Impact of mobility on QoE in wireless networks

Imen Triki

ACADÉMIE D'AIX-MARSEILLE
UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

THÈSE

présentée à l'Université d'Avignon et des Pays de Vaucluse
pour obtenir le diplôme de DOCTORAT

SPÉCIALITÉ : Informatique

École Doctorale 536 «Agrosciences et Sciences»
Laboratoire d'Informatique (EA 4128)

Rôle du vidéo streaming mobile qui dépend du contexte dans l'amélioration de la qualité d'experience

Présentée par Imen TRIKI

Soutenue le .. / .. / ..., devant le jury composé de :

M.	André-Luc Beylot	Professeur, Université de Toulouse	Rapporteur
M.	Gerardo Rubino	Directeur de Recherche, IRISA/INRIA	Rapporteur
M.	Rachid El-Azouzi	Professeur, Université d'Avignon	Directeur de Thèse
M.	Majed Haddad	Maître de Conférences, Université d'Avignon	co-Encadrant de Thèse

Laboratoire d'Informatique d'Avignon - CERI
Centre d'Enseignement et de Recherche en Informatique

ACADÉMIE D'AIX-MARSEILLE
UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

THESIS

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

In Computer Science

Doctoral School 536 «Agrosciences et Sciences»
Laboratoire d'Informatique (EA 4128)

The role of context dependent mobile video streaming in QoE enhancement

Presented by Imen TRIKI

Defended on .. / .. / .... , before the jury members:

M.	Andre-Luc Beylot	Professor, University of Toulouse	Reviewer
M.	Gerardo Rubino	Research Director, IRISA/INRIA	Reviewer
M.	Rachid El-Azouzi	Professor, University of Avignon	Advisor
M.	Majed Haddad	Professor, University of Avignon	co-Advisor

Laboratoire d'Informatique d'Avignon - CERI
Centre d'Enseignement et de Recherche en Informatique

Résumé

Abstract

The strong emergence of smartphones on human daily life as well as the high broadband access supplied by operators have triggered pervasive demand on video streaming mobile services, requiring the exploration of novel approaches on video content delivery. To afford video streaming services at sustainable quality, the idea of adjusting the streaming to the time-varying users' contexts has been actively investigated during the recent years. Since the users' perceptions on the video quality directly impact their engagement in video streaming sessions, many interests have been accorded to the user's Quality of Experience (QoE).

Today streaming solutions mostly rely on the user's contextual information such as his link capacity or his available bandwidth to provide an acceptable final QoE. Such contextual information can be easily acquired thanks to the existence of wireless sensors and dedicated smart applications on today mobile devices. At the core, lies the idea of exploiting the strong correlation between users' locations and contexts. To that end, radio maps with historical average signal strength have been geographically mapped. Various studies on users' mobility patterns also showed that people daily routes exhibit a high degree of spatial and temporal regularity, especially on public transportation or on road ways to/from frequently visited places. Coupled with radio maps, these mobility patterns can give high accuracy on context predictability along users' trips.

In this thesis, we analyse the impact of adapting video streaming to the user's context on the final QoE. We start by proposing CAMS (Context Aware Mode Switching), a context-aware resource allocation mechanism, for *real (i.e, non adaptive)* video streaming delivery to reduce the number of video stalling. CAMS is designed to be applied in a particular network topology under a particular mobility of users. Then, we explore the impact of knowing the future throughput variations on video quality adaptation and delivery cost in *adaptive* video streaming. We propose NEWCAST (aNticipating qoE With threshold sCheme And aScending biTrate levels) as a proactive algorithm for cost adjustment and quality adaptation under the assumption of a perfect throughput estimation. We then extend the study to the case where throughput prediction errors may exist and propose a bench of adaptive algorithms inspired from NEWCAST. To explore the feasibility of implementing these algorithms in real world streaming, we conduct some experiments with the DASH-If Reference player in an emulated environment. Finally, we explore the impact of knowing the future throughput variations when exploited with machine learning on the global QoE enhancement in *adaptive* video streaming. We propose a closed-loop framework based on users' feedbacks to progressively learn their QoE profiles and to fittingly optimize video deliveries. This framework is in particular suited for heterogenous populations where the QoE profiles of users are quite different and unknown in advance.

KEY-WORDS: Real Video Streaming, Adaptive Video Streaming, Quality of Experience (QoE), Mobile Network, User-Context, Resource Allocation, Throughput Prediction, Machine Learning.

Acknowledgements

Contents

List of Figures

List of Tables

List of Acronyms

3G	3rd Generation.
3GPP	3rd Generation Partnership Project.
A-STERN	Adaptive Short-TERm Newcast.
ABR	Adaptive BitRate.
ACR	Absolute Category Rating.
AMC	Adaptive Modulation and Coding.
ANN	Artificial Neural Network.
API	Application Programming Interface.
AVI	Audio Video Interleave.
AWARE	Anticipating qoe With Ascending bitRate lEvels.
BB-ABR	Buffer Based ABR.
BLER	BLock Error Rate.
BNN	Biological Neural Network.
CAMS	Context-Aware Mode Switching.
CBR	Constant BitRate.
CDF	Cumulative Distribution Function.
CP	Cyclic Prefix.
CQI	Channel Quality Indicator.
CSI	Channel State Information.
DASH	Dynamic Adaptive Streaming over Http.
DASH-IF	DASH Industry Form.
E-UTRAN	Evolved Universal Terrestrial Radio Access Network.
EPC	Evolved Packet Core.
EPS	Evolved Packet System.
FD	Frequency Domain.
FHD	Full High Definition.
FTP	File Transfer Protocol.
GBR	Guaranteed Bit-Rate.
HARQ	Hybrid Automatic Repeat reQuest.
HD	High Definition.
HetNet	Heterogeneous Network.
HSDPA	High Speed Downlink Packet Access.
HSS	Home Subscriber Server.
HTB	Hierarchy Token Bucket.
HTTP	HyperText Transfer Protocol.

INVEST INcrease with VariablE foot STep.
IP Internet Protocol.
ITU International Telecommunication Union.

LD Low Definition.
LTE Long Term Evolution.
LXC LinuX Containers.

MAC Medium Access Control.
MBS Macro Base Stations.
MCS Modulation and Coding Scheme.
MIMO Multiple Input Multiple Output.
MME Mobility Management.
MOS Mean Opinion Score.
MPC Model Predictive Control.
MPD Media Presentation Description.
MPEG Moving Picture Experts Group.

NEWCAST aNticipating qoE With threshold sCheme And aScending biTrate levels.

OFDM Orthogonal Frequency Division Multiplexing.
OFDMA Orthogonal Frequency Division Multiple Access.

PCRF Policy Control and Charging Rules Function.
PDN Packet Data Network.
PGW Packet data network GateWay.
PID Proportional-Integral-Derivative.
PRB Physical Resource Block.
PUCCH Physical Uplink Control CHannel.
PUSCH Physical Uplink Shared CHannel.

QAM Quadrature Amplitude Modulation.
QoE Quality of Experience.
QoS Quality of Service.
QPSK Quadrature Phase-Shift Keying.

RB Resource Bloc.
RE Resource Element.
RECAST shoRt-tErm Conservative newcAST.
RG Resource Grid.
RSRP Reference Signal Received Power.
RTMP Real Time Messaging Protocol.
RTSP Real Time Streaming Protocol.

SBS Small Base Stations.
SD Standard Definition.
SDT Segment Download Time.
SGW Serving GateWay.
SINR Signal-to-Interference-plus-Noise Ratio.
STERN Short-TERm Newcast.

STREET Short TeRm Enslaved nEwcasT.
SVAA Smooth Video Adaptation Algorithm.

TB Transport Block.
TB-ABR Throughput Based ABR.
TBS Transport Block Size.
TCP Transmission Control Protocol.
TD Time Domain.
TmB-ABR Time Based ABR.
TS Time Slot.
TTI Transmission Time Interval.

UDP User Datagram Protocol.
UE User Equipment.
UHD Ultra High Definition.
URI Unic Resource Identifier.
UX User eXperience.

VBR Variable BitRate.

Chapter 1

Introduction

Contents

1.1 Context

With the development of smartphones, tablets and laptops from one side, and the emergence of new opportunities in media productions such as digital video and social networking from the other side, video streaming has become one of the most prominent internet services. According to recent statistics, video traffic represents today a significant fraction of internet traffic and is expected to reach 82% by 2020 [1]. This has led the operators to rethink the way their resources are managed and their networks are dimensioned.

Unlike videos delivered using a deterministic channel with constant delay and rate, and very limited data drops (such as digital TV broadcasting), videos delivered over Internet Protocol (IP) and mobile networks are much more problematic since they are supported by a non-Guaranteed Bit-Rate (GBR) resource type due to intermittent usage of network resources [2, 3]. Note that a non-GBR bearer does not have dedicated bandwidth and is used for best-effort traffic such as file downloads. During a video streaming, the radio channel conditions can change consistently (e.g., due to mobility), which may degrade considerably the user's viewing experience and thus the user engagement. As it is hard to determine the user's real perception of a video streaming service through a direct Quality of Service (QoS) analysis , research interests have been steered toward end-user Quality of Experience (QoE) which differs from one user to another depending on a set of factors. These factors may be linked to the network conditions, to the content characteristics, to the user preferences or to the device capabilities.

To improve the user's QoE in mobile video streaming, many approaches have been proposed for both real (i.e., non adaptive) and adaptive streaming at different system levels (e.g., application and network levels). Today approaches mostly rely on the context of the user as the current technologies allow to acquire it. User context is usually related to the user's location, but it can refer to other features such that the

user's daily move-trajectory and travel time. It can be provided manually by the user himself or automatically through communication with the user's device sensors and applications. Users' geographical positions and mobility patterns may give further contextual information related to the network conditions such as channel quality and link capacity. These information can be used to improve video delivery in a *reactive* or a *proactive* manner.

Recent studies performed by Alcatel-Lucents's Bell Labs [4] showed that the long-term predictions of the users' mobility and streaming requests may be used in inter-cell resource allocation to improve the overall network efficiency without degrading the QoE. Together with the German service provider Net Mobile, Alcatel has developed context-aware video streaming to reduce the probability of poor video delivery on the move. Context-aware video streaming mainly relies on the smart use of location-based information which allows to predict upcoming network performance. The acquisition of the context is realized by using road and network coverage maps and further radio performance data. Based on this prediction, the streaming is steered to download more video seconds to the user's device in areas of good coverage before reaching poor coverage areas. This results in a highly improved QoE in terms of video stalling.

This thesis deals with video streaming services in mobile networks and exploits user context to improve some of the QoE metrics. The contextual information used are mainly linked to the users' s positions and mobility patterns, namely the variability of the channel conditions and the link capacity (throughput). Both real (i.e., non adaptive) and adaptive video streaming over HyperText Transfer Protocol (HTTP) are studied with further focuses on the network performance. More details about our objectives are detailed in the next section.

1.2 Objectives

Video streaming has two main properties that other internet services do not have: First, the durations of the media streams are known in advance, and second, the streamed data are buffered at the receiver before they are displayed to the user. In this thesis, we exploit these properties for both real (i.e., non adaptive) and adaptive streaming, to improve the end user's QoE while taking into account his context. Our main purpose is to know how and when video data should be buffered at the receiver side to ensure a smooth video playback.

Our first study is presented in Chapter 3. It is performed on real (i.e., non adaptive) video streaming under the assumption of a specific network topology and a specific mobility of users. The network is assumed to comprise small sites distributed all along a highway, and the users are assumed to move on that highway at a constant speed. Under such a high user mobility, we propose to design a cross-layer that captures the users' channel states and interacts with the network scheduler to improve the QoE and the overall spectral efficiency. Our purpose is to take advantage from the high mobility of users rather than considering it an obstacle in video streaming services.

Our second study is presented in Chapter 4. It deals with adaptive video streaming and exploits the knowledge of the user's future throughput. The prediction of the

throughput is assumed to be perfectly performed over a large horizon window (3 minutes and more). Under such assumption, we propose to design a streaming model to make a tradeoff between the operator's system utilization cost and the user's QoE. This streaming model will determine the way the user will be served and the video bitrates will be distributed over the future horizon window.

Our third study is presented in Chapter 5. It also hinges on the assumption of knowing the user's future throughput with the difference that, here we assume prediction errors. We keep the same optimization problem of Chapter 4, but we propose new streaming strategies more robust and more adapted to inaccurate throughput prediction.

To explore the performance of our strategies in real world video streaming, we propose in Chapter 6 to conduct real experiments with a real video player. Due to the lack of material, we emulate the user's future throughput in a Linux environment.

Finally, based on the same throughput prediction as in Chapter 4, we address in Chapter 7 a QoE optimization problem in which we propose to maximize the global QoE in an heterogeneous population where the users' preferences and QoE profiles are unknown and different.

We review in the next section all our contributions in respect to these objectives.

1.3 Contributions

The contributions of this thesis are organized in Chapters:

In Chapter 2, we give an overview on Long Term Evolution (LTE) networks, video streaming services and neural networks. We review some standards and some specific features to understand the basis of this work. In the LTE section, we present the network components, the network physical resources, how the data is transmitted to/from the user and how the physical resources are scheduled to users. Then, in video streaming section, we present some of digital video characteristics to understand the signification of the video parameters used in this thesis. To differentiate between real and adaptive streaming, we state all the existing streaming methods with a brief explanation on each of them. As we mostly work with adaptive streaming (Chapter 4, 5, 6 and 7), we relate more details on Dynamic Adaptive Streaming over Http (DASH). The QoE concept in video streaming is reviewed as well with a focus on the QoE metrics. In the last section, we present neural networks with a short description of the gradient descent algorithm used in Chapter 7.

In Chapter 3, we propose a resource allocation mechanism for real (i.e., non adaptive) video streaming in LTE networks under the assumption of a specific mobility pattern. More specifically, we consider a road (highway) network topology with consecutive adjacent small cells and users moving at a constant speed along the road. By exploiting the high variability of the users' channel conditions and the prefetching (buffering) property of video streaming, we propose to restrict the list of users to serve to only users having high channel conditions. This will make them download as much data as their channels allow before falling in low coverage areas. Due to their high mobility and the network topology, all the users will pass across high and low coverage

areas, which guarantees a good fairness level among them all. We call our resource allocation mechanism Context-Aware Mode Switching (CAMS). CAMS is based on the development of a cross-layer at the level of the network resource allocator (eNodeB) to filter the list of users to serve before it is passed to the scheduler. According to our simulations performed with a standard compliant LTE simulator, CAMS insures a higher QoE in terms of stalling number by comparison with the conventional scheduler; up to 87% improvement in the probability of no stalls when users move at 40 kmph. It also achieves a higher spectral efficiency; 1 bit per second per Hertz gain compared to the conventional scheduler. The contribution of this Chapter was published in our conference paper [5].

In Chapter 4, we develop and solve an optimization problem that balances user's QoE and system utilization cost by assuming a perfect knowledge of the user's future throughput variations. We define a balancing parameter to make the tradeoff between the operator's preferences and the user's preferences; a high QoE usually comes at a high system utilization cost and a low cost usually returns a low QoE. We prove the existence and the properties of the optimal solution: First (i), the transmission schedule is of a threshold type, and second (ii), the video bitrates are distributed in an ascending order. We optimally solve the problem by proposing an algorithmic approach based on graph-theory. Then, to solve it in a faster way, we propose an heuristic named aNticipating qoE With threshold sCheme And aScending biTrate levels (NEWCAST). NEWCAST is based on two sub-heuristics, one for setting the transmission threshold and one for setting the bitrate distribution. We evaluate the performance of NEWCAST under different values of the balancing parameter and different throughput real traces. Finally, by performing a comparison with baseline adaptive video streaming algorithms, we show that NEWCAST can achieve higher performance under a wise calibration of the tradeoff parameter. Some of the contributions of this Chapter were published in our conference paper [6].

In Chapter 5, we extend our study to the case where the throughput prediction is not perfect. We propose four algorithms: Short-TERm Newcast (STERN), Adaptive Short-TERm Newcast (A-STERN), shoRt-tErm Conservative newcAST (RECAST) and Short TeRm Enslaved nEwcasT (STREET) to make the approach of NEWCAST more robust to prediction errors. Each algorithm exploits the prediction of the throughput over successive short horizons. Based on our simulation results, we show that all our algorithms outperform NEWCAST in terms of video stalling mainly when the system utilization cost is prioritized. STERN and A-STERN achieve the lowest average numbers of stalls but give in return high numbers of quality switching. RECAST is the most stable in terms of quality switching. STREET is the closest to NEWCAST and gives near performances in terms of video average quality and bitrate distribution. Some of the content of this Chapter was published in our conference paper [7].

In Chapter 6, we implement both of NEWCAST and its derivative algorithms (STERN, A-STERN, RECAST and STREET) with the DASH-If-Reference player. For lack of material, we conduct our experiments in an emulated environment using the Linux bandwidth shaping tool to emulate the user's predicted throughput. We show and explain in a first time why the approach of NEWCAST, as defined in Chapter 4, is not suitable for real world streaming. We propose for that to adjust it, as well as STERN, A-STERN, RECAST and STREET, to finalize the experimental study. By distinguishing throughput under-estimation and throughput over-estimation cases, we evaluate the performance of each algorithm by repeating the same experiment many

times under different values of the balancing parameter. Our results show that, under overestimated throughput, all the derivative versions of NEWCAST achieve lower numbers of video stalls than NEWCAST. STREET being the most efficient one. Under underestimated throughput, however, NEWCAST globally achieves the highest performance.

In chapter 7, we develop a multi-user QoE optimization problem in which we maximize the users' global QoE assuming the knowledge of their future throughput variations but not their QoE preferences. To solve this problem, we define a single-user QoE problem that assumes the knowledge of the user's future throughput variations and his QoE preferences as well. We solve the single-user problem by proving the properties of the optimal solution then by proposing an heuristic inspired from the approach of NEWCAST. To solve the multi-user problem, we propose a closed-loop framework based on the single-user problem, users feedbacks and machine learning. This framework optimizes the QoE as well as feedbacks on previously optimized videos are collected. It acquires a progressive knowledge on the users' major profiles and accordantly adapts the video qualities for coming users. By using a single-layer neural network for machine learning, we show through simulations that our closed-loop framework achieves a high performance in terms of convergence and QoE maximization. The contribution of this Chapter was published in our conference paper [8].

1.4 Publications

The list of publications including international conferences and workshops are following:

International Conferences:

[C1]: **Imen Triki**, Majed Haddad, Rachid El-Azouzi, Afef Feki and Marouen Gachaoui. "Context Aware mobility Resource Allocation for QoE-Driven Streaming Services", *Wireless Communications and Networking Conference.* WCNC, April 2016, Doha, Qatar.

[C2]: **Imen Triki**, Rachid El-Azouzi and Majed Haddad. "NEWCAST: Anticipating Resource Management and QOE Provisioning for Mobile Video Streaming", *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, IEEE 17th International Symposium on A, June 2016, Coimbra, Portugal.

[C3]: **Imen Triki**, Rachid El-Azouzi and Majed Haddad. "Anticipating Resource Management and QoE for Mobile Video Streaming under Imperfect Prediction", *Multimedia (ISM), IEEE International Symposium on*, Dec. 2016, San Jose, CA, USA.

[C4]: **Imen Triki**, Rachid El-Azouzi, Majed Haddad, Quanyan Zhu and Zhiheng Xu. "Learning from Experience: A Dynamic Closed-loop QoE Optimization for

Video Adaptation and Delivery", *To be published in The 28th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications* , PIMRC, October 2017, Montreal, Quebec, Canada.

[C5]: Sudheer Poojary, Rachid El-Azouzi, Eitan Altman, Albert Sunny, **Imen Triki**, Majed Haddad, Tania Jimenez, Stefan Valentin and Dimitrios Tsilimantos. "Analysis of QoE for Adaptive Video Streaming over Wireless Networks", *To be published in The 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, WiOpt May 2018, Shanghai, China.

Workshop:

[W5]: **Imen TRIKI**, Rachid El-Azouzi, Majed Haddad. "Anticipating Resource Management and QoE Provisioning for Video Streaming." *11ème Atelier en Evaluation de Performances*, March 2016, Toulouse, France.

Chapter 2

Background

Contents

2.1 Introduction

Before starting the contribution works of this thesis, it is important to present some backgrounds about the state of the art and the different features of fields we are interested in, such as the resource allocation in 4G networks, the specifications of adaptive video streaming, the QoE concept in multimedia services and the basis of artificial neural networks. We organize this chapter in three major sections. In Section 2.2, we present the LTE standard and its main specifications that are related to our work. Then in Section 2.3, we review some important features of video streaming services, such as video characteristics, DASH standard and the QoE. Finally, in Section 2.4, we give a short overview on neural networks.

2.2 Overview on LTE mobile network

2.2.1 Architecture

LTE and LTE-advanced are standards for mobile communication technologies that appeared under the fourth generation of cellular networks. As we show in Figure 2.1, a typical LTE network is composed of the Evolved Packet System (EPS) and is directly connected to outer Packet Data Network (PDN) e.g., the Internet.

The EPS is comprised of the Evolved Packet Core (EPC) and the Evolved Universal Terrestrial Radio Access Network (E-UTRAN). The EPC is charged of taking the overall control of the User Equipment (UE), the E-UTRAN, however, is charged of controlling the radio functions as it contains the network Evolved Node Bs (eNodeBs) and the UEs. The EPC is connected to the PDN via the Packet data network Gate-Way (PGW), which is responsible for allocating the IP addresses to the UEs, the QoS enforcement, and further tasks such as packet filtering and policy enforcement.

The PGW is linked to the Serving GateWay (SGW) and the Policy Control and Charging Rules Function (PCRF). The major function of the SGW is packet routing and forwarding, but it has further tasks such as collecting the information for charging and acting as a mobility anchor for inter-eNodeB handover. The PCRF, however, is responsible for the QoS and the charging policy control.

The Mobility Management (MME) and the Home Subscriber Server (HSS) are other elements of the EPC. The MME supports the user authentication and the roaming with in addition the control and the process of the signaling between the EPC and the UE. The HSS, however, contains a database for the users' subscriptions.

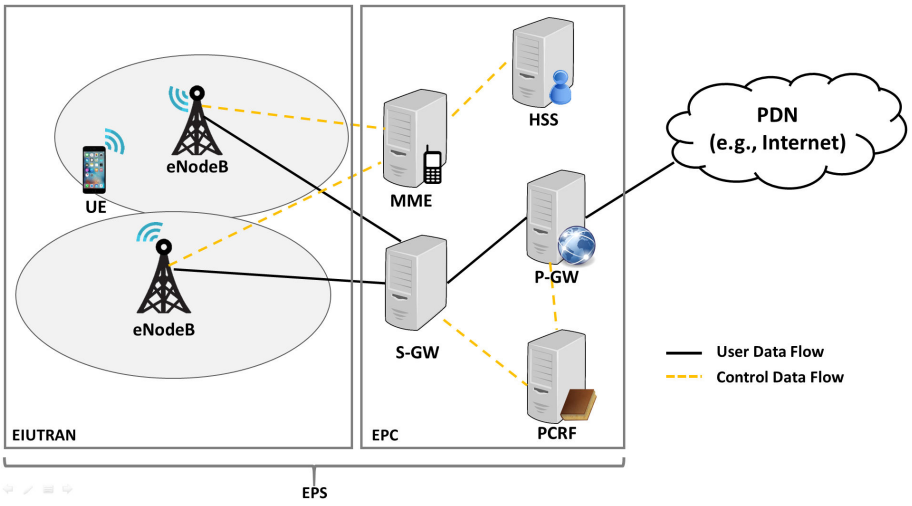


Figure 2.1: *Architecture of an LTE network.*

2.2.2 Physical resources

In LTE, the smallest element of resource allocation unit that can be assigned by the eNodeB schedular to the UEs is called Physical Resource Block (PRB). As detailed in Figure 2.2, a PRB is defined as a group of 12 consecutive subcarriers over one Time Slot (TS). In Orthogonal Frequency Division Multiple Access (OFDMA), each subcarrier brings 6 or 7 Orthogonal Frequency Division Multiplexing (OFDM) symbols (the modulated data) depending on the Cyclic Prefix (CP) type (either extended or normal) for a duration of 1 TS. Each single sub-carrier per one symbol period is called Resource Element (RE). LTE supports different cell bandwidths (1.4, 3, 5 10 15 and 20 Mhz). To each bandwidth is associated a number of PRBs. All the PRBs form over time what we call a Resource Grid (RG). This RG is shared between the UEs depending on the specifications of the norm and other scheduling metrics. For Multiple Input Multiple Output (MIMO) transmission there is one RG for each antenna.

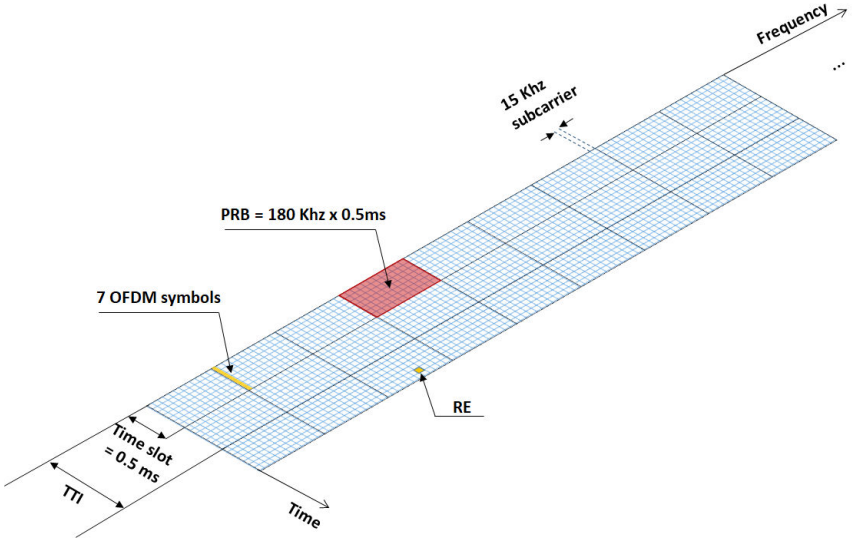


Figure 2.2: *LTE resources.*

In LTE, the PRBs are assigned per pair to the UEs, i.e., each user will have, if scheduled, one or multiple pairs of PRBs covering with that 1 Transmission Time Interval (TTI) of 1 ms duration. Each TTI represents in LTE one sub-frame of two TSs, and each 10 sub-frames represent one frame of 10 ms.

2.2.3 Data transmission

Modulation and Coding Scheme (MCS)

Each symbol of LTE RE is carried on a predefined number of bits depending on the type of modulation used by the system. As per Release 8 (R8) [9], LTE supports Quadrature Phase-Shift Keying (QPSK), 16 Quadrature Amplitude Modulation (QAM) and 64 QAM for downlink and QPSK, 16 QAM for uplink. Each modulation defines the capacity in bits (also called modulation order) for bringing the symbols.

Specifically, a QPSK symbol can be carried over 2 bits, a 16 QAM symbol can be carried over 4 bits and a 64 QAM symbol can be carried over 6 bits.

Accordingly, the higher the modulation order, the higher the number of bits transmitted per unit of time, but this does not imply a higher efficient throughput. In fact, along with the modulation scheme, there exists the modulation coding rate that defines the efficiency of the modulation. Hence, a 16 QAM modulation with a coding rate of 0.5 has 2 bits for carrying useful data and 2 bits for carrying redundant data. High coding rates are usually used with very good channel conditions.

In LTE, the combination of modulation and coding rate is called Modulation and Coding Scheme (MCS). As per R8, LTE supports 29 MCS in downlink (indexed from 0 to 28) and 23 MCS in uplink (indexed form 0 to 22). The MCS index (I_{MCS}), along with other LTE metrics, will determine for each user the amount of data to send or to receive over the following TTI. This amount of data is defined as the user Transport Block (TB).

Transport Block Size (TBS)

In LTE downlink/uplink, each user is assigned a TB (in bits) for receiving/sending the data over the system physical layer. Each TB is transmitted over 1 TTI as a part of a downlink/uplink system sub-frame. The size of a user TB, referred to as Transport Block Size (TBS), is decided upon the I_{MCS} assigned to the user and the number of PRBs (N_{PRB}) allocated to him by the network scheduler. In LTE, the TBS is indexed from 0 to 26. Each TBS index (I_{TBS}) is mapped to 1 or 2 I_{MCSs} as defined in the LTE Technical Specification 136.213 [10] (Section 7.1.7.1 for downlink and Section 8.6.1 for uplink). From the N_{PRB} and the I_{TBS}, the TBS is determined through the look up Table of [10] defined in Section 7.1.7.2 for downlink and uplink.

2.2.4 Link adaptation and scheduling

CQI reporting

The Channel Quality Indicator (CQI) is an indicator sent from the UE to the network eNodeB to inform about how the communication channel is good or bad for downlink/uplink transmission. CQI is a 4 bits integer (ranging from 0 to 15) based on the channel Signal-to-Interference-plus-Noise Ratio (SINR) value observed at the UE. It is used by the eNodeB to ensure two important functionalities: the *link adaptation* and the *scheduling*.

CQI reporting can be either periodic or aperiodic. Periodic CQIs are reported by the UE in periodic time intervals over the Physical Uplink Control CHannel (PUCCH) or the Physical Uplink Shared CHannel (PUSCH), whereas aperiodic CQIs are reported on the PUSCH and are triggered when the eNodeB wishes channel quality information e.g., in case of handover or loss of synchronization.

Report granularity of CQI can be one of three levels: wideband level, UE selected subband level, and higher layer configured subband level [11, 10]. The wideband report provides one CQI value for the entire system bandwidth. The UE selected

subband CQI report divides the entire bandwidth into multiple subbands, selects the best set of subbands for the UE, then reports one CQI value for the wideband and one averaged CQI value for the set of subbands. Note that, a subband is a set of k contiguous PRBs. The higher layer configured subband report provides the highest frequency granularity. It divides the entire bandwidth into multiple subbands, then reports one CQI for the wideband and multiple differential CQI values, one for each subband.

Link adaptation

Link adaptation is a method used to make efficient the use of the channel capacity. It is based on the Adaptive Modulation and Coding (AMC) technique [11], which adapts the modulation scheme and the code rate in the following way: if the CQI is good, higher-order modulation schemes with higher spectral efficiency are used like 64 QAM. In case of bad CQI, lower-order modulation scheme like QPSK are used, which are more robust to transmission errors but have lower spectral efficiency. The coding rate is also adjusted in function of the channel quality; high code rates come at high CQI values and low code rates come at bad CQI values. Overall, in LTE, the AMC technique has to ensure to the user a BLock Error Rate (BLER) value smaller than 10%. In Figure 2.3, we put a fraction of the CQI look up Table defined in the LTE Technical Specification [10].

CQI index	modulation	coding rate x 1024	efficiency
0	out of range		
1	QPSK	78	0.1523
2	QPSK	120	0.2344
3	QPSK	193	0.3770
⋮	⋮	⋮	⋮
13	64QAM	772	4.5234
14	64QAM	873	5.1152
15	64QAM	948	5.5547

Figure 2.3: *4-bits CQI table [10].*

Scheduling

Packet scheduling in LTE is performed in both: Time Domain (TD) and Frequency Domain (FD). The purpose of TD packet scheduling is to select among all the users the ones to be scheduled at each TTI. This selection is based on calculated priority metrics based for example on previous throughput calculation or on current channel condition. Note that TD scheduling uses wideband CQIs as it is not concerned with PRBs allocation. The purpose of FD scheduling, however, is to allocate the PRBs to the candidate users selected by TD scheduler, but it doesn't guarantee that all the users will be allocated frequency resources. In FD scheduling, there exist some specific priority metric algorithms for PRBs allocation, such as Round-Robin, Alpha-Fair and Best-CQI algorithms. The main purpose of scheduling algorithms is to make a tradeoff

between the hole system spectral efficiency (i.e., the system throughput) and fairness among users. Here are some brief specifications of some scheduling algorithms:

- Round-Robin (RR): Round-Robin is a channel independent scheduling algorithm. It allocates the PRBs cyclically to the users (one after the other) without seeing their CQI feedbacks. Hence, it provides the highest fairness, but it may provide poor performance in terms of spectral efficiency [12].
- Proportional-Fair (PF): Proportional-Fair scheduler tries to maximize the total throughput while providing each user with at least a minimal level of service. This is done by giving each user a scheduling priority that is inversely proportional to its anticipated resource consumption [12].
- Best-CQI: Best CQI is a channel dependent scheduling algorithm. Based on the users' CQI reports, each PRB is allocated to the user having on it the highest CQI. Which ensures the highest spectral efficiency [12].
- α-Fair: α-Fair scheduler defines a unifying mathematical formulation to fair throughput assignment [13]. The degree of fairness is expressed by the mean of a parameter α defined in $[0, \infty[$. α-Fair scheduler controls the tradeoff between efficiency and fairness. In particular, the case $\alpha = 0$ corresponds to the throughput maximization, the case $\alpha = 1$ corresponds to the proportional fair assignment and the case $\alpha \longrightarrow \infty$ corresponds to the max-min fairness.

2.3 Overview on video streaming services

2.3.1 Digital video characteristics

Container and format

A digital video is a file that contains video frames, audio and meta-data, all compressed and encapsulated in a video container. Video containers usually have their names in extension, which designs the video format they contain, e.g., the video format of the Audio Video Interleave (AVI) container, published by Microsoft, is .avi. Video containers are understandable by operating systems, and are mainly used for video transmission and storage, as the uncompressed native video files are huge-sized. In addition to the compressed video file, a video container may include some meta-data about the video such as its title, its resume, its duration, etc, and includes with that the list of video codecs (coder/decoder) that can be used for uncompression (decoding), e.g., an .mp4 video format can be decoded with an H.264 codec. Below in Table 2.1, we list some of the most popular video containers, their extensions and their publishers [14]. Then in Figure 2.4, we put a schematic of a typical video life stages.

Youtube actually uses 3 types of containers: FLV, MP4 and WebML. In order to provide compatibility with all browsers, devices, bandwidths and quality requirements, it converts all its uploaded videos into various formats using different codecs and stores them into video servers within its supported containers.

Container	Extension	Publisher
3GP	.3gp, .3g2	3GPP
AVI	.avi	Microsoft
ASF	.asf, . wma, .wmv	Microsoft
FLV	.flv	Adobe
MP4	.mp4, .mp4a, .mp4b, .mp4r, .mp4v, mp4p	Moving Picture Experts Group
MPEG	.mpg ,.mpeg	Moving Picture Experts Group
Quicktime	.mov, .qt	Apple
WebM	.webm	WebM project (Open source)

Table 2.1: *Some of the most popular video containers.*

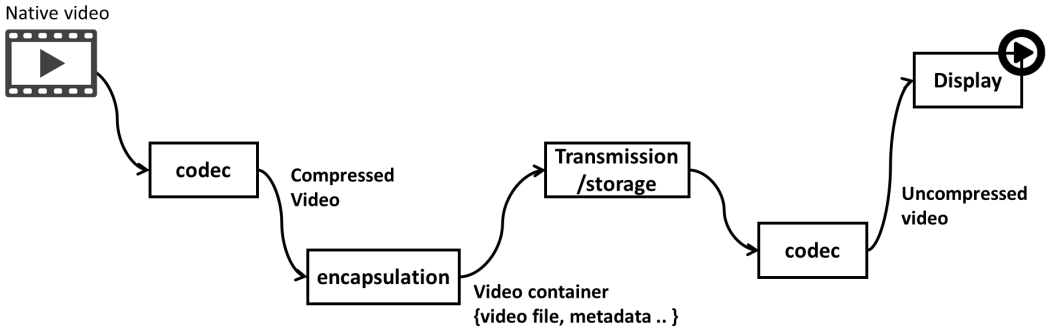


Figure 2.4: *Life stages of a typical video file.*

Codec

A video codec (coder/decoder) is a software used for coding and decoding audio-visual data. Its typical job is to compress the video file so that it can be stored or transmitted with a smaller size, then to decompress it when playing it back. Each codec has its own peculiarities, its own strengths and its own weaknesses, which makes the decision on the right codec to use a little bit complex. The most popular video codecs used in the web are DivX, FFmpeg MPEG-4, MPEG-4 (or H.264), Theora VP9, x.264 and Xvid.

Video compression is governed by two important factors: (i) Spatial redundancy and (ii) temporal redundancy. Spatial redundancy is the redundancy of information within a single video frame and is called intra-frame redundancy, e.g., repeated or similar pixel values in a large area of blue sky. Temporal redundancy, however, is the redundancy of information between a set of video frames, and is called inter-frame redundancy, e.g., in a video sequence where actors are only speaking, the scene background remains the same, which means redundant pixel values when moving from one frame to another.

The goal of using redundancy in video compression is to avoid encoding the similar or the near-similar pixel values that have already been encoded. In MPEG [15], inter-frame redundancy coding is used to create difference images, which are then

compressed spatially as shown in Figure 2.5.

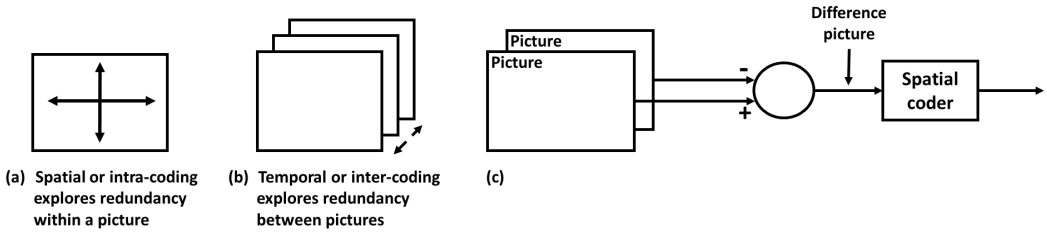


Figure 2.5: *Difference images in MPEG coding [15].*

Resolution, scan and aspect ratio

Video resolution is technically defined as the number of bits per unit of area per video frame, which also represents the density of pixels per video frame. In digital videos, the resolution is expressed only by the number of horizontal pixel lines, which is the video height in pixels, e.g., 1080p and 1080i are two video resolutions that define video heights of 1080 pixels. The number of vertical pixel lines, however, is deduced from the video resolution aspect ratio, which is the ratio between the width and the height of the video, e.g., 16:9.

Throughout history, video aspect ratio has evolved from 4:3 with digital TV and earlier computer displays, to 16:10 with widescreen computer displays then later to 16:9, which is the high-definition standard aspect ratio.

The letters "p" and "i" used at the end of the resolution names above are for indicating the type of video scan (display): "p" stands for progressive scan and "i" stands for interlaced scan. An interlaced scan paints all the odd lines first, then all the even lines of each video frame as in Figure 2.7. A progressive scan, however, paints all the lines of each video frame in sequence as in Figure 2.6.

Youtube supports many video resolutions including the Low Definition (LD) (240p and 360p) Standard Definition (SD) (480p), the High Definition (HD) (720p), the Full High Definition (FHD) (1080p) and the Ultra High Definition (UHD) (4K and 8K). It adapts the video resolution to the user bandwidth. Figure 2.8 contains a snapshot of a Youtube player with all the supported Youtube resolutions.

Figure 2.6: *Snapshot of a video progressive scan.*

Figure 2.7: *Snapshot of a video interlaced scan.*

Figure 2.8: *Youtube supported video resolutions.*

Frame rate

Video frame rate is the frequency at which consecutive video frames are displayed and is expressed in frames per second or fps. At the beginning edge of cinema, frame rate of films was set between 16fps and 24fps then was standardized to 24fps, which is defined as the minimum acceptable frame rate for acceptable video motion. With television broadcast, video frame rate increased to be matching the local frequency used for Alternating-Current (AC), hence, appeared the 30fps in US and Japan (as they use 60Hz AC) and the 25fps in Europe and Asia (as they use 50Hz AC). These frame rates have then been widely used for online videos. With the appearance of HD streaming and later, video frame rate moved to 60fps, producing with that crystal clear imagery with smoother motion.

Bitrate

Video bitrate is the unit of measure for video encoding and is expressed in bits per second. It represents the rate at which video bits are processed for display. In general, the higher the resolution the higher the birate used for encoding, since, as mentioned before, video encoding is perceived to make the video size match the channel/media support.

Video bitrate actually exists in two types: Constant BitRate (CBR) and Variable BitRate (VBR). With CBR, the entire video clip is encoded at a quasi-same bitrate regardless of its variable complexity (e.g., variable amount of motion), which makes the video perceived quality variable in time. With VBR, however, the video is encoded at different bitrates depending on the complexity of its sequences, which provides a constant perceived quality over time. VBR encoding actually produces better image quality than CBR even though it requires more processing time. CBR is particularly used in case of live event streaming or in case of satellite television broadcast where different channels are multiplexed into one limited bandwidth.

2.3.2 Video delivery methods

Web server vs. streaming server

For understanding video delivery methods, it is important to know the difference between a Web Server and a Streaming Server [16].
A web server is a computer system designed to host web sites and to serve web sites' content (web pages and files) to the clients upon requests using the HTTP. HTTP is actually the basic application protocol used to distribute information on the World Wide Web. It works over the Transmission Control Protocol (TCP) and typically uses port 80.

A Streaming server however, is a computer system solely designed to store video files for streaming delivery. It serves either compliant and directly connected-to players , or web browsers with compliant players and plugins. Streaming servers use for video delivery several real time protocols such as the Real Time Messaging Protocol (RTMP) and the Real Time Streaming Protocol (RTSP). They support User Datagram Protocol (UDP) as well as TCP.

Download method

In video download, the video file is stored in a web server and is assigned an Unic Resource Identifier (URI) to be referenced by the server web pages. When the client requests the video through his browser, he must wait for the entire video to be downloaded before starting the playback. At the end of download, a kind of meta-data are sent to the player to know how to display the video. As in typical file download, the video file is entirely stored (cached) at the user's device memory, which is an advantage and a drawback at the same time; it ensures a good display experience since the video is locally accessed by the player and can be played offline, but this may expose the user to content security issues. Overall, video download may not be the best solution for the user and may lead to the loss of bandwidth in case the user interrupts the download or doesn't like the content of the video.

Progressive-download method

Progressive download is in some references called pseudo-streaming. It acts the same way as video download with the difference of making the playback simultaneous to download upon starting the video. At the first request, the server sends to the player a meta-data about the video to know the range of prefetching before starting the playback. The whole remaining part of the video file is buffered as fast as the user's connection speed, which may cause some playback interruptions in case of low bandwidth. Progressive download enables the user to jump to different points in the video timeline even if the video data has not yet been downloaded. This is allowed by leveraging HTTP 206's Byte Range Requests and Partial OK Responses by the server upstream.

Streaming method

Video streaming is very close to progressive download in terms of the user's experience, nevertheless, instead of using a web server, it uses a streaming server for storing and streaming the video. With true streaming, the video file is splitted into chunks and is buffered to the user as soon as these chunks are requested. Unlike in progressive download, video chunks are transferred and instantly consumed by the player without being cached to the user's local memory, which looks safer. Just like progressive download, true streaming allows the user to seek to not yet downloaded parts of the video. As it is based on stateful connected protocols, the server can detect any environment changes and makes decisions on the video transfer data rate to improve the user's experience. At the beginning of streaming, initial bursts of video content are sent in a trick mode to the user to make the playback start immediately.

Adaptive streaming method

In adaptive streaming, the video can be streamed under multiple encoding bitrates depending on the user's link capacity and some quality adaptation metrics used by the media player. When the video is put on the streaming server, it is divided into multiple segments, each one encoded at different bitrates and indexed and referenced in a manifest file, this file contains meta-data about the video and is transferred to the player at the beginning of the streaming session. In adaptive streaming, the video is streamed over HTTP through an ordinary web server acting as an orchestrator to manage the player requests and the video segments transfer. Prior to each segment download, the player decides upon the most adequate segment-bitrate and sends it to the streaming server via HTTP requests. As soon as video data arrive, they are instantly consumed by the player but they are not cached to the user's device memory. Adaptive streaming, is assumed today to be the most efficient streaming technology and starts becoming widespread on video-sharing websites such as Youtube and Netflix. The norm specifications were standardized by the Moving Picture Experts Group (MPEG) under the name of Dynamic Adaptive Streaming Over HTTP (DASH or MPEGMPEG-DASH). Many proprietary solutions supporting adaptive streaming exist today such as Adobe's HTTP Dynamic Streaming, Apple's HTTP Live Streaming and Microsoft's Smooth Streaming.

