UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DESIGN AND IMPLEMENTATION OF AN ABR VIDEO STREAMING SIMULATION MODULE FOR NS-3. ANALYSIS AND COMPARISON OF ABR VIDEO STREAMING ALGORITHMS OVER VARIOUS MOBILE NETWORK SCENARIOS.

XINXIN LIU
JUNIO 2021

TRABAJO DE FIN DE MÁSTER

Título:	Diseño e implementación de un módulo de ABR video streaming para NS-3. Análisis y comparación de algoritmos de ABR video streaming sobre varios escenarios de redes móviles.
Título (inglés):	Design and implementation of an ABR video streaming simulation module for NS-3. Analysis and comparison of ABR video streaming algorithms over various mobile network scenarios.
Autor:	Xinxin Liu
Tutor:	Marcus Ihlar (Ericsson AB)
Ponente:	Carlos Mariano Lentisco Sanchez (ETSIT-UPM)
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos

TRABAJO FIN DE MÁSTER

DESIGN AND IMPLEMENTATION OF AN ABR VIDEO STREAMING SIMULATION MODULE FOR NS-3. ANALYSIS AND COMPARISON OF ABR VIDEO STREAMING ALGORITHMS OVER VARIOUS MOBILE NETWORK SCENARIOS.

Xinxin Liu

Junio 2021

Resumen

El streaming de vídeo con tasa de bits adaptativa se está convirtiendo en la técnica más utilizada por las plataformas de vídeo en línea. Con la pandemia mundial *COVID-19*, el streaming de vídeo se ha convertido en una de las principales fuentes de entretenimiento durante los confinamientos. De hecho, más de la mitad de la cuota de tráfico de la red se utiliza hoy en día para streaming de vídeo [7].

El objetivo de este Trabajo Fín de Máster (TFM) es construir un framework en *ns-3*, implementado en *C++*, para analizar y comparar algunas implementaciones de algoritmos de adaptación de vídeo sobre diferentes escenarios de red. El primer paso es estudiar *ns-3*, familiarizarse con algunos módulos de *ns-3* y construir varios escenarios de red *LTE*. El segundo paso es construir un módulo que pueda simular servidores y clientes de vídeo de *BitRate Adaptativo (ABR)*, estudiar algunos enfoques de los algoritmos de adaptación de la tasa de bits de vídeo e implementar dichos algoritmos, incluyendo soluciones basadas en el ancho de banda, en el buffer y algoritmos híbridos. Por último, podemos comparar y evaluar el rendimiento de diferentes algoritmos *ABR* en escenarios con condiciones variables con diferentes métricas objetivas de *QoE*.

Como resultado, se han probado diferentes implementaciones de algoritmos de adaptación con este nuevo módulo *ABR*. Aunque es evidente que tiene sus limitaciones siendo un entorno simulado.

Este proyecto se ha llevado a cabo con la cátedra Ericsson-UPM en software y sistemas.

Palabras clave: DASH, ABR, ns-3, streaming de video por HTTP, simulación, QoE

Abstract

Adaptive bitrate video streaming is becoming the most used technique for online video platforms. With the *COVID-19* worldwide pandemic, video streaming has become one of the primary sources of entertainment during the shutdown. In fact, more than half of the network traffic share today is used by video streaming [7].

The objective of this Master's Thesis is to build a framework in *ns-3*, implemented in *C++*, for testing video adaptation algorithms and to compare some implementations over different network scenarios. The first step is to study *ns-3*, familiarize with some *ns-3* modules, and build various LTE network scenarios. The second step is to build a module that can simulate *ABR* video servers and clients, study some approaches of video bitrate adaptation algorithms and implement those algorithms, including throughput based, buffer based and hybrid solutions. Finally we can compare and evaluate the performance of different *ABR* algorithms on scenarios with varying conditions with different objective *QoE* metrics.

As a result, different implementations of adaptation algorithms have been tested with this new *ABR* module. Although it is evident that it has its limitations being a simulated enviroment.

This project has been carried out with the Ericsson-UPM scholarship in software and systems.

Keywords: DASH, ABR, ns-3, HTTP video streaming, simulation, QoE

Acknowledgements

I would like to express my deepest appreciation to my mentor, Marcus, you have been really pacient with me and this thesis wouldn't be possible without your help. I would like to thank Carlos, my supervisor, by giving me a lot of constructive advices and proofreading my thesis.

