leading to a higher average MOS. Third, the heuristic is capable of dealing with a low buffer size and filling level. This is thanks to the fact that the heuristic attempts to respect the panic threshold at all times. In this way, a lower average buffer filling is achieved, while freezes are less likely to occur. Based on this analysis, we conclude that the proposed optimization for the FINEAS algorithm most closely fulfils the intended objectives of this dissertation.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Variable bandwidth</th>
<th>Fixed bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu_{MOS}$</td>
<td>$\sigma_{MOS}$</td>
</tr>
<tr>
<td>Learning-based MSS</td>
<td>2.730</td>
<td>0.645</td>
</tr>
<tr>
<td>Learning-based FINEAS</td>
<td>2.863</td>
<td>0.712</td>
</tr>
</tbody>
</table>

Table 4.6: Performance for the two proposed optimizations.
Chapter 5

Conclusion

This chapter concludes this dissertation. General conclusions are presented in Section 5.1 summarizing results and discussing the applicability of the proposed approach. A number of possible improvements are discussed in Section 5.2 providing an opportunity for future research.

5.1 General Conclusions

In this dissertation, we proposed to use a Q-learning algorithm in the implementation of two existing rate adaptation algorithms, to adaptively change the parameter configuration according to network conditions. The main rationale for this decision was that, using a hard coded implementation with fixed parameter values, suboptimal results are perceived when the client is introduced to a changing network environment. On the one hand, while an acceptable QoE is generally perceived when a high buffer size is used, the average play-out delay is unnecessarily high in a fixed bandwidth scenario. On the other hand, while a low play-out delay is achieved when a low buffer size is used, the QoE is unacceptable in a highly variable bandwidth scenario. Using the proposed approach, it is possible to achieve both an acceptable average MOS when high variations in the available bandwidth occur, as well as a significantly lower average buffer filling when the available bandwidth is fixed. Promising results were achieved both for the MSS and the FINEAS algorithm, yet results for the FINEAS algorithm turned out to be significantly better. This is because the quality selection heuristic for this rate adaptation algorithm is based on the considered model for the QoE, allowing the client to achieve both a higher average MOS and a lower average buffer filling.
Even though results are promising, we argued that better results can theoretically be achieved. Using a simple hard coded approach, it is possible to achieve a lower level for the average buffer filling when the perceived bandwidth is fixed, while a higher QoE is achieved when the perceived bandwidth is highly variable. This approach is however too extreme, and is not recommended since the algorithm is unable to adapt to changing conditions for the available bandwidth. It does however show that further improvements are possible, either through new definitions for the reward function or through other metrics for the available bandwidth.

During the course of this research, certain challenges had to be faced. For one, the noise on the estimated bandwidth was a difficult issue to deal with. A solution was proposed, in which a larger number of bandwidth samples is used to correctly discriminate between a variable and a fixed bandwidth scenario. This approach however affects the responsiveness of the agent, so that other solutions are possibly more appropriate. Another problem was that in the last month of this research, it was impossible to launch a large number of experiments on the HPC. Because of this, we were forced to reduce the evaluated parameter sweep by optimizing certain parameter values separately. This is not an optimal approach, since parameter values are often dependent on one another. A more thorough evaluation is recommended to show that the parameter configuration we finally selected, is indeed truly optimal.

In general however, we can conclude that our initial objectives are reached. Using the proposed approach, the user is provided with an acceptable QoE at all times, while a significantly lower average buffer filling is obtained when the perceived bandwidth is fixed. We hope that this research is a next step towards a truly adaptive video streaming solution and that it can be a basis for future work.

5.2 Future Work

In this dissertation, we showed that there are several advantages to adaptively changing the parameter configuration in existing HAS implementations. Within the context of our research, future work could focus on a new definition for the reward function of the Q-learning client, making it possible to achieve even better results. As for the environmental state, other criteria might be taken into account, such as the absolute buffer filling, the CPU power etc. Of course, future research should not be limited to Q-learning: other machine learning techniques can be applied as well, possibly leading to further improvements.
5.2 Future Work

In this dissertation, focus was on a single-client scenario only. As was discussed in Section 2.2.4, issues may arise when multiple clients are streaming video content from the same server or within the same network. Future work could focus on a multi-client scenario, possibly using the same Q-learning approach. The research by Petrangeli et al. however showed that convergence issues can arise when the number of clients is too high, so that other solutions might be more appropriate [32]. Providing fairness among clients might be possible through a modified version of the FINEAS approach, or through the definition of a global reward function that should be maximized. Several solutions are possible, providing an interesting topic for future work.

Another possible improvement is in the generality of the solution, since the proposed approach is in fact designed to suit the experimental setup presented in Section 3.1. Even when only changing the available quality bit rates, boundaries for the state elements should be redefined. This means that our approach is possibly hard to integrate in a real video streaming environment, as the algorithm would have to be trained again for every video that is offered at a new quality bit rate. Even though a single content provider generally uses the same encoder and bit rate levels for all of the available videos, providing a more general approach is recommended.

“Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.”

- Winston Churchill
5.2 Future Work
Bibliography


List of Figures

2.1 Reinforcement learning scheme. .................................................. 6

2.2 HTTP Adaptive Streaming concept. ........................................... 10

3.1 Simulated network topology. .................................................... 16

3.2 Impact of the buffer size on the average MOS and buffer filling, for thresholds
defined by Famaey et al. [36] ............................................................ 20

3.3 Impact of the buffer size on the average MOS and buffer filling, for thresholds
optimized with respect to the MOS. .................................................. 21

3.4 Quality bit rate and buffer filling in a fixed bandwidth scenario, for a buffer size
of 12 seconds and a panic threshold of 2 seconds. .............................. 23

3.5 Impact of the buffer target on the average MOS and buffer filling, for a buffer
size of 10 seconds and a panic threshold of 2 seconds. ......................... 29

3.6 Impact of the panic threshold on the average MOS, using a buffer target equal to
the maximum buffer filling. ............................................................... 30

3.7 Impact of the buffer size on the average MOS and buffer filling, using the appro-
priate panic threshold and a buffer target equal to the maximum buffer filling. 31

4.1 Estimated bandwidth when using the traditional FINEAS quality selection heuristic. 34

4.2 Results for the standard deviation and mean absolute difference, using 15 band-
width samples. ............................................................................ 35

4.3 Relative interval occurrence for the mean absolute difference and the median
absolute deviation, using 50 bandwidth samples. ............................ 35
4.4 Results for the standard deviation and mean absolute difference when a bandwidth transition occurs, using 50 bandwidth samples. ........................................... 36

4.5 Impact of the decision interval and the reward window on the average MOS for \( \phi = 1 \) and on the average buffer filling for \( \phi = 0 \). ........................................... 39

4.6 Convergence relative to a buffer size of 12 seconds, a panic threshold of 4 seconds, a lower threshold of 6 seconds and an upper threshold of 8 seconds, for \( \phi = 1 \). . . 40

4.7 Convergence relative to a buffer size of 6 seconds, a panic threshold of 2 seconds and a lower and upper threshold of 3 seconds, for \( \phi = 0 \). ........................................... 41

4.8 Impact of the trade-off parameter on the average MOS and buffer filling, for \( D = 15 \) and \( W = 10 \). ........................................... 42

4.9 Impact of the trade-off parameter on the average MOS and buffer filling, for \( D = 15, W = 10 \) and \( 0.6 \leq \phi \leq 1 \). ........................................... 43

4.10 Impact of the Q-learning parameters on the algorithm performance, for \( D = 15, W = 10 \) and \( \phi = 0.75 \). ........................................... 44

4.11 Average rewards for a variable and a fixed bandwidth scenario, relative to results for a fixed buffer size and optimal threshold values. ........................................... 47

4.12 Impact of the trade-off parameter on the average MOS and buffer filling, for \( D = 15, W = 10 \) and \( 0.6 \leq \phi \leq 1 \). ........................................... 49

4.13 Average rewards for a variable and a fixed bandwidth scenario, relative to results for a fixed buffer size and optimal parameter values. ........................................... 51
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Bit rates for the <em>Big Buck Bunny</em> video trace.</td>
</tr>
<tr>
<td>3.2</td>
<td>Impact of thresholds on the MOS and buffer filling, for a buffer size of 12 seconds and a panic threshold of 2 seconds.</td>
</tr>
<tr>
<td>3.3</td>
<td>Defined actions with corresponding parameter configuration.</td>
</tr>
<tr>
<td>3.4</td>
<td>Performance for the traditional FINEAS algorithm, for different parameter configurations.</td>
</tr>
<tr>
<td>3.5</td>
<td>Defined actions with corresponding parameter configuration.</td>
</tr>
<tr>
<td>4.1</td>
<td>Defined interval ranges.</td>
</tr>
<tr>
<td>4.2</td>
<td>Evaluated parameter configurations.</td>
</tr>
<tr>
<td>4.3</td>
<td>Evaluated Q-learning parameter configurations.</td>
</tr>
<tr>
<td>4.4</td>
<td>Performance for the traditional MSS algorithm and the proposed optimization.</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance for the traditional FINEAS algorithm and the proposed optimization.</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance for the two proposed optimizations.</td>
</tr>
</tbody>
</table>
Self-learning Optimization of Adaptive Video Streaming
Parameters

Jeroen van der Hooft

Supervisors: Prof. dr. ir. Filip De Turck, Dr. Jeroen Famaey
Counsellors: Ir. Maxim Claeys, Stefano Petrangeli

Master's dissertation submitted in order to obtain the academic degree of Master of Science in de ingenieurswetenschappen: computerwetenschappen

Department of Information Technology
Chairman: Prof. dr. ir. Daniël De Zutter
Faculty of Engineering and Architecture
Academic year 2013-2014