2.3.3 DASH: Dynamic Adaptive Streaming over HTTP

History

Since its inception by Move Networks [17] in 2007, HTTP adaptive streaming technology has been widely used by vendors and service providers. Many companies such as Microsoft, Apple and Adobe have quickly introduced their proprietary solutions; Microsoft released its Smooth Streaming (MSS) [18] in 2008, Apple introduced its HTTP Live Streaming (HLS) [19] for use on iOS devices in 2009 then Adobe released its HTTP Dynamic Streaming (HDS) [20] in 2010 with Flash player.

The idea of standardizing this technology was issued by MPEG with a Call for Proposal for an HTTP streaming standard in April 2009. It is then in that year that MPEG started collaborating with experts and other standard groups such as the 3rd Generation Partnership Project (3GPP) to develop the specification of the norm. More than 50 companies were involved in the standardization work such as Microsoft, Netflix, and Adobe, and the effort was coordinated with other industry organizations such as the World Wide Web Consortium (W3C). Two years later (2011), that MPEG ratified its draft standard for DASH [21] by unanimous positive votes from 24 national bodies. In April 2012, DASH was published as ISO/IEC 23009-1:2012 [22].

In the same spirit of cooperation, the DASH Industry Form (DASH-IF) [23] was formed by the leading streaming companies to promote the adoption of MPEG-DASH and help transitioning it from a specification into a real business. DASH-IF was formally incorporated in September 2012 and accounts today 67 members spread throughout the world (Microsoft, Netflix, Google, Ericsson, Samsung, Adobe, etc.). "Dash.js" was therefore announced as the official player of DASH-IF Reference and Production.

Today, DASH becomes the standard choice for live events' streaming and on demand services like those provided by Amazon, LoveFilm and Netflix.

Typical system

In a typical DASH system, the video file is encoded at different bitrates called representations. Each representation is split into video segments of equal durations, e.g. 2 seconds. The video is then stored into a web server and streamed to the client via subsequent HTTP requests. DASH specifications include in the standard the definition of a Media Presentation Description (MPD) file to provide the player with meta-data about the video: how it is encoded, the temporal and structural relationships between the segments, the types of included streams (audio, video) and further.

The formats of MPD files and segments are defined in the standard via profiles. Each profile has its own specifications and inquires about the use of features to process the Media Presentation (MPD file and segments), for instance in the *onDemand* profile the video is streamed via byte-range requests and the representations are of single video files, in the *live* profile, however, each representation is split into segments and the video is streamed via segments' requests.

MPD files are XML documents written in a hierarchical data models as shown in the example of Figure 2.9. They are organized in periods. Each period describes a part of the content with a start time and duration and is organized in one or multiple adaptation sets. Commonly, each adaptation set stands for a type of content, e.g., one adaptation set for video streams, and many adaptation sets for audio streams (one for each language). Adaptation sets can also comprise subtitles or arbitrary meta-data. Video and audio adaptation sets are then organized in representations, each representation contains media segments with the same characteristics, either with the same encoding rate or the same resolution or the same codec etc. Further information about the segments' locations can be provided in the MPD file using a unique baseURL per representation, or using the entire list of segments or using a segment template. In some MPD files, representations may be organized in sub-representations to provide

further meta-data about the streams, and the media segments can be split into smaller sub-segments.

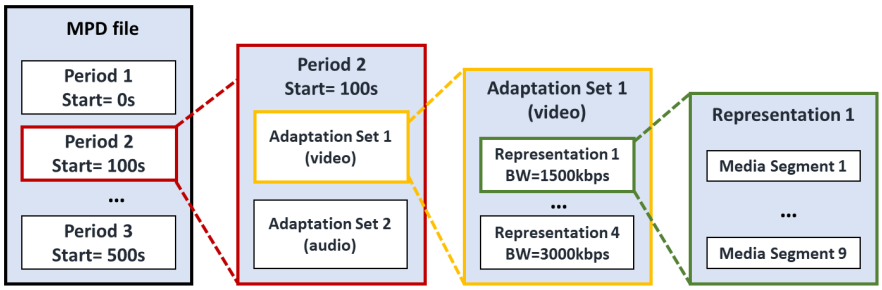


Figure 2.9: *Example of an MPD file hierarchical data model.*

DASH doesn't include specifications on how the bitrate adaptation should be performed, rather, it delegates to the client the task of defining its own adaptation strategy. As in general, the adaptation logic tends to provide the highest possible video quality. In the literature, many algorithms of bitrate adaptation have been developed for DASH streaming but research is still ongoing there. In the next section we review some of the referenced Adaptive BitRate (ABR) algorithms.

Adaptive Bitrate (ABR) algorithms

The difference between existing ABR algorithms mainly lies in the input information they use for bitrate selection, which may leverage on the network characteristics, the application-layer parameters or both of them. Each ABR algorithm behaves in a way to improve the end user experience, either in a *reactive* manner or in a *proactive* manner also called predictive.

With reactive ABR algorithms, the action is taken as soon as the event is detected. For instance, at the drop of the available bandwidth, the video bitrate is reduced to avoid buffer underflows in the future. These algorithms may mitigate some encountered streaming issues but they never give an insurance on how the system will behave after the problem. In counterpart, with proactive algorithms, the problem is identified in advance by predictive mechanisms through past observations and handled just before it comes out. For instance, if the bandwidth drop time is known in advance by the player, the video bitrate can be reduced long before it happens, which guarantees a freeze-less playback experience.

Apart from this classification which may look a bit general, ABR algorithms can be classified in three main classes depending on the input information they use for bitrate selection. These information give insight on how the network conditions are between the client and the server, they may be related to the bandwidth-rate, or to the playback buffer level or to the distribution of video segments download times. Although some algorithms may exploit a combination of these factors, usually one of them is dominant. Therefore, and according to the latest researches [24, 25], ABR algorithms can be classified Throughput Based ABR (TB-ABR), Buffer Based ABR (BB-ABR) or Time Based ABR (TmB-ABR).

- **Throughput-based algorithms:** TB-ABR algorithms are typically based on throughput estimation. Their commonly adopted rule in bitrate adaptation is to never chose a bitrate larger than the estimated throughput unless it is the lowest bitrate. According to [24, 25], a very common TB-ABR scheme [26, 27] follows a four-step adaptation model; First (i), the network bandwidth is estimated. Second (ii), the estimated throughput is smoothed using noise filters to avoid estimation errors due to bandwidth variation. Third, (iii) the bitrate is selected in function of the previously smoothed throughput. Finally (iv), the next segment request is scheduled upon estimating the next inter-request time. Almost all of today commercial adaptive-streaming players implement the estimation and the request scheduling parts in a similar way, though, they may differ in the way of smoothing the throughput and selecting the bitrate [28, 29, 26].

  According to [28], throughput estimate is either *instant* or *smoothed*; The *instant* throughput of a segment $i+1$ is the throughput measured during the download of segment i, whereas the *smoothed* throughput is the weighted sum of the n previous throughputs measured during the download of the n latest segments, $n > 1$ [30]. The main drawback of using the *instant* throughput is that it makes the bitrate selection react quickly to sudden throughput variations, which may impair the user experience.

  As for the bitrate selection, it was mentioned in [29] that it may be *aggressive* or *conservative*; With the *aggressive* method, the bitrate can jump from one level to another without caring about the jump size, whereas with the *conservative* method, the bitrate is increased/decreased progressively to not bother the user's perception.

  In conventional TB-ABR algorithms, the inter-request time is determined using a bi-modal scheduler which stipulates that the next segment request should be scheduled with a constant timer delay if the buffer is full or immediately if not. A very recent algorithm, PANDA [27], was conceived with a more sophisticated scheduler that drives the buffer occupancy towards the maximum buffer level, while matching, at the same time, the inter-request time to the necessary duration for current segment download.

  In Chapter 4, we chose a TB-ABR algorithm similar to that defined in "Microsoft's Smooth Streaming" to serve as a baseline algorithm for comparison with NEWCAST, our proposed bitrate adaptation method.

- **Buffer-based algorithms:** BB-ABR methods potentially adapt the bitrate based on the current buffer occupancy and the rate of buffer changes. Some of them use rate map functions that maps the bitrate to the buffer occupancy [31, 32]. Rate maps often define non-decreasing functions. Some others divide the buffer into ranges by introducing thresholds, then depending on the buffer occupancy they select the bitrate in order to keep the buffer evolve in bounded region [33, 34, 35]. This group usually use control loop feedback mechanisms, such as the Proportional-Integral-Derivative (PID) controller, to guide the bitrate adaptation [33, 34].

  In [32], a baseline BB-ABR algorithm was designed with a rate map function stipulating that : (i) If the buffer level is lower than a predefined minimum threshold, then the lowest bitrate is selected. (ii) If it is higher than a predefined maximum

threshold, then the highest bitrate is picked, and (iii) if it is in the bounded region, then the bitrate is selected according to a linear function. Two improved versions of this algorithm were later proposed by authors. In the first version, the buffer level minimum threshold is dynamically set and a segment size map is used rather than a rate map. In the second version, the same updates were maintained with in addition a throughput-based bitrate selection in the startup phase.

In [34], authors proposed a control loop feedback mechanism called Smooth Video Adaptation Algorithm (SVAA) that uses buffered video time in order to guide the bitrate adaptation. The basis of the SVAA is to pick the bitrate that matches the estimated TCP throughput multiplied by a buffer-based adjustment factor. This adjustment factor is itself a product of three sub-factors: the buffer size adjustment, the buffer trend adjustment, and the video segment size adjustment. The buffer size and the buffer trend adjustments are used to rectify the difference between the observed and the target buffer occupancy. The video segment size adjustment, however, is used to enable fast rate increase and compensate for the TCP's slow start phase in case of non-persistent HTTP connections.

SVAA strikes the balance between responsiveness and smoothness in adaptive streaming over HTTP. As shown by authors, due to the inherent TCP throughput variations, the buffer size smoothness and the video bitrate smoothness become conflictual. Globally, SVAA smoothly increases the birate when the available bandwidth increases and promptly reduces it in response to sudden TCP-congestion. The smoothness of the bitrate is realized by seeing the current buffer occupancy and the past m consecutive segments' bitrates. Although it uses throughput estimate, [24] classifies it as a BB-ABR algorithm as it tends to keep the buffer occupancy in a bounded region around a target level.

Authors of [36] conducted a comparison between some existent buffer-based algorithms that leverage on buffer ranging [37, 35, 38, 39]. According to their studies, the *most stable* approach is the one addressed in [39] since the criterion to maintain the bitrate depends only a range of buffer levels defined by two thresholds B2 and B3. In Chapter 4, we use this algorithm as a baseline BB-ABR for comparison with NEWCAST, our proposed bitrate adaptation method.

- **Time-based algorithms:** The TmB-ABR algorithms aim at synchronizing the video Segment Download Time (SDT) to the segment playout time Γ by selecting the most suitable video bitrate. SDT is determined by the time interval that separates the time of sending the request and the time of completely receiving the segment, which mainly depends on the segment bitrate (i.e., the segment size) and the bandwidth variation.

  In [40], authors proposed a TmB-ABR approach based on the calculation of the ratio between Γ and SDT for each segment. They defined for the bitrate adaptation two thresholds T_{down} and T_{up}. If the ratio is higher that T_{up}, then the next higher bitrate is selected, and if the ratio is lower than T_{down}, the highest lower bitrate is selected. The purpose being to make the ratio vary in between the thresholds. This approach achieves conservative switch-up and aggressive switch-down, but may lead to unnecessarily bitrate oscillations when the download rate varies significantly.

Authors in [41] proposed therefore an approach called ABMA based on the estimated SDT distribution and an introduced analytical model of the playback buffer to sustain the probability of video rebuffering under a predefined negligible threshold while at the same time optimizing the video bitrate adaptation. An improved version of this algorithm, ABMA+, was proposed in [24]. It consists of using, in addition, a pre-computed playback buffer map to select the maximum video bitrate that guarantees a smooth content playback. It takes into account VBR aspect as well.

2.3.4 QoE concept in video streaming services

2.3.4.1 QoE definition

Since the late 90's, the QoE concept has emerged in the field of communication services as a complement to the QoS concept to have more global views on the systems performance and their impact on the users' experience. Many definitions have been proposed for the QoE notion. In 2013, the European Network on Quality of Experience in Multimedia Systems and Services QUALINET, defined the QoE as [42]

> "The degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state."

This definition got a wide acceptance from community, and was afterwards adopted in 2016 by the International Telecommunication Union (ITU) [43].

QoE concept actually focuses on the entire user experience and is relevant to the User eXperience (UX). However, the UX is only limited to the use of a system or a service, whereas the QoE encompasses other features related to the content itself.

QoE is even different than the QoS but is most of the times related to. According to the ITU-T E.800 [44], the QoS is defined as

> "The totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service."

This definition of the QoS diverges from what the QoE is. QoS mostly deals with physical and measurable networks and delivery performance factors such as packet loss, jitter, throughput and delay, and, may address, in some cases, application-level factors such as encodings–in video streaming services–and their effect on the underlying network's performance.

A good QoS does not necessarily imply a good QoE, since the QoE depends on many complex features related to the user himself such as his mood, his expectations and his culture. A good QoE however requires a minimum goodness of QoS, but may not improve at higher QoS levels, since the service quality improvement may not be naturally perceived by the user.

2.3.4.2 QoE metrics

Quantifying the QoE is a very hard task as it encompasses many features that may be subjectively determined by the user. However, objective QoE features may sometimes be sufficient to reflect some of the user's experience. In video streaming services many objective key QoE metrics were defined to rate the quality of the streaming and to accordantly approximate the user's satisfaction. In this thesis we only focus on five objective QoE metrics: the startup delay, the video average quality, the video stalls ratio, the rebuffering delay ratio and the rate of bitrate-switching in case of adaptive video streaming.

- **The startup delay:** It is the initial waiting time before starting playing the video. The startup delay is measured in seconds and corresponds to the downloading time of a predefined number of video frames, also called startup threshold.
- **The video average quality:** In CBR streaming it defines the encoding birate of the hole video. In VBR streaming or adaptive VBR streaming, it defines the average per chunk (segment) video bitrate. It is measured in kilo bits per second.
- **Video stalls ratio:** A video stall is defined by the interruption of the video playback due to a lack of video frames in the playback buffer. A video stall is also called starvation or rebuffering event. Video stalls ratio is defined as the number of video stalls per video length in second.
- **Rebuffering delay ratio:** Rebuffering delay is the duration is second of a video stall. The ratio is defined by the quotient of all rubeffering delays over the video length in second.
- **Rate of bitrate-switching:** Bitrate-switching, also referred to as quality-switching, is the number of times the video quality changes during a streaming session. The rate is defined as the quotient of the total number of switches over the total number of video segments.

In the literature, subjective QoE metrics are often presented through the Mean Opinion Score (MOS) [45] or the user engagement [46].

MOS is commonly used for rating the quality of media services. It was originally set to evaluate the quality of audio delivery in telephony industry but was then generalized to evaluate video and audio-visual content. MOS is an arithmetic average of the users' subjective evaluation on a given system quality. Users' evaluation consists of assigning integer scores to their personal opinions typically ranging from 1 to 5, where 1 is for the lowest perceived quality and 5 is for the highest perceived quality.

The Absolute Category Rating (ACR) is one of the most commonly used method in media quality evaluation, it maps the quality rating from Bad to Excellent to numbers from 1 and 5, as seen in table 2.2. Other standardized MOS ranges exist and may be differently used for evaluation in function of the underlying purposes of tests. Most of them are defined in the ITU-T recommendations P.800 [45] and P.910 [47].

In Chapter 7 we use the MOS in the design of a QoE optimization framework that expresses the QoE in function of the aforementioned objective QoE metrics.

Rating	Label
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Table 2.2: *Absolute Category Rating (ACR).*

2.3.4.3 QoE estimation

In the literature, different studies are performed to model the user's QoE as an explicit function of measurable and quantitative metrics. Usually, the models are proposed then evaluated through the computation of MOS. Some works claim that the QoE can be directly mapped to the QoS metrics such as the throughput, the jitter and the packet loss [48, 49, 50]. In [48], a machine learning technique was proposed to build models in an online fashion from customer feedback. These models map the network parameters (NQoS) and the application-level parameters (AQoS) with the users QoE. In the same direction, authors in [49], investigated the correlation between QoS and QoE through subjective measurements of the users experience under different network conditions. They showed how Machine Learning methods such as Naive Bayes, Support Vector Machines, k-Nearest Neighbors, Decision Tree, Random Forest and Neural Networks can help in building accurate and objective QoE models that correlates low-level parameters (delay, packet-loss and jitter) to high-level quality features. Since Machine Learning techniques require a large number of datasets and are extremely time and process consuming, authors in [50] proposed in their QoE modeling to use the Factor Analysis technique which is a statistical method that allows parsing a large set of variables (generally uncorrelated) into a small set of uncorrelated factors required for extracting the model. To explore the QoE function, the authors were based on various parameters relevant the QoS, the bitstream and to video quality.

Other recent works found that the QoE could be expressed through some application metrics such as the frequency of video freezing (stalls), the startup delay, the average video quality and the dynamic of the quality changing during the streaming session [51, 52]. In [51], authors were seeking for expressing the user engagement as function of the video quality metrics, where the user engagement can be the video playtime or the number of visits to a website. They argued that the hidden interdependencies between the metrics and the confounding effects of the user personal features may make the task complex but can be handled by choosing a suitable Machine Learning approach and by carefully incorporating the features into the learning process. In [52], the QoE was modeled as a linear function of four objective metrics: The average quality, the startup delay, the rebuffering and the average quality variation. Further studies in [53] revealed that the HTTP Adaptive Streaming (HAS) profile–defined as the sequence of segments quality levels–is quite sufficient and more accurate than the radio scenario parameters to predict the QoE. Whereas in [54], the QoE was modeled by exploiting a psychological phenomenon known as "Primacy Effect and Recency Effect" which says that people are more affected in their short-term memory by the initial and the most recent information. The impact of video bitrate distribution was quantified based on that. A more sophisticated QoE evaluation was

then realized by mapping the real-time bitrate and the video content type into the QoE model.

In Chapter 4, we design an approach for bitrate distribution that takes into account the recency effect and some of the QoE objective metrics. A similar approach is proposed in chapter 7 with, in addition, the incorporation of neural networks.

2.4 Overview on Artificial Neural Networks for Machine Learning

2.4.1 Overview on Machine Learning

Machine Learning is nowadays one of the most used methods in artificial intelligence that gives computers the ability to learn tasks based on raw data without being explicitly programmed. It is applied in a wide range of applications such as data mining, natural language processing and image recognition.

Machine Learning tasks can be typically categorized in three major classes: supervised [55] and unsupervised [56], depending on wether the learning is performed through incoming signals or feedbacks. Halfway between these classes, a mixture class known as semi-supervised [57] class may be distinguished.

In supervised learning, the system is given a predefined set of labeled training samples comprised of input data and their outputs. The goal of the program is to learn the general rule that maps inputs to outputs to be then able to accurately estimate the output when given new input data. The mapping rule is generally called *Hypothesis* and is approximated by training the data samples using a learning algorithm. Among supervised Machine Learning methods we may cite the Support Vector Machines [58], used for classification, regression and outliers detection, the Gaussian Processes [59], used for regression and probabilistic classification and the Naive Bayes methods [60], used for classification.

In unsupervised machine learning, only unlabeled input data are provided to the system and it is up to the program to find patterns and relationships between them. Among unsupervised learning methods we may cite the Dimensionality Reduction [61] and the k-Means Clustering [62].

Artificial Neural Networks (ANNs) [63] are also part of Machine Learning methods and may be used with both supervised and unsupervised learning. They are used for clustering, classification, pattern recognition, and further applications.

2.4.2 Concept of Artificial Neural Networks

Analogy with Biological Networks

ANNs are computing systems inspired from Biological Neural Networks (BNNs) that constitute the brain. They are similar to BNNs in two major points: First, both of

them acquire knowledge through learning, and, second their acquired knowledge is stored within synaptic weights that constitute inter-neuron connection strengths.

In Figure 2.10, we put a schematic representation of the human biological neuron structure (on the left) and its mathematical model (on the right).

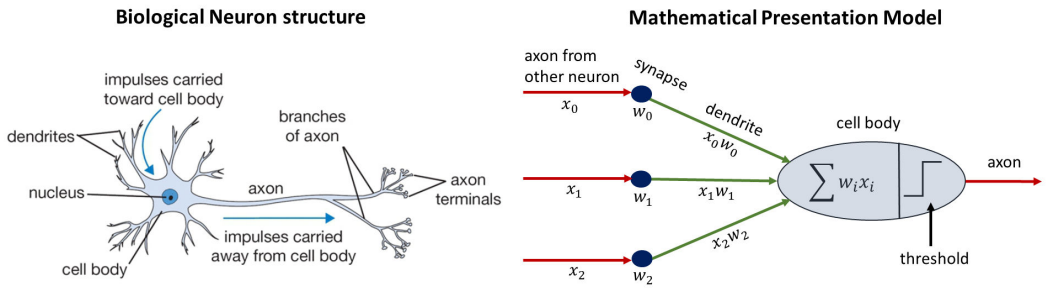


Figure 2.10: *Biological neuron structure and mathematical model [64].*

As depicted in the figure, a biological neuron is composed of dendrites, a soma (cell body) and an axon. The dendrites are responsible for carrying signals received from other neurons. At the cell body, the incoming signals are summed up to form the neuron own signal, if the formed signal intensity reaches a threshold value, the neuron fires the signal to the next connected neurons through the axon. The neurons are interconnected with each other via synapses. The strengths of the synapses, called synaptic weights, are responsible for weakening or strengthening the signals before transmission. Therefore, in the mathematical presentation model each dendrite is assimilated to a weighted input $w_i x_i$ where x_i is the signal input received from other neuron and w_i is its corresponding weight.

Artificial neurons are also presented with the same mathematical model. However, to scale up the neuron response, a *bias* may be added to the neuron weighted inputs, and, to get the output at non-binary values, other *Activation Functions* may be used, such as linear and sigmoidal functions. In Figure 2.11 we put a general representation of an artificial neuron. Some of the commonly used Activation Functions are listed in Figure 2.12.

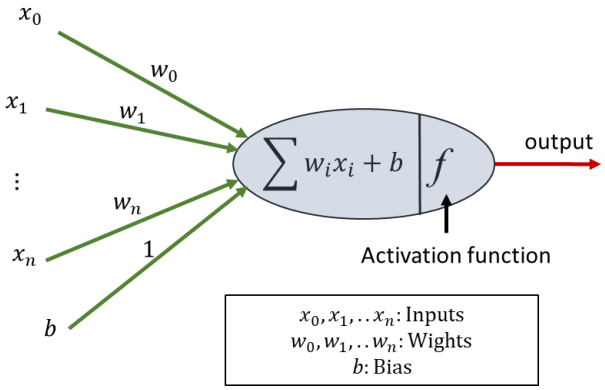


Figure 2.11: *Artificial neuron representation.*

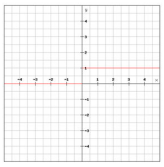
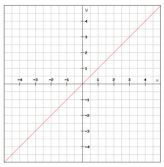
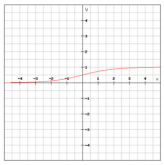
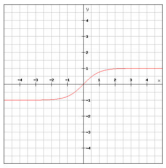
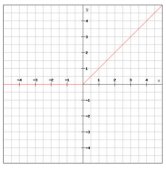
Name	Step Binary	Linear	Sigmoid	Tanh	Rectified Linear
Plot					
Equation	$f(x) = \begin{cases} 1 \; if \; x \geq 1 \\ 0 \; if \; x < 1 \end{cases}$	$f(x) = c\,x$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x) = \begin{cases} x \; if \; x \geq 1 \\ 0 \; if \; x < 0 \end{cases}$

Figure 2.12: *Activation Functions in Artificial Neuron Networks.*

Structures of ANNs

A typical ANN is comprised of one or more interconnected neurons, thus can be viewed as a weighted directed graph in which the neurons represent the nodes and the connections between the nodes represent the weighted edges. there exists many types of neural networks, each type is characterised by its structure and its Activation Function(s). *Feed-Forward Networks* [65], for instance, have the simplest structure as they don't implement cycles/loops in their graphs. They move the data forward from input nodes to output nodes in only one direction. *Recurrent Networks* [66], however, process memory and implement cycles in their graphs, which makes them able to exhibit dynamic temporal behavior.

The most commonly used ANNs are structured in layers: input, hidden (if exist) and output layers. Each layer comprises one or multiple nodes. As in Feed-Forward Networks (Figure 2.13), the input layer is the first layer and comprises *passive* nodes where each node receives single input data from outside world and duplicates it to the following nodes without processing. The output layer is the last layer and comprises the last *active* nodes of the graph. In between these layers, one or more hidden (intermediate) layers may exist. They comprise *active* nodes where each node receives information from previous layer nodes and forwards it after processing to next layer nodes.

To compute the depth of an ANN, only hidden layers are taken into account. A network with more than 3 hidden layers is often called *Deep Neural Network*. In chapter 7 we deploy a single-layer neural network (without hidden layers) with a single output node that uses a linear Activation Function (the Identity function).

Learning in supervised ANN

Supervised ANNs learn by adjusting their weights and biases iteratively to yield desired outputs; the input data are trained using a specific set of rules known as a learning algorithm. Many learning algorithms exist for supervised training models in ANNs. Among the simplest and the most popular ones we cite the *Gradient Descent* [67] which trains the ANN by supervising the error between the network output and the desired output. The weights in the Gradient Descent approach are updated in a way to minimize the error by moving oppositely to its Gradient vector. The *Gradient*

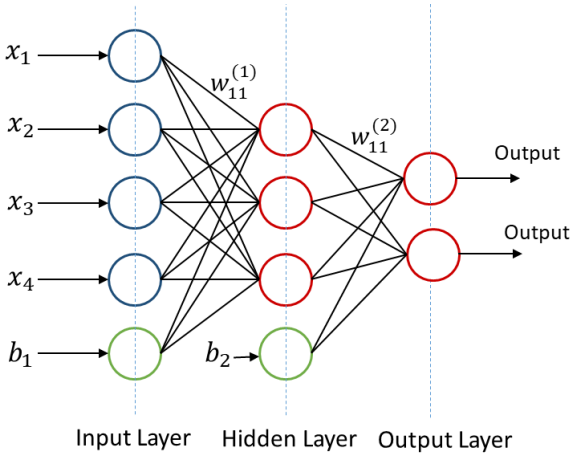


Figure 2.13: *Example of a Feed-Forward ANN with one hidden layer.*

Delta rule is the basic version the Gradient Descent and is used in case of single-layer networks. For multi-layer networks, an extended version, called the *Back Propagation rule*, can be used. It adjusts the weights of the nodes' connections by propagating the error backward from output layer to input layer.

There are two general approaches for supervised training in ANNs (also for unsupervised training), usually called *batch* (offline) learning and *online* learning [68]. A mixture of these two approaches is in some references called *mini-batch* learning [69]. In batch learning, the adjustment of the weights and biases is done only if all the training dataset is presented to the network (Figure 2.14.a), which may be very consuming in processing time and memory when the training dataset is big sized. In online learning, the adjustment of the weights and biases is done at the presence of each training sample to the network (Figure 2.14.c), which was proven to yield better models with the Back-Propagation rule. In the mini-batch learning, the adjustment of the weights and biases is performed as soon as m training samples are presented to the network (Figure 2.14.b), m is defined as the *mini-batch size*.

In Chapter 7 we implement the Gradient Delta Rule with the mini-batch training approach to learn the QoE function within an optimization QoE problem.

2.4.3 Learning with the Gradient Descent Delta rule

Consider a training dataset $\{(X^{(i)}, y^{(i)})\}_{1<i<m}$ of m samples where each training sample i is a couple of input vector $X^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ and a desired scalar output $y^{(i)}$, and let $h_{W,b}$ be the Hypothesis function of the single-layer neural network described in Figure 2.15 such that

$$h_{W,b}(X^{(i)}) = f(\sum_{k=1}^{n} w_k x_k^{(i)} + b) = f(W^T X^{(i)} + b) \tag{2.1}$$

where $W = (w_1, \ldots, w_n)$ is the network vector of weights and f is a derivable Activation Function of the network.

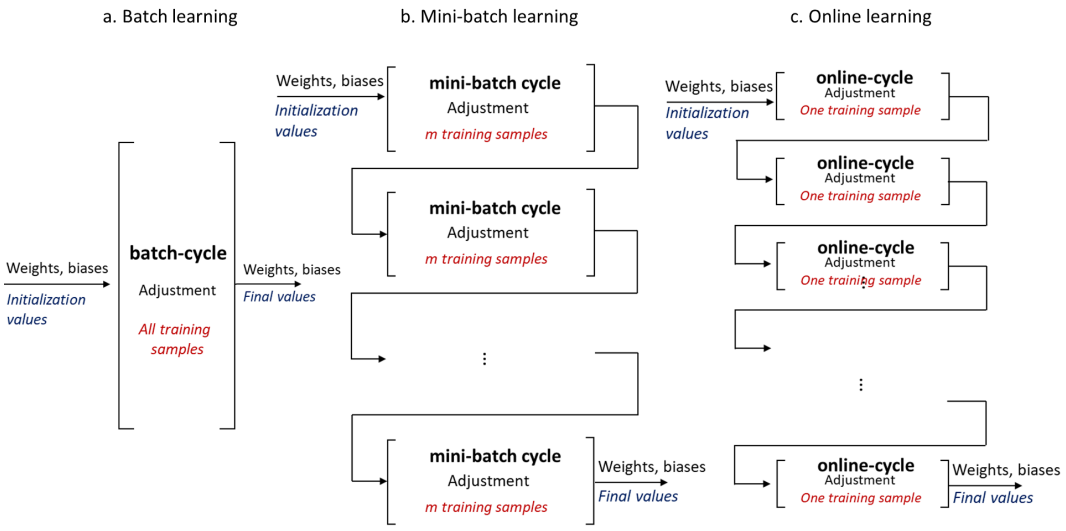


Figure 2.14: *Batch, mini-batch and online learning.*

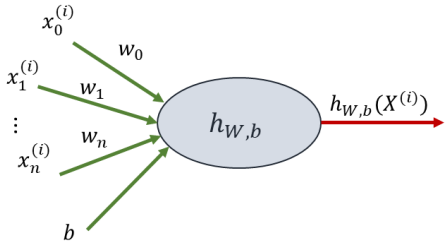


Figure 2.15: *Single-layer ANN.*

$h_{W,b}(X^{(i)})$ actually denotes the i^{th} estimated output of the network and should be driven to follow the desired output $y^{(i)}$.
For each training sample $(X^{(i)}, y^{(i)})$, we define the i^{th} sample squared-error cost function as:

$$J(W, b; X^{(i)}, y^{(i)}) = \frac{1}{2}(h_{W,b}(X^{(i)} - y^{(i)})^2$$

Overall the m training samples, the global cost function is defined as:

$$J(W, b) = \frac{1}{m}\sum_{i=1}^{m} J(W, b; X^{(i)}, y^{(i)})$$

In case of batch Gradient Descent, the weights and bias are iteratively updated in a way to reduce $J(W, b)$, i.e, they move oppositely to the direction of the cost function Gradient vector $\nabla J(W, b)$:

$$w_k = w_k - \alpha \frac{\partial}{\partial w_k} J(W, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(W, b)$$

α here denotes the learning rate. A large α may lead the weights to oscillate infinitely

without converging to the desired values, and a small α may lead the learning to converge slowly. Therefore, it should be wisely set before starting the training.
As in general, the weights and the bias should be initialized to very small values.

2.5 Conclusion

In this Chapter, we have reviewed some backgrounds of this thesis namely the resource allocation in LTE networks, the specifications of DASH standard, the QoE concept in video streaming services and the basis of neural networks. By looking through the literature, we have distinguished three classes of adaptive bitrate algorithms where the most prominent ones are the throughput based and the buffer based algorithms. We have further related some existing works on QoE modelling and have shown that objective QoE metrics are as important as subjective metrics in QoE estimation and quantification. However, to the best of out knowledge, no explicit function has been proposed as an exact formula to quantify the QoE, and researches are still ongoing there. The QoE models used in this thesis are partially inspired from state of the art QoE modelling and analysis.

In the next Chapter, we present our first contribution which studies the QoE through some objective QoE metrics in real (i.e., non adaptive) video streaming under the assumption of a specific network topology and a specific mobility of users.

Chapter 3

CAMS: Context-Aware Resource Allocation for Real Video Streaming

Contents

3.1 Introduction

Most of the existing works on scheduling algorithms in mobile networks are only valid for slow varying fading scenarios. Nevertheless, in mobile environments, such an assumption generally fails especially with high speed users, as the Channel State Information (CSI) varies over a faster time scale.

In this Chapter, we propose a fast and efficient resource allocation mechanism for real (i.e., non adaptive) video streaming that considers the high mobility of users as an opportunity rather than an obstacle [70, 71]. This mechanism, which we call CAMS, profits from the users' mobility context along with the packet prefetching process of the streaming video player to improve the overall spectral efficiency and the QoE. In particular, we deploy an Heterogeneous Network (HetNet) topology with Macro Base

Stationss (MBSs) and Small Base Stationss (SBSs) [72] distributed all along a highway, and assume that the users move at a constant speed over that highway.

The key idea of CAMS is to exploit the diversity of the users channel states by making them switch from active to inactive states and vice-versa during their streaming sessions. Active users will be allocated resources to prefetch video packets as much as their channel states allow, whereas inactive users will only play their already prefetched video data. To do so, we set a threshold of SINR value and serve only users with higher SINR.

A similar idea to CAMS has been adopted in HetNet for intelligent cell selection. The concept consists of dynamically distributing the users between macro and small cells depending on their signal strengths and the network load level in order to enhance their throughput and their experiences [73, 74]. While this concept globally increases the network capacity, CAMS improves the spectral efficiency and the users' QoE.

We organize the Chapter as follows: In Section 3.2, we describe the system model, namely the network topology, the mobility of users and the video streaming model. In Section 3.3, we present the approach of CAMS, its brought and how it can be implemented in a Base Station. Then in Section 3.4, we review the simulations setting and the numerical results. Section 3.5 concludes the Chapter.

3.2 The system model

3.2.1 The network topology

In this work, we adopt an HetNet topology with MBSs and SBSs covering a highway. We assume the MBSs to be very spaced and very distant from the highway and the SBSs to be very close and distributed one after the other along the highway. We set the SBSs to have the same configuration (same transmission power, same antennas, same scheduler etc ..) and to be equally distant from each other to form similar coverage areas as depicted in Figure 3.1.

3.2.2 The mobility of users

We adopt a high vehicular mobility model in which K users move along the highway in the same direction at a constant speed. We assume the users to be initially distributed over 3 equidistant lines as depicted in Figure 3.2. During their streaming sessions, all the users undergo multiple handover processes to switch from one cell to another. We adopt the Event $A3$ handover based on the measurement of the Reference Signal Received Power (RSRP)[1]. If the RSRP received from a neighboring cell exceeds the RSRP measured on the serving cell by a predefined offset, then the handover event is triggered [75].

[1] RSRP is a measure of the received signal strength of a cell at a UE and it is measured based on the strength of predefined reference signals that cells broadcast.

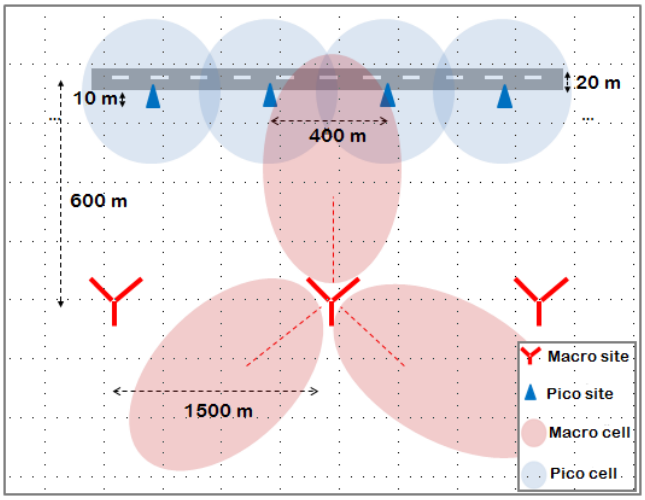


Figure 3.1: *The highway network topology.*

Figure 3.2: *The Users' mobility.*

3.2.3 The video streaming traffic

We consider real (i.e., non adaptive) video streaming where video files are stored in video streaming servers at unique resolutions (e.g., $720p$ [76]). We assume that each video file is divided into chunks of equal durations. Each chunk is comprised of video frames and each frame is divided into slices (or packets) of equal sizes.

When a client requests for a video file to a streaming server, the video slices are streamed one by one from server to the client's serving SBS, then from the SBS to the client. The transmission rate of the video on the client's radio link will depend on the link conditions.

At the arrival of video slices, video frames are reconstructed by the player then displayed after a threshold number x_0 of frames (hereinafter called startup threshold) is prefetched. During the playback, the client continues streaming the video till the end of download. When the transmission rate declines and the playback buffer falls empty of frames, a video stall (also called starvation) occurs and a prefetching stage is introduced till having again x_0 frames in buffer. The playback is resumed afterward.

In Figure 3.3, we illustrate the streaming process from server to end-user.

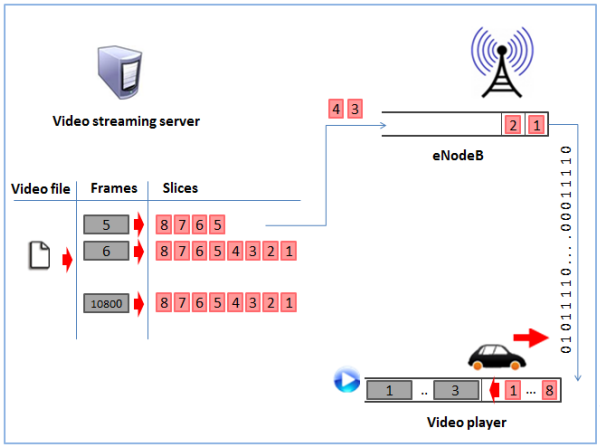


Figure 3.3: *The video streaming traffic.*

3.3 CAMS: Context-Aware Mode Switching for real video streaming

3.3.1 Concept of CAMS

The mode switching approach has been already widely used by a variety of services and mobile applications, including elastic services such as e-mail and instant messaging. Instant messaging is for instance only occasionally available when the receiver is in a covered zone, otherwise, the message holds offline till being effectively sent the next time the receiver is covered.

The mode switching approach we propose in CAMS is designed to make users alternate from active to inactive states and vice versa depending on their radio channel conditions. We call active zone each area of the network where the radio conditions are similar or better than a predefined state. In contrast, an inactive zone corresponds to an area of worse radio conditions. We define the limit between active and inactive zones as a threshold number of SINR value. A user is then assumed to be in active state when his latest measured SINR value is above this threshold, and in inactive state otherwise. In Figure 3.4, we illustrate the switching mode approach of CAMS under the highway network topology and the perpetual mobility of users.

When the user is active, he is admitted in the list of users to schedule the next TTI, otherwise he is removed from the scheduling list to not receive data the next TTI. As the users are in permanent motion across the cells, they end up all passing through active and inactive zones. This is due to the fact that the channel conditions of mobile devices are time-varying and location-dependent.

3.3.2 Brought of CAMS

The merit of CAMS approach lies in the fact that high SINR users will be given more chance to store as much video data as their channel conditions allow since worse

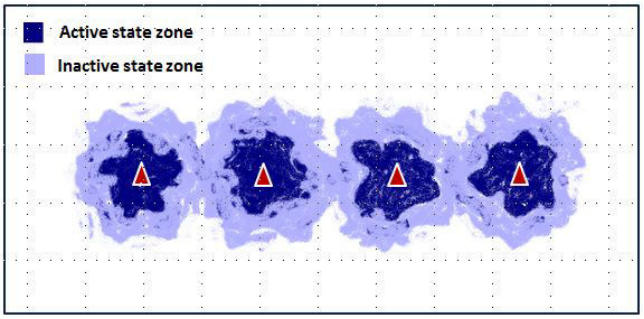


Figure 3.4: *Active and inactive zones as defined in CAMS.*

SINR users will be omitted from the scheduler. Note that, when the channel conditions are bad, the packet loss rate is high and the spectral efficiency is low, which makes video prefetching more interesting in areas of good channel conditions.

We resume the brought of CAMS mechanism in three major points:

- It increases the overall spectral efficiency since low SINR users will release their Resource Blocs (RBs) to be more efficiently used by higher SINR users. The spectral efficiency will depend on the setting of the SINR threshold.
- It gives high SINR users more chance to be allocated resources than with conventional schedulers. This will allow them to quickly finish streaming their videos, giving the chance to other users in the queue to be scheduled afterward.
- CAMS will not condemn users with low SINR values, this is because all the users are in perpetual motion. Given the network topology, all the users will be given the chance to go across active zones. Moreover, by virtue of video buffering mechanism, users can remain inactive for some time without degrading their QoE.

To conclude, the approach of CAMS allows for better radio resource management, provides improved overall mobile-broadband performance, and allows operators to maintain a more seamless user experience.

3.3.3 Implementation of CAMS

CAMS can be deployed with few modifications in the existing network architectures. A simple way to do this consists of adding a kind of cross layer behind the conventional schedulers to capture the users latest SINR values and to restrict the list of users to serve each TTI. By doing so, users with SINR values under the threshold will be prevented from being allocated resources. However, the efficiency of CAMS will depend on the threshold setting, the conventional scheduling metrics and the strategy of the scheduler.

3.4 Simulations and numerical results

3.4.1 Simulation tool

For the simulation part, we use the LTE Vienna simulator, implemented with Matlab and published under an academic non-commercial use license by the TU Vienna Institute of Telecommunications [77]. This simulator implements a set of 3GPP LTE standard specifications. It uses simplified models and capture the essential characteristics of the standard with high accuracy and low computational complexity. The simulator is available under uplink and two downlink versions. For real video streaming we use the downlink system level version. The main event of a simulation is the transmission of a subframe from eNodeB to end-users, which corresponds to a TTI in the LTE standard.

3.4.2 Simulations setting

We consider an HetNet with MBSs and SBSs covering a highway as shown in Figure 3.5 and detailed in Table 3.1. We set the number of users to 12 and distribute them initially over 3 lines. We fix the inter-distance between users to 20 meters and the speed of vehicles to 40 kmph then to 120 kmph.

We configure the video streaming as detailed in Section 3.2. Each video is 2 minutes of length and defines an HD quality with a frame rate equal to 60 fps (HD $720p60$). For analysis purposes, we do not consider transmission coding errors. Thus, we disable the Hybrid Automatic Repeat reQuest (HARQ) algorithm defined at the user Medium Access Control (MAC) layer. Further details on the video traffic configuration are put in Table 3.2.

We apply CAMS with the conventional Alpha-Fair scheduler at the level of both MBSs and SBSs.