Thanks also to my girlfriend, you always have been by my side and made this journey more pleasant. I am also very thankful for the company of classmates, even though we could not be in class physically this year because of the pandemic, but I will remember this stage of life with much gratitude.

Lastly, special thanks to Ericsson for the opportunity to do an internship during this course and to develop the project for this Master's thesis.

Contents

List of Figures

List of Tables

Listings

Glossary

3GPP - 3rd Generation Partnership Project

ABR - Adaptive BitRate

AMC - Adaptive Modulation and Coding

API - Application Programming Interface

ARP - Address Resolution Protocol

ASCII - American Standard Code for Information Interchange

BOLA - Buffer Occupancy based Lyapunov Algorithm

CDF - Cumulative Distribution Function

CDN - Content Delivery Network

CPU - Central Processing Unit

CQI - Channel Quality Indicator

DASH - Dynamic Adaptive Streaming over HTTP

DHCP - Dynamic Host Configuration Protocol

DRM - Digital Rights Management

EARFCN - E-UTRA Absolute Radio Frequency Channel Number

e-NodeB - enhanced Node B

EPC - Evolved Packet Core

EPS - Evolved Packet System

GSM - Global System for Mobile communications

HARQ - Hybrid Automatic Repeat reQuest

HDS - HTTP Dynamic Streaming

HLS - HTTP Live Streaming

HSS - Home Subscriber Server

HTTP - HyperText Transfer Protocol

IEC - International Electrotechnical Commision

IETF - Internet Engineering Task Force

IIS - Internet Information Services

IP - Internet Protocol

ISO - International Organization for Standarization

ITU-T - International Telecomunication Union - Telecomunication standarization

JFI - Jain Fairness Index

KPI - Key Performance Indicator

LENA - LTE-EPC Network simulAtor

LTE - Long Term Evolution

MAC - Medium Access Control

MCS - Modulation and Coding Scheme

MIMO - Multiple Input Multiple Output

MME - Mobility Management Entity

MMS - Multimedia Message Service

MPEG - Moving Picture Experts Group

MPD - Media Presentation Description

MSS - Microsoft Smooth Streaming

NAT - Network Address Translation

NR - New Radio

ns-3 - network simulator 3

OFDMA - Orthogonal Frequency Division Multiple Access

OSMF - Open Source Media Framework

PCRF - Policy Charging and Rule Function

PGW - Packet data network GateWay

PHY - LTE PHYsical Layer

QoE - Quality of Experience

QoS - Quality of Service

RB - Resource Block

RE - Resource Element

REM - Radio Environment Map

RLC - Radio Link Control

ROHC - RObust Header Compression

SC-FDMA - Single-Carrier Frequency Division Multiple Access

SGW - Serving GateWay

SRS - Souding Reference Signal

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UE - User Equipment

UHD - Ultra High Definition

UMTS - Universal Mobile Telecomunications System

URL - Universal Resource Locators

XML - eXtensible Markup Language

Chapter 1 Introduction

1.1 Context

There is no doubt about the importance of online video streaming. According to Sandvine [7], in 2020, 57% of the global internet traffic was used by video streaming. Moreover, one of the key predictions made by Cisco in 2018 [8] stated that by year 2022, video traffic will make up 82% of all *IP* traffic.

Consequently, many challenges arise. Due to the growth in the number and diversity of connected video-capable devices, and the increasing bandwidth and higher quality content available, the video client needs to adapt the multimedia content to the network and the devices. The technique of taking account the varying network conditions and computing resources of the user device to choose the adequate quality level is denominated as *Adaptive BitRate (ABR)*. Adaptation may be performed by monitoring different parameters such as estimated bandwidth, client's buffer level, CPU load or screen size.

The *Dynamic Adaptive Streaming over HTTP (DASH)* is the standard that implements adaptive bitrate video streaming and was developed by the *Moving Picture Experts Group (MPEG)* [18]. *MPEG-DASH* enables provisioning and delivering media using existing *HTTP*-delivery networks supports dynamic adaptation with seamless switching. By using *HTTP*, the player will not have firewall problems. The quality selection relays on the client thus providing better scalability.