Parameters	**Macro sites**	**Small sites**
Number of sites	7	16
Inter-sites-distance	1000 m	400 m
Tx-Power	40 watts	1 watt
Scheduler	Alpha-Fair: α=0.6	
Bandwidth	20 Mhz	
Frequency	2.14 GHz	
RB bandwidth	180 KHz	
Path-loss	TS36942 / urban model	
Shadowing	"Claussen" model	

Table 3.1: *Network configuration.*

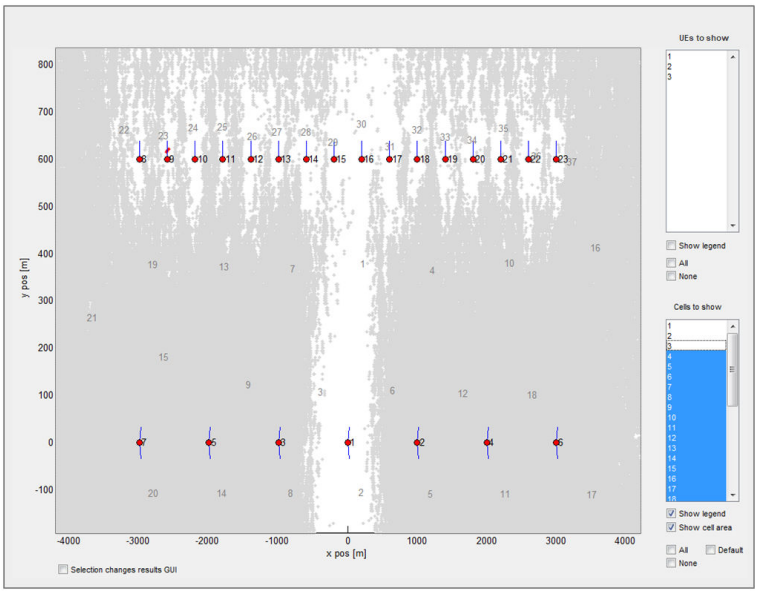


Figure 3.5: *HetNet map topology with Vienna simulator.*

Parameters	values
Video quality	HD 720p60
Frames/video	10800
Slices/frame	8
Slices/TTI	900
Slice size	1302 bytes
HARQ	disabeled

Table 3.2: *Video traffic configuration.*

3.4.3 Simulations results

We evaluate the performance of CAMS by reviewing the network performance and some objective key QoE metrics. The setting of the SINR threshold is performed based on empirical observations and not on analytical studies. For the sake of clarity and consistency, we only show results of SINR thresholds with significant performance gain.

Spectral efficiency

We define the spectral efficiency of a given user in a given TTI as the total number of bits allocated by the network to that user in that TTI divided by the user's allocated bandwidth (RBs length in Hz). Thus, we compute the average spectral efficiency of each user by dividing the sum of all his spectral efficiency values over the simulation length in TTIs.

In Figure 3.6, we show the variation of the average spectral efficiency per user in function of the SINR threshold for the two aforementioned speeds: 40 kmph and 120 kmph.

As a first observation, we notice that the variation of the SINR threshold does not impact the spectral efficiency in the same way for the two speeds in question. Results show that the spectral efficiency varies more significantly from one threshold to another with the lowest speed. This can be explained by the fact that, when a user moves slower, he spends longer time in active zones which makes him capture different spectral efficiency values. This highlights the impact of varying the threshold.

A maximum spectral efficiency is interestingly noticed for an SINR threshold equal to 12 dB at 40 kmph with almost 1 bps/Hz more than the reference model (without CAMS). Which corresponds to a gain of 23%. This can be justified by the fact that the RBs used by low SINR users in the reference model have been released with CAMS to be more efficiently used by higher SINR users.

When the threshold increases and goes beyond 12 dB, CAMS becomes very selective and the users spend much time in inactive states. Wich explains the degradation of the average spectral efficiency.

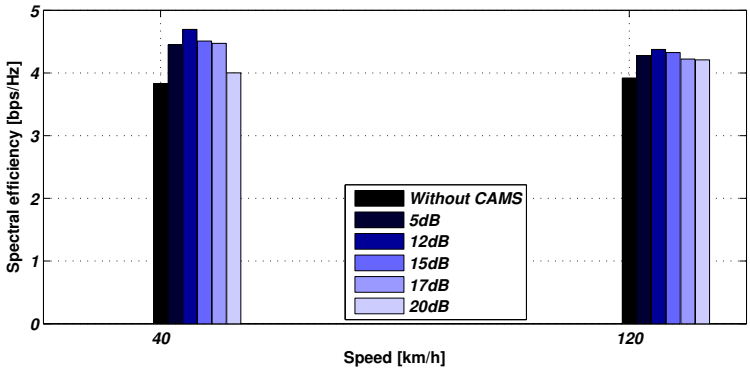


Figure 3.6: *Spectral efficiency.*

Fairness

Along with the spectral efficiency, we examine the fairness degree among all users. A commonly used metric is the Jain's fairness index [78] which is given by:

$$J(r_1, r_2, \ldots, r_K) = \frac{(\sum_{k=1}^{K} r_k)^2}{(K \cdot \sum_{k=1}^{K} r_k^2)}, \tag{3.1}$$

where K is the number of users and r_k is the throughput of user k divided by the number of the RBs assigned to him.

Figure reffig:fairness illustrates the Jain's fairness index computed with the reference conventional scheduler (without CAMS) and with CAMS at low and high speeds and different SINR thresholds. With the conventional scheduler, we obtain a fairness index very close to 1. When we apply CAMS, the fairness index slightly decreases proportionally to the threshold value. The impact of varying the threshold is rather obvious with low speed than with high speed, but globally, the fairness index remains relatively high and very close to 1. We mainly notice a Jain's index equal to 0.952 and

0.960 at low and high speeds, respectively, with a threshold of 12 dB, versus 0.994 and 0.995 with the conventional scheduler.

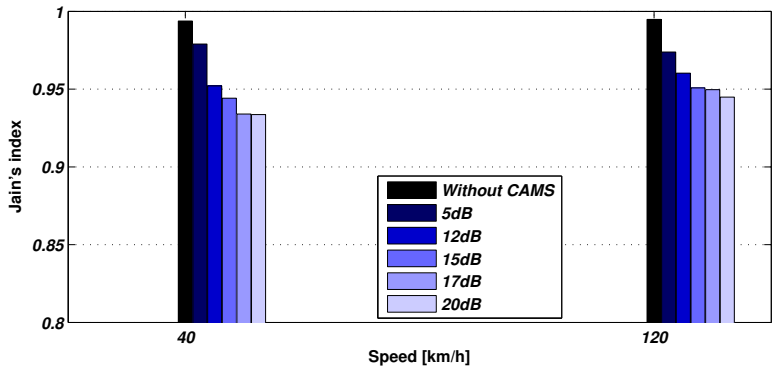


Figure 3.7: *Jain's fairness index.*

Startup delay

The startup delay is defined as the waiting time required by the player to initially prefetch x_0 frames before starting displaying the video.

In Figure 3.8, we show the variation of the startup delay with speed and SINR threshold for two different values of x_0 (10 and 30). In both figures, we remark that the startup delay decreases with the speed. This is even more significant when the threshold is high. In fact, if we observe through a same period of time, we find that users moving at high speed will visit more active zones than users moving at low speed, inducing with that shorter startup delays.

With CAMS, the startup delay is larger than with the conventional scheduler and increases proportionally to the SINR threshold. As we previously explained, the impact of varying the threshold becomes more significant with low speed than with high speed.

When we compare the two plots of Figure 3.8, we observe that the startup delay becomes larger when x_0 is higher. This is mainly noticed when low SINR thresholds are used with CAMS or when the conventional scheduler is used.

To not greatly degrade the QoE in terms of the startup delay, it is better to use CAMS with high values of x_0 and high speeds.

Probability of starvations

In Figures 3.9 and 3.10, we show the empirical Cumulative Distribution Function (CDF) of the number of starvations experienced with different SINR thresholds and different startup thresholds. Each of the four graphs below corresponds to a given x_0 and a given speed.

As a first observation, we remark that the performances are approximately the same for low and high speeds. Figure 3.9.a and Figure 3.9.b show that when users

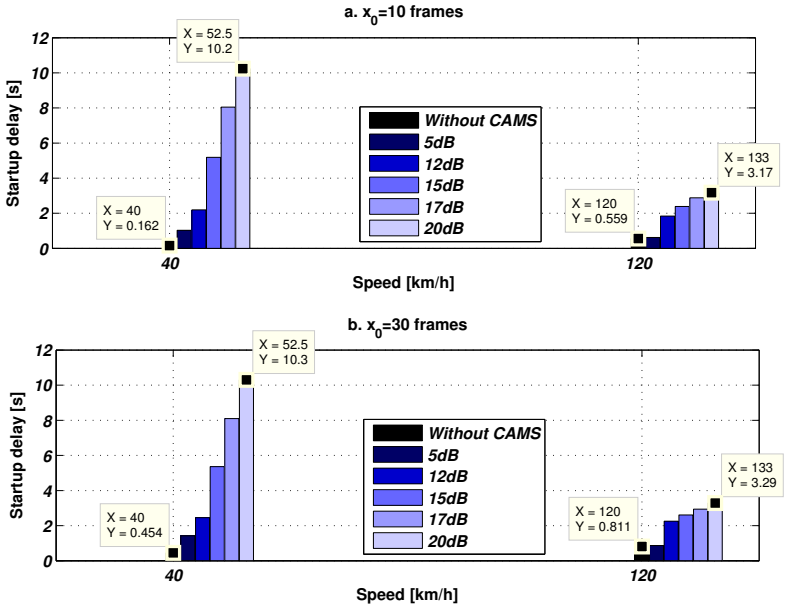


Figure 3.8: *The startup delay.*

move slower, they are more likely to experience high number of starvations. Another noteworthy observation that emerges from the results is that, when CAMS is used, the probability of having no starvation increases. In particular, we notice a gain of 87% at 40 kmph and a gain of 70% at 120 kmph with an SINR threshold of 17 dB compared to the conventional scheduler. The same interpretations hold for $x_0 = 30$ (Figure 3.10.a and Figure 3.10.b).

From the results, we also notice that the probability of having low numbers of starvations slightly increases when x_0 increases. This becomes more visible when the conventional scheduler is used.

Rebuffering delay

To proceed further with the QoE analysis, we plot the average duration of the n^{th} starvation for $n = \{1, 2, 3, 4\}$. Figure 3.11 and Figure 3.12 refer to $x_0 = 10$ frames and $x_0 = 30$ frames respectively.

Figure 3.11 shows that when users move faster, the waiting time for rebuffering becomes shorter. The same interpretation holds as for the startup delay variation. By comparing Figure 3.11 to Figure 3.12, we notice that the waiting time becomes more important when the value of x_0 increases. This is even more obvious when low SINR thresholds are used with CAMS or when the conventional scheduler is used.

To not greatly degrade the QoE in terms of rebuffering delay, we suggest to use CAMS with high values of x_0 and with high speeds.

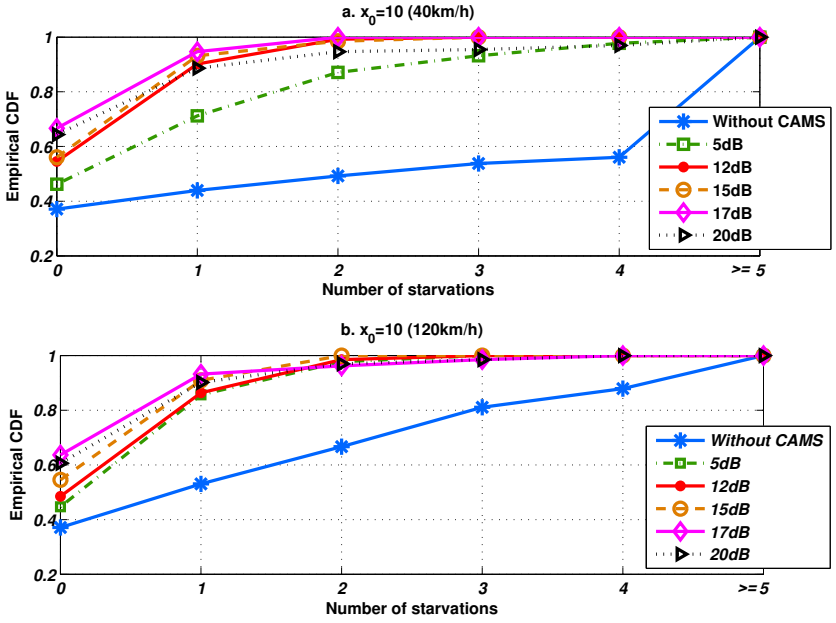


Figure 3.9: *Empirical CDF of starvation with $x_0 = 10$ frames.*

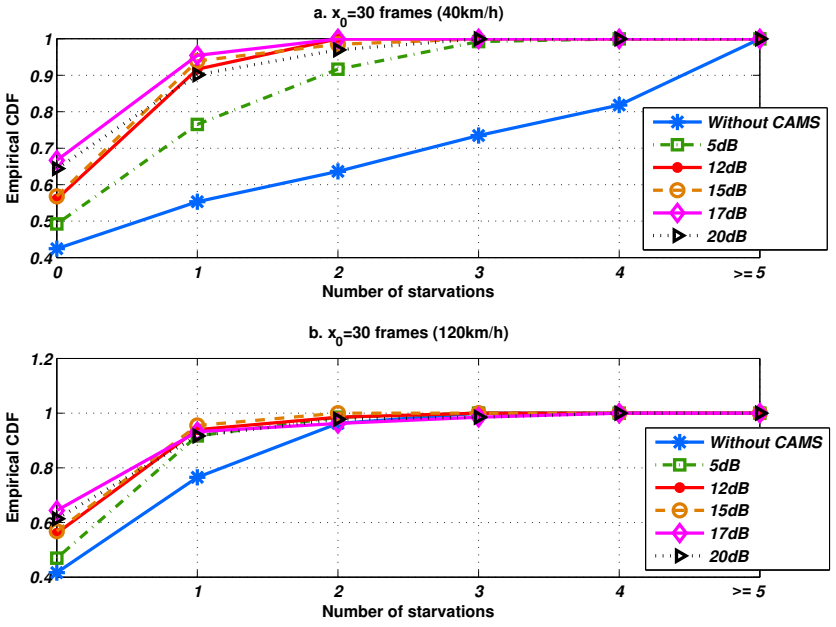


Figure 3.10: *Empirical CDF of starvation with $x_0 = 30$ frames.*

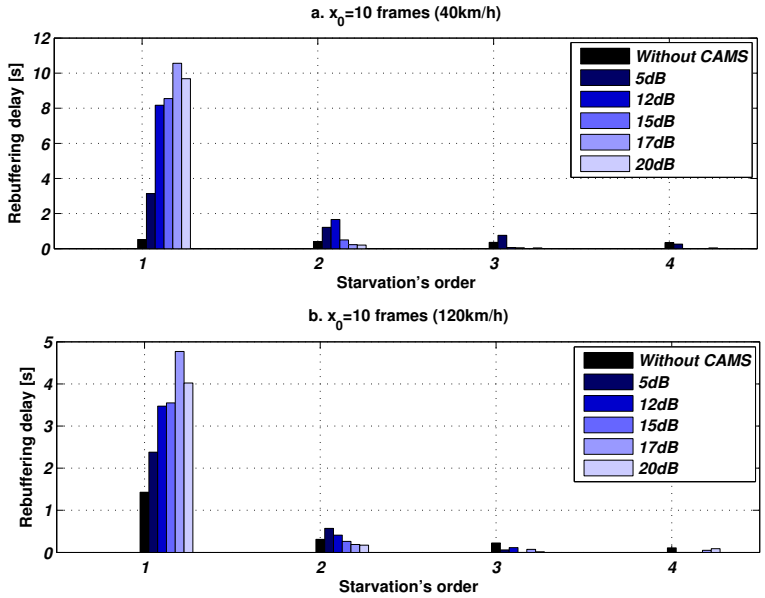


Figure 3.11: *Rebuffering delay with* $x_0 = 10$ *frames.*

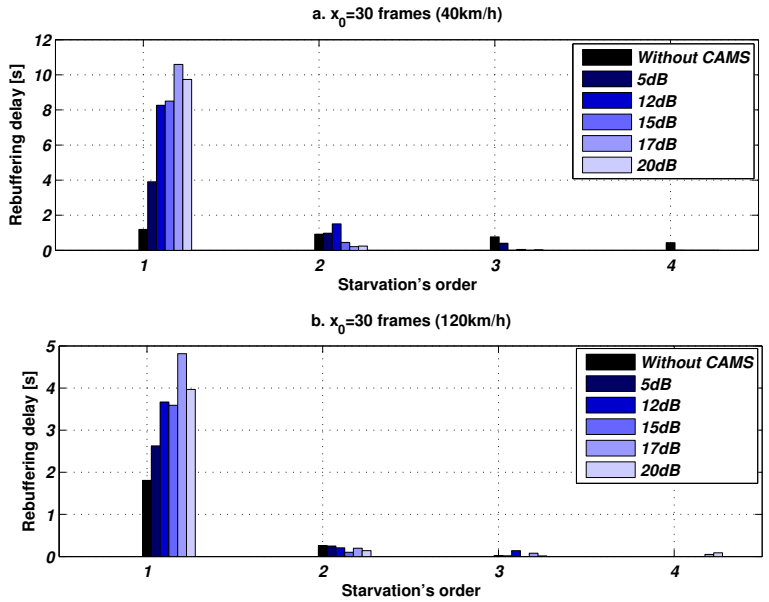


Figure 3.12: *Rebuffering delay with* $x_0 = 30$ *frames.*

3.5 Conclusion

In this Chapter, we have designed CAMS, a context-aware resource allocation mechanism for real (i.e., non adaptive) video streaming that exploits the prefetching property of the application layer and the multi-user diversity at the physical layer to improve the network performance and the users' QoE. CAMS not only enables the wireless network to achieve high spectral efficiency but also guarantees a high fairness among users. We have evaluated the QoE based on three objective metrics: the startup delay, the probability of starvations and the rebuffering delay.

Our simulation results have shown that, in order to globally improve the users' QoE, CAMS should be rather used with high vehicular speeds provided a wise setting of the SINR threshold. As it adapts the resource allocation to the users' previous contexts, we classify it within reactive approaches.

In the next Chapter, we explore an other delivery method for adaptive video streaming that depends on the user's future context. Unlike the approach of CAMS, we classify this delivery method within proactive approaches.

Chapter 4

NEWCAST: Context-Aware Delivery of Adaptive Video Streaming Under Perfect Throughput Prediction

Contents

4.1 Introduction

The knowledge of future throughput variations in mobile networks becomes more and more possible today thanks to the rich contextual information provided by mobile applications and services, and smartphone sensors. Recent studies on contextual information have revealed a promising possibility of accurately predicting the future

available resources over a medium horizon. This rises the opportunity to efficiently design video streaming by exploiting the knowledge of future capacity variations .

Although there is a rich literature on methods used for optimizing the QoE in video streaming services, very few papers have exploited the knowledge of future throughput variations [79, 80, 81]. The main idea of this Chapter is inspired from [80], where authors designed a QoE-driven optimization framework that exploits the knowledge of future throughput variations to minimize the system utilization cost while avoiding rebuffering events. The main shortcoming of their approach is that it is only suited for real (i.e., non adaptive) video streaming as it ignores important visual quality metrics related to adaptive streaming.

In this Chapter, we assume a perfect knowledge of the user's future throughput variations and provide a general optimization framework for adaptive video streaming that accounts for the user's QoE and the operator's preferences. Under the constraint of no rebuffering events, we formally obtain an optimal solution where the transmission schedule is of a threshold type and the bitrate distribution is of an ascending order. We propose an efficient heuristic, which we call NEWCAST, that performs close to the optimal approach. We evaluate the performance of NEWCAST through simulations using Matlab. Under the constraint of no rebuffering events, we study the characteristics of NEWCAST in terms of robustness (using real traces) and complexity, then we compare it to baseline adaptive bitrate algorithms.

We organize the Chapter as follows: In Section 4.2, we introduce the system model and formulate the optimization problem. In Section 4.3, we discuss the properties of the optimal solution. Then in Section 4.4, we propose optimal approaches and heuristic algorithms for the problem resolution. Section 4.5 is dedicated for both simulations and numerical results, and Section 4.6 concludes the Chapter.

4.2 Problem formulation

We consider a video file stored in a video streaming server and divided into N segments of equal length in second. Each segment is composed of S frames and encoded at L different bitrates $\{b_1, \cdots, b_L\}$, such that $b_i < b_j$ for $i < j$. To stream the video, the client requests the segments to the server one by one and indicates at each request the video quality (bitrate) needed for the streaming. Denote by $b(t)$ the video bitrate being streamed at time t, and by $\gamma(t)$ the quotient $\frac{b_L}{b(t)}$ where b_L is the highest video bitrate. We assume that, at the client side, the video frames are played at a rate of λ frames per second (fps), and that, before starting the video, a prefetching stage is introduced till having Q_0 frames in the playback buffer. To avoid buffer overflows, we assume that the playback buffer is very large.

In our problem modelling, we exploit the knowledge of the user's future available throughput (hereinafter called network capacity) to optimize the system usage cost and the QoE. Let $c(t)$ be the network future capacity at time t and $r(t)$ be the transmission bit-rate of the user at that time, note that $0 \leq r(t) \leq c(t)$. Inspired from [80], we define the system utilization cost as

$$\sigma = \frac{1}{T} \int_0^T \frac{r(t)}{c(t)} dt, \tag{4.1}$$

where $\frac{r(t)}{c(t)}$ is the proportion of resources that will be allocated to the user at time t (can be interpreted as the proportion of time the user will be occupying the network if we discretize the time), and T defines the video length in second.

We compute the number of frames that will be streamed with quality level j during the streaming session as

$$\int_0^T \frac{\delta_{\{b(t)=b_j\}} r(t)\lambda}{b(t)} dt = \int_0^T \frac{\gamma_j(t) r(t)\lambda}{b_L} dt, \tag{4.2}$$

where

$$\gamma_j(t) = \begin{cases} \gamma(t) & \text{if } b(t) = b_j \quad j \in [1 \dots L] \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Assume that the user's perception on the video quality levels can be expressed by the mean of weights $\{w_1, \cdots, w_L\}$ such that w_j corresponds to quality level j and $w_i < w_j$ for $i < j$. Hence, we define the weighted average quality of the video as

$$\rho = \frac{\sum_{j=1}^{j=L} w_j \int_0^T \gamma_j(t) r(t) \lambda dt}{b_L \times (N \times S)} = \frac{\sum_{j=1}^{j=L} w_j \int_0^T \gamma_j(t) r(t) dt}{S_L}, \tag{4.4}$$

where S_L represents the video total size in bits when it is encoded with the highest bitrate level b_L, i.e.,

$$S_L = \frac{b_L \times N \times S}{\lambda}.$$

Normally, a high video quality comes at a high cost, and a reduced system usage returns a low quality. However, it may happen that the user's preferences in terms of the QoE and the operator's preferences in terms of the system usage mismatch. To cover such situations, we define a positive balancing parameter π to make the tradeoff between system utilization cost and video quality. Therefore, we define our optimization cost function as

$$\mathcal{F} = \sigma - \pi \times \rho.$$

Let $u(t)$ be the *cumulative* number of arrival frames at time t and $l(t)$ be the *cumulative* number of frames being already played at that time. Therefore, we define the buffer underflow constraint as $u(t) \geq l(t)\ \forall t \leq T$. Given the transmission bit-rate $r(t)$ and the corresponding video bitrate $b(t)$, we express the *network frame rate* as $\frac{\lambda\, r(t)}{b(t)}$.

Denote by (r, γ) the video transmission strategy during the streaming session, where r defines the transmission schedule and γ characterizes the distribution of video bitrates. We start with the case where no rebuffering events will happen during the streaming session.

Hence, we summarize our optimization problem, as follows

$$\min_{(r,\gamma)} \mathcal{F}(r,\gamma) = \frac{1}{T}\int_0^T \frac{r(t)}{c(t)}dt - \pi \times \frac{\sum_{j=1}^{j=L} w_j \int_0^T \gamma_j(t) r(t) dt}{S_L} \tag{4.5}$$

$$s.t \begin{cases} \int_0^t \frac{\lambda\, c(t)\gamma_1}{b_L} \geq l(t) & \forall t \leq T \\ \int_0^t \sum_{j=1}^{j=L} \frac{\lambda\, r(t)\gamma_j(t)}{b_L} \geq l(t) & \forall t \leq T \\ \int_0^T \sum_{j=1}^{j=L} \frac{\lambda\, r(t)\gamma_j(t)}{b_L} = l(T), \end{cases}$$

where the first constraint ensures the existence of at least one solution which is a mono-quality streaming using the lowest video bitrate and the whole resources. At the end of Section 4.5, we study the case where only one rebuffering event is tolerated during the streaming session.

4.3 Properties of optimal solution under no rebuffering events

4.3.1 The threshold scheme for transmission schedule

Definition 1. *Giving the network capacity c, we define the threshold transmission schedule by*

$$r_{th}(t) = \begin{cases} c(t) & \text{if } c(t) \geq \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

Proposition 1. *Assume that there exists a feasible solution that satisfies the constraints in (4.5), then there exists an optimal strategy $(r_{th}, \gamma_{r_{th}})$ of optimization problem (4.5), where r_{th} is a threshold transmission schedule.*

This property has been actually inspired from [80]. However, authors in [80] assumed a real (i.e., non adaptive) video streaming, whereas we assume adaptive video streaming with multiple encoding rates. This makes our optimization problem different and the threshold proof too (see 8.2).

In practice, the setting of the transmission threshold α doesn't follow the data shifting process of the proof, since if not, it becomes complicated to generate such a threshold scheme. In Section 4.4, we design a rapid approach to build a threshold strategy for the transmission schedule.

4.3.2 Ascending bitrate level strategy

In this Section we study the properties of the bitrate level strategy under a threshold based transmission schedule. Precisely, we analyze the impact of the video quality levels' order on the setting of α.

Definition 2. *We say a bitrate level strategy is* **ascending** *if the quality levels of the video segments increases during the session, i.e., for all* $0 \leq t \leq t' \leq T$

$$b(t) \leq b(t') \text{ i.e., } \gamma(t) \geq \gamma(t').$$

Proposition 2. *Assume that there exists a threshold-based solution* (r_{th}, γ) *that satisfies the constraints in (4.5), then there exists a threshold-based ascending bitrate level solution* (r'_{th}, γ') *that optimizes problem (4.5).*

We put the details of the proof in 8.2.

4.4 Algorithmic approaches

In this Section we solve optimization problem (4.5) through algorithmic approaches based on the properties discussed in the previous Section. We give the optimal approach then we propose our heuristic under the assumption of no rebuffering events during the session. Afterwards, we extend the study to the case where the network capacity is insufficient to avoid video stalls with the lowest video quality.

4.4.1 Approaches under no rebuffering events

4.4.1.1 Approach for an optimal solution

Global algorithm: We summarize our global optimal approach in three main steps as illustrated in Figure 4.1:

1. We look for all the possible values of $\alpha \in [\alpha_{min}, \alpha_{max}]$ that satisfy the constraints in (4.5) and associate to each one the birate levels strategy that gives the highest possible weighted average quality.
2. For each couple of threshold and bitrate strategy, we compute the resulting cost function $\mathcal{F}$.
3. We pick the optimal strategy which corresponds to the minimum value of $\mathcal{F}$.

Computing the thresholds: To find the optimal thresholds α with the lowest complexity, we propose to sort the future capacity in an ascending way, then try its ascendent values as thresholds till reaching the one that causes video stalls. This approach will determine all the possible thresholds $[\alpha_{min}, \alpha_{max}]$. Figure 4.6 illustrates the example used in the simulation Section.

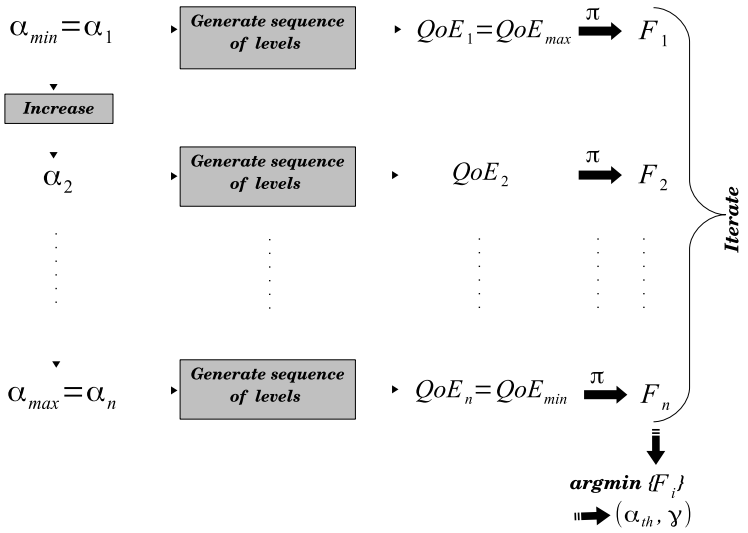


Figure 4.1: *Global algorithm for an optimal threshold-based solution with ascending bitrates.*

Generating the bitrates: Our approach to generate an optimal ascending bitrate level strategy consists of using a tree of choice of N levels as depicted in Figure 4.2, where each level corresponds to a video segment. The nodes of a tree level i correspond to all possible quality levels that can be assigned to segment i. The parent of a node (if it exists) has either a worse or equal quality. The children (if they exist) have either a better or equal quality. We construct the tree level by level to form the path that gives the optimal sequence of bitrates. At each level, we compute all the possible values that may take the weighted average quality, then we remove the nodes whose paths will cause a constraint violation. The optimal sequence of bitrates corresponds to the path that maximizes the weighted average quality at the bottom of the tree. The complexity of this algorithm may reach up to $\mathcal{O}((L+1)^N)$, which makes it unsuited for real world streaming.

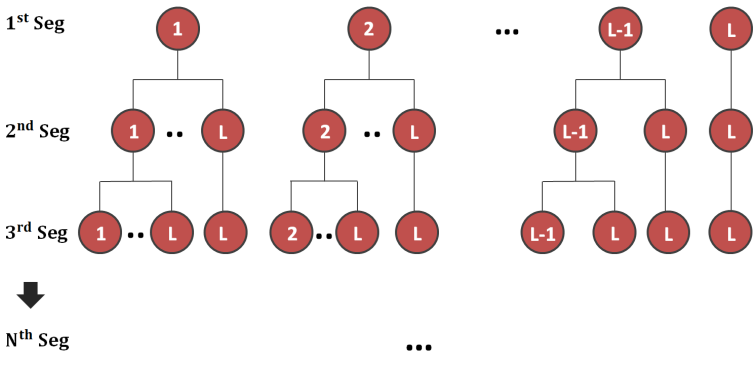


Figure 4.2: *Tree of choice for optimal ascending bitrate.*

4.4.1.2 Heuristic for a sub-optimal solution

Global algorithm: aNticipating qoE With threshold sCheme And aScending biTrate levels (NEWCAST): Our heuristic follows the same principle as the optimal global approach, but it uses two heuristics INcrease with VariablE foot STep (INVEST) and Anticipating qoe With Ascending bitRate lEvels (AWARE) for respectively computing the thresholds and generating the sequence of bitrates. Let γ_α and $\mathcal{F}_\alpha$ be the ascending bitrate level strategy and the cost function under r_α-based transmission schedule. The main steps of this heuristic are described in Algorithm 1.

Computing the thresholds: INcrease with VariablE foot STep (INVEST): This heuristic also follows the same principle as the optimal approach. However, instead of trying all the sorted capacity values as thresholds till violating the contraints, it defines a variable foot step to increase the threshold initially set to c_{min}. The values taken by this foot step will depend on the dynamic of the network capacity; Let $\{\alpha_1, \cdots, \alpha_M\} \subset [\alpha_{min}, \alpha_{max}]$ such that $\alpha_{i+1} > \alpha_i$. To compute α_{i+1} knowing α_i, we set the number of bits that we want to abandon through increasing the threshold (denote it by Q), then we select the capacity value (threshold) that allows doing that as described in Figure 4.3. $\alpha_{i+1} - \alpha_i$ will define the i^{th} foot step. (See Algorithm 2).

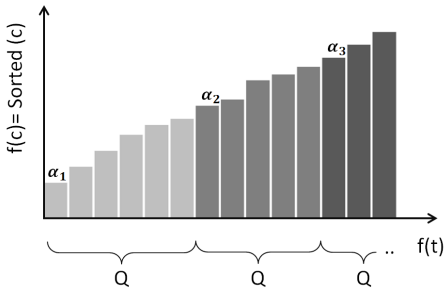


Figure 4.3: *INVEST: INcrease with VariablE foot STep.*

Generating the bitrates: Anticipating qoe With Ascending bitRate lEvels (AWARE): This heuristic has a polynomial complexity and is quite faster than the optimal approach. Our simulation results show that its outcoming solution approaches the optimal solution at almost 98% in terms of the video average quality. We summarize its steps in the few following points:
At the beginning, we assign the lowest bitrate to all video segments. Then, starting from the end of the video (latest segment) back to the beginning, we increase the bitrate of each segment by one level as long as the stall constraints are satisfied. We repeat this step many times till reaching the highest available bitrate (See Figure 4.4). By following this approach, the number of times the bitrate will be increased is at most equal to $L - 1$ (see Algorithm 3). To reduce the startup delay, which is a prominent key QoE factor (but not included in our optimization problem), we set the startup-segments to the lowest bitrate and stream them using a greedy[1] transmission rather than a threshold-based transmission. As shown in Figure 4.5, an inherent advantage of this algorithm is that it ensures a progressive increase of the bitrate instead of an

[1] A greedy transmission uses all the available network capacities.

aggressive increase as given by the optimal approach, which is quite more appreciated by the users.

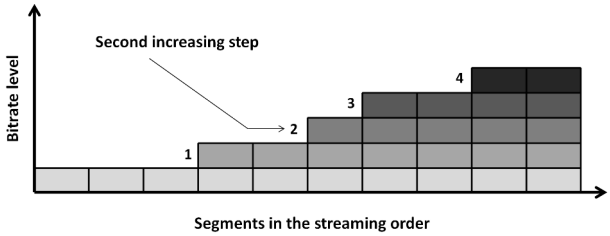


Figure 4.4: *Sketch of proof of the ascending strategy.*

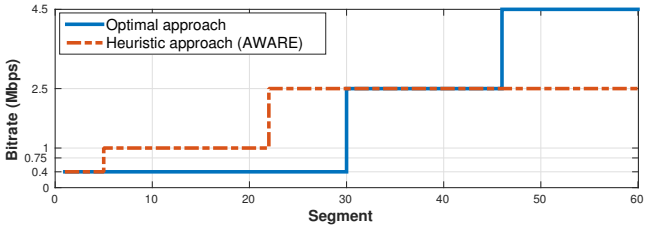


Figure 4.5: *Comparative example between optimal approach and AWARE.*

Algorithm 1: NEWSCAST: aNticipating qoE With threshold sCheme And aScending biTrate levels

Data: c, VideoProperties, L, w, Q;
α=c_{min}; i=1;
[PossibleTransmission, r_α, γ_α]=**AWARE**(c,α, videoProperties, L);
while *PossibleTransmission* **do**
    $\mathcal{F}_\alpha$=computeObjFunction (c,r_α,γ_α,w);
    i=i+1;
    α = **INVEST**(c,i,Q);
    [PossibleTransmission, r_α, γ_α]=**AWARE**(c,α, videoProperties, L);
end
$\mathcal{F}^*_{\alpha^*}$=min$\{\mathcal{F}_\alpha\}$;
α_{th}=α^*;
return (α_{th},$\gamma_{\alpha_{th}}$)

Algorithm 2: INVEST: INcrease with VariablE foot STep

Data: c,i,Q
SortedC=sort(c);
CumSortedC=CumulativeSum(SortedC);
ind = max(find (CumSortedC $\leq$ i $\times$ Q));
return SortedC(ind)

Algorithm 3: AWARE: Anticipating QoE With Ascending bitRate lEvels

```
Data: c,α, videoProperties, b_1 ··· b_L;
s=1; SegmentsBitrates[1:N]=b_s;
while s < L do
    s=s+1;
    Start=FirstSegmentOfBitrate(b_{s-1});
    End=N;
    middle = (End-Start) div2 +1;
    while middle ≥ 1 and End ≥ Start and middle ≤ End ) do
        init=SegmentsBitrates;
        SegmentsBitrates[middle:End]=b_s ;
        SegmentsBitrates[1:StartupSegments]=b_1 ;
        Test = ExistViolation(SegmentsBitrates,c,α,videoProperties);
        if Test then
            SegmentsBitrates[middle:End] = init[middle:End];
            middle=middle+(End-middle) div2 +1;
        else
            End=middle-1;
            middle=Start+(End-Start) div2 +1;
        end
    end
end
[r_α,γ_α]=TransmitVideo(c,α, VideoProperties, SegmentsBitrates);
Test = ExistViolation(SegmentsBitrates,c,α,VideoProperties);
return (T̄est,r_α,γ_α)
```

4.4.2 Approaches under rebuffering events

So far, we have assumed no rebuffing events during the streaming session, meaning that the future capacity has been assumed quite sufficient to allow streaming the hole video at the lowest bitrate. In extreme cases, the capacity may not be sufficient and may cause the player to have video stalls even with the lowest quality level. To go further with the analysis, we adapt our approach to a similar case where K stalls will inevitably occur during the streaming session. This is how we proceed: First, we spot the K segments at the level of which the stalls will take place. Then, we divide the video into $K + 1$ independent parts in-between the stalls. On each part, we run NEWCAST as if we had a new streaming session. By doing this, we will optimize the streaming approach, specifically at the last part of the video.

4.5 Simulations and numerical results

4.5.1 Simulation tools and setup

We perform all our simulations using Matlab server R2015b on a Dell PowerEdge T420 Intel Xeon running Ubuntu 14.04. The streaming session is configured based on some DASH and Youtube parameters [76, 82] and the network capacity is randomly generated around an average throughput value. We put all our parameter settings in Table 4.1

To the best of our knowledge, no explicit way really exists to compute the weights that can be accorded to the video bitrates. In [54], authors explored a QoE estimation model in which they assigned to each video segment a QoE metric that varies logarithmically in function of the bitrate and the motion factor. In [83], however, authors used a per quality MOS factor to reflect the user's satisfaction toward each quality level.

In this work, we assign the weights proportionally to the bitrate values as following

$$w_i = \frac{b_i}{\sum_{i=1}^{L} b_i},$$

where b_i is the i^{th} bitrate level and w_i is its corresponding weight. All the parameters are listed in Table 4.1. For accuracy, we explore the values of the threshold α using the optimal approach. Our heuristic (INVEST) will be later discussed in Section 4.5.4.

Parameters	values
Window size	3 min 10 s
Average throughput	2 Mbps
Capacity time slot	1 s
Video length	3 min
Segment length	1s
Video frame rate	30 fps
Startup threshold	4s
Video bitrates (Mbps)	[0.4 0.75 1 2.5 4.5]
Levels weights	[0.09 0.17 0.22 0.55 1]

Table 4.1: *Parameters of Matlab simulations.*

4.5.2 Framework performance

Figure 4.6 outlines the dynamic of the capacity used in the simulation Section as well as its corresponding values of the threshold α. Note that, when α exceeds its maximum value, a stall constraint will be violated.

By the sequel, we define our benchmark as the case where all the future capacity is used and the highest possible video quality is delivered, i.e., $\alpha = c_{min}$.

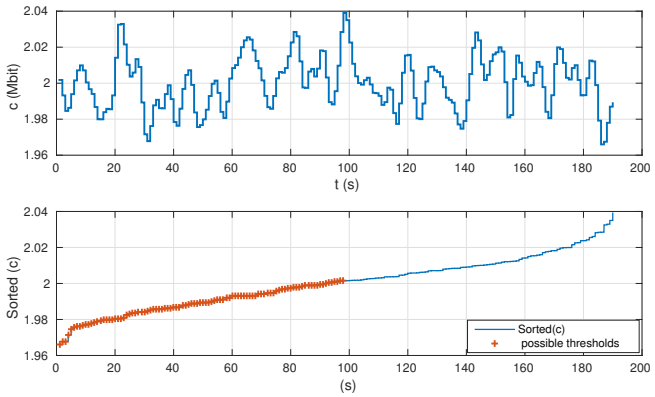


Figure 4.6: *Network capacity and threshold α.*

By executing NEWCAST under the aforementioned configuration using different values of π, we noticed that the system performance only varies for π ranging from 4.50 to 4.70. Beyond the limits of this interval, the performance remains constant. In the following analysis, we will focus on only three values of π: *low*, *medium* and *high*. Denote by α_π the threshold returned by NEWCAST after execution using the balancing parameter π.

In Figure 4.7 we show the variation of α_π in function of π; a small value of π results in a high α_π as it prioritizes the system utilization cost. A big value of π, however, results in a low threshold as it accords more importance to the average quality. Whereas a medium π leads to an in-between threshold that balances QoE and system cost.

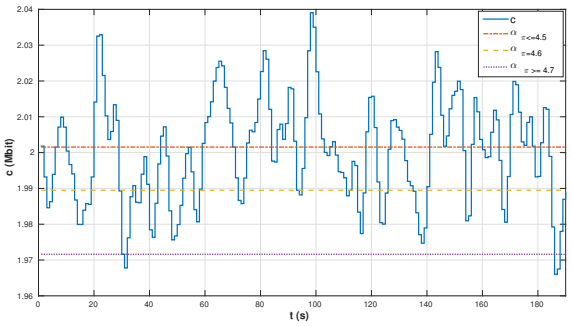


Figure 4.7: *Variation of α_π as function of π.*

In Figure 4.8 we plot the playback buffer state evolution over time and its corresponding sequence of bitrates for the three aforementioned values of π. When π is small, many silent times are noticed and the buffer state evolves with high slopes (mainly at the beginning and at the middle of the video). This is actually due to the low quality of the segments being streamed. Note that the player streams as much frames as the bitrate is low. For the medium value of π, more flexibility is noticed with shorter silent times and better quality. As for the big value of π, no silent times are noticed since almost all the network resources are used. The buffer state evolves gradually with low slopes, given the fact that segments of high-order quality are being streamed.

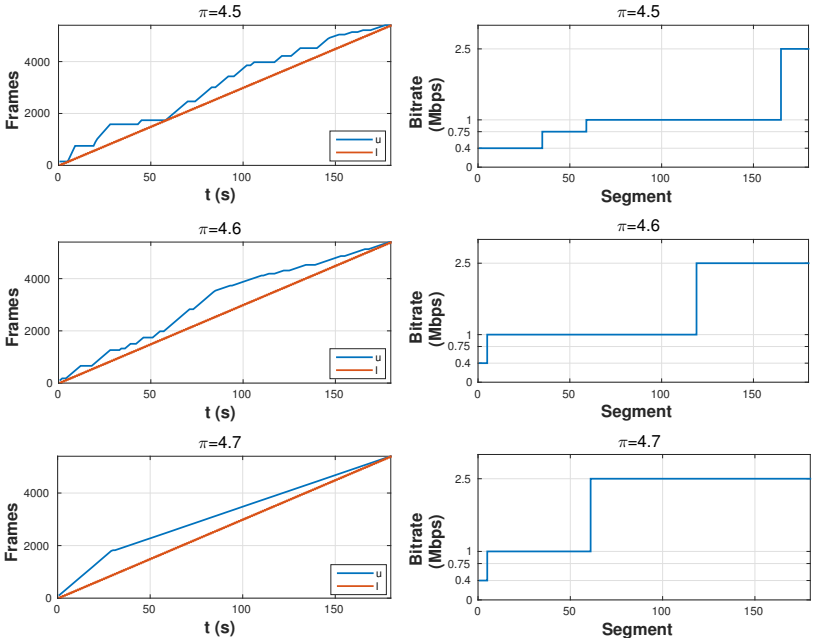


Figure 4.8: *Playback buffer state evolution and corresponding sequence of bitrates for different values of π.*

Now we explore the idea of enforcing a stall during the streaming session. Let $\mathcal{F}_{or}$ be the original cost function before enforcing a stall, and $\mathcal{F}_{st}$ be the resulting cost function after enforcing a stall.

In Figure 4.9, we plot again the playback buffer state evolution over time for the three values of π, and plot below the variation of $\mathcal{F}_{st}$ in function of the stall emplacement (1^{st} segment, 2^{nd} segment, $\cdots$). As depicted in the figure, for $\pi = 4.5$, $\mathcal{F}_{st}$ experiences high fluctuations around $\mathcal{F}_{or}$ mainly when the stalls are enforced at the beginning of the video. The lowest values of $\mathcal{F}_{st}$ are noticed when the stalls are enforced at the moments where the original buffer state is critical, i.e., *a low quality with no much flexibility toward the stall constraint*.

Note that, the critical states of the buffer at these moments were preventing NEWCAST from setting a higher threshold. When a stall is enforced there, the video is divided into two independent parts and the streaming strategy is optimized before and after the stall, leading to two different thresholds that reduce the overall system utilization cost.

Now, by increasing π, we observe a quasi-constant but lower $\mathcal{F}_{st}$; a stall enforcement certainly enhances the quality at the beginning part of the video, but it condemns the flexibility and the average quality at the rest of the video. The degradation in the global quality induces a reduction in the global system cost that outweighs the resulting $\mathcal{F}_{st}$.

To sum it up, a stall enforcement may be only interesting when the value of π is low since it may further reduce the system cost. A judicious choice of its emplacement should correspond to the moments where the original buffer state is critical.

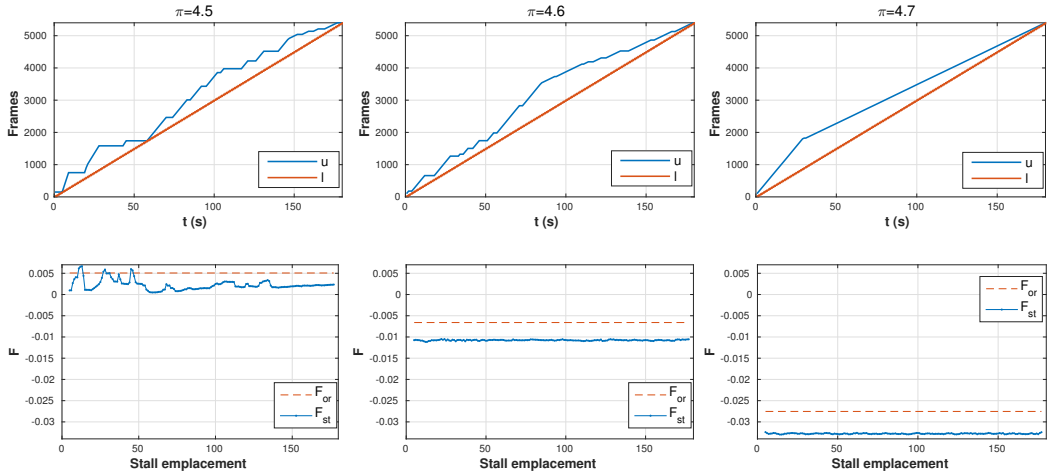


Figure 4.9: *System performance with buffer stall enforcement.*

4.5.3 Robustness under prediction errors

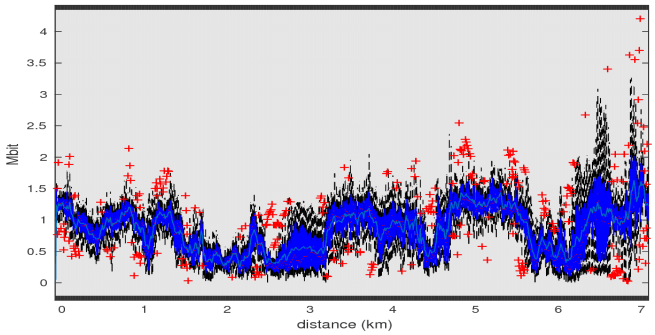


Figure 4.10: *Experimental spatial variations of the capacity for the tramway Ljabru-Jernbanetorget trajectory.*

The key limitation of our framework is that there is still no explicit approach that can accurately predict the network capacity over more than ten seconds in the future. In order to evaluate the robustness of NEWCAST, we use the real throughput traces available online in the High Speed Downlink Packet Access (HSDPA) dataset [84]. This dataset consists of 30 minutes of continuous throughput measurements of a moving device in Telenor's 3rd Generation (3G)/HSDPA wireless mobile network.

We use the traces of the Ljabru-Jernbanetorget trajectory as they present the smallest variance in the throughput spatial variation (see Figure 4.10). From this spatial variation, we compute the temporal variation of the throughput by supposing the user moving at a speed of 50 Kmph.

Under the same video configuration of Table 4.1, we compute the performance $\mathcal{P}_{av}$ of NEWCAST by using the average throughput of all throughput realizations.

Then, by using each throughput realization apart, we compute its performance $\mathcal{P}_{real}$ ($\mathcal{P}$ stands for system cost or video average quality).

We evaluate the robustness of the framework by computing the performance error rate using each throughput realization as

$$\mathcal{P}_{error} = \left| \frac{\mathcal{P}_{real} - \mathcal{P}_{av}}{\mathcal{P}_{av}} \right|.$$

In Figure 4.11, we plot the average error rate of the system cost and the average error rate of the video average quality in function of π. A noteworthy observation here is that the average error rate does not exceed 15% for both metrics. Results even show a low sensitivity of the system cost to prediction errors when π is small, and a low sensitivity of the average quality to prediction errors when π is high.

In general, we can claim that our scheme performs well under the presence of real prediction errors.

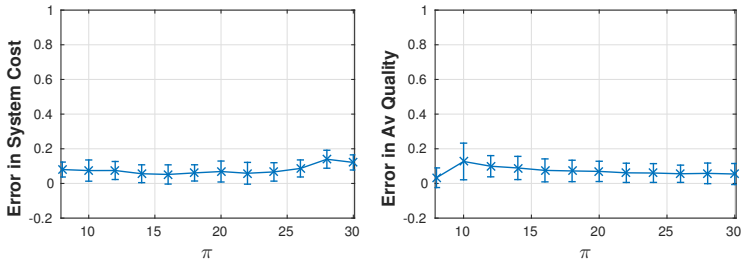


Figure 4.11: *Average error rate of the system performance under real throughput prediction errors.*

4.5.4 Complexity

Framework performance under bigger time slots

In Figure 4.12, we compute the mean execution time of NEWCAST (using optimal thresholds) by averaging results over 100 (randomly generated) capacities and using different time slots (from 1s to 5s). It takes almost 4s to compute the final strategy with a time slot equal to 1s. As expected, using bigger time slots takes much shorter time. However, this comes at the expand of the final result accuracy depending on the value of π. In the same figure, we show the system response (through $\mathcal{F}$) for each time slot by averaging results over the 100 capacities. We compute an accuracy rate factor (≤ 1) by comparing the obtained results with the results of 1s time slot.

In our model, we assume that in a time slot only one bitrate level can be streamed, which may condemn the QoE under bigger time slots. For high values of π, we notice a very slight degradation of the accuracy of $\mathcal{F}$, since the system tends to use all the network resources. However, for low values of π, we observe a high degradation of the accuracy since the system tends to use less network resources. With the constraint

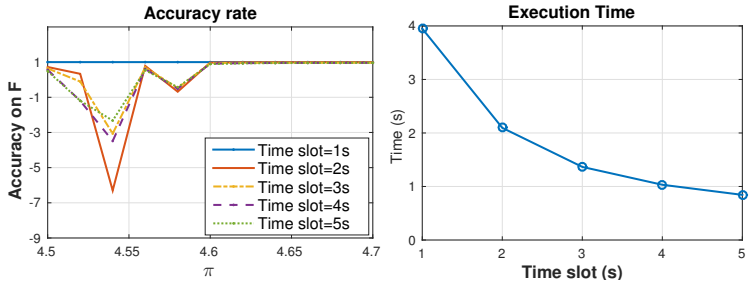


Figure 4.12: *Accuracy and complexity variations with different time slots.*

of using one bitrate per time slot, the QoE is highly degraded and, by the sequel, the system cost is highly reduced.

Framework performance under different values of Q

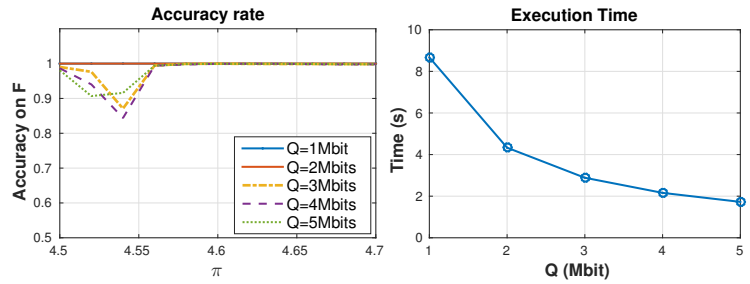


Figure 4.13: *Accuracy and complexity variations under different values of Q.*

Here, we set the time slot to 1s and run NEWCAST using different values of Q (between 1Mbit and 5Mbits) by averaging results over the same 100 capacities. Results in Figure 4.13 show that setting Q to the average throughput (2Mbps) leads to a high accuracy rate (≈ 1) with an execution time of 4s (as for optimal thresholds). Setting lower values of Q, increases the execution time and keeps almost the same accuracy on $\mathcal{F}$. For higher Q, the complexity is notably reduced, but slight degradations are noticed on the accuracy rate (less than 16%). A judicious choice of Q should then be made depending on the operator's preferences: a high Q gives a high QoE and a very low complexity, whereas, a low Q gives a low system cost and a higher complexity.

4.5.5 Comparison with baseline ABR algorithms

In this Section, we compare NEWCAST to two baseline ABR algorithms: one is TB-ABR, the other is BB-ABR. We develop each algorithm on Matlab and simulate its behaviour on different video streaming sessions. We keep all the parameter settings of Table 4.1.

TB-ABR and BB-ABR configuration: criteria of choice for comparison with NEWCAST

The main characteristic of NEWCAST is that it increases the quality of segments *progressively* to avoid bothering the user with sudden quality jumping. For this reason, we configure the TB-ABR and the BB-ABR algorithms to be both *conservative*. For TB-ABR we use *the smoothed throughput* estimation such that

$$\hat{T}(i+1) = \sum_{k=i-3}^{i} p_k T(k); \tag{4.7}$$

$$p_1 = 0.5,\ p_2 = 0.3,\ p_2 = 0.15 \text{ and } p_2 = 0.05,$$

where $T(i)$ designs the throughput measured after downloading segment i, and $\hat{T}(i+1)$ is the throughput estimate of segment $i+1$. As for the bitrate selection, we use a method close to that defined in "Microsoft's Smooth Streaming" (see Algorithm 4). For BB-ABR, we use the *most stable* method cited in [85]. We define three thresholds B_{low}, B_{min}, and B_{high} (respectively equal to 4, 8 and 12 segments), and define multiple strategies of bitrate adaptation depending on the range of the buffer level (see Algorithm 5).

Capacity samples

To be as close as possible to real world throughput variations, we generate the capacity samples by using the standard-complaint Ns3 simulator; We conduct extensive simulations of an LTE-network by varying the mobility or/and the number of users each time. All the throughput samples resulting from these simulations are used for the evaluation of both NEWCAST and the ABR algorithms. Table 4.2 summarizes the parameters used for the configuration of the simulated LTE-network.

Main comparison points

From the execution of NEWCAST and the two aforementioned ABR algorithms using all the throughput samples, we notice that in 0.3% of cases, the TB-ABR algorithm encounters video stalls (at least one stall), and that in 7.8% of cases, the BB-ABR algorithm encounters video stalls. NEWCAST, however, succeeds at achieving zero stall during all the streaming sessions. Hence, we find it more judicious to perform the comparison by distinguishing the cases where the number of stalls encountered by each ABR algorithm is also equal to zero. Our analysis is driven by the three metrics that mostly characterize NEWCAST: The system cost, the average per segment video quality, and the average number of quality switching. In Figure 4.14, we plot each of these metrics in function of π; π ranging from 0.2 to 26.

Parameters	values
Number of macro cells	1
Number of UEs per cell	25 - 30 - 35 - 40 - 45
Number of simulations	1000
eNb Tx Power	46 dBm
eNb noise figure	5 dB
UE noise figure	9 dB
Pathloss model	COST 231
MAC scheduler	Proportional fair 100 RBs
Fading model	Pedestrian
Transmission model	MIMO Transmit diversity
Mobility model	RandomWalk2dMobilityModel
Velocity of users	Uniform [5,16] m/s
EPS bearer	NGBR-VIDEO-TCP-DEFAULT
Simulation length	190 s

Table 4.2: *Ns3 simulation setting parameters.*

TB-ABR vs NEWCAST : According to Figure 4.14, the main advantage of NEWCAST is that it can achieve the same quality as TB-ABR with a system utilization cost reduced by at least 30%, and that it can achieve the same system cost with an average quality enhanced by up to 19%. This is mainly due to the smart threshold-based-strategy of NEWCAST that uses the less expensive resources depending on the value of π. It is then up to the operator to make the tradeoff and to wisely calibrate the value of π to outperform the TB-ABR algorithm. A further important observation lies in the very reduced number of quality switching achieved by NEWCAST (at most 2.5) compared to that achieved by TB-ABR (around 11).

BB-ABR vs NEWCAST : We notice from Figure 4.14 that BB-ABR is very greedy toward the resource usage compared to TB-ABR, which makes it give near performance to NEWCAST when applied with high values of π. Actually, for some values of π, NEWCAST outperforms BB-ABR, but this outperformance is marginal. In fact, the same average quality can be achieved with a system cost reduced by 12%, and the same system cost can be achieved giving an average quality increased by 4%. The greedy character of BB-ABR can be either emphasised or de-emphasised depending on the thresholds set for the playback buffer (B_{min}, B_{low} and B_{high}), so it may happen that BB-ABR uses all the resources and gives a higher average quality than NEWCAST, but this outperformance won't exceed 2% since the heuristic used by NEWCAST approximates the optimal quality arrangement by 98%. All things considered, the most worth citing advantage of NEWCAST, is that it gives a far less number of quality switching (at most 2.5 against 19 with BB-ABR), which is quite better for the users' perception.

In conclusion, when the knowledge of the future throughput is perfect, NEWCAST can perform better than the baseline TB-ABR and BB-ABR algorithms. By mean of a wise calibration of the value of π, the tradeoff between system utilization cost and QoE can be steered to either save more resources or increase the average quality. In all cases, the number of quality switching remains the most suitable for the end user's perception.

Algorithm 4: TB-ABR : Throughput-Based ABR

```
for segments of startup phase do
    set the quality to the lowest bitrate b_1
end
for segments of post-startup phase do
    estimate the throughput based on the three previous downloaded segments
    if the throughput ≤ b_1 then
        set the quality to b_1
    else
        if the throughput ≤ the previous bitrate then
            set the quality to the highest bitrate below the throughput
        else
            if the next higher bitrate ≤ the throughput then
                increase the bitrate by one level
            else
                Keep the same quality
            end
        end
    end
end
```

Algorithm 5: BB-ABR : Buffer-Based ABR

```
for segments of startup phase do
    set the quality to the lowest bitrate b_1
end
for segments of post-startup phase do
    if 0 ≤ BufferState ≤ B_min then
        set the quality to the lowest bitrate b_1
    end
    if B_min < BufferState ≤ B_low then
        if the BufferState is increasing then
            keep the same quality
        else
            decrease the bitrate by one level if possible, or keep it the same
        end
    end
    if B_low < BufferState ≤ B_high then
        keep the same quality
    end
    if B_high > BufferState then
        increase the bitrate by one level if possible or keep it the same
    end
end
```

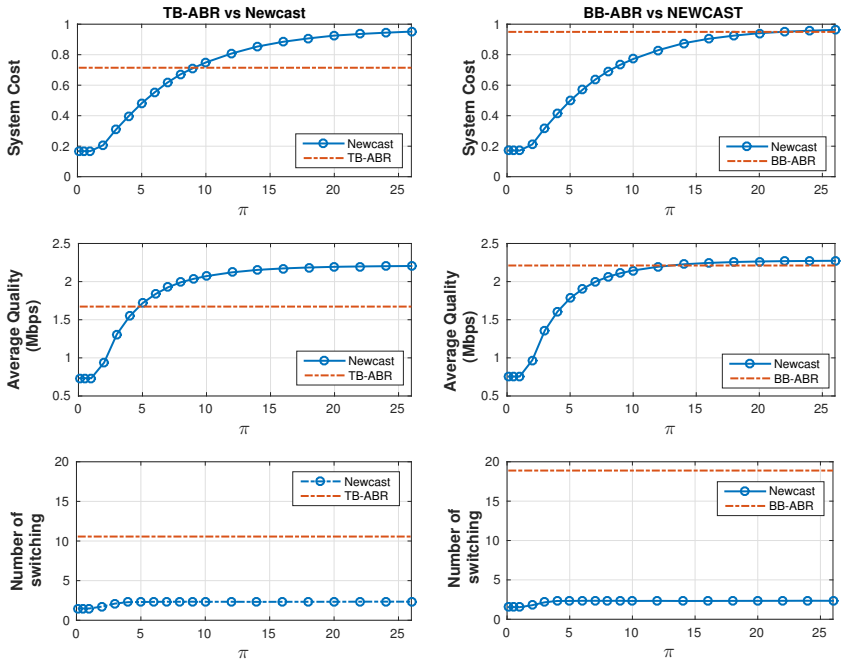


Figure 4.14: *TB-ABR vs NEWCAST and BB-ABR vs NEWCAST (without stalls).*

4.6 Conclusion

In this Chapter, we have developed a new framework, which we called NEWCAST, to optimize the delivery of video streaming content under the knowledge of the user's future throughput variations. We have designed the framework to make a tradeoff between system utilization cost and video quality by leveraging on some key QoE metrics such as the average per segment video quality and the rebuffering events (video stalls).

Our numerical results have revealed the possibility of using NEWCAST as an online algorithm for DASH delivery. By comparison with baseline adaptive algorithms, we have found that NEWCAST outperforms the basic streaming approaches in terms of the tradeoff made between system cost and QoE. It even shows a lower complexity since it adapts the quality of all video segments at once at the beginning of the streaming session.

In the next Chapter, we extend NEWCAST to other versions in order to make it more robust to eventual throughput prediction errors.

Chapter 5

NEWCAST Derivative Versions for Context-Aware Adaptive Video Delivery Under Unperfect Throughput Prediction

Contents

5.1 Introduction

In the previous Chapter, we proposed a framework for adaptive video streaming delivery to strike a balance between network utilization cost and user's QoE. In particular, we designed NEWCAST, a proactive video content delivery algorithm, that adjusts the video quality over a long-term future horizon, assuming a perfect throughput prediction.

However, from a practical point of view, perfect throughput prediction is not always possible [86], which presents a key limitation to NEWCAST. In this Chapter, we

take a step further in the analysis of NEWCAST performance under throughput prediction errors. We propose four algorithms STERN, A-STERN, RECAST and STREET as derivative versions from NEWCAST to mitigate the problem of inaccurate throughput prediction.

The first two algorithms STERN and A-STERN are a direct application of NEWCAST over successive short-term future horizons and aim at reducing the number of video stalls. Whereas the two other algorithms RECAST and STREET aim at reducing the number of quality-switching. Our numerical results lead us to believe that these algorithms can be efficient and robust in realistic environments even if the prediction of the capacity variation is not accurate.

We organize the Chapter as follows: In Section 5.2, we highlight the shortcoming of NEWCAST under throughput prediction errors. In Section 5.3 we present STERN and A-STERN and evaluate their performances. We compare STERN to NEWCAST and A-STERN to STERN. Then in Section 5.4, we present RECAST and compare its performance to A-STERN. Finally in Section 5.5, we present STREET and compare it to RECAST. A recapitulation of the five above algorithms is afterwards reviewed in Section 5.6. Section 5.7 concludes the Chapter.

5.2 NEWCAST under unperfect prediction over a long future horizon

5.2.1 The usage of NEWCAST in real environments

In real environments, NEWCAST should be implemented at the client side as an independent framework. It should be able to communicate the threshold α^* to the network scheduler and the set of video bitrates γ^* to the media player as described in Figure 5.1. We can imagine for sending α^* a kind of a cross layer that also allows to apply the threshold-based transmission scheme. The set of video bitrates γ^*, however, can be directly sent to the player just at the beginning of the streaming session. These bitrates will then be consecutively requested by the player to the streaming server. Note that, in our analytical model, the variable $\gamma*$ is set to describe the variation of the video bitrate *in function of time*. In real implementation, the player will not use it that way, it will rather use the bitrate variation *in function of the segments' orders*, which can be directly returned by NEWCAST.

5.2.2 Prediction error model

As claimed in [87], a perfect prediction of the capacity may not be feasible over a large horizon window. However, it is plausible that the prediction becomes accurate over a short horizon window. In the literature, we find that prediction accuracy depends on three major factors: (i) the accuracy on the user's mobility model, (ii) the space mapping of the users' average throughput, and (iii) the variation of the real throughput around the user's space-mapped average throughput [86].

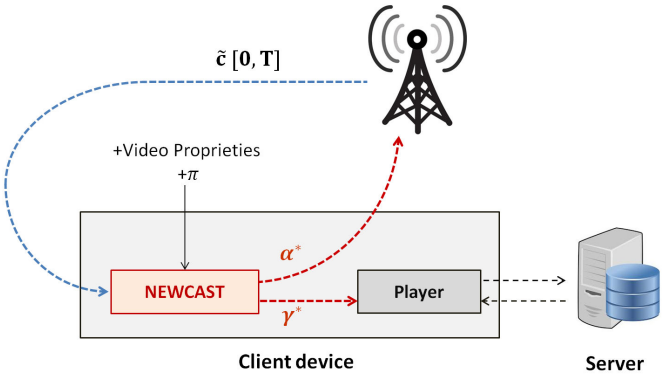


Figure 5.1: *Illustration of NEWCAST's interactions with the network scheduler and the media player.*

Accordingly, in this Chapter, we assume that the prediction error increases as long as we move forward in time. Thus, we model the real capacity c_{real} as a Gaussian white noise with, as mean, the predicted capacity $\tilde{c}$, namely

$$\forall\ t\ in\ [0,T],\ c_{real}(t) = \mathcal{N}\left(\tilde{c}(t), \mathcal{SD}_{err}(t)\right),$$

where $\mathcal{SD}_{err}(t) = \mathcal{SD}_{err} \cdot \log(t)$ is the error standard deviation at time t.

5.2.3 NEWCAST robustness to video stalls

As in the previous Chapter, we evaluate the robustness of NEWCAST through simulations using Matlab and the parameter settings of Table 4.1. We maintain the same predicted capacity $\tilde{c}$ and generate around many samples of possible real throughput variations c_{real} according to our adopted prediction error model. We use for that different values of $\mathcal{SD}_{err}$.

We run NEWCAST under the knowledge of $\tilde{c}$ and set the future streaming strategy (α^*,γ^*). Then, using each real capacity c_{real}, we apply this strategy and compute the real system utilization cost and the number of stalls during the streaming session.

In our simulations, we set π to a small value ($\pi = 3$) to prioritize the system cost and make the system more sensitive to prediction errors. As depicted in Figure 5.2, we evaluate the robustness of NEWCAST by representing the distribution of video stalls during the streaming session in function of $\mathcal{SD}_{err}$.

First, we observe that the probability of having zero stalls is relatively low and decreases as the error on the predicted capacity increases: From 0.49 with $\mathcal{SD}_{err}$=10^{-4} to 0.17 with $\mathcal{SD}_{err}$ =10^{-2}. Second, we notice that the average number of video stalls increases: From 1.53 to 1.89 with the same values of $\mathcal{SD}_{err}$. Thus, when the prediction of the capacity is imperfect over a long future horizon, the approach of NEWCAST fails to guarantee a good QoE. This leads us to consider short future horizons in order to obtain more accurate throughput prediction.

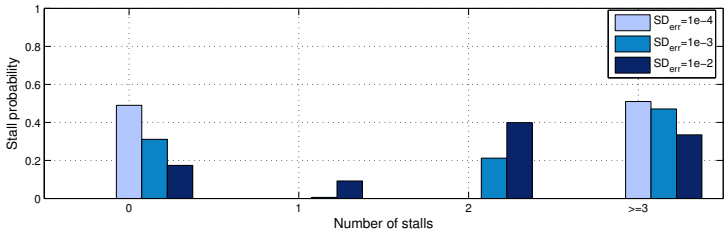


Figure 5.2: *NEWCAST: Distribution of video stalls as function of $\mathcal{SD}_{err}$.*

5.3 NEWCAST short-term versions for better stall avoidance

Both algorithms we propose in this section leverage on NEWCAST strategy over long and short future horizons.

5.3.1 Short-TERm Newcast (STERN)

Algorithm

STERN is a modified version of NEWCAST that considers short-term throughput prediction over *successive* short future horizons. It takes as input the evolution of the buffer occupancy initially computed by NEWCAST over the long future horizon (using the long-term predicted capacity $\tilde{c}$) in order to adjust the number of segments to stream at each short horizon. Let $\tilde{u}(t)$ be the number of cumulative received frames computed by NEWCAST for future time t, $t \in [0, T]$, and u_{real} be the real cumulative received frames function. At the beginning of each short horizon, STERN computes through the variation of $\tilde{u}$ the number of segments to stream over that horizon to make the playback buffer evolve as was initially scheduled by NEWCAST, i.e., to make u_{real} follow $\tilde{u}$. To reduce the number of video quality-switching when moving from one short horizon to another, STERN ignores the startup phase mode of NEWCAST that enforces the first segments to be streamed at the lowest video bitrate under a greedy transmission mode.

Let $\mathcal{H}_i$, $i > 0$, be the i^{th} short future horizon, $\tilde{c}_{\mathcal{H}_i}$ the short-term predicted capacity over that horizon, and $X_{\mathcal{H}_i}$ the number of segments to stream over it. Figure 5.3 depicts the interactions of STERN with NEWCAST, the network scheduler and the media player. Algorithm 6 and Figure 5.4 describe its main steps.

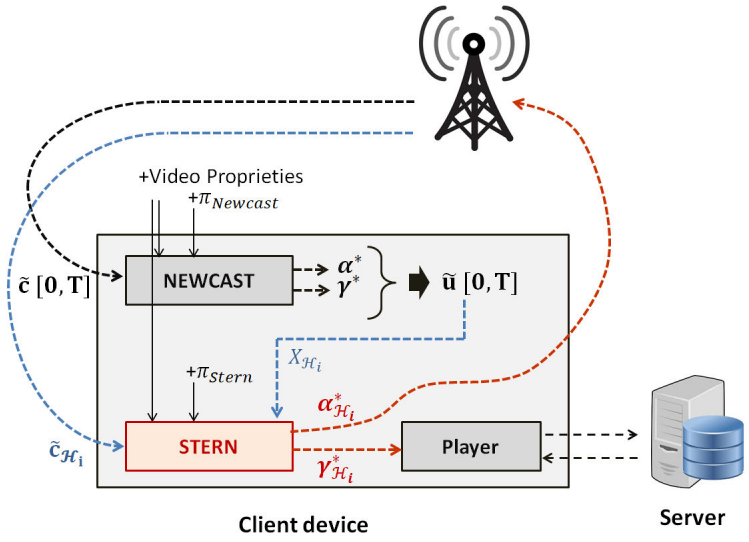


Figure 5.3: *Illustration of STERN interactions with NEWCAST, the network scheduler and the media player.*

Algorithm 6: short-term Newcast (STERN)

Input: $\mathcal{H}_1 = [t_0, t_1], \tilde{u}, X_{\mathcal{H}_1} = \frac{\tilde{u}(t_1)}{SegmentSize}, i = 1;$

while *Still segments to stream* **do**

  Predict $\tilde{c}_{\mathcal{H}_i}$*;*

  Check if it is possible to stream $X_{\mathcal{H}_i}$ *segments using* $\tilde{c}_{\mathcal{H}_i}$*, otherwise, reduce* $X_{\mathcal{H}_i}$*;*

  $[\alpha^*_{\mathcal{H}_i}, \gamma^*_{\mathcal{H}_i}]$=***NEWCAST***$(\tilde{c}_{\mathcal{H}_i}, X_{\mathcal{H}_i}, \pi_{STERN}$, *BufferState*);

  $QualityVect_{\mathcal{H}_i}$ = *DeduceQualityVector*$(\gamma^*_{\mathcal{H}_i})$;

  $[u_{real}, t_{fin}]$=*StreamVideo*$(c_{real}[t_{i-1} : end], \alpha^*_{\mathcal{H}_i}, QualityVect_{\mathcal{H}_i})$

  $t_i = t_{fin} + 1;$

  $t_{i+1} = t_i + length(\mathcal{H}_i);$

  $\mathcal{H}_{i+1} = [t_i, t_{i+1}];$

  $X_{\mathcal{H}_{i+1}} = \frac{\tilde{u}(t_{i+1}) - u_{real}(t_{fin})}{SegmentSize};$

  $i = i + 1;$

end

return $(\{\alpha^*_{\mathcal{H}_i}\}, \{\gamma^*_{\mathcal{H}_i}\})$

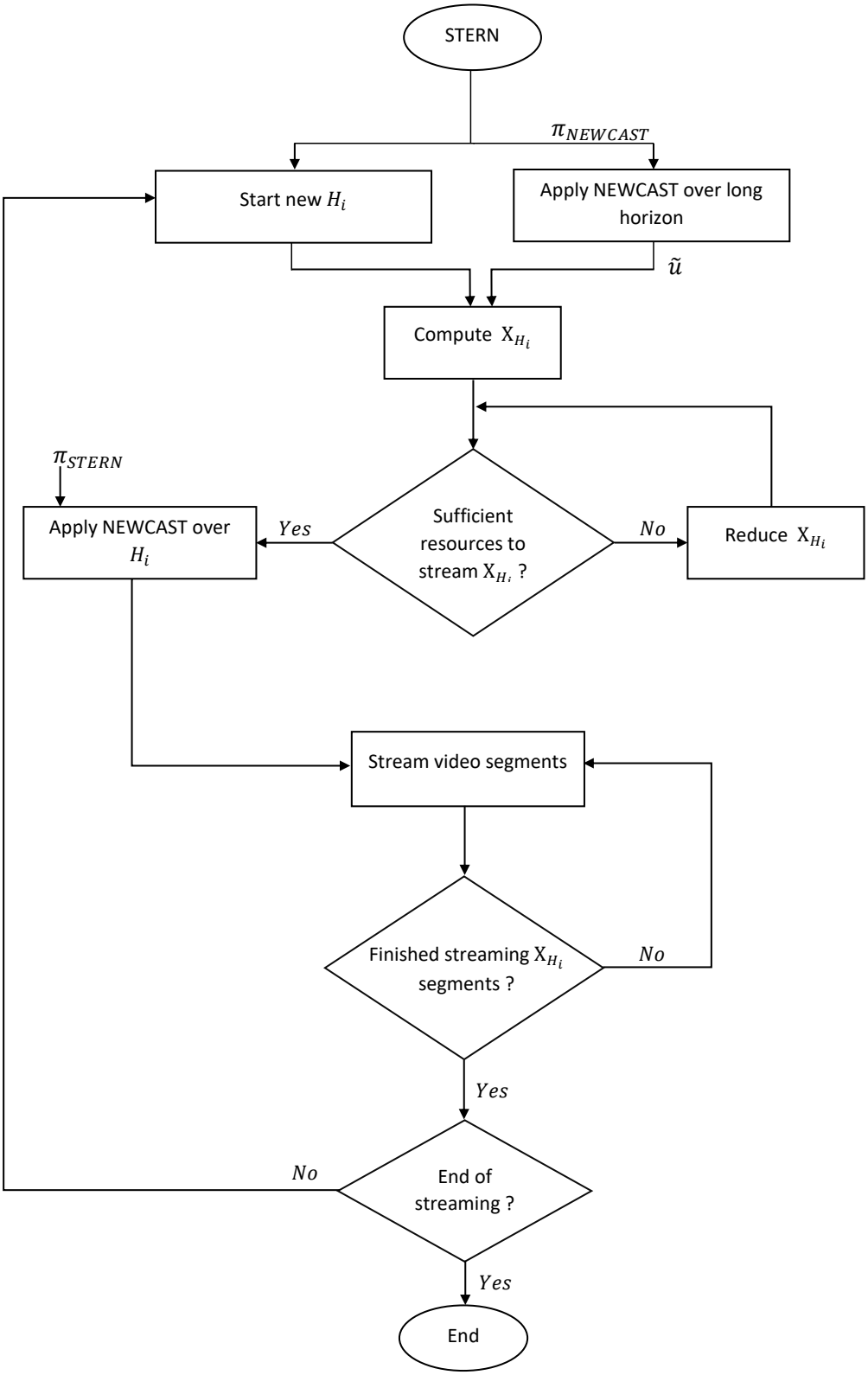


Figure 5.4: *Flowchart of STERN algorithm.*

Performance

In our simulations, we set π_{STERN} to be one time equal to $\pi_{NEWCAST}$ ($\pi_{STERN} = 3$) and one time greater than $\pi_{NEWCAST}$ ($\pi_{STERN} = 6$).

In Figure 5.5 we plot the distribution of video stalls in function of $\mathcal{SD}_{err}$ using NEWCAST then using STERN with the tow values of π_{STERN}. As a general observation, STERN succeeds at making the system more robust to video stalls compared to NEWCAST. Indeed, for $\pi_{STERN} = 3$, the probability of having zero stalls increased from 0.17 (using NEWCAST) to 0.24, and for $\pi_{STERN} = 6$ it increased to 0.95 at the highest prediction error. Which induced a reduction in the average number of video stalls: From 1.90 (using NEWCAST) to 1.02 and 0.05 for $\pi_{STERN} = 3$ and $\pi_{STERN} = 6$ respectively.

In Figure 5.6, we plot the average number of video quality-switching during the streaming session using NEWCAST then using STERN for the same values of $\mathcal{SD}_{err}$. By comparison with NEWCAST, the number of quality-switching induced by STERN is notably high: We count 18 and 20 switching for $\pi_{STERN} = 3$ and $\pi_{STERN} = 6$ respectively, versus only 3 switching with NEWCAST. For the sake of illustration, we put in Figure 5.7 two snapshots of the video quality variation using STERN under a randomly generated real throughput c_{real}.

As for the system performance, we notice from Figure 5.8 that STERN is a bit greedier than NEWCAST; at the highest prediction error, it increased the system cost from 32% (using NEWCAST) to 46% and 66% for $\pi_{STERN} = 3$ and $\pi_{STERN} = 6$ respectively. Which induced a rise in the average video quality.

All things considered, we can claim that STERN succeeds at ameliorating the overall video quality mainly by reducing the average number of video stalls compared to NEWCAST, which was our first motivation. Nevertheless, it produces higher numbers of quality-switching and higher system utilization costs.

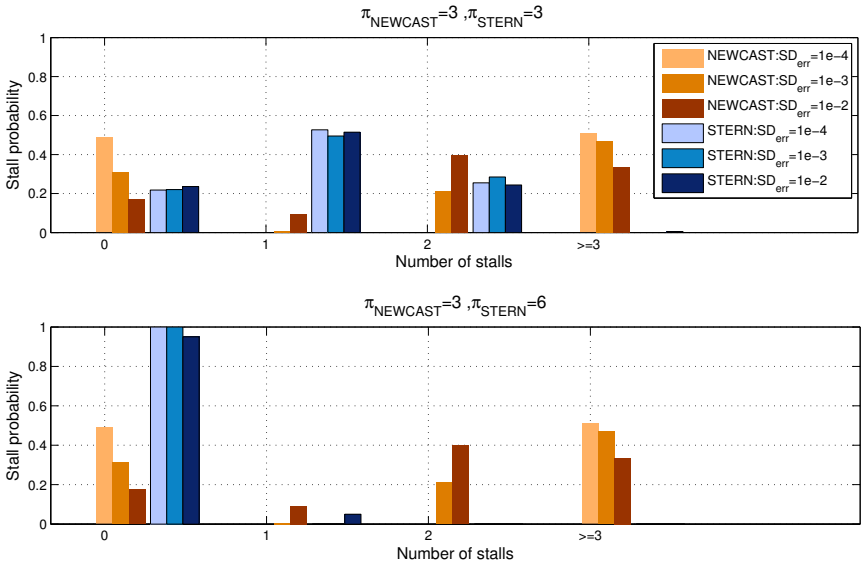


Figure 5.5: *STERN vs. NEWCAST: Probability of having stalls in function of $\mathcal{SD}_{err}$ and π_{STERN} for a short horizon length of 10s.*

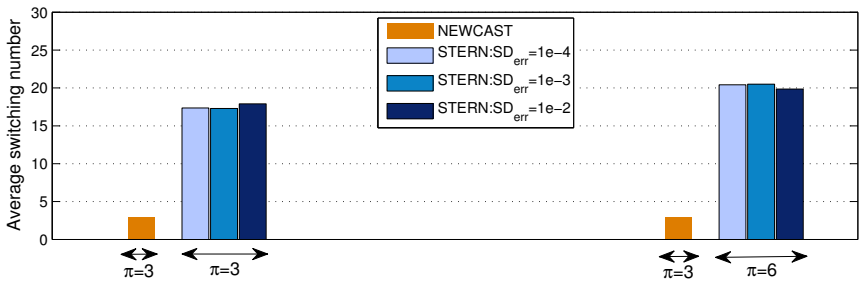


Figure 5.6: *STERN vs. NEWCAST: Average number of quality-switching in function of $\mathcal{SD}_{err}$ and π_{STERN} for a short horizon length of 10s.*

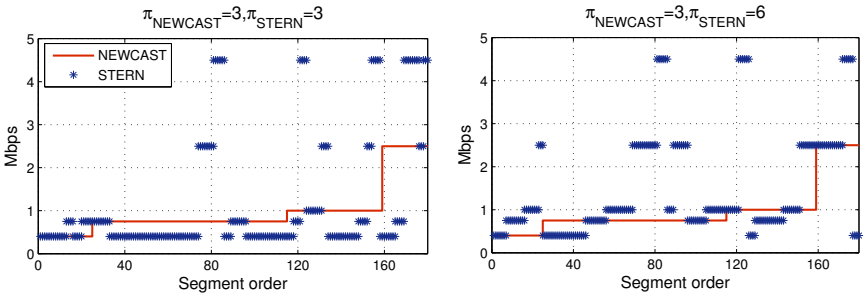


Figure 5.7: *STERN vs. NEWCAST: Snapshots of video quality variation in function of π_{STERN} for $\mathcal{SD}_{err} = 10^{-2}$ Mbits and a short horizon length of 10s.*

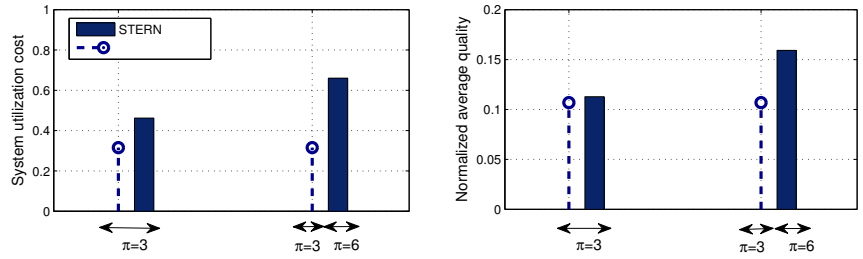


Figure 5.8: *STERN vs. NEWCAST: System utilization cost and normalized average video quality in function of π_{STERN} for $\mathcal{SD}_{err} = 10^{-2}$ Mbits and a short horizon length of 10s .*

5.3.2 Adaptive Short-TERm Newcast (A-STERN)

Algorithm

A-STERN is the dynamic adaptive version of STERN; a supervision on the real playback buffer state is continuously performed to see whether the real cumulative number of received frames matches the predicted function $\tilde{u}$ or not.
If $|u_{real}(t) - \tilde{u}(t)| \geq \epsilon \ , t \in \mathcal{H}_i, \ i > 0$ where ϵ is a predefined constant, then we update the streaming strategy; we stop streaming the video and build a new strategy over a new short horizon $\mathcal{H}_{i+1}$ using the approach of STERN (see Algorithm 7 and Figure 5.9).

Algorithm 7: Adaptive Short-TERm Newcast (A-STERN)

Input: $\mathcal{H}_1 = [t_0, t_1], X_{\mathcal{H}_1} = \frac{\tilde{u}(t_1)}{SegmentSize}, i = 1;$
while *Still segments to stream* **do**
  $t = t_{i-1};$
  Predict $\tilde{c}_{\mathcal{H}_i}$*;*
  Check if it is possible to stream $X_{\mathcal{H}_i}$ *segments over* $\tilde{c}_{\mathcal{H}_i}$*, otherwise, reduce* $X_{\mathcal{H}_i}$*;*
  $[\alpha^*_{\mathcal{H}_i}, \gamma^*_{\mathcal{H}_i}]$=***NEWCAST***$(\tilde{c}_{\mathcal{H}_i}, X_{\mathcal{H}_i}, \pi_{A-STERN}$, *BufferState*$)$;
  $QualityVect_{\mathcal{H}_i}$=*DeduceQualityVector*$(\gamma^*_{\mathcal{H}_i})$;
  $[u_{real}, t_{fin}]$=*StreamVideo*$(c_{real}[t_{i-1} : end], \alpha^*_{\mathcal{H}_i}, QualityVect_{\mathcal{H}_i})$;
  ***** ***Do simulataneously to streaming***
  while $t \in [t_{i-1}, t_i[$ **do**
    if $\frac{|u_{real}(t) - \tilde{u}(t)|}{SegmentSize} \geq \epsilon$ **then**
      Update the streaming strategy:
      $t_i = t + 1;$
      $t_{i+1} = t_i + length(\mathcal{H}_i);$
      $\mathcal{H}_{i+1} = [t_i, t_{i+1}];$
      $X_{\mathcal{H}_{i+1}} = \frac{\tilde{u}(t_{i+1}) - u_{real}(t)}{SegmentSize};$
      $i = i + 1;$
      $t = 0;$
    else
      $t = t + 1;$
    end
  end
  *****
  if *the*$X_{\mathcal{H}_i}$*segments have been completely received* **then**
    $t_i = t_{fin} + 1;$
    $t_{i+1} = t_i + length(\mathcal{H}_i);$
    $\mathcal{H}_{i+1} = [t_i, t_{i+1}];$
    $X_{\mathcal{H}_{i+1}} = \frac{\tilde{u}(t_{i+1}) - u_{real}(t_{fin})}{SegmentSize};$
    $i = i + 1;$
  end
end
return $(\{\alpha^*_{\mathcal{H}_i}\}, \{\gamma^*_{\mathcal{H}_i}\})$

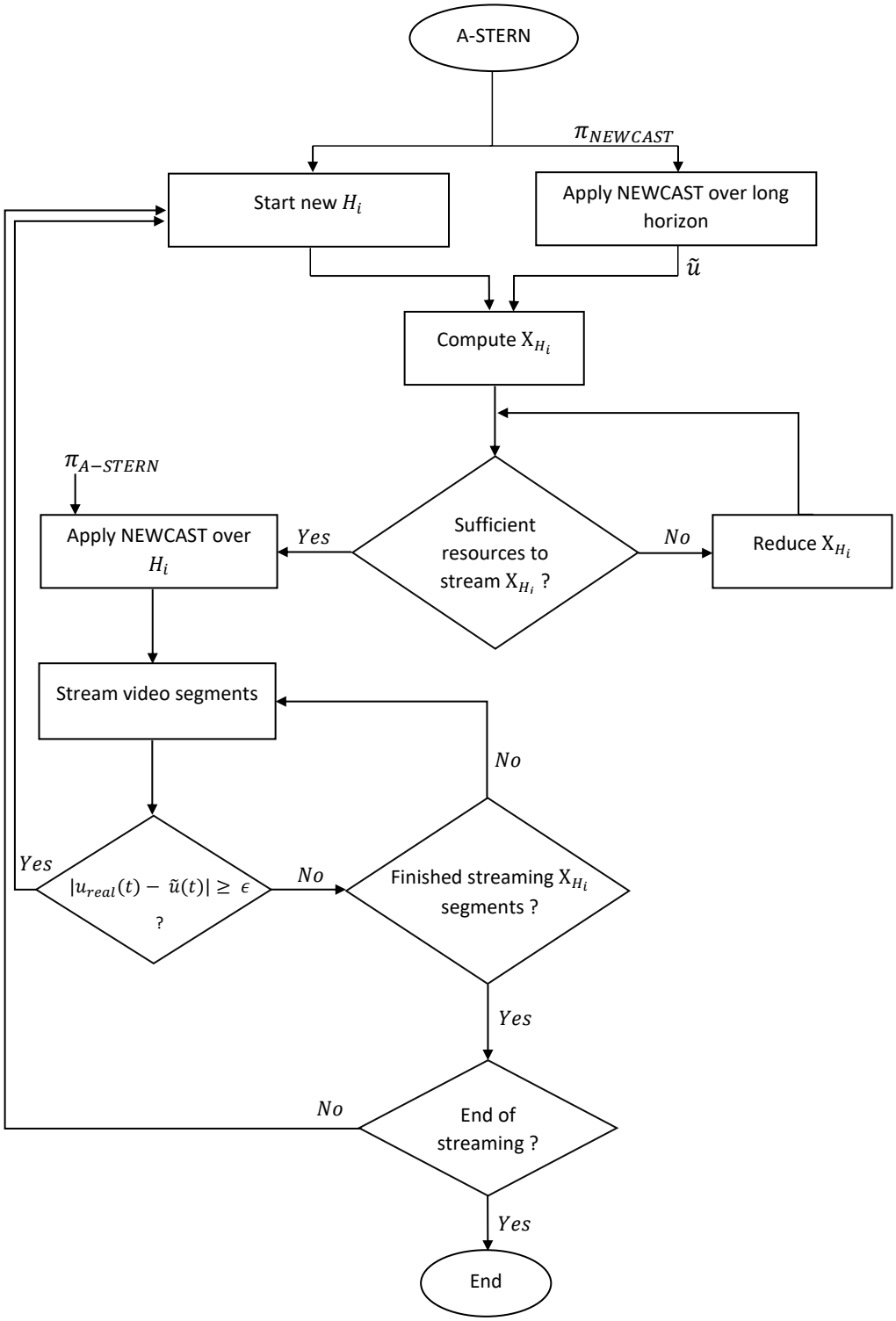


Figure 5.9: *Flowchart of A-STERN algorithm.*

Performance

In Figure 5.10, we compare A-STERN to STERN by computing the distribution of the number of stalls during the streaming session. From the plots, we notice that A-STERN increases the probability of having zero stalls compared to STERN: From 0.24 to 0.64 and from 0.95 to 0.97 for $\pi_{STERN} = \pi_{A-STERN} = 3$ and $\pi_{STERN} = \pi_{A-STERN} = 6$ respectively, at the highest prediction error. This mainly induced a decrease in the average number of stalls: From 1.02 to 0.38 and from 0.05 to 0.03 at the same values of π and at the same prediction error. A noteworthy observation here is that this improvement is mainly prominent at the lowest value of π, since there, the system is very sensitive to prediction errors. Which implies that the updates made on the playback buffer become more useful at the lowest values of π.

As for the number of switching and the system performance, we make from Figure 5.11 and Figure 5.13 several interesting observations: First, when π is low, the system utilization cost and the number of quality-switching are both reduced with A-STERN by comparison with STERN, which is quite interesting for the operator and the end user. Second, when π is high, the video average quality holds the same with A-STERN as with STERN, whereas the number of quality-switching increases, which is not quite appreciated for the user's perception. Overall, whatever is the value of π the quality-switching rate is considered to be relatively high, the reason for which we propose the two following algorithms RECAST and STREET.

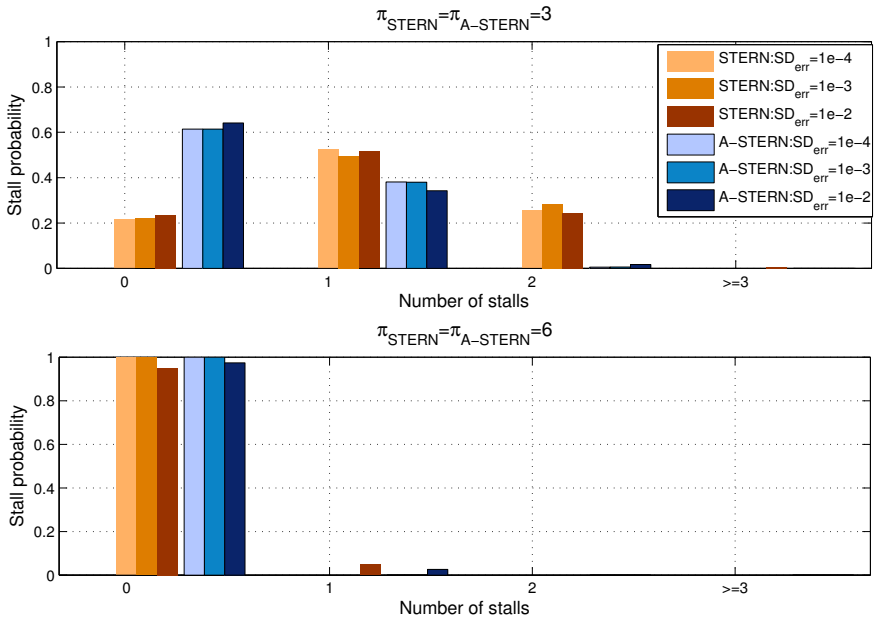


Figure 5.10: *A-STERN vs. STERN: Probability of stalls as function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s and ϵ= 3 segments.*

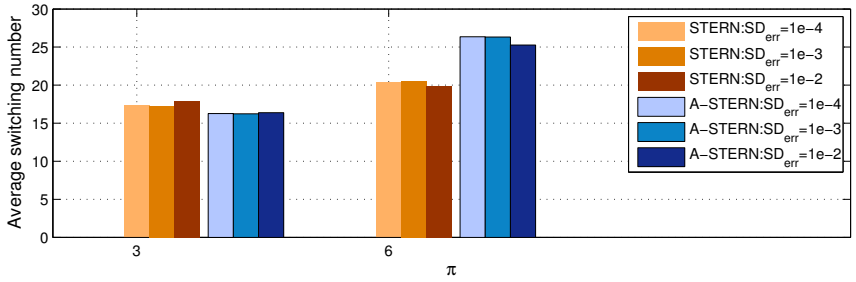


Figure 5.11: *A-STERN vs. STERN: Average quality-levels' switching number as function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s and ϵ= 3 segments.*

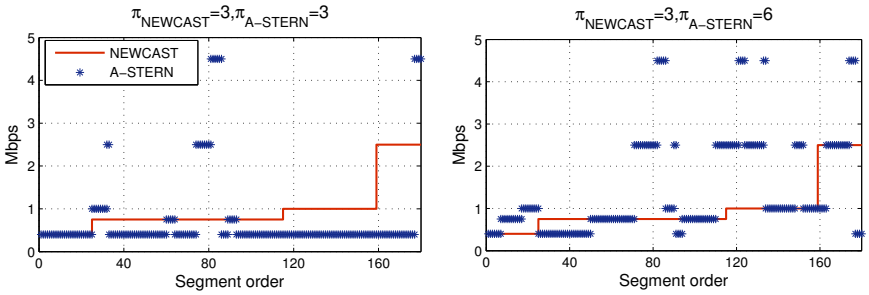


Figure 5.12: *A-STERN vs. NEWCAST: Snapshots on quality-level variation as function of π for$\mathcal{SD}_{err} = 10^{-3}$ Mbits, a short horizon length of 10s and ϵ= 3 segments.*

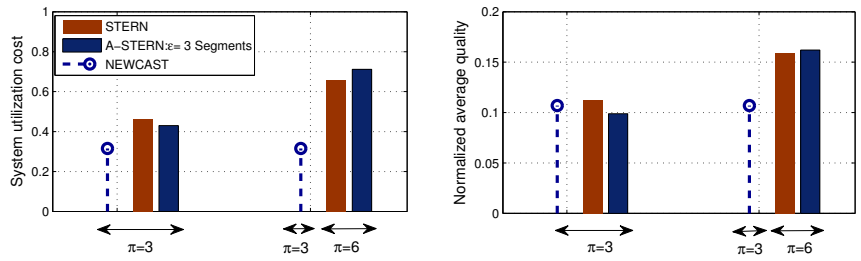


Figure 5.13: *A-STERN vs. STERN and NEWCAST: System utilization cost and normalized average quality as function of π for $\mathcal{SD}_{error} = 10^{-2}$ Mbits and a short horizon length of 10s.*

5.4 shoRt-tErm Conservative newcAST for smoother quality variation: (RECAST)

Algorithm

Our previous numerical results clearly showed that short-term NEWCAST versions (STERN and A-STERN) succeed at decreasing the probability of stalls compared to NEWCAST. However, this comes at the expense of an increased number of quality-switching. In this Section, we propose to focus more on the quality-switching rate rather than the number of stalls. To do so, we define RECAST as a modified version of STERN. While STERN uses the same *ascending* bitrate strategy as NEWCAST over

each short horizon $\mathcal{H}_i, i > 0$, RECAST enforces the bitrate strategy to be *constant*. The final quality $\gamma_{\mathcal{H}_i}$ and the final threshold $\alpha_{\mathcal{H}_i}$ are accordantly set depending on the value of π_{RECAST} (see Figure 5.14).

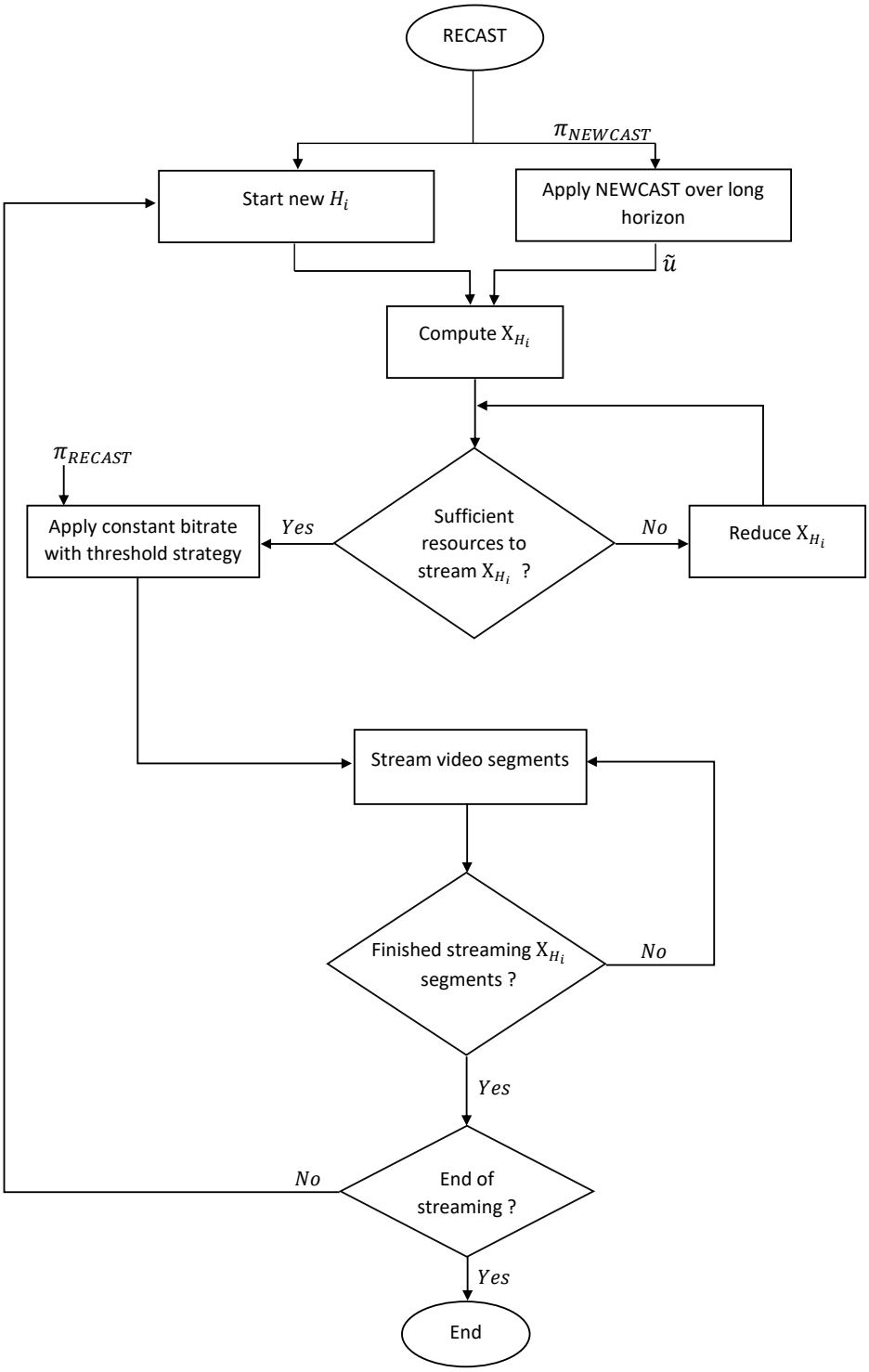


Figure 5.14: *Flowchart of RECAST algorithm.*

Performance

Figure 5.15 plots the average number of quality-switching induced by RECAST compared to the previous algorithm A-STERN. It is shown from the results that the number of quality-switching is decreased from 16.39 to 9.86 and from 25.27 to 12.29 for $\pi_{A-STERN} = \pi_{RECAST} = 3$ and $\pi_{A-STERN} = \pi_{RECAST} = 6$, respectively, at the highest prediction error. Which is what we intended to do by setting the constant bitrate strategies. However, this comes at the expense of the probability of stalls and the video average quality; according to Figure 5.17 and Figure 5.18, the number of stalls is increased from 0.38 to 0.67 at the lowest value of π (where the system is the most sensitive to prediction errors) whereas the video average quality is decreased from 0.67 Mbps to 0.479 Mbps at the highest value of π (where the QoE part should be the most prioritized). No big amelioration is noticed for the system utilization cost, except a slight reduction when π is high.

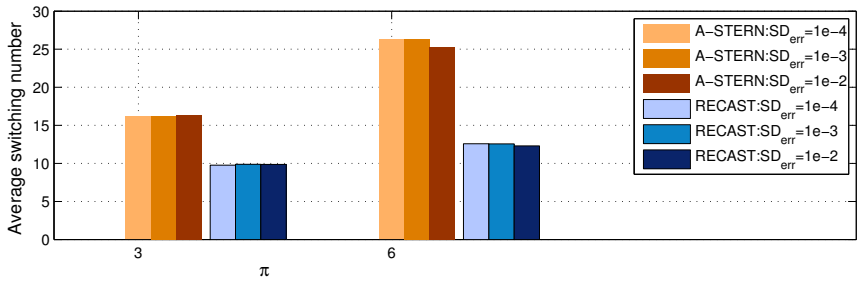


Figure 5.15: *RECAST vs. A-STERN: Average number of quality-switching in function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s.*

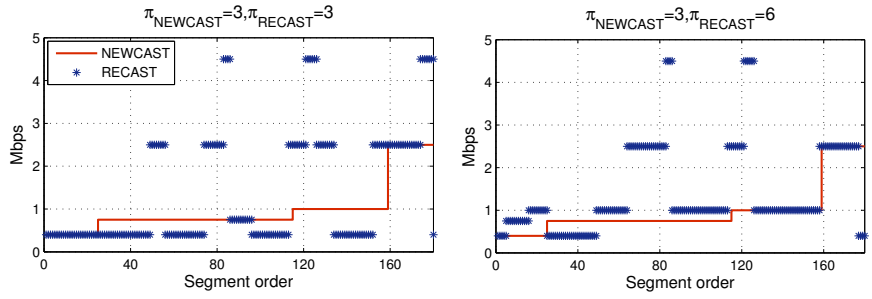


Figure 5.16: *RECAST vs NEWCAST: Snapshots of quality variation in function of π for $\mathcal{SD}_{err} = 10^{-2}$ Mbits and a short horizon length of 10s.*

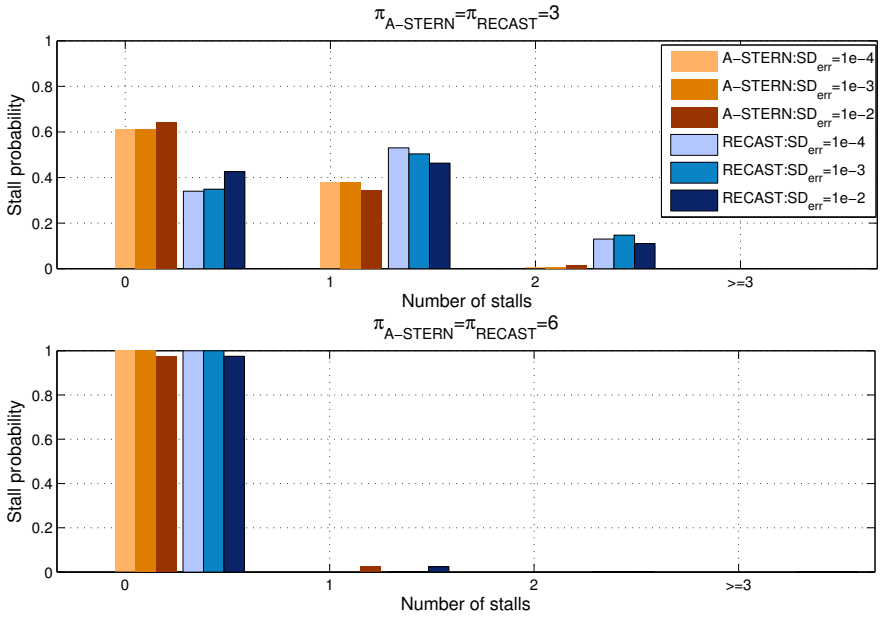


Figure 5.17: *RECAST vs. A-STERN: Probability of stalls as function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s and ϵ= 3 segments.*

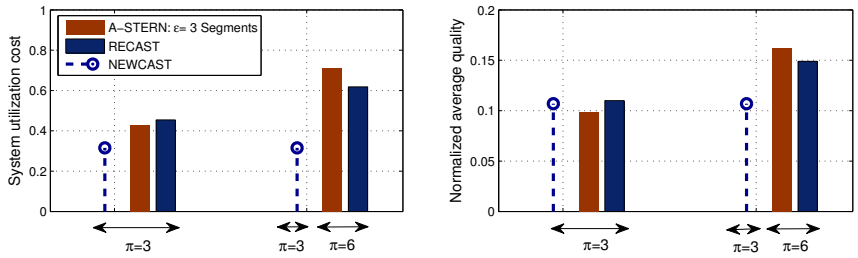


Figure 5.18: *RECAST vs. A-STERN and NEWCAST: System utilization cost and normalized average quality as function of π for $\mathcal{SD}_{err} = 10^{-2}$ Mbits and a short horizon length of 10s .*

5.5 Short-TeRm Enslaved nEwcasT (STREET)

Algorithm

Unlike the previous algorithms that set their bitrate strategies independently of the set of bitrates generated by NEWCAST (using the long-term predicted capacity $\tilde{c}$), STREET algorithm attempts to follow the same set of bitrates as NEWCAST in the hope of giving similar performance with a reduced number of stalls. Here is how it performs: At the beginning of each short horizon $\mathcal{H}_i$, i>0, it computes the number of segments $X_{\mathcal{H}_i}$ to stream by seeing $\tilde{u}$. If it is possible to perform the streaming with the same set of bitrates as NEWCAST, it sets up the highest possible threshold $\alpha_{\mathcal{H}_i}$ to reduce as possible the system utilization cost over $\mathcal{H}_i$. Otherwise it reduces the quality of the segments by following the approach of STERN algorithm. Detailed steps are described in Algorithm 8 and Figure 5.19.

Algorithm 8: short-term Enslaved nEwcasT (STREET)

Data: $\mathcal{H}_1 = [t_0, t_1], X_{\mathcal{H}_1} = \frac{\tilde{u}(t_1)}{SegmentSize}, i = 1$, *Start=1* ;
while *Still segments to stream* **do**
  Predict $\tilde{c}_{\mathcal{H}_i}$;
  $QualityVect_{\mathcal{H}_i}$= *NEWCASTQuality*[*Start:Start+*$X_{\mathcal{H}_i} - 1$] ;
  if *It is possible to stream* $X_{\mathcal{H}_i}$ *segments under* $QualityVect_{\mathcal{H}_i}$, *over predicted* $\tilde{c}_{\mathcal{H}_i}$ **then**
    find the best threshold $\alpha^*_{\mathcal{H}_i}$;
  else
    Check if it is possible to stream $X_{\mathcal{H}_i}$ *segments over* $\tilde{c}_{\mathcal{H}_i}$, *otherwise, reduce* $X_{\mathcal{H}_i}$;
    $[\alpha^*_{\mathcal{H}_i}, \gamma^*_{\mathcal{H}_i}]$=***NEWCAST***$(\tilde{c}_{\mathcal{H}_i}, X_{\mathcal{H}_i}, \pi_{STREET}$, *BufferState*);
    $QualityVect_{\mathcal{H}_i}$= *DeduceQualityVector*$(\gamma^*_{\mathcal{H}_i})$;
  end
  $[u_{real}, t_{fin}]$=*StreamVideo*$(c_{real}[t_{i-1} : end], \alpha^*_{\mathcal{H}_i}, QualityVect_{\mathcal{H}_i})$;
  $t_i = t_{fin} + 1$;
  $t_{i+1} = t_i + length(\mathcal{H}_i)$;
  $\mathcal{H}_{i+1} = [t_i, t_{i+1}]$;
  $X_{\mathcal{H}_{i+1}} = \frac{\tilde{u}(t_{i+1}) - u_{real}(t_{fin})}{SegmentSize}$;
  $i = i + 1$;
  start=$\frac{u_{real}(t_{fin})}{SegmentSize} + 1$;
end
return $(\{\alpha^*_{\mathcal{H}_i}\}, \{\gamma^*_{\mathcal{H}_i}\})$

Performance

Figure 5.20 and Figure 5.21 show the performance of STREET in terms of video quality-switching by comparison with NEWCAST and the previous algorithm RECAST. We notice from the results that STREET succeeds at following a near trend of bitrate variation as NEWCAST with a near number of quality-switching (around 3.03 for both values of π_{STREET}). The performance in terms of system utilization cost and video average quality is evenly near to NEWCAST's performance (around 0.39 and 0.48 Mbps for both values of π_{STREET}). As for the distribution of the number of stalls, it is depicted from Fig .5.23 that the probability of having zero stalls decreased compared to RECAST from 0.97 to almost 0.32 for $\pi_{STREET} = 6$, under the highest prediction error. As a result, the average number of stalls increased from 0.03 to 0.71 but remained relatively low compared to NEWCAST. Overall, we consider STREET as the best algorithm that mostly achieves the same performance as NEWCAST (in terms of system cost, average quality and quality-switching), while reducing the risk of video stalls by around 75%.

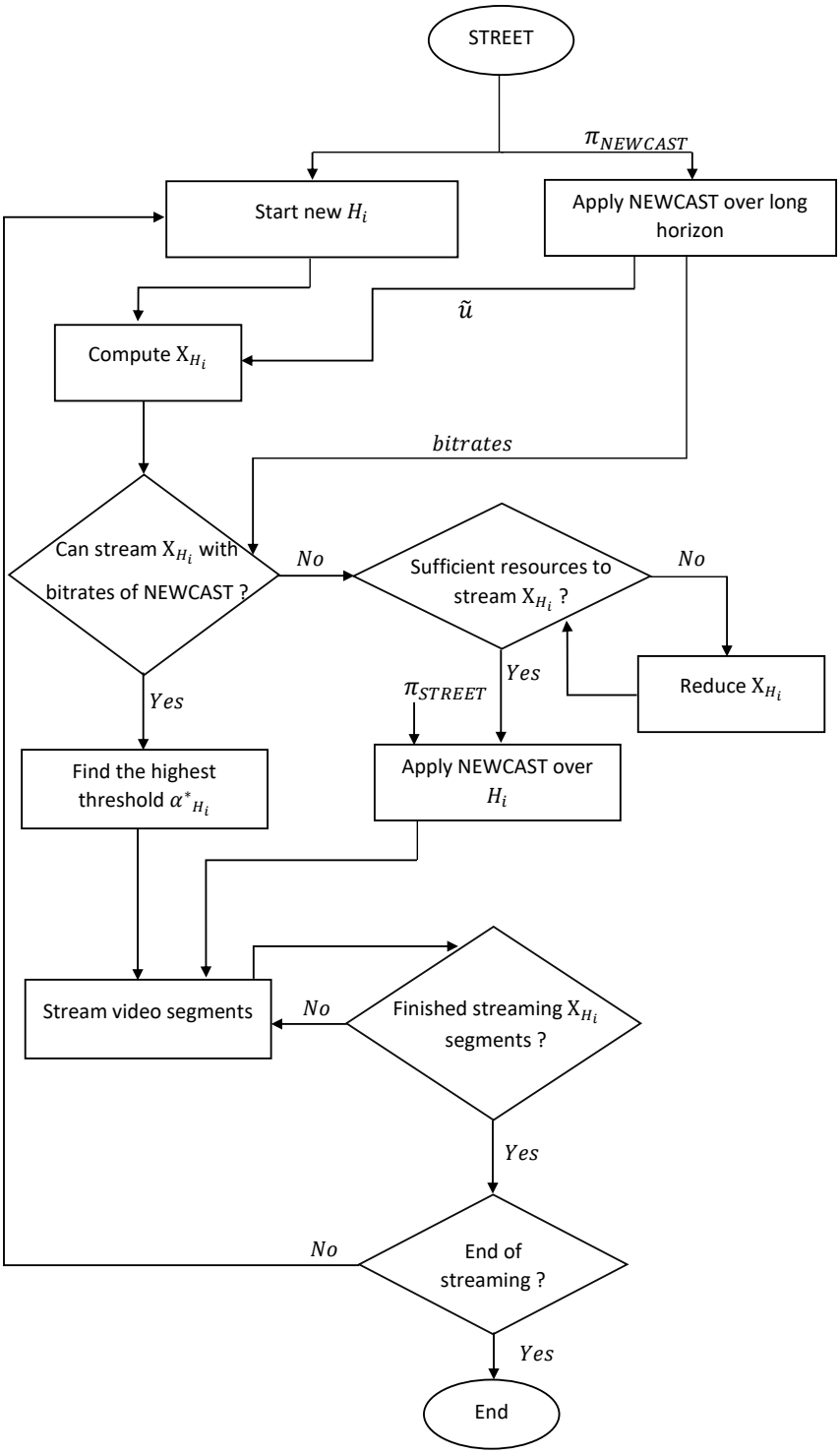


Figure 5.19: *Flowchart of STREET algorithm.*

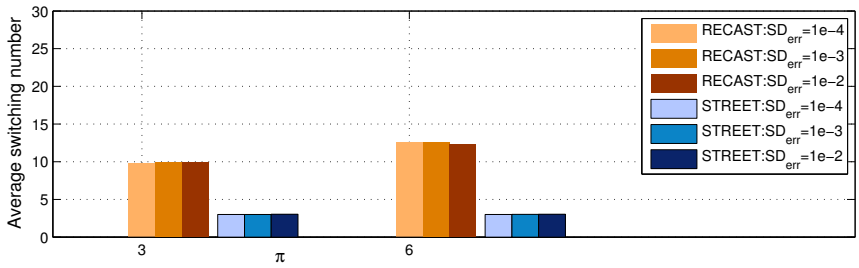


Figure 5.20: *STREET vs. RECAST: Average number of quality-switching in function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s.*

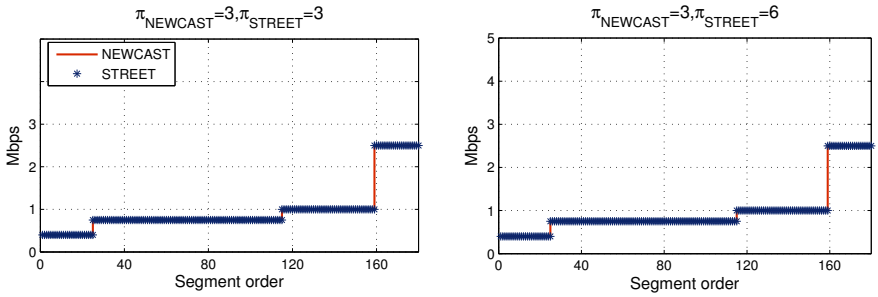


Figure 5.21: *STREET vs. NEWCAST: Snapshots of quality variation in function of π for $\mathcal{SD}_{err} = 10^{-2}$ Mbits and a short horizon length of 10s.*

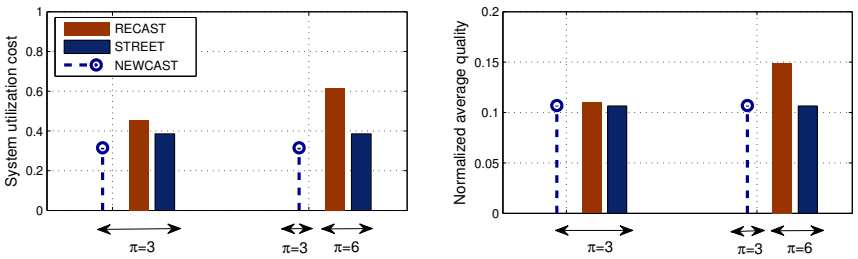


Figure 5.22: *STREET vs. RECAST and NEWCAST: System utilization cost and normalized average quality in function of π for $\mathcal{SD}_{err} = 10^{-3}$ Mbits and a short horizon length of 10s.*

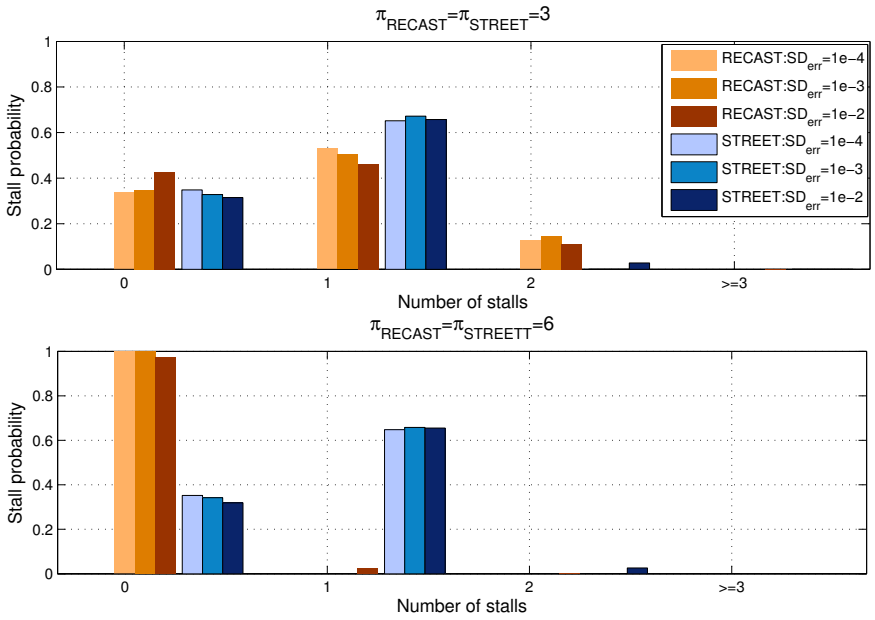


Figure 5.23: *STREET vs. RECAST: Probability of stalls in function of $\mathcal{SD}_{err}$ and π for a short horizon length of 10s.*

5.6 Summary of numerical results

In order to gain insights into the performance of the five proposed algorithms, we summarize in Table 5.1 the average system utilization cost, the average number of stalls, the video average quality and the average number of quality-switching for $\mathcal{SD}_{err} = 10^{-3}$ Mbits and for short horizons of $10s$. The performances of all the algorithms are obtained by averaging results over 1000 simulations using different randomly generated capacities c_{real} and one long-term predicted capacity $\tilde{c}$. A-STERN is evaluated for $\epsilon = 3$ segments.

Here is a summary of the most worthy observations we make:

- NEWCAST achieves the lowest system utilization cost but gives the highest number of video stalls.
- STERN and A-STERN are the most costly in terms of system usage but are the most resistant to video stalls mainly for low values of π. A-STERN is the best one in reducing the average number of stalls: it produces 0.38 and 0.03 stalls in average against 1.02 and 0.05 stalls with STERN. The video average quality increases sensibly with STERN and A-STERN compared to NEWCAST, whereas the number of quality-switching increases tremendously.
- Compared to STERN and A-STERN, RECAST performs better in terms of quality-switching, but it produces more video stalls when π is small.
- STREET performs the best in terms of quality-switching and achieves almost the same performance as NEWCAST while it reduces the average number of stalls by almost 75%.

	System utilization cost		Average number of stalls		Average quality (Mbps)		Average number of switches	
	$\pi = 3$	$\pi = 6$	$\pi = 3$	$\pi = 6$	$\pi = 3$	$\pi = 6$	$\pi = 3$	$\pi = 6$
STERN	0.46	0.66	1.02	0.05	0.51	0.72	17.91	19.86
A-STERN	0.43	0.71	0.38	0.03	0.44	0.73	16.39	25.27
RECAST	0.45	0.62	0.69	0.03	0.49	0.67	9.86	12.29
STREET	0.39	0.39	0.71	0.71	0.48	0.48	3.03	3.03
NEWCAST $\pi = 3$	**0.32**		**1.90**		**0.48**		**3**	

Table 5.1: *Performance summary of the five algorithms for $\mathcal{SD}_{err} = 10^{-3}$, a short horizon length of 10s, and ϵ=3 segments.*

To clearly see the broughts and the drawbacks of STERN, A-STERN, RECAST and STREET by comparison with NEWCAST, we add a colored map of performance in Figure 5.24.

As depicted in the figure, each algorithm achieves its own tradeoff, thus, we cannot judge which one is the best unless we know the preferences of the user and the operator as well.

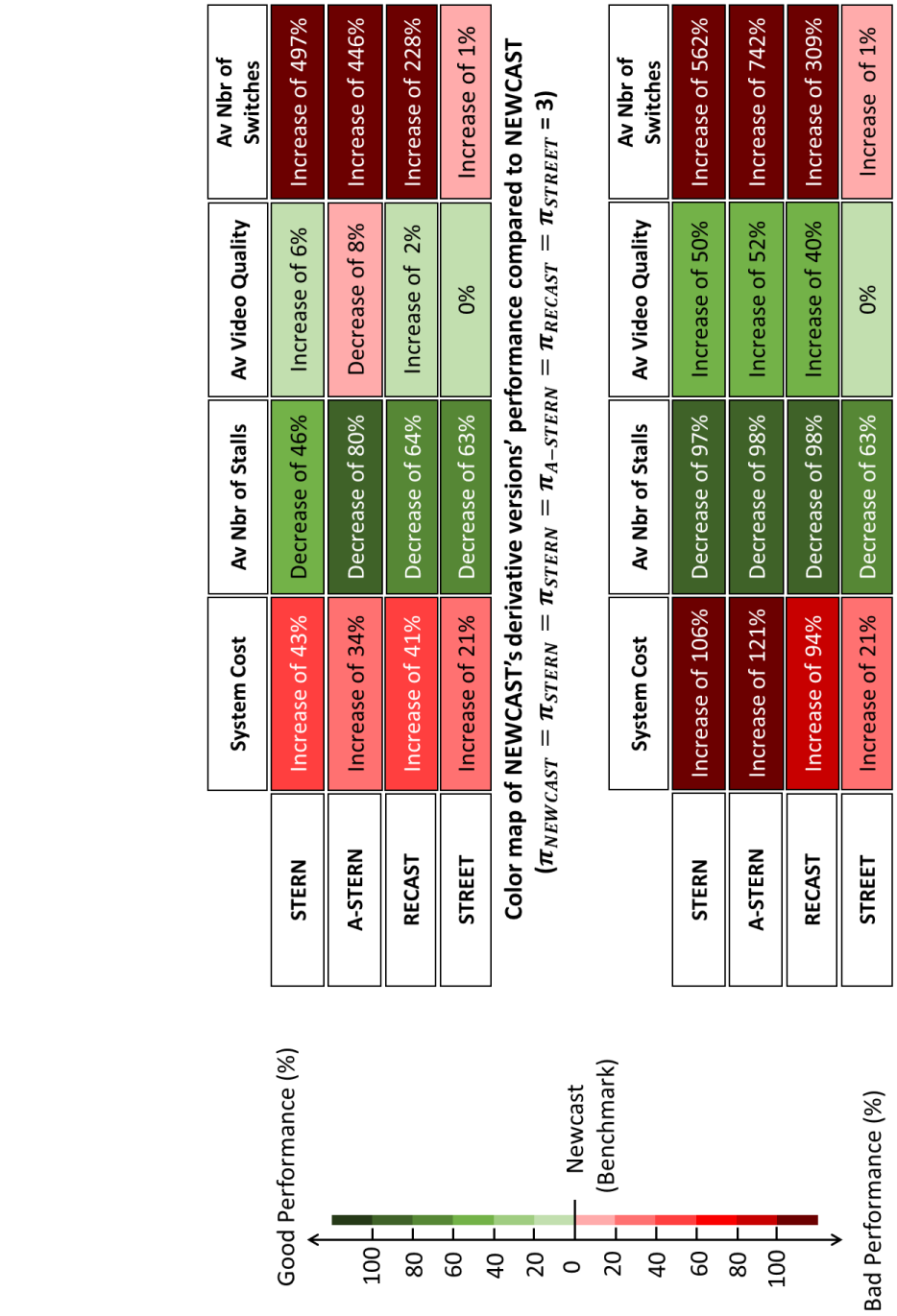


Figure 5.24: *NEWCAST's derivative versions' broughts and drawbacks by comparison with NEWCAST.*

5.7 Conclusion

In this Chapter, we have proposed four derivative versions of NEWCAST to make its streaming approach more robust to throughput prediction errors. STERN, A-STERN, RECAST and STREET are the names of our proposed algorithms. Each algorithm leverages on the short-term throughput prediction over successive short horizons and the playback buffer evolution initially computed by NEWCAST over the long future horizon.

We have proposed the first two algorithms STERN and A-STERN to reduce the average number of video stalls returned by NEWCAST. We unfortunately have found that they produce high numbers of quality-switching. Which has led us to rethink the way the video bitrates are distributed. Thus, we have proposed RECAST and STREET with the purpose of making smoother quality variations.

Our numerical results, obtained through extensive Matlab simulations, have revealed the efficiency of all our proposed solutions. In the next Chapter, we implement NEWCAST and all its derivative versions with a real DASH player to explore to what extend can they be used in realistic environments.

Chapter 6

Implementation of NEWCAST and its Derivative Versions with a Real DASH Player

Contents

6.1 Introduction

In the two previous Chapters we proposed NEWCAST and its derivative versions as frameworks for adaptive video delivery assuming the knowledge of the user's future throughput. In a first step, we assumed that the throughput prediction is accurately performed over long-term future horizons. Based on adaptive streaming properties, we developed our analytical streaming model, then accordingly, we designed the framework of NEWCAST to strike a balance between system utilization and QoE. In a second step, we assumed the presence of throughput prediction errors. Based on the same analytical model, we designed STERN, A-STERN, RECAST and STREET as derivative versions from NEWCAST to mitigate the problem of inaccurate throughput prediction.

In our analytical model, we made some key assumptions to make it easy design the frameworks. These assumptions are mainly related to the video segments' bitrates and the HTTP requests' durations. In this Chapter, we show through experiments how these assumptions will impact the performance of NEWCAST, and subsequently its derivative versions, in a realistic streaming environment. We propose some changes inside the frameworks and evaluate their performances as we did in the simulation Sections.

We organize the Chapter as follows: In Section 6.2, we describe throw sequence diagrams how each framework should be deployed in realistic environments. Then in Section 6.3, we review our experimental environment and how it is configured. We detail the changes we propose inside NEWCAST as well. In Section 6.4, we analyze the performance of each framework. Finally, in Section 6.5, we conclude the Chapter.

6.2 Interactions within realistic environments

6.2.1 Entities involved in standard mobile video streaming

In a standard mobile video streaming, the main entities that are directly involved in the streaming process are potentially the client (player) and the video server. In adaptive video streaming, the client requests video segments one after the other to the server through independent HTTP requests. It indicates at each time the bitrate of the segment being requested. Disposing of all video representations, the server replies by streaming the requested segments at their requested bitrates. In a mobile environment, video segments are transmitted to the client through the operator's network resources.

6.2.2 NEWCAST deployment in mobile video streaming

To integrate NEWCAST in a user's mobile device and allow the exchange of the future capacity c and the threshold α^*, two cross layers should be implemented: One at the level of the user's device and the other at the level the operator's resource allocator

(e.g., eNodeB in LTE). In Figure 6.1 we design the main interactions that may take place between NEWCAST and the real streaming entities:

As soon as the streaming session is opened, the player notifies NEWCAST to compute the list of segments' qualities before starting the streaming. Consequently, NEWCAST requests to the operator the user's future available throughput, then starts computing the adequate streaming strategy. After processing, NEWCAST sends the list of segments' qualities back to the player and the transmission threshold α^* to the operator's scheduler. While the scheduler applies the threshold-based transmission scheme, the player requests the segments one after the other to the server respecting in that the list of qualities computed by NEWCAST.

The most challenging point of this scenario is the way to apply the threshold-based transmission scheme. In a cellular network where multiple users with different network conditions are computing for the resources, it is hard to determine beforehand the exact amount of data to serve to a specific user. This problem will be handled in a future research work. At this level, we may propose a simple way to apply the threshold-based transmission scheme: At each scheduling time slot, if the amount of data (TBS in LTE) allocated to the user is above the threshold, then the user will be served. Otherwise, the resources are released to be allocated to other users.

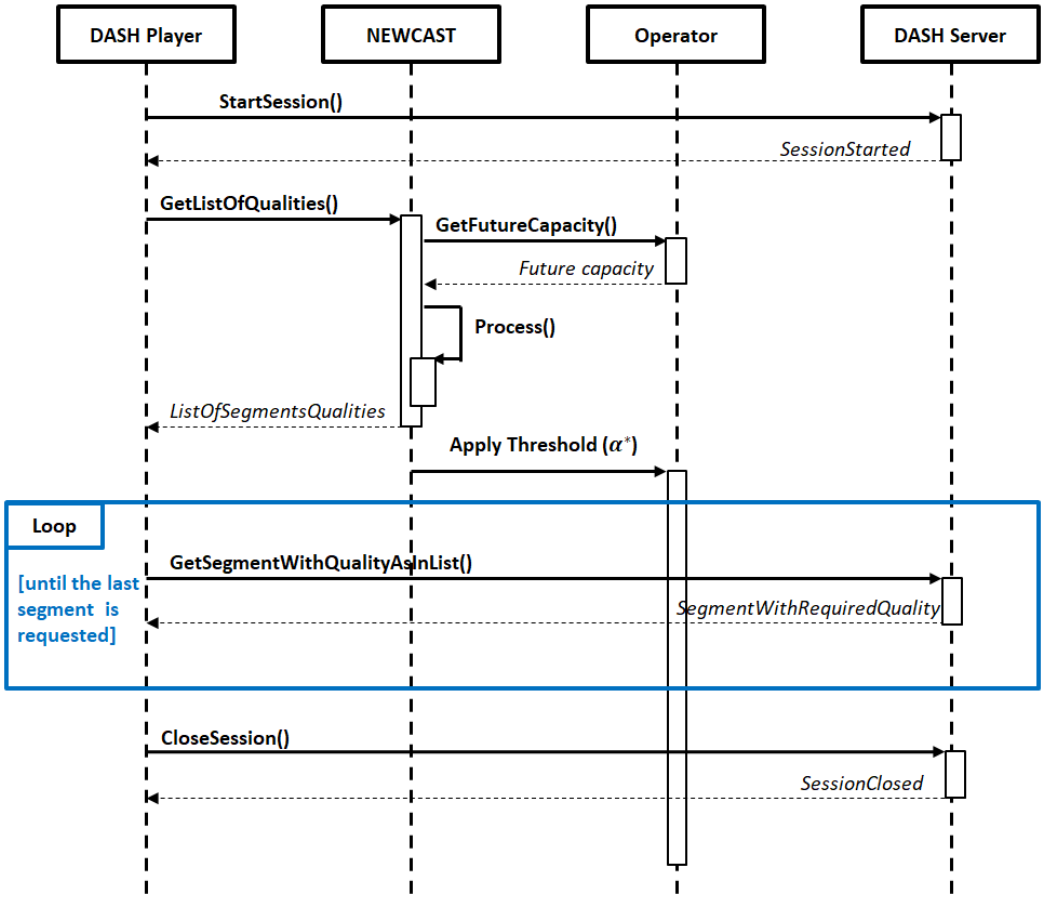


Figure 6.1: *Sequence diagram of a video streaming session using NEWCAST.*

6.2.3 NEWCAST's derivative versions deployment is mobile video streaming

The same type of interactions holds for NEWCAST's derivative versions except few differences. In the following, we show the sequence diagrams of the four proposed algorithms (STERN, A-STERN, RECAST and STREET).

STERN and RECAST

Figure 6.2 depicts the main interactions we conceive between NEWCAST, STERN (or RECAST) and the other streaming entities. The role of NEWCAST relies in generating the *path* [1] for STERN (or RECAST) to know in advance the number of segments to stream during each short horizon. The process of computing the streaming strategy is repeated by STERN (or RECAST) at the beginning of each short horizon. In accordance to each strategy, the operator's scheduler applies the threshold-based transmission scheme and the player requests the segments at their indicated bitrates.

A-STERN

With A-STERN, the interactions hold the same as with STERN and RECAST, nevertheless, a supervision process on the real time playback buffer evolution is scheduled upon each segment arrival. If the difference between the *path* $\tilde{u}$ and the real cumulative number of received frames u_{real} is greater than ϵ, then A-STERN starts a new short horizon even if the current horizon has not yet finished (see Figure 6.3).

Remark 1. *In the simulation part of the previous Chapter, we conceived the supervision to be performed at each time slot. In the experiments, we perform the supervision only at the moments where video segments are received. This will return a lower complexity and a lower power consumption at the user device.*

STREET

STREET interacts with the streaming entities in the same way as STERN and RECAST. Nevertheless, when asking for the *path* from NEWCAST, it asks for the corresponding set of segments' qualities, since, as explained in the previous Chapter, it attempts to follow the same strategy as NEWCAST (in terms of bitrate distribution). We mark this with red stars in Figure 6.4.

[1] The path is the cumulative number of received frames function $\tilde{u}$ defined in the previous Chapter.

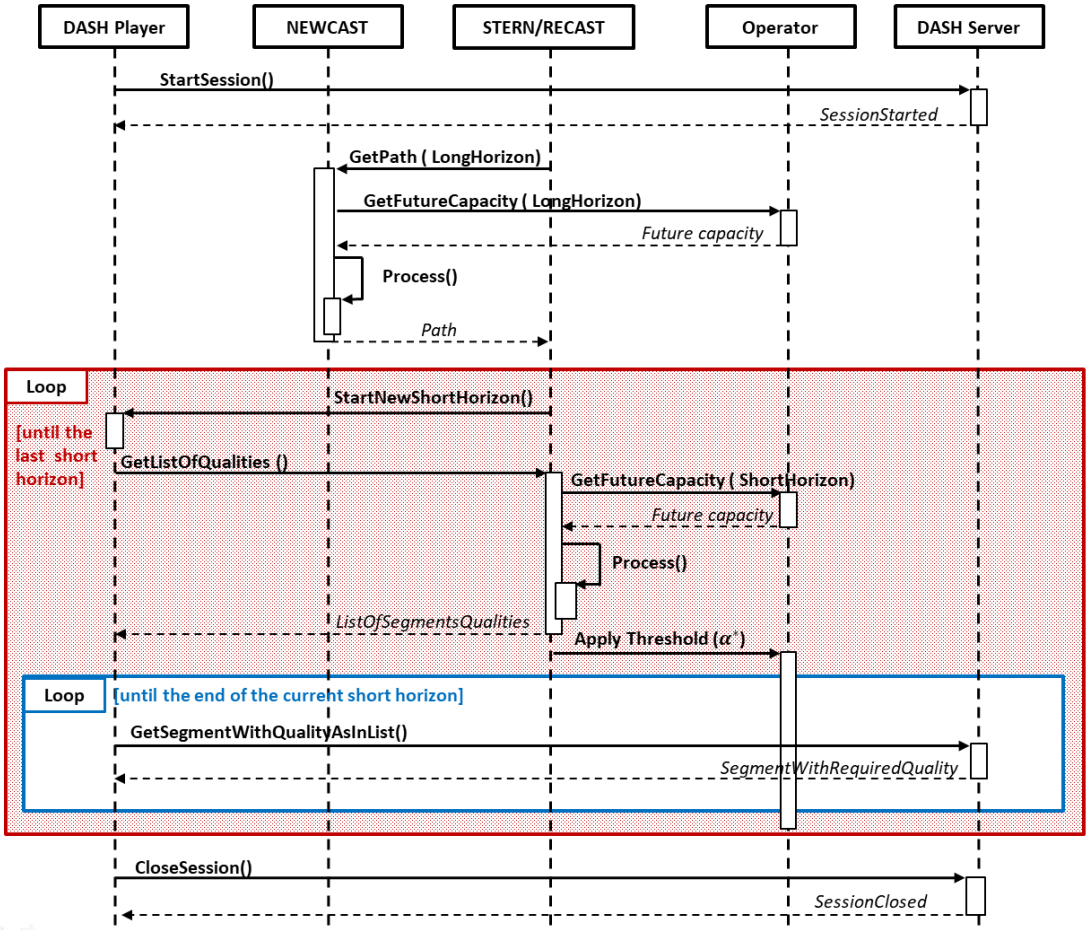


Figure 6.2: *Sequence diagram of a video streaming session using STERN/RECAST.*

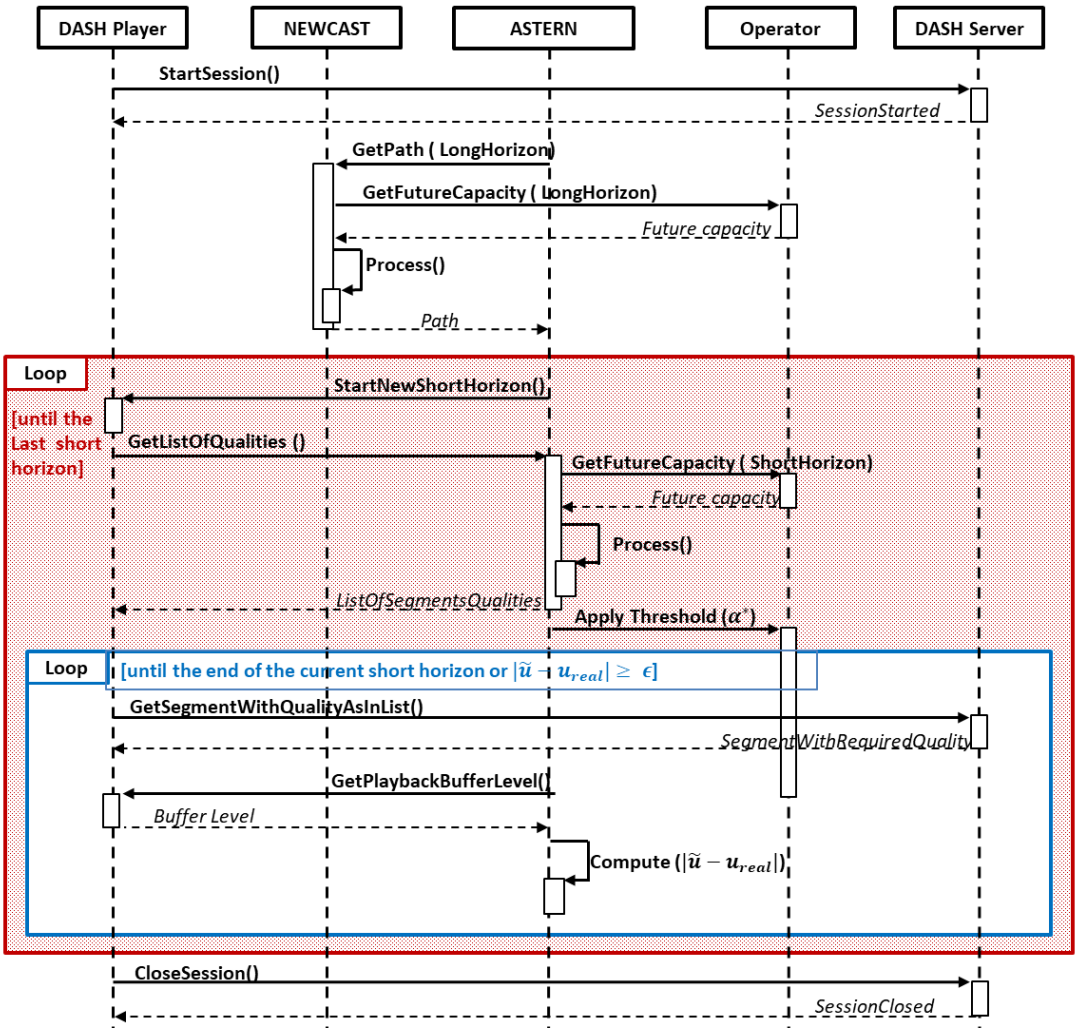


Figure 6.3: *Sequence diagram of a video streaming session using A-STERN.*

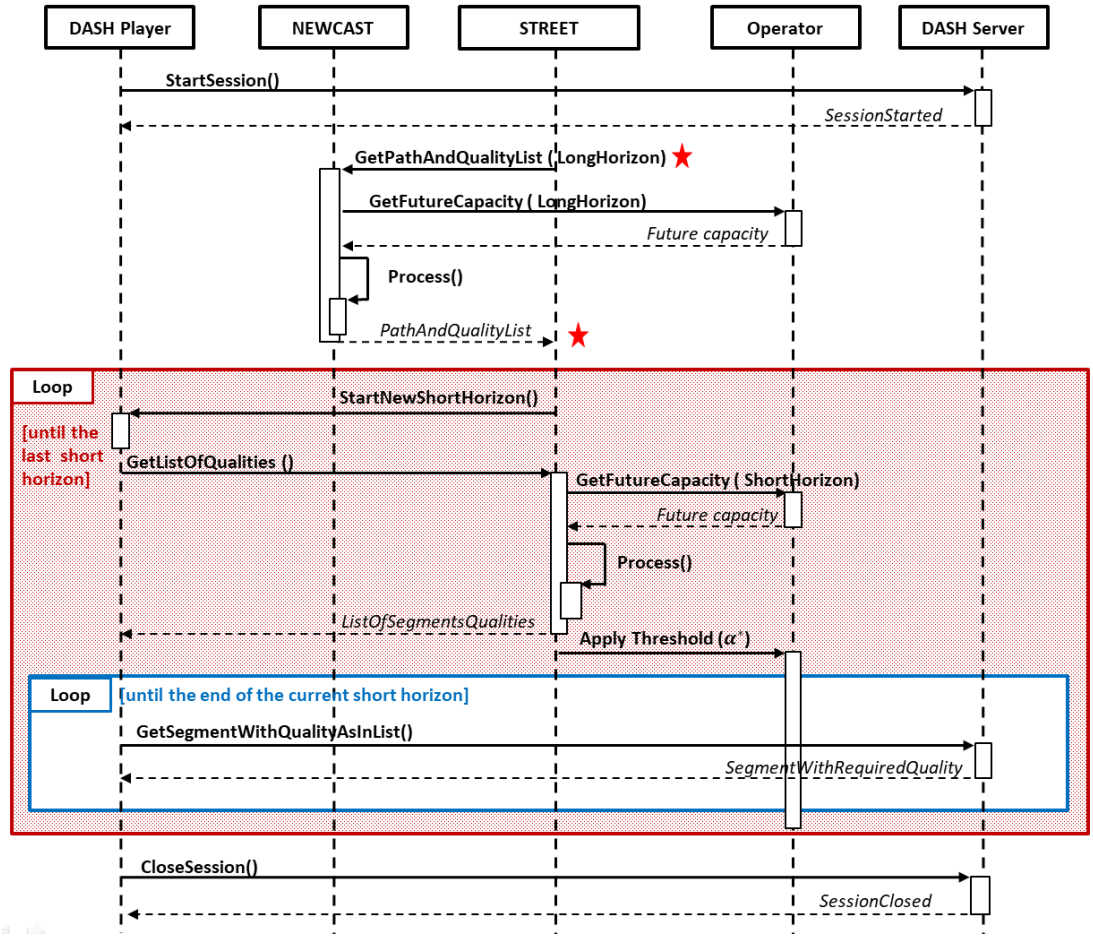


Figure 6.4: *Sequence diagram of a video streaming session using STREET.*

6.3 Implementation tools and configurations

6.3.1 Environment and tools

For the implementation, we use two virtual machines with Linux distributions: One is used as a DASH server and the other is used as a DASH client. In the DASH server, we install the HTTP Apache server and put inside the *Dashjs* framework [23] with the *Envivio* video segments (encoded at different quality levels) and their corresponding *.mpd* file [88]. In the DASH client, we just install *google chrome* browser.

We configure the two virtual machines to be able to communicate through their Ethernet interfaces. To emulate the network schedule and make the bandwidth between the two machines follow a predefined variation (considered as the predicted capacity), we use the Linux *tc-tool* for traffic shaping as shown in Figure 6.5.

To implement NEWCAST and make it interact with the *Dashjs* player, we use Javascript and other basic web languages. We put NEWCAST with the player call function in a same *.php* file that the DASH client requests to start the video streaming session.

In Table 6.1 we put more details on the hardware/software tools used for the implementation.

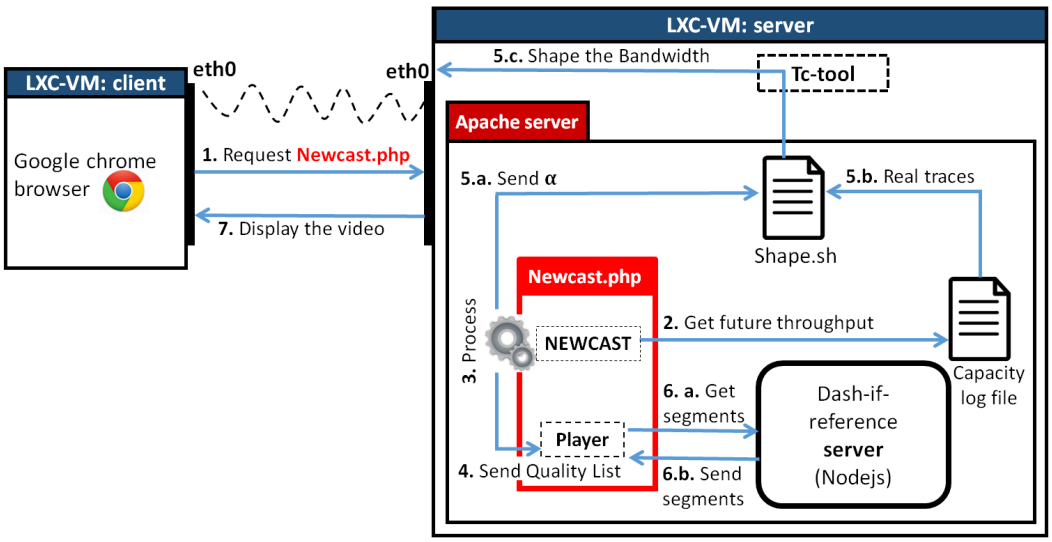


Figure 6.5: *Architecture of the system used for experiments.*

6.3.2 Client and server configurations

We create the virtual machines by the mean of the *LinuX Containers (LXC)* tool. Actually, the installation of the LXC package on the host machine creates by default a host-shared-bridge *lxcbr0* that is linked to the host machine's physical interface *eth0*.

Host machine	Optiplex 7010 Intel Core i7-3770 CPU 3.40Ghz
Distribution	Ubuntu 14.04.5 LTS
Virtual machines	Linux Container Lxc 1.0.9
Apache	2.4.7
Dashjs	2.4.0
Google Chrome	55.0.2883.87

Table 6.1: *Details on the software/hardware tools used for real implementation.*

This bridge has by default the first ip address in the network 10.0.3.0/24. Upon creation, each container will have one of its Ethernet interfaces connected to that bridge with an IP address automatically generated in the same network. This configuration will permit to the containers to be connected to the internet through the host machine and, thus, will permit to install on them all the required dependencies for the experiments. In Figure 6.6, we depict the general IP configuration of our system.

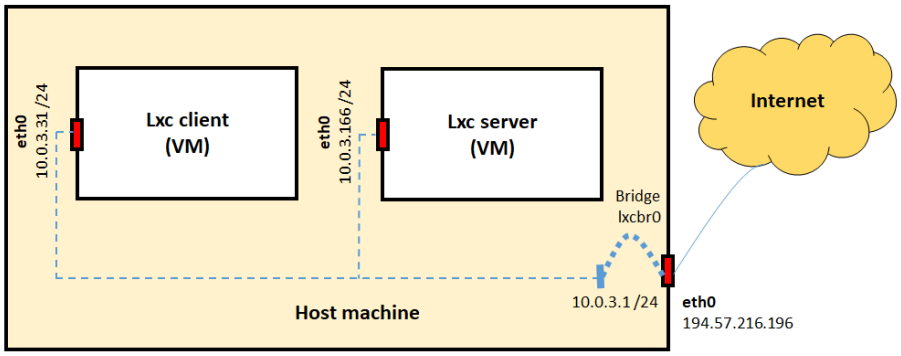


Figure 6.6: *General IP configuration of the system used for experiments.*

6.3.3 Player configuration

To make NEWCAST interact with the DASH player, we make some changes inside the *Dashjs* framework:

- We add a new event `"FRAGMENT_VIDEO_LOADING_COMPLETED"` to the player class to detect the moments where a segment of type ”video” is completely loaded to the playback buffer.
- The size of the playback buffer is by default limited to 42 seconds. As soon as this threshold is reached, the player keeps on delaying the requests of coming segments till having the buffer level reduced, which may disturb the continuous streaming aspect assumed by NEWCAST. In the *”MediaPlayerModel.js”* file, we change this restriction to fit the infinite buffer size and to eventually avoid delaying the requests of video segments.
- The size of the playback cache is by default limited to 30 seconds. As soon as this threshold is reached, the player keeps on removing the earliest played segments till having the cache level reduced. To avoid removing the segments and keep

track on the real time cumulative number of received frames u_{real}, we raise this threshold in the *"MediaPlayerModel.js"* file.

- By default, a video stall occurs when the playback buffer level is less than a predefined threshold (slightly bigger than zero). The rebuffering stage after a stall occurs stops when the buffer level becomes higher than this same threshold. As we assume independent thresholds in NEWCAST for both the stall occurrence and the rebuffering duration, we define in the "`check-IfSufficientBuffer()`" function a new independent threshold for the prefetching stage.
- Since we are only interested in the "video traffic", we delete the "audio traffic" from the *.mpd* file to not be requested by the player.

6.3.4 Throughput emulation

Bandwidth shaping using tc-tool

The Linux *tc-tool* is a user-space utility program used to configure the *"Traffic Control"* in the Linux kernel. It allows to (i) shape the traffic on egress, (ii) schedule the transmission of packets, (iii) police the traffic on ingress and (iv) drop the exceeding traffic. When a traffic is shaped, its rate of transmission is under control; that may include lowering the available bandwidth or smoothing out the bursts.

Traffic processing is controlled by three kinds of objects: *qdiscs* (for queueing discipline), *classes* (defined under some qdiscs) and *filters* (used by classful qdiscs).

To emulate the user throughput and make the bandwidth follow a predefined variation over time, we perform the traffic shaping at each time slot (each second) using the Hierarchy Token Bucket (HTB), which is a classful *qdisc*.

Here in Figure 6.7, we put the *key* commands used to develop the shell script file *"shape.sh"* for shaping the bandwidth:

```
/sbin/tc class add dev eth0  parent 1: classid 1:1 htb rate "$value" "kbit" ceil "$value" "kbit"  (1)
/sbin/tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip dst 10.0.3.31/24 flowid 1:1  (2)
/sbin/tc qdisc del dev eth0  root (3)
```

Figure 6.7: *The key tc-tool commands for bandwidth shaping.*

Where (1) limits the bandwidth rate of the Ethernet interface *eth0* to "`$value`" kbits, (2) applies the shaping on the outbound traffic going to the client and (3) deletes the qdisc being already created for *eth0*.

By following the throughput traces stored at the capacity log file, we execute these commands repeatedly in a way to set the bandwidth rate to the throughput trace at the considered time slot.

Bandwidth shaping performance with continuous traffic (FTP)

To make sure that the throughput emulation using the Linux *tc-tool* and the capacity log file performs well, we conduct some experiments of a progressive file transfer between the client and the server (using File Transfer Protocol (FTP)). We then compare between the throughput log traces and the real throughput measurements obtained through the Linux real time statistics applied on the server's Ethernet interface *eth0* (under the file /sys/class/net/eth0/statistics/tx-bytes).

In the first plot of Figure 6.8, we show a snapshot of the real throughput variations as well as the predicted throughput variations of the capacity log file. According to our results, the real throughput obtained through the *tc-tool* shaping is on average equal to 76.3% of the predicted throughput, i.e., if we multiply the predicted throughput values by a factor of 0.763, hereinafter called the *Shaping-Factor*, we almost obtain the same values as the real throughput.

In the second plot of Figure 6.8, we show a snapshot of the real throughput variations, as well as the variations of the predicted throughput multiplied by the *Shaping-Factor*. Through the plot, we notice a quasi-perfect superposition of the two curves, which implies that our bandwidth shaping performs well with continuous traffic by considering this *Shaping-Factor*.

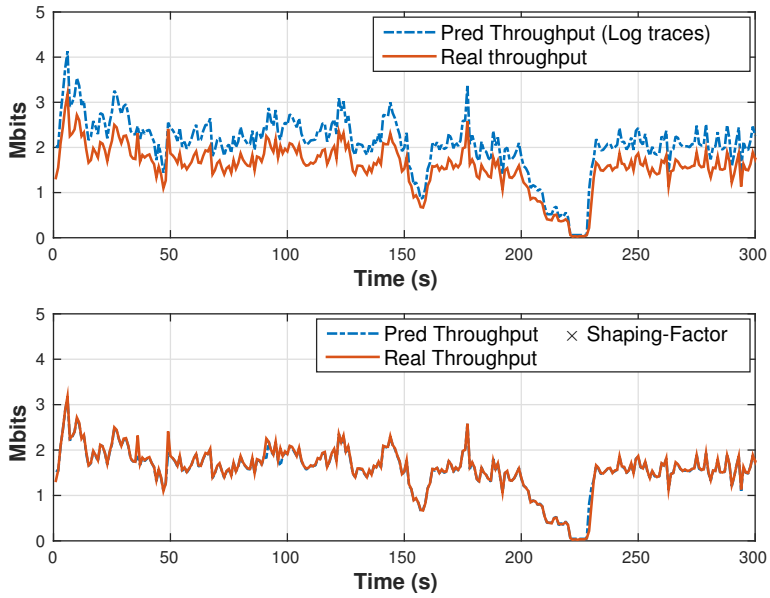


Figure 6.8: *Bandwidth shaping performance with continuous FTP traffic.*

6.3.4.1 Bandwidth shaping performance with DASH traffic

DASH traffic is different than FTP traffic because of the regular HTTP requests sent in between segments download. In this Section, we test the throughput emulation with some DASH traffics to evaluate its performance.

We conduct the experiments with three DASH traffics, each traffic corresponds to a constant bitrate delivery of the *Envivio* video. We set the bitrates to three values: 5.3 Mbps (high), 0.750 Mbps (medium) and 0.480 Mbps (low).

In Figure 6.10, we show four snapshots of the real throughput variations (we plot the predicted throughput multiplied by the *Shaping-Factor* as well). The first snapshot corresponds to an FTP traffic and the last three snapshots correspond to the predefined DASH traffics.

Through the curves, we notice a mismatching between the real throughput variations and the predicted throughput variations with all DASH traffics, unlike with FTP. This mismatching increases as the video bitrate decreases. We explain this by the following interpretation:

During a time slot ($1s$ in our experiements), the lower is the bitrate, the higher is the number of segments the client downloads and the higher is the number of HTTP requests. In contrast, the higher is the bitrate, the lower is the number of segments and the lower is the number of HTTP requests. From the client side, each HTTP request duration is assimilated to a silent time where the dedicated bandwidth is not used. Thus, the real throughput measurement on a given time slot gives a lower value when the bitrate is low and a higher value when the bitrate is high. Which explains the fact that when the bitrate increases, the shaping using DASH traffic approaches the shaping using FTP. In Figure 6.9, we give an illustration of this interpretation.

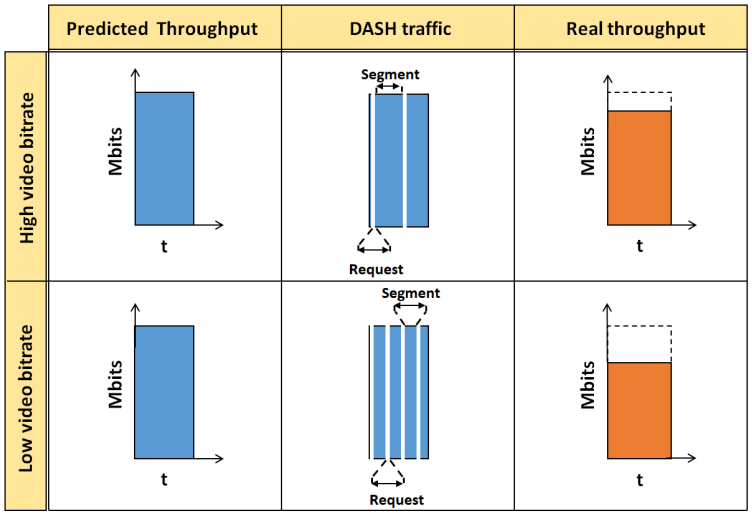


Figure 6.9: *Influence of DASH HTTP requests in the performance of the bandwidth shaping.*

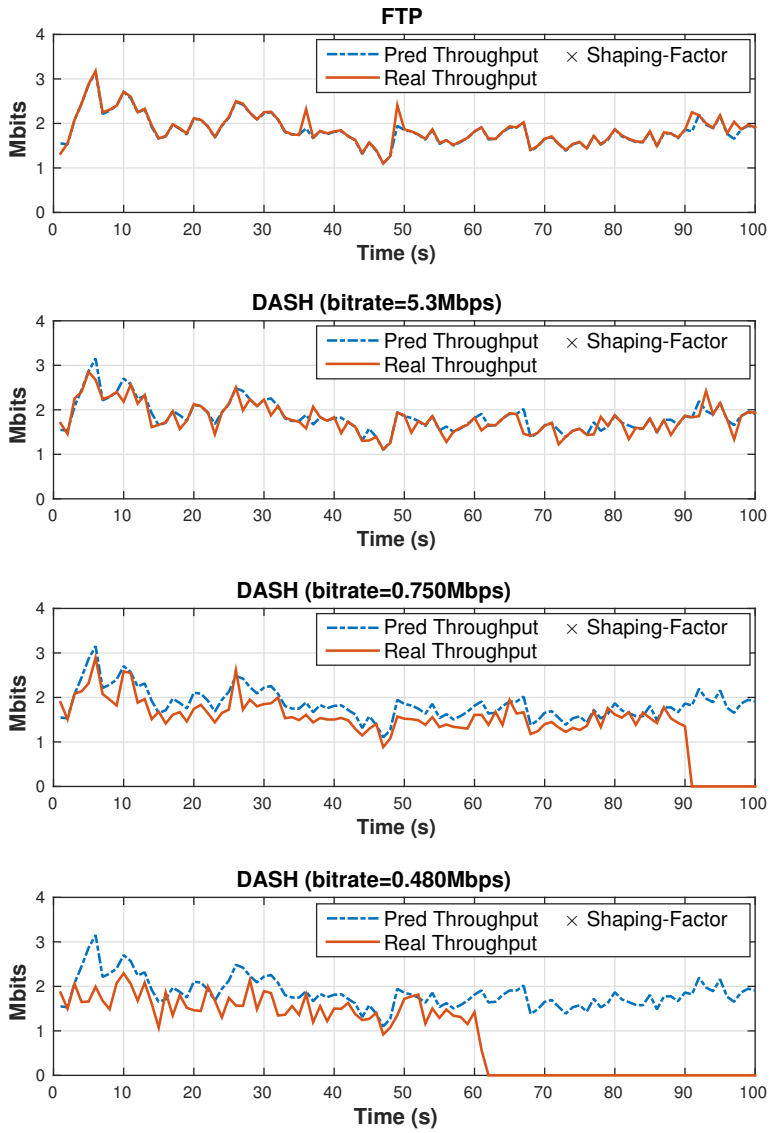


Figure 6.10: *Bandwidth shaping performance with FTP and different constant-bitrate DASH traffics.*

6.3.5 Changes inside NEWCAST

To adapt NEWCAST to realistic video streaming environments, we perform these few changes:

- As explained in the previous analysis, the real throughput of a user running NEWCAST over DASH will not be the same as the throughput "r" modelled by NEWCAST. This is principally due to the existence of the regular HTTP requests in between video segments. In our first design of NEWCAST, we have neglected the durations of all HTTP requests, which is not really valid.

  According to our measurements with the *Wireshark* traffic analyzer, the average duration of an HTTP request is approximately equal to 0.06 seconds. To take it into account in the real implementation of NEWCAST, we assimilate each HTTP

request to a virtual file of size $0.06 \times c(t)$, where $c(t)$ is the predicted capacity multiplied by the *Shaping-Factor* at the request time slot t (see Figure 6.11).

In Figure 6.12, we show the variation of the new throughput "r" modelled by NEWCAST after changes using a small and a high value of π. We plot the predicted throughput multiplied by the *Shaping-Factor* as well.

- When the resulting threshold α^* is high, video stalls with long durations may occur. To shorten the rebuffering stages, we disable the threshold transmission schedule during prefetching by pushing the system to use all the available bandwidth (as if α^* was equal to zero).
- To keep up with marginal throughput prediction errors, we add a *security-marge* of 1 second to the stall constraints, i.e.,

$$u(t) \geq l(t) + \lambda \times 1 \; \forall \; t \; \in [0, T]$$

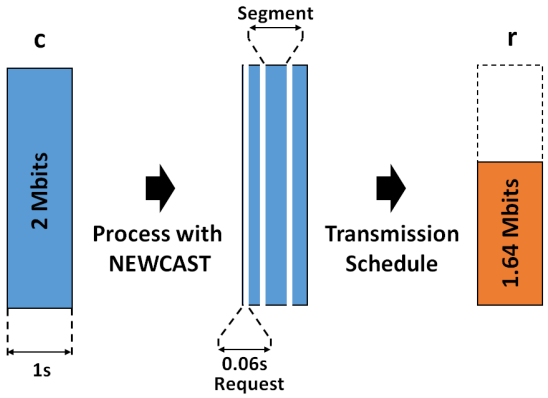


Figure 6.11: *Illustrative example of how "r" is computed with NEWCAST after changes.*

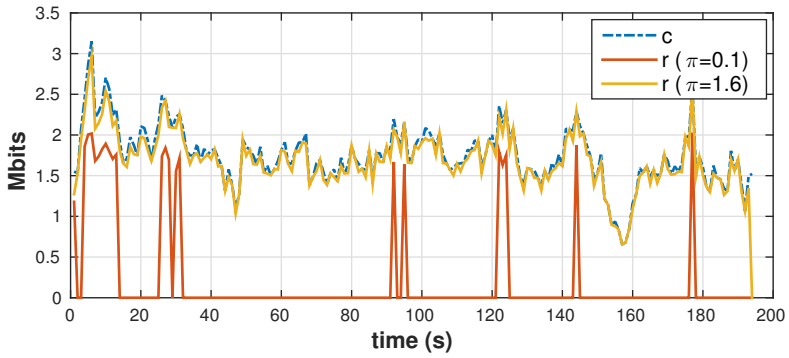


Figure 6.12: *Variation of "r" in function of π after changes inside NEWCAST.*

6.3.6 Required player APIs for interactions

To make the player use the segments quality list set by NEWCAST, we potentially use three main Application Programming Interfaces (APIs) of the *Dashjs* framework:

- The `setAutoSwitchQuality()` API: We use this API to disable the *Dashjs*'s adaptive algorithm and its quality auto-switch mode.
- The `setQualityFor()` API: We use this API to enforce the quality of each coming segment as soon as the previous segment is loaded to the playback buffer.
- The `on()` API: We use this API to detect the moments where video segments are completely loaded to the playback buffer.

6.4 Streaming performance using NEWCAST and its derivative versions

6.4.1 Parameter settings

As mentioned in the previous Sections, we use for the experiments the *Envivio* video file available online in [88]. We set the predicted capacity "c" to one of the throughput traces of the HSDPA dataset [84]. In table 6.2 we put all our parameter settings.

Parameters	values
Window Size	3 min 14 s
Capacity Time Slot	1 s
Video Length	3 min 14 s
Segment Length	2s
Video frame rate	30 fps
Startup threshold	4s
Video bitrates (Mbps)	[0.2 0.3 0.48 0.75 1.2 1.85 2.85 4.3 5.3]
Levels weights	[0.011 0.017 0.27 0.44 0.070 0.107 0.165 0.250 0.308]

Table 6.2: *Parameters of the experiments.*

6.4.2 Graphical interfaces for real time supervision

To make sure that our frameworks work correctly, we develop two graphical interfaces for real time supervision. One for NEWCAST and the other for its derivative versions: STERN, A-STERN, RECAST and STREET.

Figure 6.13 depicts a snapshot of NEWCAST's graphical interface. In this interface, we include some metrics of the real time streaming such as the number of stalls, the average per segment video quality, the number of quality switching (at the left corner) and some metrics of the bandwidth shaping such as the throughput prediction error (at the right corner). To compare between the model set by NEWCAST and the real streaming performance, we include some plots for the throughput variations, the playback buffer evolution and the distribution of video bitrates. As depicted in

Figure 6.13, the first line of the plots is dedicated for the model set by NEWCAST, whereas the second line is dedicated for the real time streaming. The real time transmission mode (threshold based or greedy) is indicated at the bottom of the interface with the running time in second.

In Figure 6.14, we show a snapshot of STERN's graphical interface [2]. In this interface, we include the same metrics and plots as in NEWCAST's interface. However, to survey how the strategies are computed over the successive short-horizons, we include additional metrics at the bottom of the interface such as the total number of downloaded segments, the target number of segments to stream over the current short-horizon and the beginning time of the following short-horizon.

6.4.3 NEWCAST performance under material induced prediction errors

We first evaluate NEWCAST without introducing throughput prediction errors (only material induced errors interfere in the performance of the streaming). We do this to check whether the bandwidth shaping is trustworthy or not.

We repeat the same experiment many times using different values of π (around 100 times per value of π). Results shown by Figure. 6.15 depict a high instability of the system performance when the value of π is small (between 0.1 and 0.7), i.e., when the threshold α_π is high. This instability has provoked high numbers of video stalls. However, when the value of π is high (between 0.8 and 1.6), we notice more stability in the performance.

Through the plots, we also observe a close similarity between the system performance and what was modeled by NEWCAST: A difference of 5.2% in the system utilization cost and a difference of 0.53% in the average number of video stalls.

Although we repeat the same experiment for each value of π, we observe a strict positive variance of the system utilization cost and of the average number of stalls . We potentially link this to the casual errors of the bandwidth shaping and the variation of the HTTP requests' durations.

Overall, these results offer hope that, under high values of π, the exploitation of NEWCAST in real environments becomes feasible, unless an accurate throughput prediction is available. Under low values of π, however, the quality of the streaming risks to be degraded since the system becomes sensitive to the tiniest prediction error.

In the next Section, we restrict our studies to the cases where π is higher than 0.8

[2] the same interface is used for A-STERN, RECAST and STREET

Figure 6.13: *Snapshot of NEWCAST's graphical interface for real time supervision.*

Figure 6.14: *Snapshot of STERN's graphical interface for real time supervision.*

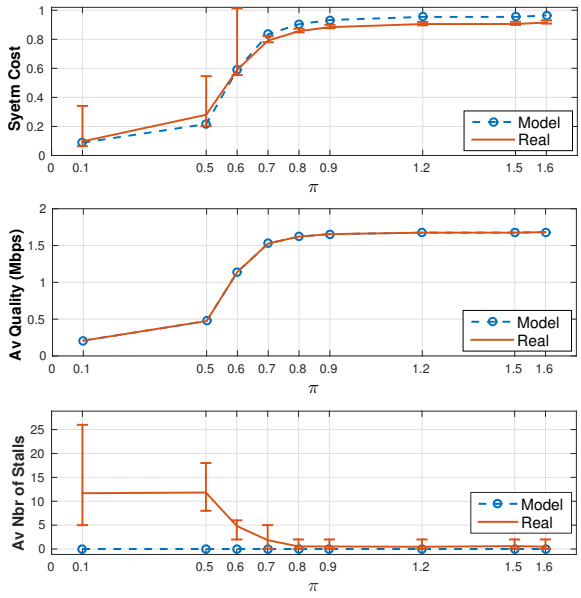


Figure 6.15: *Experiments results: NEWCAST performance without introduced prediction errors.*

6.4.4 NEWCAST and its derivative versions performance under introduced prediction errors

6.4.4.1 Prediction error model

Unlike in the simulation Sections, each experiment lasts at most 5 minutes (by considering eventual video stalls). Thus, conducting a lot of experiments per each value of π would be very time consuming, one of the reasons for which we do not adopt our previous prediction error model defined in 5.2.2; the latter model is random-based and acquires extensive experiments to compute the average system performance.

One more reason for which we do not adopt our previous error model, is that when the bitrate is low, the system does not exhibit a stable performance. Thus, we need a deterministic throughput variation rather than a random-based variation. To counter each instability of the system, we consider a same throughput traces for all the experiments. We mainly distinguish between two types of throughput estimation: Throughput over-estimation with a constant over-estimation marge of 300 Kbits per time slot and throughput under-estimation with a constant under-estimation marge of 300 Kbits per time slot. Which respectively translates:

$$\tilde{c}(t) = c_{real}(t) + 300\ Kbits\ ,\ \forall t$$

$$\tilde{c}(t) = c_{real}(t) - 300\ Kbits\ ,\ \forall t$$

6.4.4.2 Case of overestimated throughput

Here, we set $\pi_{NEWCAST}$ to 0.8 since in the previous study (with no introduced prediction errors), the system showed a quasi-stable performance for π higher than 0.8. For STERN, A-STERN, RECAST and STREET we set the value of π to be equal to 0.8 then to be higher than 0.8.

In Figure 6.16, we show the performance of each algorithm in terms of real system utilization cost, real average quality, real number of switching and real number of video stalls. A noteworthy observation that arises from these results is that NEWCAST ensures the highest number of video stalls ($\simeq 13$) by comparison to the other algorithms, and that the number of stalls induced by each short-term algorithm decreases as the value of π increases. However, this comes at the expense of other performance metrics. Here are some characteristics of each algorithm:

- **NEWCAST**: NEWCAST is the leader of all its derivative versions. It gives the highest average quality (+), the lowest quality switching number (+) but the highest number of stalls (-).
- **STERN**: STERN succeeds at reducing the average number of stalls (+), but in return, it decreases both of the average quality and the system cost (-) and increases the number of quality switching (-). As π increases, STERN gets closer to the performance of NEWCAST in terms of the system cost and the average quality (+) but it brings about a far higher number of quality switching ($\simeq 16$) (-).
- **A-STERN**: A-STERN performs closely to STERN mainly when π is small, but it seems to be slightly greedier when π is high; it increases both of the average quality and the system cost (+), but gives in return a higher number of quality switching ($\simeq 18$) (-).
- **RECAST**: RECAST is the best in terms of ensuring the lowest number of quality switching (+), but this comes at the expense of the average quality (-), which gives a low system cost (+). Compared to the other short-term algorithms, it brings about the highest number of stalls (-) and seems to be the second least sensitive to the variation of π (-).
- **STREET**: We proposed STREET to potentially follow the same strategy as NEWCAST in terms of the bitrate distribution. Unlike in the simulation part, when the short-term predicted throughput is insufficient to stream the quality set by NEWCAST, STREET uses all the available throughput and simply runs AWARE without setting a threshold for the transmission schedule[3], which makes it insensitive to the value of π. Accordingly, it succeeds at giving the closest performance to NEWCAST in terms of the system cost and the average quality (+), with even a far lower number of stalls ($\simeq 1$) (+), but it gives in return a very high number of stalls ($\simeq 18$) (-).

In Figure 6.17, we show some snapshots of video bitrate distribution with all the proposed algorithms. Mainly we show how the average quality and the number

[3]Otherwise, STREET will be similar to STERN since the throughput is all the time overestimated

of quality switching increase when the value of π becomes higher with STERN, A-STERN and RECAST. By comparison with these algorithms, we show how STREET is the best one in achieving the closest bitrate distribution to that of NEWCAST.

To make the performance analysis of STERN, A-STERN, RECAST and STREET clearer, and to obviously distinguish their broughts from their drawbacks, we put in Figure 6.20 a colored map in which we include the percentage of gain/loss of each metric by comparison with NEWCAST. Each green color is interpreted as a brought (good), and each red color is interpreted as a drawback (bad) from the point of view operator and/or client.

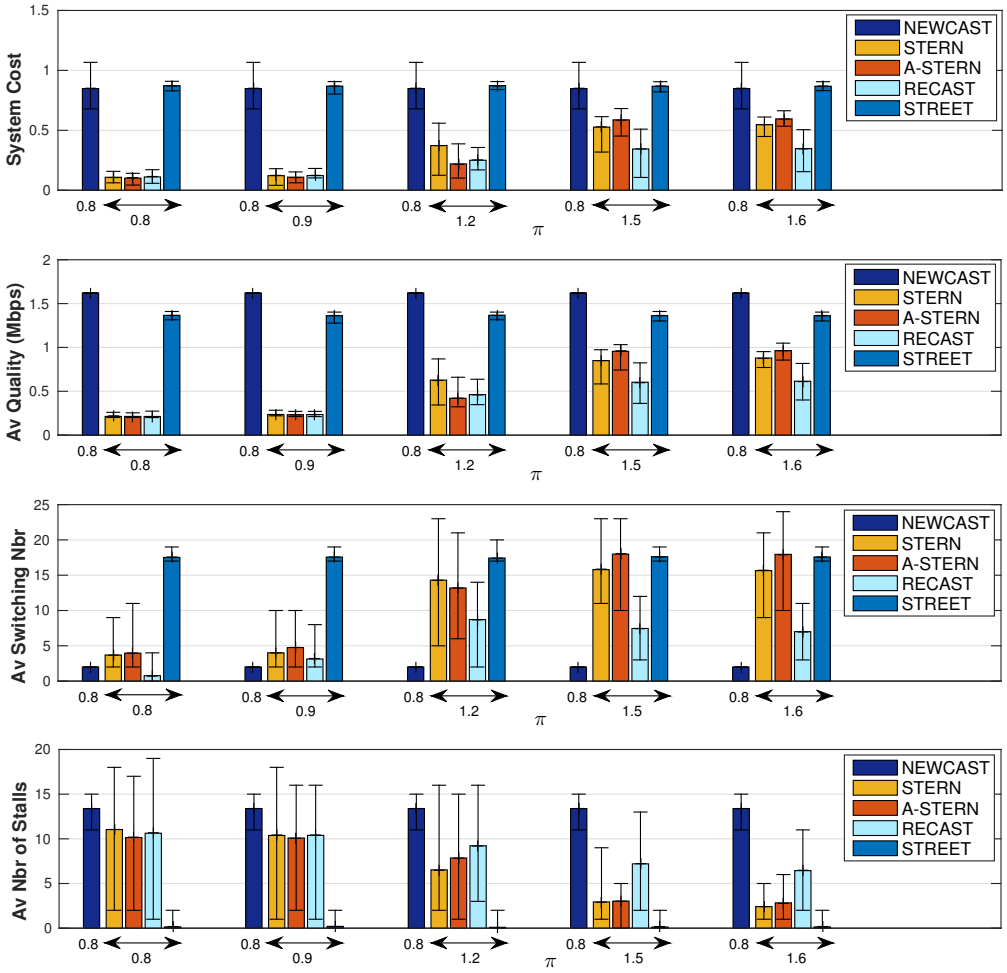


Figure 6.16: *Experiments results: Global performance of the streaming using NEWCAST and its derivative versions under overestimated throughput.*

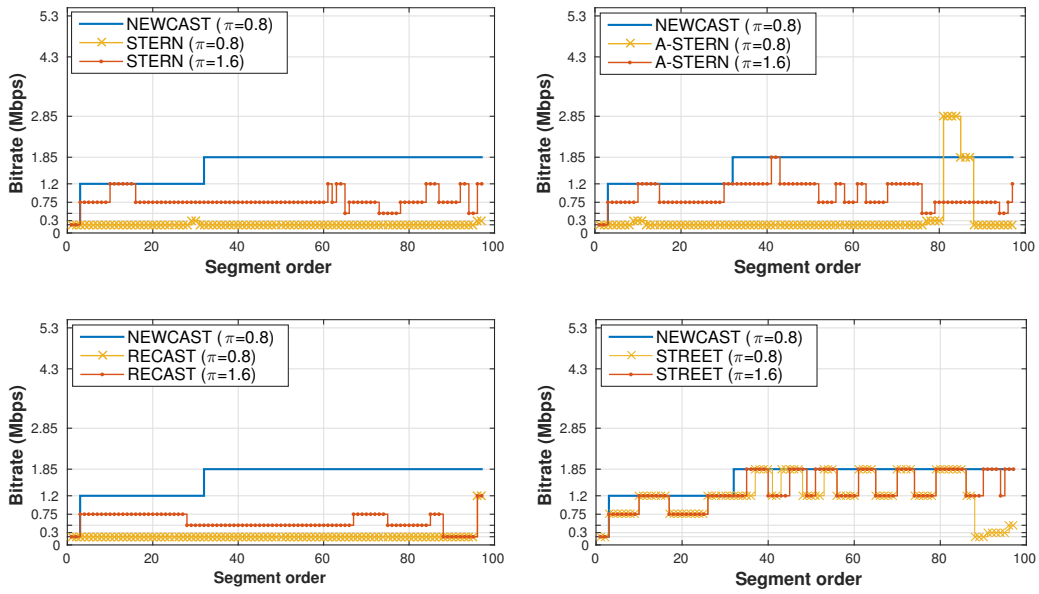


Figure 6.17: *Experiments results: Snapshots of video bitrate distribution using NEWCAST and its derivative versions under overestimated throughput.*

6.4.4.3 Case of underestimated throughput

Here, we set the values of π as in the previous case. Since the number of stalls is evidently equal to zero, we do not include it in the analysis.

In Figure 6.18, we show the performance of all the algorithms in terms of the system utilization cost, the average quality and the average number of quality switching. Here are some important observations we can make according to the results:

First, all the short-term algorithms reduce the average video quality compared to NEWCAST at the exception of A-STERN, which is the only algorithm that reacts to the throughput under-estimation by increasing the average quality (+) and the number of switching (-). Second, NEWCAST is the best one in achieving a global good performance (+). When π is high, STERN performs very close to NEWCAST in terms of average quality and system cost (+), but gives a very high number of quality switching (-). RECAST condemns the video average quality (-) as it follows a constant bitrate strategy per short-term horizon. And finally, STREET is the best one in following a global performance close to that of NEWCAST (+).

As in the previous case, we show in Figure 6.19 some snapshots of bitrate distribution for all the algorithms. We hold the same analysis as with over-estimated throughput.

For more clarity, we put in Figure 6.21 a colored map in which we highlight the broughts and the drawbacks of each algorithm by comparison with NEWCAST.

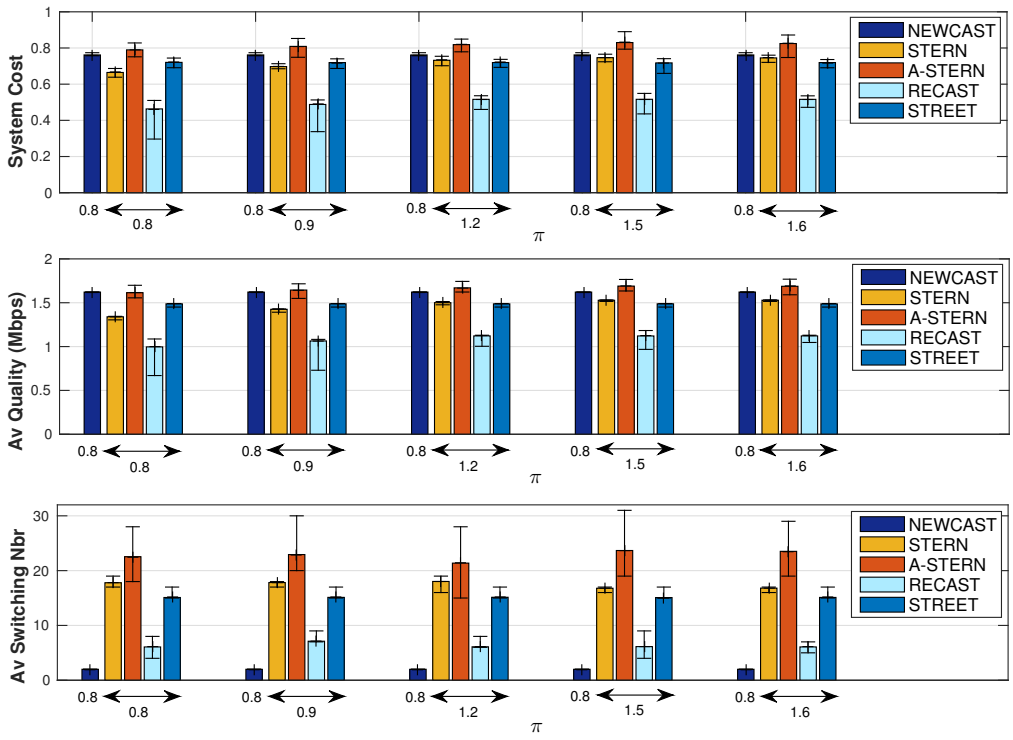


Figure 6.18: *Experiments results: Global performance of the streaming using NEWCAST and its derivative versions, under underestimated throughput.*

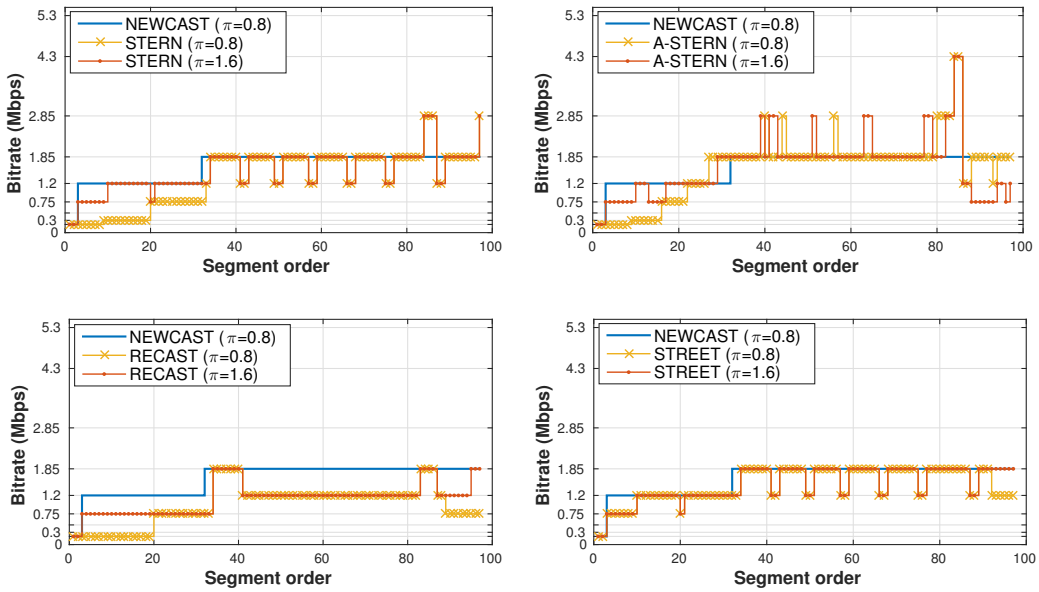
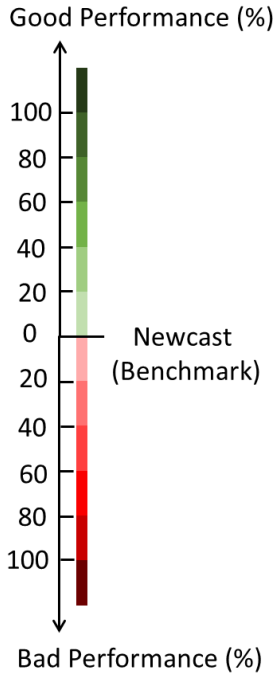


Figure 6.19: *Experiments results: Snapshots of video bitrate distribution using NEWCAST and its derivative versions under underestimated throughput.*



	System Cost	Av Nbr of Stalls	Av Video Quality	Av Nbr of Switches
STERN	Decrease of 87%	Decrease of 18%	Decrease of 87%	Increase of 85%
A-STERN	Decrease of 88%	Decrease of 24%	Decrease of 87%	Increase of 98%
RECAST	Decrease of 87%	Decrease of 21%	Decrease of 87%	Decrease of 61%
STREET	Increase of 4%	Decrease of 99%	Decrease of 15%	Increase of 776%

Overestimation

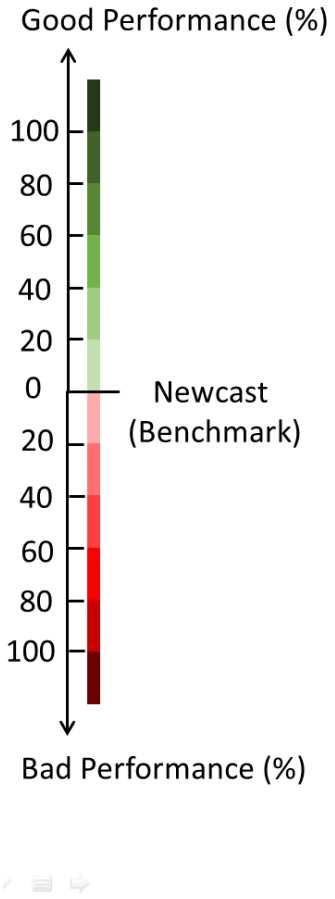
$(\pi_{NEWCAST} = 0.8,\ \pi_{STERN} = \pi_{A-STERN} = \pi_{RECAST} = \pi_{STREET} = 0.8)$

	System Cost	Av Nbr of Stalls	Av Video Quality	Av Nbr of Switches
STERN	Decrease of 35%	Decrease of 82%	Decrease of 46%	Increase of 683%
A-STERN	Decrease of 29%	Decrease of 97%	Decrease of 41%	Increase of 798%
RECAST	Decrease of 58%	Decrease of 52%	Decrease of 62%	Increase of 250%
STREET	Increase of 4%	Decrease of 99%	Decrease of 16%	Increase of 780%

Overestimation

$(\pi_{NEWCAST} = 0.8,\ \pi_{STERN} = \pi_{A-STERN} = \pi_{RECAST} = \pi_{STREET} = 1.6)$

Figure 6.20: *Colored map of NEWCAST's derivative versions' broughts and drawbacks by comparison with NEWCAST, under underestimated throughput.*



	System Cost	Av Video Quality	Av Nbr of Switches
STERN	Decrease of 13%	Decrease of 17%	Increase of 790%
A-STERN	Increase of 4%	Decrease of 1%	Increase of 1027%
RECAST	Decrease of 39%	Decrease of 38%	Increase of 204%
STREET	Decrease of 5%	Decrease of 9%	Increase of 655%

Underestimation

$(\pi_{NEWCAST} = 0.8,\ \pi_{STERN} = \pi_{A-STERN} = \pi_{RECAST} = \pi_{STREET} = 0.8)$

	System Cost	Av Video Quality	Av Nbr of Switches
STERN	Decrease of 1%	Decrease of 5%	Increase of 741%
A-STERN	Increase of 8%	Increase of 4%	Increase of 1076%
RECAST	Decrease of 32%	Decrease of 30%	Increase of 204%
STREET	Decrease of 5%	Decrease of 8%	Increase of 656%

Underestimation

$(\pi_{NEWCAST} = 0.8,\ \pi_{STERN} = \pi_{A-STERN} = \pi_{RECAST} = \pi_{STREET} = 1.6)$

Figure 6.21: *Colored map of NEWCAST's derivative versions' broughts and drawbacks by comparison with NEWCAST under overestimated throughput.*

6.5 Conclusion

In this Chapter, we have discussed step by step the implementation of NEWCAST and its derivative versions in a realistic video streaming environment. For lack of material resources to predict the future throughput of commonly used mobile devices, we have proceeded with the throughput emulation by using the Linux *tc-tool* to shape the bandwidth between the client and the server.

Our experimental results have shown the efficiency of the four proposed algorithms STERN, A-STERN, RECAST and STREET in terms of reducing the number of stalls when the throughput is overestimated. Nevertheless, by comparing with the simulation results, all the short-term versions induce higher numbers of quality switching.

The evaluation of our algorithms has been objectively made in terms of our predefined QoE metrics and the system cost. Thus, we cannot judge which algorithm is the best one. A subjective QoE evaluation should then be more judicious. In the next Chapter, we develop a QoE problem in which the users' perceptions on video qualities are a key prominent factor for resolution.

Chapter 7

Exploring the Role of Machine Learning with Context-Aware Adaptive Video Streaming in QoE Enhancement

Contents

7.1 Introduction

The QoE is known to be subjective and context-dependent. Identifying and calculating the factors that affect the QoE is indeed a difficult task. Recently, a lot of effort

has been devoted to estimate the users' QoE in order to improve video delivery. In the literature, most of the QoE-driven optimization schemes that realize tradeoffs among different quality metrics have been addressed under the assumption of homogenous populations. Nevertheless, people perceptions on a given video quality may not be the same, which makes the QoE optimization harder. The study of this chapter aims at taking a step further in order to address this limitation and to meet the users' profiles within heterogeneous populations. To do so, we propose a closed-loop control framework based on the users' (subjective) feedbacks to learn the QoE function and to optimize it at the same time. We suggest for this framework that all the users' future throughputs are known over long-horizon windows.

Our contribution in this chapter is twofold: First, (i) we exploit the knowledge of future throughput variations in order to solve the optimization problem addressed in [52] in a smoother and faster manner based on similar mathematical analysis than in [6]. Second, (ii) we design a closed-loop framework based on client-server interactions to learn the overall users' perceptions and to fittingly optimize the quality of the streaming. The performance of our proposed framework is obtained using Matlab and Ns3 simulations under multi-user scenario.

We organize the chapter as follows: In Section 7.2, we formulate the single-user QoE-optimization problem. Then, in Section 7.3, we discuss the strategy of the optimal solution and propose an heuristic that performs close to the optimal solution. In Section 7.4, we address the multi-user case and propose a closed-loop based framework using neural networks. Finally, in Section 7.5, we evaluate the performance of this framework through some numerical results.

7.2 Single-user QoE problem formulation

7.2.1 The video streaming model

We model a video as a set of S segments (or chunks) of equal durations in second. Each segment is composed of N frames and is stored on the streaming server at different quality representations. Each representation designs a video encoding rate (hereinafter called bitrate). Denote by $b_1, b_2, \ldots, b_L$ the available video bitrates where $b_i < b_j$ for $i < j$. We suppose that all the video frames are played with a deterministic rate λ, e.g., 30 frames per second (fps).

Prior to each segment download, the player indicates to the server the quality needed to stream it. Let $b^{(s)}$ be the bitrate associated to segment s and $\mathcal{B} = \{b^{(1)}, \ldots, b^{(S)}\}$ be the set of bitrates associated to all video segments. We assume that the video playback buffer is big enough to avoid eventual buffer overflow events. We denote by $B(t_k)$ the number of segments that the playback buffer contains at time t_k. At the beginning of the streaming session, a prefetching stage is introduced to avoid future buffer underflows; T_0 seconds of video (corresponding to x_0 segments) have to be completely loaded to the buffer before starting playing the video. When there are no segments in the playback buffer, the video stops and a new prefetching stage is introduced to load again T_0 seconds of video before pursuing the lecture. This event is, hereinafter, referred to as video stall.

In this study, we exploit the knowledge of the user's future throughput over a given horizon window $\mathcal{H}$=$[t_1 \dots t_T]$. Before starting the streaming, we propose to set all the video segments' bitrates to be optimally streamed over that horizon. We denote by $r(t_k)$ the user's estimated throughput at time t_k, $k \in [1 \dots T]$ and by b_{t_k} the video bitrate scheduled to be streamed at that time. Note that b_{t_k} only depends on the throughput variation and the set of segments' bitrates $\mathcal{B}$. From now on, we denote it by $b_{t_k}(\mathcal{B}, r)$.

To model the dynamic of the playback buffer, we define two phases:

- The start-up/rebuffering phase: referred to as **BaW-phase** (for Buffer and Wait), where the media player only downloads the video without playing it.
- The playback phase: referred to as **BaP-phase** (for Buffer and Play), where the player downloads and plays the video at the same time.

Depending on the state of the buffer at each time of observation t_k, we define two variables $S_{BaP}(t_k)$ and $\tau_{BaW}(t_k)$ such that:

1. If the player is on a BaP-phase, $S_{BaP}(t_k)$ defines the time at which that phase has started,
2. If the player is on a BaW-phase, $S_{BaP}(t_k)$ defines the time at which the *next* BaP-phase will start,
3. If the buffer is empty, $\tau_{BaW}(t_k)$ determines the duration of the resulting BaW-phase,
4. If the buffer is not empty, $\tau_{BaW}(t_k)$ is set to zero.

This translates mathematically as

$$S_{BaP}(t_k) = max\{S_{BaP}(t_{k-1}) \ , \ \delta(B(t_k) = 0) \cdot (t_k + \tau_{BaW}(t_k))\}, \tag{7.1}$$

$$\tau_{BaW}(t_k) = \delta(B(t_k) = 0) \cdot \{\tau \ ; \ \sum_{t=t_k}^{t_k+\tau} \frac{\lambda \cdot r(t)}{N \cdot b_t(\mathcal{B}, r)} = x_0\}, \tag{7.2}$$

where

$$\delta(X) \begin{cases} 1 \text{ if } X = \text{is true} \\ \\ 0 \text{ otherwise.} \end{cases}$$

Hence, the dynamic of the playback buffer can be written as

$$B(t_k) = \{B(t_{k-1}) + \frac{\lambda \cdot r(t_k)}{N \cdot b_{t_k}(\mathcal{B}, r)} - \frac{\lambda}{N} \cdot \delta(t_k \geq S_{BaP}(t_k))\}^+, \tag{7.3}$$

where $\{x\}^+ = max\{x, 0\}$ is used to ensure that the playback buffer occupancy cannot be negative.

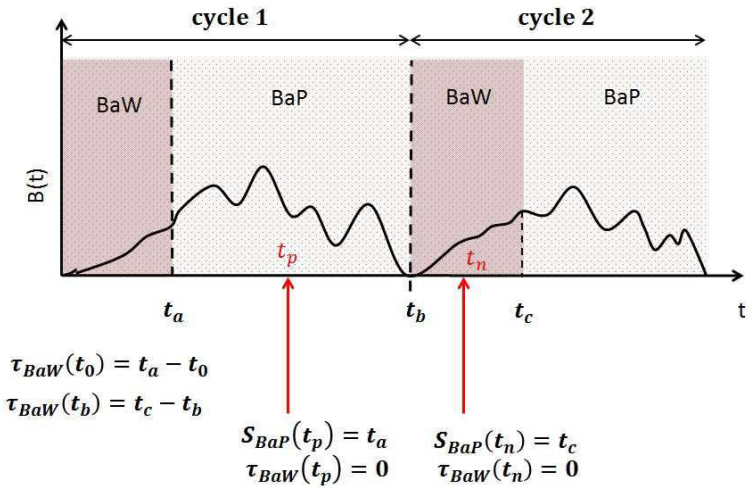


Figure 7.1: *Illustration of the playback buffer BaW-BaP cycles.*

7.2.2 The QoE-optimization problem

The goal of bitrate adaptation in video streaming services is to improve the users' perceived quality of the video. However, it is too challenging to quantitatively define the QoE as it encompasses many complex factors such as the user's mood, the time and the way he watches the video, the video context, etc. In this work, we use five of the most common key QoE metrics to express our objective QoE function.

1. The average video quality (denoted by ϕ_1), which is the average per-segment quality over all segments given by

$$\phi_1(\mathcal{B}) = \frac{1}{S}\sum_{s=1}^{S} b^{(s)}. \tag{7.4}$$

2. The startup delay ratio (denoted by ϕ_2), which is the proportion of time that takes the first BaW-phase before starting the video:

$$\phi_2(\mathcal{B}) = \frac{\tau_{BaW}(t_0)}{T}, \tag{7.5}$$

   where T is the video length in seconds.

3. The average number of video quality switching (denoted by ϕ_3) given by

$$\phi_3(\mathcal{B}) = \frac{1}{S-1}\sum_{s=2}^{S} \delta\{b^{(s)} \neq b^{(s-1)}\}. \tag{7.6}$$

4. The number of video stalls (denoted by ϕ_4) given by

$$\phi_4(\mathcal{B}) = \sum_{k=1}^{T} \delta\{B(t_k) = 0\}. \tag{7.7}$$

5. The rebuffering delay ratio (denoted by ϕ_5), which is the proportion of time that take all the rebuffering events, namely

$$\phi_5(\mathcal{B}) = \frac{1}{T}\sum_{k=1}^{T}\delta\{B(t_k)=0\}\cdot\tau_{BaW}(t_k). \tag{7.8}$$

As the user's preference on each of these QoE metrics may not be the same, we assign to each metric ϕ_i a weighting parameter ω_i to adjust its impact on the global QoE variation. As done in a previous work [52], we model our global QoE as a linear function of the weighted five aforementioned QoE metrics, namely

$$\mathcal{Q}(\mathcal{B}) = \sum_{i=1}^{5}\omega_i\phi_i(\mathcal{B}), \tag{7.9}$$

where $\omega_1 \geq 0$ and $\omega_i \leq 0,\ \forall i \in 2,\ldots,5$.

Let $\mathcal{W} = (\omega_1,\ldots,\omega_5)^\top$ be the vector of weights and $\Phi(\mathcal{B}) = (\phi_1(\mathcal{B}),\ldots,\phi_5(\mathcal{B}))^\top$ be the vector of QoE metrics. If we assume that the user tolerates at most p stalls during the hole session, we end up formulating our single-user QoE optimization problem as follows

$$\max_{\mathcal{B}}\ \mathcal{Q}(\mathcal{B}) = \mathcal{W}^\top\Phi(\mathcal{B}) \tag{7.10}$$

$$\text{s.t.}\ \begin{cases} \sum_{k=0}^{N}\frac{\lambda\cdot r(t_k)}{N\cdot b_{t_k}(\mathcal{B},r)} = S \\ \\ \phi_4(\mathcal{B}) \leq p\ ;\ p\in\mathbb{N}, \end{cases}$$

where the first constraint ensures that the whole video will be streamed by the end of the future horizon.

7.3 Proposed solution for single-user QoE optimization

The QoE optimization problem defined in (7.10) is a combinatorial problem with a very high complexity (NP hard). In [52], authors addressed a similar problem, but they assumed an inaccurate throughput estimation, which justifies their choice to adopt the Model Predictive Control (MPC) to solve their QoE optimization problem. In our study, we characterize an important property of the optimal strategy and propose for resolution an heuristic algorithm that performs close to the optimal approach.

7.3.1 property of optimal solution: Ascending bitrate strategy per BaW-BaP cycle

Definition 3. *A bitrate strategy is said to be ascending per BaW-BaP cycle, if the quality level of the segments increases during each BaW-BaP cycle of the streaming session.*

Proposition 3. *Assume that there exists a solution $\mathcal{B}$ that satisfies the constraints in (7.10). Then, there exists an ascending bitrate per BaW-BaP cycle solution $\mathcal{B}_{as}$ that optimizes the problem in (7.10).*

We put the details of the proof in 8.2.

7.3.2 Algorithm for optimal solution

In this section, we describe the main steps for building an optimal solution of at most p stalls during the streaming session. The key idea of our algorithm is *stall enforcement*: As we assume knowing the future throughput, we are able to enforce video stalls at any moments of the streaming session. Once we locate the stalls' positions (at the level of witch segment each stall will happen), we divide the session into multiple BaW-BaP cycles then we look for the optimal ascending bitrate strategy over each cycle. We obtain the optimal number of stalls through an exhaustive research. We start computing the optimal strategy with zero stalls, then with one stall up to p stalls. The distribution of stalls is also obtained through an exhaustive research. Here are the steps to obtain an optimal ascending bitrate strategy over one BaW-BaP cycle:

1. Find all the possible ascending bitrate combinations of the BaW-phase that allow to build an ascending bitrate strategy over the hole BaW-BaP cycle (step A and B in Figure 7.2).
2. For each BaW-phase combination, find all the possible ascending strategies that satisfy the constraints of (7.10) (steps 1, 2 and 3 in Figure 7.2).
3. For each strategy, compute the resulting QoE metrics then apply the vector of weights $\mathcal{W}$ to find the best solution.

In order to find *all* the possible ascending bitrate strategies, we can use a tree of choice similar to that described in Figure 4.2.

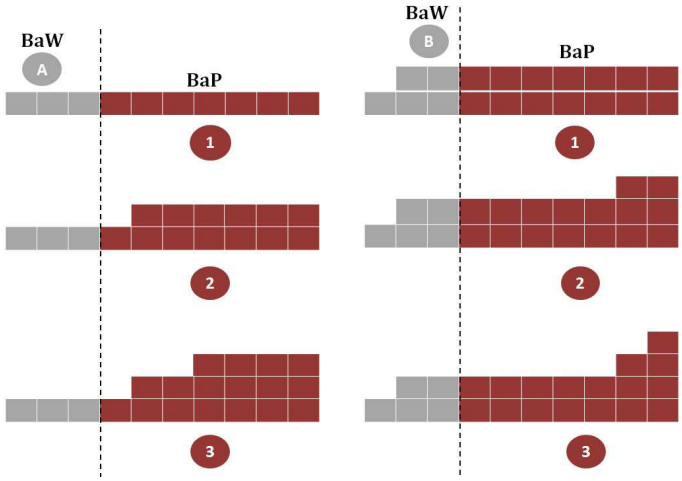


Figure 7.2: *Steps for building an increasing bitrate strategy over a BaW-BaP cycle.*

7.3.3 Heuristic for a sub-optimal solution

Our heuristic differs from the optimal approach in two major points: First, (i) the way we build the ascending bitrate strategy on each BaW-BaP cycle, and second, (ii) the way we set the number of stalls. Here are some explanatory details:

- Once we set the bitrates' combination of the BaW-phase, we progressively increase the bitrates of the BaP-phase starting from the last segment back to the beginning. We stop increasing the bitrate when we reach the point (segment) at the level of which a stall happens if we keep increasing the bitrate (5^{th} segment in step A-2 and 7^{th} segment in step A-3 of Figure 7.2). Given that the number of segments of the BaW-phase is small in general, it does not take much time to find all the possible ascending BaW-phase combinations, which makes our heuristic converge fast (see Algorithm 9).
- Instead of doing an exhaustive research on the number of stalls, we proceed as follows: We start by finding the optimal strategy with zero stalls. Then, we check if the global QoE will increase with one stall enforcement. If it does, we try to enforce a second stall. If not, we stop and return the latest strategy. We keep increasing the number of stalls that way till reaching p stalls or till the QoE function decreases. We put more details in Algorithm 10, where K_i, $i \leq p$ denotes the position of the i^{th} stall.

7.4 Multi-user QoE optimization problem

7.4.1 Problem formulation

In this section, we extend the QoE optimization problem to the multi-user case. We propose to find the vector of weights $\mathcal{W}^*$ that maximizes the QoE among all users. The main objective is to maximize the users' feedbacks on the video delivery using a synthetic QoE dataset. The QoE problem of the multi-user case can be mathematically expressed as

$$\mathcal{W}^* \in \underset{\mathcal{W}}{\operatorname{argmax}} \left\{ \sum_{u=1}^{U} \mathbb{E}_{r_u}\{\mathcal{F}_{r_u}(\mathcal{W})\}, \right\} \tag{7.11}$$

where r_u is the throughput of user u and $\mathcal{F}_{r_u}(\mathcal{W})$ is his feedback on the quality delivered after QoE optimization (7.10) using vector $\mathcal{W}$.

Algorithm 9: MAESTRO: MAximizing qoE with aScending biTRate strategy over One-cycle

Data: $\{b_l\}_{l\leq L},\ c\ ,\ S\ ,\ W$;
$M = [\,]\ , b^{(s)} = b_1\ \ \forall s \in 1, \ldots, x_0$;
for $l_1 = 1 : L$ **do**
  $\{b^{(s)}_{Previous}\}_{1\leq s\leq S} = \{b^{(s)}\}_{1\leq s\leq S}$;
  $b^{(s)} = b_{l_1} \forall\ s \in x_0 + 1, \ldots, S$;
  check if it is possible to stream $\{b^{(s)}\}_{1\leq s\leq S}$ *without stalls* ;
  if *no stall happens* **then**
    $s = x_0$;
    while $s \geq 1$ ***and*** *No stall happens* **do**
      $b^{(s)} = b_{l_1}$;
      $s = s - 1$;
      check if it is possible to stream $\{b^{(s)}\}_{1\leq s\leq S}$ *without stalls* ;
    end
    if *a stall happens* **then**
      $b^{(s)} = b^{(s)}_{Previous}$;
    end
    $I_{l_1,l_1} = \{b^{(s)}\}_{1\leq s\leq S}$;
    Compute $\Phi_{l_1,l_1} = (\phi_1, \phi 2, \phi 3)$;
    for $l_2 = l_1 + 1 : L$ **do**
      $\{b^{(s)}_{Previous}\}_{1\leq s\leq S} = \{b^{(s)}\}_{1\leq s\leq S}$;
      $s = S$;
      while $s > x_0$ ***and*** *No stall happens* **do**
        $b^{(s)} = b_{l_2}$;
        $s = s - 1$;
        check whether it is possible to stream $\{b^{(s)}\}_{1\leq s\leq S}$ *without stalls* ;
      end
      if *a stall happens* **then**
        $b^{(s)} = b^{(s)}_{Previous}$;
      end
      $I_{l_1,l_2} = \{b^{(s)}\}_{1\leq s\leq S}$;
      Compute $\Phi_{l_1,l_2} = (\phi_1, \phi_2, \phi_3)$;
      $M = [M; \Phi_{l_1,l_2}]$;
    end
  end
end
return ($\Phi^* = \underset{M(i,:),i\geq 1}{\textbf{Argmax}}(M[w_1, w_2, w_3]^T)$)

Algorithm 10: CASTLE: asCending bitrAte STrategy over muLti-cycle sEssion

Data: $\{b_s\}_{s\leq L},\ c\ ,\ x_0,\ S,\ p,\ W$;
Bound$_{Inf} = x_0$, Bound$_{Sup} = S$, i=1;
Previous QoE= QoE without stalls;
Previous $\Phi = \Phi$ without stalls;
for $i \leq p$ **do**
  for $K_i \in \{x_0 + 1, .., S - x_0\}$ **do**
    if *there are some stalls already positioned* **then**
      Bound$_{Inf} = \max\{K_j; K_j < K_i\}_{j\leq i}$ or x_0;
      Bound$_{Sup} = \min\{K_j; K_j > K_i\}_{j\leq i}$ or S;
    end
    BaW-BaP$_{PreStall} = \{$Bound$_{Inf}..K_i - 1\}$;
    BaW-BaP$_{PostStall} = \{K_i..$Bound$_{Sup}\}$;
    MAESTRO$([w_1, w_2, w_3]^T$, BaW-BaP$_{PreStall})$;
    MAESTRO$([w_1, w_2, w_5]^T$, BaW-BaP$_{PostStall})$;
    compute $\Phi_{K_i} = (\phi_1, \phi_2, \phi_3, i, \phi_5)$;
    $M = [M; \Phi_{K_i}]$;
  end
  if *max*$\{\{MW^T\}_{j\geq 1}\}$ > *Previous QoE* **then**
    K_i=argmax$_{j\geq 1}\{\{MW^T\}_{j\geq 1}\}$;
    Previous QoE= Resulting QoE;
    Previous $\Phi = \Phi_{K_i}$;
    i=i+1;
  else
    return (Φ^* = Previous Φ)
  end
end
return (Φ^* = Previous Φ)

7.4.2 Practical solution: Closed-loop- based framework with users' feedbacks

Framework design

The multi-user QoE optimization problem requires to solve problem (7.10) for each user $u \in \{1, \ldots, U\}$, knowing the exact value of vector $\mathcal{W}^*$ that meets all the users' preferences. The challenge is then to combine single user QoE optimization with a QoE training mechanism in a closed-loop manner to progressively learn the value of $\mathcal{W}^*$. To do so, we develop two sub-frameworks and make them interact together within a closed-loop based framework: One is for single user QoE optimization (as described in (7.10)) and the other one is for QoE training (see Figure 7.3).

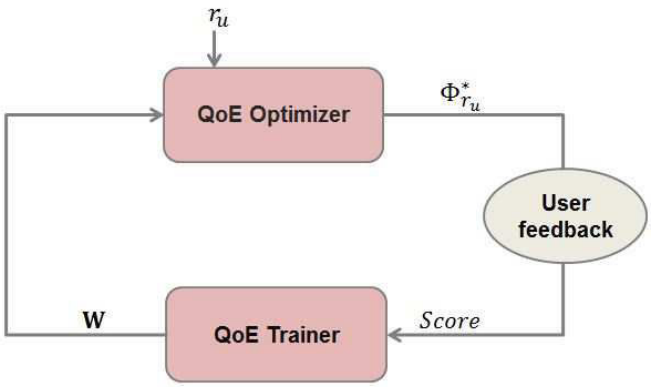


Figure 7.3: *Closed-loop based framework for multi-user QoE-optimization.*

QoE training tool

To compute $\mathcal{W}^*$, we use a simple neural network [89], where the training samples are couples of QoE metrics and user feedback. We define the training dataset as $\{({\Phi^*}_{r_u}, \mathcal{F}_{r_u})\}_{1\leq u\leq U}$, where ${\Phi^*}_{r_u}$ is the vector of QoE metrics delivered by (7.10) under throughput r_u and vector $\mathcal{W}$. $\mathcal{F}_{r_u}$ being the corresponding feedback.

We define the Hypothesis function of the neural network as a linear function

$$h_{\mathcal{W}}(\Phi) = \mathcal{W}^\top \Phi,$$

where Φ is the input vector and $\mathcal{W}$ is the vector of weights to learn (See Figure 7.4).

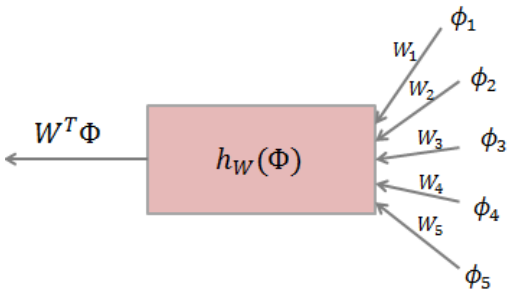


Figure 7.4: *Architecture of the QoE trainer.*

For the learning, we use a mini-batch algorithm based on the gradient descent. The goal of using the gradient descent is to minimize the average error rate between $\mathcal{F}_{r_u}$ and the network output $h_{\mathcal{W}}({\Phi^*}_{r_u})$, $u \in \{1 \cdots U\}$.

Let $\textbf{Loss}(\mathcal{W}, {\Phi^*}_{r_u}, \mathcal{F}_{r_u})$ be the half squared error corresponding to the u^{th} training sample and $\textbf{Loss}(\mathcal{W}, m)$ be the averaged error among m training samples, namely

$$\textbf{Loss}(\mathcal{W}, {\Phi^*}_{r_u}, \mathcal{F}_{r_u}) = \frac{1}{2}|h_{\mathcal{W}}({\Phi^*}_{r_u}) - \mathcal{F}_{r_u}|^2 \tag{7.12}$$

and

$$\textbf{Loss}(\mathcal{W}, m) = \frac{1}{m}\sum_{u=1}^{m} \textbf{Loss}(\mathcal{W}, {\Phi^*}_{r_u}, \mathcal{F}_{r_u}). \tag{7.13}$$

To reduce the average loss, the gradient descent updates the vector of weights $\mathcal{W}$ in a way that it moves oppositely to the direction of the gradient vector ∇**Loss**$(\mathcal{W}, m)$. The algorithm stops when a predefined minimum loss ϵ is reached or when the number of updating steps is above a given threshold T_{rs} (See Algorithm 11).

The partial derivatives of **Loss**$(\mathcal{W}, m)$ in function of the weights ω_k, $k \leq 5$ are given by

$$\frac{\partial \textbf{Loss}(\mathcal{W}, m)}{\partial \omega_k} = \frac{\partial}{\partial \omega_k} \frac{1}{m} \sum_{u=1}^{m} \textbf{Loss}(\mathcal{W}, {\Phi^*}_{r_u}, \mathcal{F}_{r_u}) \tag{7.14}$$

$$= \frac{1}{m} \sum_{u=1}^{m} {\Phi^*}_{r_u,k} (\mathcal{W}^\top {\Phi^*}_{r_u} - \mathcal{F}_{r_u})$$

Algorithm 11: The mini-batch Gradient descent

Input: $\{({\Phi_{r_1}}^*, \mathcal{F}_{r_1}), .., ({\Phi_{r_m}}^*, \mathcal{F}_{r_m})\}$, ϵ, μ, T_{rs}, $[\alpha_{min}, \alpha_{max}]$;

GoodConvergence =0 ; SlowConvergence =0 ; Divergence = 0; Set α in $[\alpha_{min}, \alpha_{max}]$; Set $\mathcal{W}$ very small;

repeat

  repeat

    $\mathcal{W} = \mathcal{W} - \alpha.\nabla$**Loss**$(\mathcal{W}, m)$;

  until *GoodConvergence or* $(\alpha_{max} - \alpha_{min}) \leq \mu$

  if ***Loss***$(\mathcal{W}, m) \leq \epsilon$ **then**

    GoodConvergence =1;

  else

    if ***Loss***$(\mathcal{W}, m)$ *is decreasing* **then**

      SlowConvergence = 1;

      increase (α_{min});

      Set α in $[\alpha_{min}, \alpha_{max}]$;

    else

      Divergence =1;

      decrease(α_{max});

      Set α in $[\alpha_{min}, \alpha_{max}]$;

    end

  end

until ***Loss***$(\mathcal{W}, m) \leq \epsilon$ *or* T_{rs} *iterations are done*

return $\mathcal{W}^* = \mathcal{W}$;

7.5 Numerical results

7.5.1 Simulation environment

To evaluate the performance of our proposed framework we conduct several simulations using Ns3 and Matlab. We use Ns3 mainly to generate standard-compliant correlated throughput samples; we run several simulations of an LTE network by varying the mobility of users each time. In Table 7.1, we put the parameter settings of all Ns3 simulations.

The QoE optimization sub-framework and the QoE trainer were both developed with Matlab. As in real world, we consider users' feedbacks as scores rated from 1 to 5. When a quality $\Phi_r{}^*$ is delivered to a user, we look through the predefined synthetic dataset to find the score it may give.

In the dataset, we have put all the possible values of vector $\Phi_r{}^*$ in a specific priority order verifying

$$|w_{satll}| >> |w_{rebuffering}| >> |w_{average\text{-}quality}| >> |w_{startup}| >> |w_{switching}|.$$

Then, we have grouped them in classes. To each class we have associated a MOS and a specific distribution of scores.

As soon as a vector $\Phi_r{}^*$ is delivered, we determine through the QoE dataset the class to which it belongs. Then, according to that class we randomly generate a score based on its corresponding distribution of scores. In Figure 7.5 we give insight on the form of our QoE dataset. The distribution of MOS in function of the class rank is plotted in Figure 7.6.

Note that the throughput samples used at the level of the QoE optimization sub-framework are randomly selected (according to a Uniform distribution) among 1000 throughput samples generated with Ns3.

All Matlab parameter settings are listed in Table 7.2.

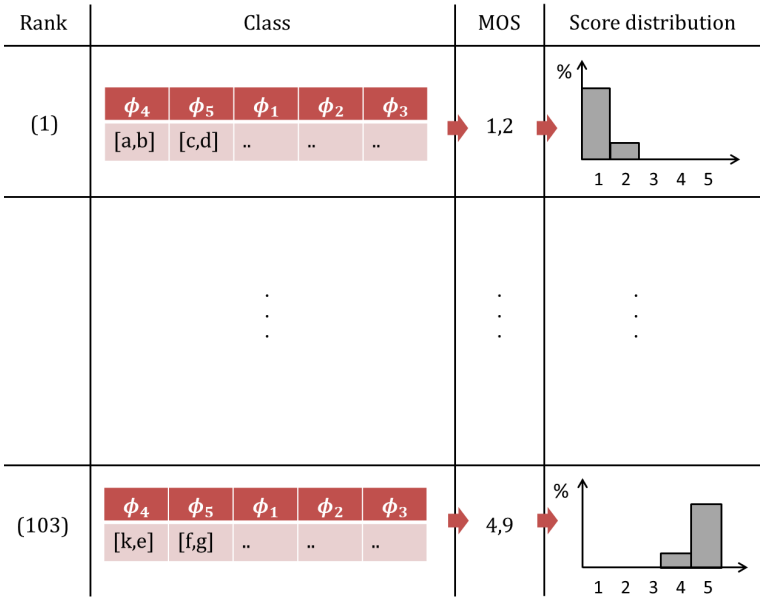


Figure 7.5: *Synthetic-dataset for scores' generation.*

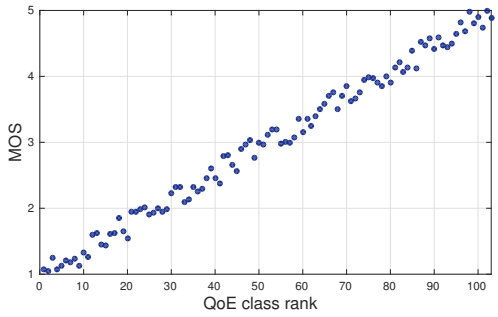


Figure 7.6: *Synthetic MOS as function of the QoE class rank.*

Number of macro cells	1
Number of UEs per cell	10
eNb Tx Power	46 dBm
eNb noise figure	5 dB
UE noise figure	9 dB
Pathloss model	COST 231
MAC scheduler	Proportional fair 50 RBs
Fading model	Pedestrian
Transmission model	MIMO Transmit diversity
Mobility model	RandomWalk2dMobilityModel
Velocity of users	Uniform [5,16] m/s
EPS bearer	NGBR-VIDEO-TCP-DEFAULT
Fading model	Pedestrian
Simulation length	70 s

Table 7.1: *Ns3 simulation setting parameters.*

Window Size	70 s
Throughput Time Slot	1 s
Video Length	30 s
Segment Length	1s
Video frame rate	30 fps
Playback cache	5s
bitrate levels Mbps	[0.4 0.75 1 2.5 4.5]
Maximum number of stalls (p)	1

Table 7.2: *Matlab simulation setting parameters.*

7.5.2 Performance results

According to our simulation results, two noteworthy observations can be made regarding the performance of our closed-loop based framework: First, the learning process ultimately converges to a steady state where the output vector $\mathcal{W}^*$ has a quasi-constant value, and second, this vector $\mathcal{W}^*$ achieves the highest QoE in terms of MOS compared to the other vectors computed throughout the learning process.

In Figure 7.7, we show the evolution of the mean square variation of vector $\mathcal{W}$ during the learning process for different mini-batch sizes (5, 10, 50 and 100 scores). According to the results, this variation tends to zero whatever the mini-batch size, although the decrease is slow in some cases (case of 5 and 50 scores). A fast convergence is however noticed in the case of 10 scores. The difference in the convergence time is actually due to the randomness of the throughput selection and the scores generation.

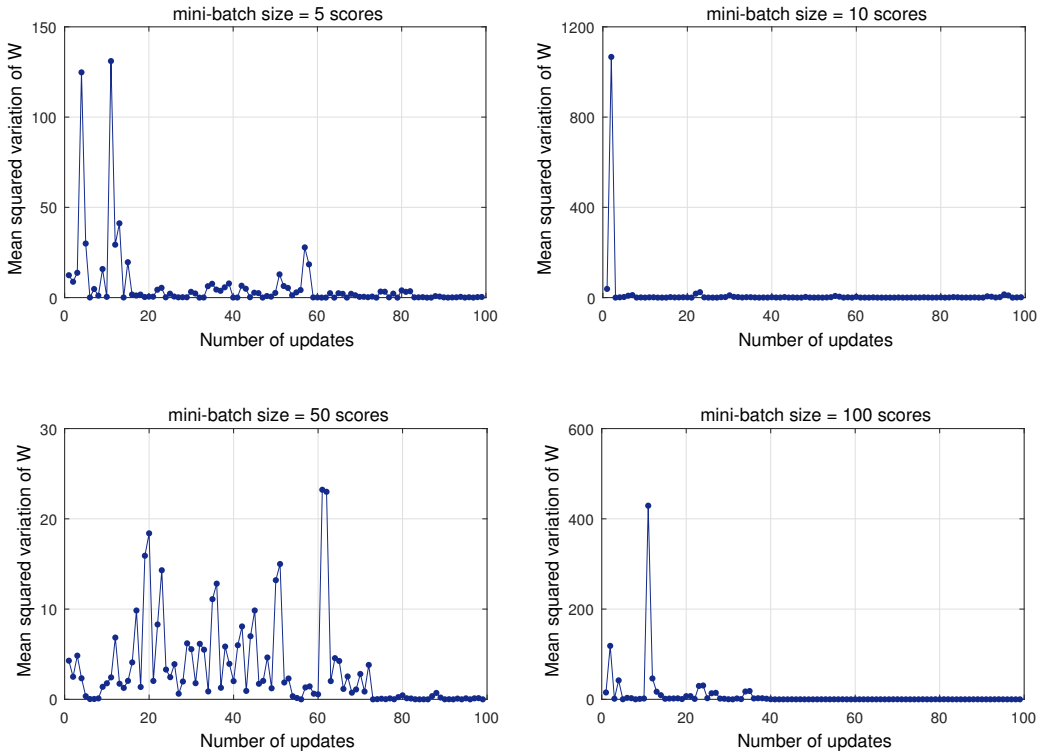


Figure 7.7: *The mean square variation of vector $\mathcal{W}$ during the learning process.*

A comparison between the final outputs $\mathcal{W}^*$ of the four mini-batch sizes shows that they are not exactly the same. To go further in the analysis, we compare between all the updated values of vector $\mathcal{W}$ for each mini-batch size. We perform the comparison by evaluating the MOS given by each vector $\mathcal{W}$ when the QoE-optimization sub-framework is separately run under 1000 randomly selected throughput samples.

Figure 7.8 shows that for the four mini-batch sizes, the MOS experiences some fluctuations with the first values of $\mathcal{W}$. Then, when it tends to the values obtained at the steady state, it converges to the highest MOS value (around 4.8 for the four cases). These results offer hope that the proposed closed-loop based framework can be designed around QoE optimization for video adaptation and delivery in real world environment.

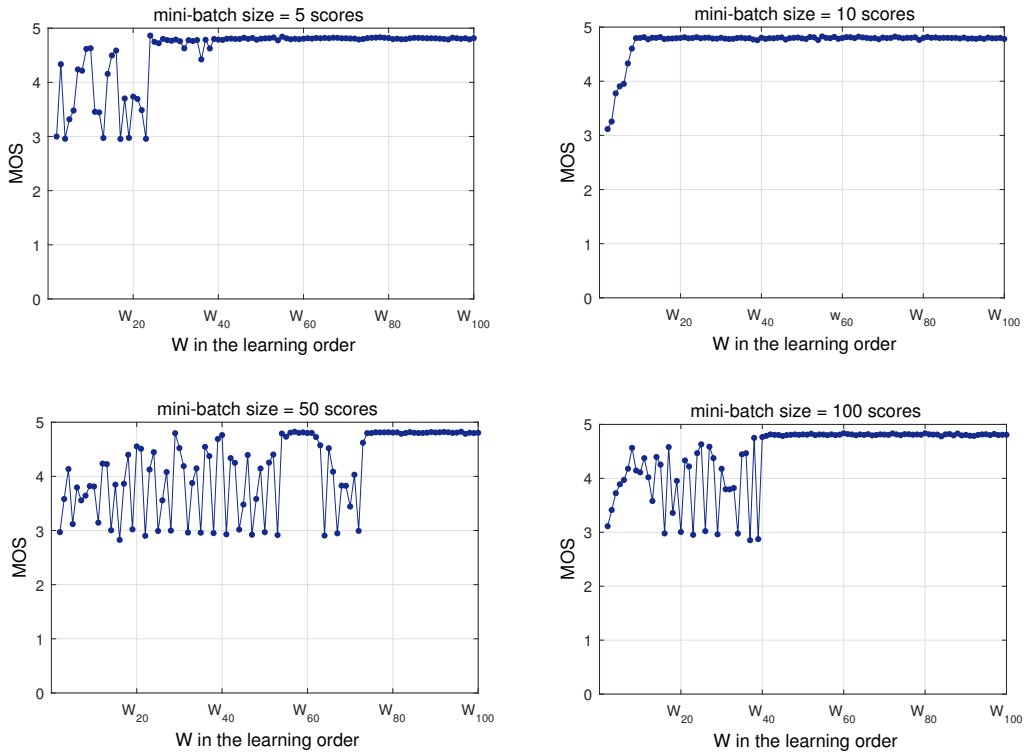


Figure 7.8: *The MOS of the QoE-optimization sub-framework using the updated values of vector $\mathcal{W}$.*

7.6 Conclusion

In this Chapter, we have addressed a QoE optimization problem in which the users' profiles are the key prominent factor for resolution. We have proposed a closed-loop framework based on the users' feedbacks to learn the QoE objective function and to proceed to the QoE optimization. By using a synthetic QoE dataset, we have proven the efficiency of the proposed framework. Indeed, the QoE function learned at the steady state ensures a high quality delivery for the majority of users. These promising results allow us to gain insight on how QoE optimization problems can be handled in heterogeneous populations where the users' preferences and profiles are different.

To go further with this study, and explore the robustness of our closed-loop system, a real QoE dataset needs to be constructed. Thus, real scores on real video streaming sessions should be collected. To optimize the framework performance, the Hypothesis function of the neural network should be carefully reviewed, which is the subject of our future works.

Chapter 8

Conclusions and Perspectives

Contents

8.1 Conculsions

Video streaming services become more and more present in human daily life thanks to the high broadband access afforded by operators and the technological advances in multimedia services.

In this thesis, we have addressed some problems related to video streaming delivery. We have mainly focused on two major properties of the service that common internet services do not have: First, the duration of a video file is known in advance, and second, video data can be buffered (prefetched) at the user's local device before it is displayed without degrading the QoE. We have exploited these two properties with some of the user's contextual information to improve the final QoE. Both real (i.e., non adaptive) streaming and adaptive streaming have been addressed, with a more focus on the DASH standard. To quantify the QoE, we have been based on some key QoE metrics that mostly affect the users such as video stalling, startup delay, rebuffering and bitrate switching in case of adaptive video streaming.

Our works have been based on the assumption that the user's context (channel quality or link capacity) can be easily acquired by the network resource allocator (e.g, eNodeB) or/and the application layer of the user device. Our proposed streaming approaches have mainly focused on the way and time video data should be transmitted to the consumer to afford a minimum acceptable QoE or higher. We have proposed to adapt the video delivery in a reactive manner (Chapter 3) or in a proactive manner (chapter 4, 5, 6 and 7) depending on the user's context.

Through our first study, we have shown that the high mobility of users, usually seen as an obstacle, can be exploited as a profit to improve the overall spectral efficiency and the global QoE in mobile video streaming. We have shown that existing scheduling algorithms for resource allocation can perform better when the users' contexts are acquired and utilized in restricting the access to the scheduler. We have designed a cross-layer to prevent low SINR users from being scheduled by defining a threshold of SINR value. Under the high mobility of users and the assumed road

network topology, we have shown that the number of video stalling has decreased by comparison with the conventional scheduler. The idea behind implementing such a cross-layer is to allow users with high SINR values to download as much data as their channel conditions allow before going across low coverage areas. We have classified our approach as reactive since the scheduling is performed in function of the users' latest channel states.

In our following study, we have proposed proactive streaming approaches by assuming the knowledge of the user's future context. We have shown that the prediction of the long term throughput variations allows to avoid future video stalling and useless video bitrate switching by wisely setting the strategy of the streaming long before the end of the session. In our proposed approaches, we have taken into account the user's QoE as well as the operator's preferences in terms of reducing the network utilization cost. We have shown that the knowledge of the future throughput variations can be profitable for both of the user and the network owner. The key idea of our approaches is to allow downloading as much data as the user's link capacity allows from the beginning times of the streaming session. This is ensured by assigning low video bitrates to the first video segments. As well as time goes forward, the video bitrate is increased in a progressive manner, which returns a low number of quality switching. This idea has been applied under the assumption of a perfect throughput prediction with NEWCAST (Chapter 4), then under the assumption of unperfect throughput prediction with STERN, A-STERN, RECAST and STREET (Chapter 5). We have shown through our simulations that the latter algorithms are more robust to prediction errors than NEWCAST. Which offers hope to deploy them in realistic environments.

The idea of NEWCAST and its derivative algorithms requires the implementation of a cross-layer at the level of the network resource allocator to allow predicting the future throughput variations and to apply the user's scheduling scheme. According to our proof, to reduce the network utilization cost, the scheduling of the user should be of a threshold type. The value of the threshold is determined in function of the preferences of the user and/or the operator.

Our experiments have shown that NEWCAST can interact with real video players as an additional framework to the application layer, but they have revealed its sensitivity to throughput prediction errors. STERN, A-STERN, RECAST and STREET have exhibited, however, a better robustness in terms of video stalling. Which offers hope again to deploy our solutions in realistic environments.

In our last study, we have proposed another proactive streaming approach based on the knowledge of the users' future throughput variations and machine learning. We have defined and solved a QoE optimization problem for heterogenous populations with unknown and different QoE profiles. We have shown that using machine learning in combination with the users' s long term future contexts can be profitable for maximizing the global QoE. In one hand, knowing the future throughput of each user allows maximizing his QoE as shown previously with NEWCAST. In another hand, training the feedbacks of users on previously optimized video qualities allows learning the real QoE profiles of users. We have designed a closed-loop framework based on single-user QoE optimization and feedbacks training, and have shown its high efficiency in terms of convergence and maximizing the QoE.

8.2 Perspectives

The resource allocation with CAMS

Many interesting perspectives can be suggested for our resource allocation mechanism CAMS. First, by always considering a constant speed for all users, the SINR threshold can be analytically determined by defining a mathematical model in which the users' speed will be mapped to the global QoE. We may optimize a QoE objective function under some constraints related to the stalling number or frequency or even duration. Second, by varying the speed of users, we may design a smart cross-layer that defines the SINR threshold per user context. Which may give higher QoE and higher network performance. The cross-layer may even communicate with the application layer to have feedbacks on the user's QoE and to accordingly adjust his SINR threshold. Third, applying CAMS with DASH and reactive adaptive algorithms may degrade the users' QoE since the users will switch perpetually from active to inactive states. A cross-layer optimization taking into account the application layer and the user state may be envisioned to ensure a smooth video bitrate switching and a low number of stalls.

NEWCAST and its derivative versions

We deigned NEWCAST to tradeoff QoE and system utilization cost for a single user. In a real network where different users of different network conditions are competing for the resources, the threshold-based transmission schedule would be a good alternative to fairly serve the users and to ensure a seamless QoE among them. As a perspective, we suggest to adapt our first optimization problem to the multi-user case. We may envisage a same threshold-based schedule for all the streaming users or a threshold-based schedule per user context. The purpose being to reduce the overall system utilization cost and to improve the overall QoE. NEWCAST derivative versions may also be tested in a multi-user environment to explore wether they improve or degrade the users' global QoE with the presence of prediction errors.

In the analysis of NEWCAST performance, we conducted a comparison with throughput-based and buffer-based baseline adaptive algorithms. As a perspective, we propose to do the comparison again with up-to-date algorithms such as BOLA, FESTIVE and PANDA. It would be even more interesting to perform the comparison with real experiments.

The deployment of NEWCAST and its derivative versions in real systems requires the implementation of a cross-layer at the level of the network resource allocator. This cross-layer should allow predicting the user's future throughput variations and applying the threshold transmission scheme. As a future work, we propose to design the interactions between this cross-layer and the network scheduler.

The QoE optimization with the closed-loop based framework

The idea of implementing the closed-loop based framework for video QoE optimization is quite original mainly in heterogenous populations. Nevertheless, the way we implemented it lacks some authenticity as we were based on our own synthetic QoE dataset. As a future work, we propose to collect real data from real video streaming users to obtain more authentic results. We may even fashion the video playbacks in our way to ensure a high number of QoE classes in our QoE dataset. One other challenging point, consists of defining a more suitable Hypothesis function than the identity function for the training sub-framework. We may use deep neural networks for that. As another perspective, we may define an additional resource allocation problem to manage the network resources between simultaneous streaming users. The purpose being to improve the overall QoE by always considering heterogenous populations with different QoE profiles.

Appendix

A.1 Proof of Proposition 3

Let c and r be the network capacity and the user transmission bit-rate on a given interval of time $[0, \epsilon]$. Without loss of generality [1] and for the sake of illustration, we choose an interval of time where c is monotonically decreasing as shown in Figure 1.

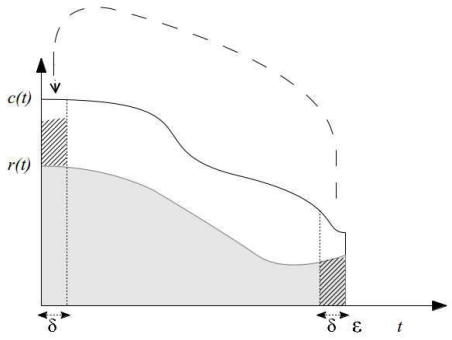


Figure 1: *Sketch of proof of the threshold strategy.*

As we have $r(t) \leq c(t)\ \forall\, t \in [0, \epsilon]$, then $\exists\ (\delta, \beta) \in [0, \frac{\epsilon}{2}] \times [0, 1]$ such that $\forall\, t \in [0, \delta]$

$$c(t) \geq c(t + \epsilon - \delta) \tag{1}$$

and

$$\int_0^{\delta} \frac{r(t) + \beta r(t + \epsilon - \delta)}{c(t)} dt \leq \delta, \tag{2}$$

where inequality (1) derives from the decreasing pace of c, and relation (2) derives from the fact that some data at the end can be transmitted beforehand. (According to the figure above, the hatched area on the right can be entirely shifted to the left, which gives a value of β equal to 1). On the other hand, we have

$$\begin{aligned} \int_0^{\epsilon} \frac{r(t)}{c(t)} dt = \int_0^{\delta} \frac{r(t) + \beta r(t + \epsilon - \delta)}{c(t)} dt + \int_{\delta}^{\epsilon - \delta} \frac{r(t)}{c(t)} dt \\ + \int_{\epsilon - \delta}^{\epsilon} \frac{r(t)}{c(t)} dt - \int_0^{\delta} \frac{\beta r(t + \epsilon - \delta)}{c(t)} dt. \end{aligned} \tag{3}$$

Using inequality (1), we obtain

[1] The proof still holds for a monotonically increasing c.

$$\int_0^{\epsilon} \frac{r(t)}{c(t)}dt \geq \int_0^{\delta} \frac{r(t)+\beta r(t+\epsilon-\delta)}{c(t)}dt + \int_{\delta}^{\epsilon-\delta} \frac{r(t)}{c(t)}dt + \int_{\epsilon-\delta}^{\epsilon} \frac{r(t)}{c(t)}dt) - \int_{\epsilon-\delta}^{\epsilon} \frac{\beta r(t)}{c(t)}dt. \quad (4)$$

Obviously, if

$$\int_0^{\delta} \frac{r(t)+\beta r(t+\epsilon-\delta)}{c(t)}dt = \delta,$$

then all the given capacities in $[0, \delta]$ will be used, i.e., all the white surface in Figure 1 will be filled. In that case, we define a new transmission schedule r' such that

$$r'(t) = \begin{cases} c(t) & \mathsf{t} \in [0, \delta] \\ r(t) & \mathsf{t} \in]\delta, \epsilon - \delta[\\ (1-\beta)r(t) & \mathsf{t} \in [\epsilon - \delta, \epsilon], \end{cases} \quad (5)$$

which gives

$$\int_0^{\epsilon} \frac{r(t)}{c(t)}dt \geq \int_0^{\epsilon} \frac{r'(t)}{c(t)}dt.$$

Otherwise, if

$$\int_0^{\delta} \frac{r(t)+\beta r(t+\epsilon-\delta)}{c(t)}dt < \delta, \quad (6)$$

then β will be equal to 1 since our objective is to shift as much data as possible from the times where the capacity is low to the times where the capacity is high. Therefore, to completely use the highest capacities, we must repeat the same shifting operation on $[0, \epsilon - \delta]$ considering a new transmission function r' verifying

$$\begin{cases} \int_0^{\delta} \frac{r'(t)}{c(t)}dt = \int_0^{\delta} \frac{r(t)+\beta r(t+\epsilon-\delta)}{c(t)}dt \\ r'(t) = r(t) \; \forall \; t \in [\delta, \epsilon - \delta]. \end{cases} \quad (7)$$

In both cases, inequality (4) holds correct, which means that the highest capacities are less expensive than the lowest capacities in terms of network utilization cost if they are used for transmitting data. If we keep repeating the shifting operation on all the future horizon, we end up having all the highest capacities entirely used and all the lowest one unused, which is clearly a threshold transmission schedule as defined in Definition 1.

Now we assume that, knowing c, there exists a feasible solution (r, γ) that satisfies the constraints in (4.5). To perform the data shifting operation on the transmission schedule, three main conditions should be verified:

- The shifted data must have the same video bitrate as the bitrate used in the shifted-to time,
- data shifting should not interrupt a segment transmission schedule,
- data shifting should not violate the stall constraints.

Actually, shifting the data transmission can be either done to the left (earlier) or to the right (later). As we assume a very large playback buffer, sending the video data at earlier times will not cause packets rejection and, thus, will not cause video stalls. Meaning that any data shifting to earlier times of higher capacities will be performed without violating the stall constraints. However, when the higher capacities come later, the data shifting must be checked whether it violates the stall constraints or not. As we only shift the data transmission without changing their corresponding video bitrates, we end up having a new bitrate level strategy $\gamma_{r_{th}}$ that gives the same weighted average quality as given by γ. Thereby, the resulting strategy $(r_{th}, \gamma_{r_{th}})$ outperforms strategy (r, γ), which completes the proof.

A.2 Proof of Proposition 2

Pick a suite of N segments with a non ascending quality levels' order, in a way that it can be streamed without video stalls over the future horizon. Then, according to this quality order, set a threshold-based solution (r_{th}, γ) with threshold α such that, beyond this threshold, the first constraint violation will occur at time $t = s_n$.

Suppose that, under this solution, two bitrate levels b_1 and b_2 will be respectively streamed over $[\tau, \tau + \delta]$ and $[\tau', \tau' + \delta']$ as depicted in Figure 2, such that

$$\tau + \delta < s_n, \;\; \tau' > s_n, \;\; b_1 > b_2$$

and

$$\int_{\tau}^{\tau+\delta} r_{th}(t)dt = \int_{\tau'}^{\tau'+\delta'} r_{th}(t)dt$$

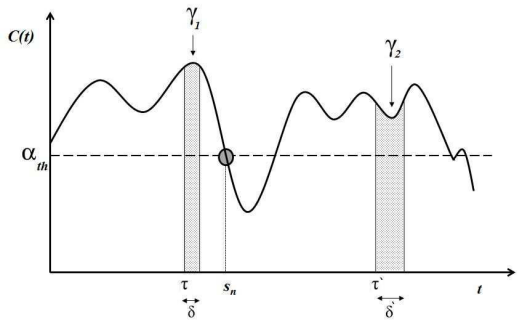


Figure 2: *Sketch of proof of the scending bitrate strategy.*

Let $fr_{th}(t)$ be the *network frame rate* at time t. As we have $b_1 > b_2$, then the number of frames that will be streamed during $[\tau', \tau' + \delta']$ is greater than the number of frames that will be streamed during $[\tau, \tau + \delta]$. Therefore, $\exists\ \beta > 0$ such that

$$\int_{\tau'}^{\tau'+\delta'} fr_{th}(t)dt = \int_{\tau}^{\tau+\delta} fr_{th}(t)dt + \beta. \tag{8}$$

Suppose that we switch between b_1 and b_2 over these two intervals of time. Then, the number of cumulative received frames at s_n will be increased by β. Let u and u' be the cumulative number of arrival frames functions before and after switching the bitrates. Therefore,

$$u'(s_n) = u(s_n) + \beta. \tag{9}$$

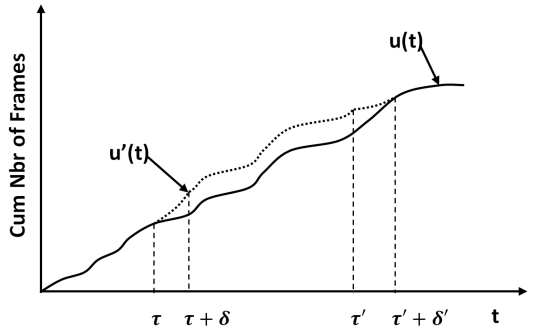


Figure 3: *Impact of bitrates switching on the cumulative number of arrival frames u.*

Actually, if $u'(s_n)$ is large enough and allows increasing the threshold beyond α without violating the stall constraint at $t = s_n$ and later, then the cost function will be reduced. Otherwise, the threshold remains the same without changing the system performance. In fact, as explained in the previous section, streaming the data beforehand will only add more flexibility toward the stall constraints since the buffer is assumed to be very large. We show by the sequel that, even if we switch between the two bitrate levels, the streaming will remain without video stalls since $u' \geq u(t)\ \forall t \in [0, T]$ (see Figure 3).

Let fr'_{th} be the *network frame rate* function after switching. Then, we have

$$fr'_{th}(t) \ > fr_{th}(t)\ \forall t \in [\tau, \tau + \delta[\tag{10}$$

$$fr'_{th}(t) \ < fr_{th}(t)\ \forall t \in [\tau', \tau' + \delta'[\tag{11}$$

$$\int_{\tau}^{\tau+\delta} fr'_{th}(t) - fr_{th}(t)\, dt = \int_{\tau'}^{\tau'+\delta'} fr_{th}(t) - fr'_{th}(t)\, dt = \beta. \tag{12}$$

We further define u' as

$$u'(t) = \begin{cases} u(t) & \text{t} < \tau \\ u(\tau) + \int_{\tau}^{t} fr'_{th}(s)\, ds & \text{t} \in [\tau, \tau + \delta[\\ u(t) + \beta & \text{t} \in [\tau + \delta, \tau'[\\ u(\tau') + \beta + \int_{\tau'}^{t} fr'_{th}(s)\, ds & \text{t} \in [\tau', \tau' + \delta'[\\ u(t) & \text{t} \geq \tau + \delta'. \end{cases} \tag{13}$$

Actually, the cumulative watched frames function l will remain the same as the playback frame rate λ holds the same for all bitrate levels. Now, we see clearly that $\forall$ $t \notin [\tau', \tau' + \delta'[\ ,\ u'(t) \geq u(t)$. However, for $t \in [\tau', \tau' + \delta'[$, we have

$$u'(t) - u(t) = \beta - \int_{\tau'}^{t} fr_{th}(s) - fr'_{th}(s)\, ds, \tag{14}$$

which is positive according to (11) and (12).

To conclude, putting the segments in an ascending bitrates' order may allow a higher transmission threshold which further reduces the cost function without degrading the average quality of the video.

A.3 Proof of Proposition 3

We shall show that for any feasible strategy $\mathcal{B}$ that satisfies the constrains in (7.10), there exists an ascending bitrate per BaW-BaP cycle strategy $\mathcal{B}_{as}$ such that

$$\mathcal{Q}(\mathcal{B}_{as}) \geq \mathcal{Q}(\mathcal{B}).$$

Here, we distinguish two cases: (i) Case where the session is composed of one BaW-BaP cycle, i.e., no stall during the session, and (ii) case where the session is composed of more than one BaW-BaP cycle, i.e., one or more stalls during the session.

- **Case (i) :** Without loss of generality, and for the sake of illustration, we assume that we can stream and play the video in a smooth way under a non-ascending bitrate strategy $\mathcal{B}$. Then, there $\exists\ m \leq n$ such that $b^{(m)} \geq b^{(n)}$. Let t_p and t_q be the requesting times of $b^{(m)}$ and $b^{(n)}$, respectively, as illustrated in Figure 4.

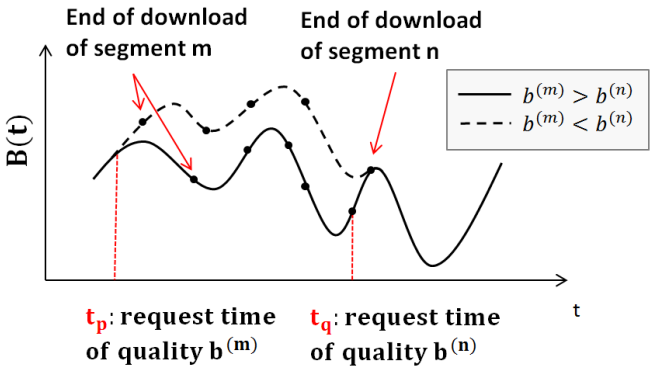


Figure 4: *Impact of bitrate switching on the buffer state evolution.*

If we switch between qualities of segments m and n, then the buffer state will be more relaxed toward the stall constraint since segment m will be streamed in a shorter time and, then, the following segments will be loaded sooner to

the buffer, which will not induce buffer stalls. That said, if we reorder $\mathcal{B}$ in an ascendant way, the video will not experience any stall. Let $\mathcal{B}_{as}$ be the resulting set after reordering $\mathcal{B}$ in an ascending way, then we have

$$\phi_4(\mathcal{B}_{as}) = \phi_4(\mathcal{B}) = 0 \text{ and } \phi_5(\mathcal{B}_{as}) = \phi_5(\mathcal{B}) = 0.$$

As we keep the same selected bitrates in $\mathcal{B}_{as}$ as in $\mathcal{B}$, the average per segment bitrate will not change, which gives

$$\phi_1(\mathcal{B}_{as}) = \phi_1(\mathcal{B}).$$

Since $\mathcal{B}_{as}$ is an ascending strategy, the video session will start with the lowest qualities defined in $\mathcal{B}$. Hence, the startup delay will be reduced when using $\mathcal{B}_{as}$ compared to $\mathcal{B}$. Therefore,

$$\phi_2(\mathcal{B}_{as}) \leq \phi_2(\mathcal{B}).$$

Now, let L be the number of qualities defined in $\mathcal{B}$. Thus, the number of quality switching under strategy $\mathcal{B}$ will be at least equal to L. On the other hand, strategy $\mathcal{B}_{as}$ will experience exactly $L-1$ quality switching since the video bitrate increases during the session. Therefore, we have

$$\phi_3(\mathcal{B}_{as}) \leq \phi_3(\mathcal{B}).$$

All things considered, we have

$$\mathcal{Q}(\mathcal{B}_{as}) \geq \mathcal{Q}(\mathcal{B}).$$

- **Case (ii) :** Here, we assume that, for a given horizon window, we can stream the video under a non-ascending bitrate strategy $\mathcal{B}$ with ϕ_4 stall events over the session ($\phi_4 \geq 1$).

  Undoubtedly, reordering all the segments' bitrates in an ascending way will add more protection to the buffer against the stall constraint, which may reduce the number of stalls ϕ_4. However, this does not mean that the global QoE will increase, because the stalls' durations will change depending on their new occurrence moments, the new variation of the bitrate and the dynamic of the user's throughput. For these reasons, our ascending bitrate strategy will not work per a hole session.

  In an other hand, when a stall happens, the buffer state becomes independent of its previous states prior the stall, which makes all the BaW-BaP cycles independent from each other. Let's write $\mathcal{B} = \{\mathcal{B}_1, \cdots, \mathcal{B}_{\phi_4+1}\}$, where $\mathcal{B}_i$ denotes the set of bitrates used in the i^{th} BaW-BaP cycle. If we apply our previous ascending strategy on each of the $(\phi_4 + 1)$ BaW-BaP cycles, we end up reducing the duration of all the BaW-phases (including the startup and the rebuffering delays) and the global number of quality switching, while maintaining the same number of stalls and the same average quality.

Let $\mathcal{B}_{as} = \{\mathcal{B}_{1_{as}}, \cdots, \mathcal{B}_{\phi_4+1_{as}}\}$, be the set of bitrates derived from $\mathcal{B}$, where $\mathcal{B}_{ias}$ is the ascending bitrate form of $\mathcal{B}_i$, then we have

$$\mathcal{Q}(\mathcal{B}_{as}) \geq \mathcal{Q}(\mathcal{B}).$$

This concludes the proof.

Bibliography

[1] White Paper. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update". In: *Cisco Systems*. 2014.

[2] Majed Haddad et al. "A Survey on YouTube Streaming Service". In: *Valuetools*. Paris, France, 2011.

[3] Ashwin Rao et al. "Network Characteristics of Video Streaming Traffic". In: *ACM CoNEXT* (Dec. 2011).

[4] Stefan Valentin Hatem Abou-zeid and Hossam Hassanein. *Context-Aware Resource Allocation for Media Streaming: Exploiting Mobility and Application-Layer Predictions.* URL: https://pdfs.semanticscholar.org/25b4/8ae1c799b9cdc56e4af23146a6ad4031281f.pdf.

[5] Imen Triki et al. "Context-Aware Mobility Resource Allocation for QoE-Driven Streaming Services". In: *IEEE WCNC*. 2016.

[6] Imen Triki, Rachid El-Azouzi, and Majed Haddad. "NEWCAST: Anticipating Resource Management and QoE Provisioning for Mobile Video Streaming". In: *17th International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM)*. 2016. URL: http://arxiv.org/abs/1512.05705.

[7] Imen Triki, Rachid El Azouzi, and Majed Haddad. "Anticipating Resource Management and QoE for Mobile Video Streaming under Imperfect Prediction". In: *IEEE International Symposium on Multimedia, ISM, San Jose, CA, USA, December 11-13, 2016*. 2016, pp. 93–98.

[8] I. Triki, R. El-Azouzi, Q. Zhu, M. Haddad and Z. Xu. "Learning from Experience: A Dynamic Closed-Loop QoE Optimization for Video Adaptation and Delivery". In: (2017). Available at arxiv.org/abs/1703.01986.

[9] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Radio Resource Control (RRC); Protocol specification (Release 8)". In: 3GPP TS 36.331 V8.0.0 (2007).

[10] *ETSI TS 136 213 V8.3.0 (2008-11) Technical Specification*. URL: http://www.etsi.org/deliver/etsi_ts/136200_136299/136213/08.03.00_60/ts_136213v080300p.pdf.

[11] Ricardo Toguchi Caldeira and Gilberto Gambugge Neto. "Link Adaptation in LTE Systems". In: *Long Term Evolution: 4G and Beyond*. Ed. by Alberto Paradisi et al. Springer International Publishing, 2016. URL: https://doi.org/10.1007/978-3-319-23823-4_6.

[12] Ronak D. Trivedi and Mittal C. Patel. "Comparison of Different Scheduling Algorithm for LTE". In: 2014.

[13] E. Altman, K. Avrachenkov, and A. Garnaev. "Generalized alpha-fair resource allocation in wireless networks". In: 2008.

[14] *Quel est le meilleur format vidéo pour le web ?* URL: https://www.kalyzee.com/quel-meilleur-format-video-web/.

[15] J. Watkinson. *The MPEG Handbook: MPEG-1, MPEG-2, MPEG-4*. Focal Press, 2001. URL: https://books.google.fr/books?id=0gbO-Ebo6W8C.

[16] *Windows Media Server or Web Server ?* URL: https://docs.microsoft.com/en-us/iis/media/windows-media-services/windows-media-server-or-web-server.

[17] *Move Networks*. URL: https://www.crunchbase.com/organization/movenetworks.

[18] *Smooth Streaming*. URL: https://www.iis.net/downloads/microsoft/smooth-streaming.

[19] *HTTP Live Streaming*. URL: https://developer.apple.com/streaming/.

[20] *ADOBE HTTP DYNAMIC STREAMING (HDS) TECHNOLOGY CENTER*. URL: http://www.adobe.com/devnet/hds.html.

[21] *MPEG ratifies its draft standard for DASH*. URL: https://web.archive.org/web/20120820233136/http://mpeg.chiariglione.org/meetings/geneva11-1/geneva_press.htm.

[22] *ISO/IEC 23009-1:2012 Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. URL: https://www.iso.org/standard/57623.html.

[23] *Dash-Industry-Forum*. URL: https://github.com/Dash-Industry-Forum/dash.js/.

[24] A. Beben et al. "ABMA+: Lightweight and Efficient Algorithm for HTTP Adaptive Streaming". In: *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016.

[25] Theodoros Karagkioules et al. "A Comparative Case Study of HTTP Adaptive Streaming Algorithms in Mobile Networks". In: *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2017.

[26] Junchen Jiang, Vyas Sekar, and Hui Zhang. "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE". In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. 2012.

[27] Zhi Li et al. "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale". In: *IEEE Journal on Selected Areas in Communications* (2014), pp. 719–733.

[28] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. "An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP". In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. ACM, 2011.

[29] Te-Yuan Huang et al. "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard". In: *Proceedings of the 2012 Internet Measurement Conference*. ACM, 2012.

[30] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP". In: *IEEE International Conference on Communication, ICC 2015, London, United Kingdom, June 8-12, 2015, Workshop Proceedings*. 2015.

[31] Te-Yuan Huang, Ramesh Johari, and Nick McKeown. "Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming". In: ACM, 2013.

[32] T. Y. Huang et al. "A buffer-based approach to rate adaptation: Evidence from a large video streaming service". In: *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2014.

[33] Luca De Cicco et al. "ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)." In: IEEE, 2013.

[34] Guibin Tian and Yong Liu. "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming". In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. 2012.

[35] Konstantin Miller et al. "Adaptation algorithm for adaptive streaming over HTTP." In: *PV*. IEEE, 2012.

[36] Truong Cong Thang et al. "An evaluation of bitrate adaptation methods for HTTP live streaming". In: *Selected Areas in Communications, IEEE Journal on* 32.4 (2014), pp. 693–705.

[37] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. "An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP". In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. ACM, 2011.

[38] Christopher Müller, Stefan Lederer, and Christian Timmerer. "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments". In: ACM, 2012.

[39] Stefan Lederer, Christopher Müller, and Christian Timmerer. "Dynamic Adaptive Streaming over HTTP Dataset". In: *Proceedings of the 3rd Multimedia Systems Conference*. ACM, 2012.

[40] C. Liu, I. Bouazizi, and M. Gabbouj. "Rate adaptation for adaptive HTTP streaming". In: *ACM Multimedia Syst.* 2011.

[41] Piotr Wisniewski et al. "On delimiting video rebuffering for stream-switching adaptive applications". In: *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*. IEEE, 2015.

[42] Kjell Brunnström et al. *Qualinet White Paper on Definitions of Quality of Experience*. Mar. 2013. URL: https://hal.archives-ouvertes.fr/hal-00977812.

[43] *Amendment 5: New definitions for inclusion in Recommendation ITU-T P.10/G.100*. URL: https://www.itu.int/rec/T-REC-P.Imp10-201601-I/en.

[44] *Definitions of terms related to quality of service*. URL: https://www.itu.int/rec/T-REC-E.800-200809-I/en.

[45] *Methods for subjective determination of transmission quality*. URL: https://www.itu.int/rec/T-REC-P.800-199608-I/en.

[46] Karsten Weide. *Streaming video quality and user engagement*. 2011. URL: http://www.frenchweb.fr/wp-content/uploads/2012/04/WP_IDC_Streaming.pdf.

[47] *Subjective video quality assessment methods for multimedia applications*. URL: https://www.itu.int/rec/T-REC-P.910-200804-I/en.

[48] V. Menkovski, G. Exarchakos, and A. Liotta. "Machine Learning Approach for Quality of Experience Aware Networks". In: *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*. 2010.

[49] M. S. Mushtaq, B. Augustin, and A. Mellouk. "Empirical study based on machine learning approach to assess the QoS/QoE correlation". In: *Networks and Optical Communications (NOC), 2012 17th European Conference on*. 2012.

[50] Miguel García Pineda, Santiago Felici-Castell, and Jaume Segura-Garcia. "Using Factor Analysis Techniques to Find Out Objective Video Quality Metrics for Live Video Streaming over Cloud Mobile Media Services". In: *Network Protocols & Algorithms* (2016).

[51] Athula Balachandran et al. "A Quest for an Internet Video Quality-of-experience Metric". In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012.

[52] Xiaoqi Yin et al. "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP". In: *SIGCOMM Comput. Commun. Rev.* (2015), pp. 325–338.

[53] J. De Vriendt, D. De Vleeschauwer, and D. C. Robinson. "QoE model for video delivered over an LTE network using HTTP adaptive streaming". In: *Bell Labs Technical Journal* (2014).

[54] Yun Shen et al. "A method of QoE evaluation for adaptive streaming based on bitrate distribution". In: *IEEE International Conference on Communications, ICC 2014, Sydney, Australia, June 10-14, 2014, Workshops Proceedings*. 2014.

[55] S. B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques". In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. IOS Press, 2007. URL: http://dl.acm.org/citation.cfm?id=1566770.1566773.

[56] Jennifer G. Dy and Carla E. Brodley. "Feature Selection for Unsupervised Learning". In: *J. Mach. Learn. Res.* (2004).

[57] Xiaojin Zhu et al. *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers, 2009.

[58] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 2008.

[59] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[60] Harry Zhang. In: *Proceedings of the Seventeenth International Florida Artificial Intelligence publisher = AAAI Press, title = The Optimality of Naive Bayes, year = 2004*.

[61] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 2007.

[62] J. A. Hartigan and M. A. Wong. "A k-means clustering algorithm". In: *JSTOR: Applied Statistics* (1979).

[63] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[64] *Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-1/.

[65] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Netw.* (1989). URL: http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[66] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., 1999.

[67] *Training Hidden Units: The Generalized Delta Rule.* URL: https://web.stanford.edu/group/pdplab/originalpdphandbook/Chapter$\%$205.pdf.

[68] Nick Littlestone. "From On-line to Batch Learning". In: *Proceedings of the Second Annual Workshop on Computational Learning Theory*. Morgan Kaufmann Publishers Inc., 1989.

[69] Mu Li et al. "Efficient Mini-batch Training for Stochastic Optimization". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2014.

[70] P. Gupta and P.R. Kumar. "The capacity of wireless networks". In: *Information Theory, IEEE Transactions on* 46.2 (2000), pp. 388–404.

[71] Matthias Grossglauser and D.N.C. Tse. "Mobility increases the capacity of ad hoc wireless networks". In: *Networking, IEEE/ACM Transactions on* 10.4 (2002), pp. 477–486.

[72] A. Damnjanovic et al. "A survey on 3GPP heterogeneous networks". In: *IEEE Wireless Communications* 18.3 (2011), pp. 10–21.

[73] AT&T and Cisco. "WLAN/Cellular Intelligent Network Selection". In: *3GPP TSG-RAN WG2 #81bis* (2013).

[74] Majed Haddad et al. "Automated Dynamic Offset Applied to Cell Association". In: *INFOCOM*. Toronto, Canada, 2014.

[75] Stefania Sesia, Issam Toufik, and Matthew Baker. *LTE - The UMTS Long Term Evolution: From Theory to Practice*. 2nd ed. Wiley, 2011.

[76] *Live encoder settings, bitrates and resolutions*. URL: https://support.google.com/youtube/answer/2853702?hl=en (visited on 07/18/2015).

[77] J.C. Ikuno, M. Wrulich, and M. Rupp. "System Level Simulation of LTE Networks". In: *Vehicular Technology Conference (VTC 2010-Spring), IEEE 71st*. 2010.

[78] D. Chiu R. Jain and W. Hawe. "A quantitative measure of fairness and discrimination for resource allocation in shared computer system". In: *DEC Research Report TR-301* (1984).

[79] C. Yim and A. C. Bovik. "Evaluation of temporal variation of video quality in packet loss networks". In: *Signal Processing: Image Communication*. 2011.

[80] Zheng Lu and G. de Veciana. "Optimizing stored video delivery for mobile networks: The value of knowing the future". In: *INFOCOM, 2013 Proceedings IEEE*. 2013, pp. 2706–2714.

[81] Xuan Kelvin Zou et al. "Can Accurate Predictions Improve Video Streaming in Cellular Networks?" In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. 2015.

[82] Stefan Lederer, Christopher Müller, and Christian Timmerer. "Dynamic Adaptive Streaming over HTTP Dataset". In: *Proceedings of the 3rd Multimedia Systems Conference*. 2012.

[83] Ali El Essaili et al. "Quality-of-experience driven adaptive HTTP media delivery". In: IEEE, 2013, pp. 2480–2485.

[84] *DATASET: HSDPA-bandwidth logs for mobile HTTP streaming scenarios*. URL: http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/.

[85] Truong Cong Thang et al. "An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming". In: *IEEE Journal on Selected Areas in Communications* (2014).

[86] Hatem Abou-zeid et al. "Evaluating mobile signal and location predictability along public transportation routes". In: *IEEE Wireless Communications and Networking Conference, WCNC*. 2015.

[87] Sami Mekki and Stefan Valentin. "Anticipatory quality adaptation for mobile streaming: Fluent video by channel prediction". In: *16th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, WoWMoM 2015, Boston, MA, USA*. 2015.

[88] *Index of /129021/dash/envivio/Envivio-dash2*. URL: http://dash.edgesuite.net/envivio/Envivio-dash2.

[89] George D. Magoulas and Michael N. Vrahatis. "Adaptive algorithms for neural network supervised learning: a deterministic optimization approach". In: *International Journal of Bifurcation and Chaos* (2006).