The *MPEG-DASH* standard was published in 2012 and revised in 2019 by the *International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC)* as *MPEG-DASH ISO/IEC 23009-1:2019* [14]. In addition, the *3rd Generation Partnership Project (3GPP)* defines the use of *DASH* as the standard continuous for delivering of multimedia content in mobile networks, specifically in *LTE* and 5G networks [2].

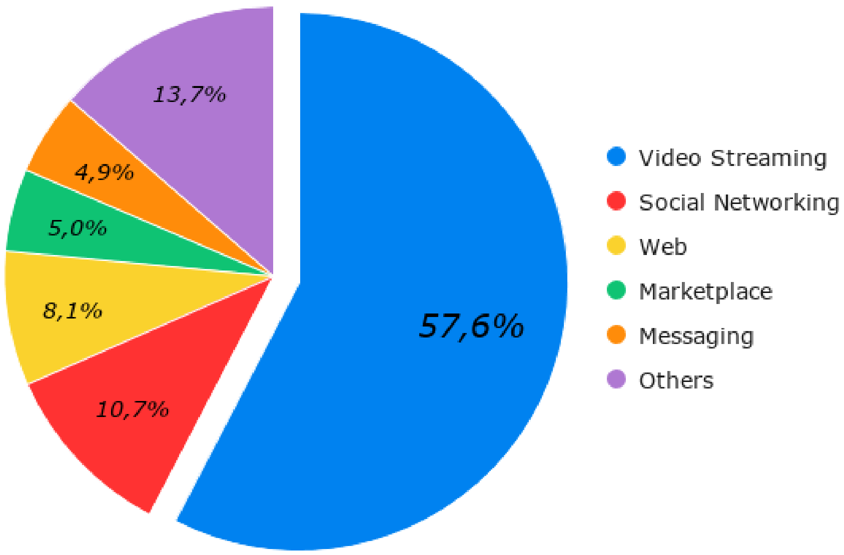

Figure 1.1: *Global application category total traffic share during COVID-19 lockdown. Source: Sandvine [7]*

DASH splits the input stream into small chunks or segments which are defined in the the *Media Presentation Description (MPD)*, which is an XML manifest file that contains the *Universal Resource Locators (URL)* of the segments. The *MPD* contains information for each representation such as the codec, bandwidth, resolution or framerate. Different qualities are defined as representations.

However, the DASH Standard [14] only defines the data formats for the media reproduction and do not provide any description on the adaptation algorithm. This thesis will analyze and compare a small number of adaptation algorithms. The *DASH Industry Forum* [9] provides an open source *MPEG-DASH* player implemented in *JavaScript* with different adaptation algorithms. Similarly, *hls.js* is an implementation of a *HTTP Live Streaming*[1] client.

The adaptation algorithms needs to be tested in different scenarios (they can be simulated) and be tweaked to provide the maximum perceived quality by the users. Also, there are algorithms that perform better in some specific scenarios and worse in others. The adaptation algorithm is the responsible for avoiding problems that may have a negative impact on the *Quality of Experience (QoE)* such as service disrruption or frequent changes

[1]HTTP Live Streaming is a HTTP-based adaptive bitrate streaming protocol developed by Apple Inc. [4]

on the bitrate. One problem is that, the algorithm can overestimate the bandwidth, this means requesting segments of a superior quality that the channel can support, and it would cause a pause in the reproduction because all the segments in the buffer are emptied. The algorithm can also underestimate the bandwidth, the video player requests media segments with inferior quality than the quality at which the bandwidth available of the network can allow. Lastly, the algorithm should avoid constant bitrate switches result of bandwidth fluctuations, and provide a smooth and seamless video watching experience.

This project will study and analyze the adaptation algorithms using The *ns-3* simulator is an open-source and extensible discrete-event network simulator. The extensible nature of this tool allows us to develop a new module for *ns-3* mimicking the behaviour of *ABR* clients and servers. With this new module, *ns-3* will be able to simulate diverse mobile network scenarios and test the performance of adaptation algorithms.

1.2 Objectives

The objectives of this thesis is to build a framework for testing *ABR* adaptation algorithms, and implement some adaptation algorithms and compare them in various mobile network scenarios with different objective *QoE* metrics. In order to achieve the proposed objectives, the following steps will be proposed:

1. Study and understand *ns-3* and basic modules such as the core module, the internet module, applications module, *LTE* module among others. Build basic *LTE* scenarios tweak radio parameters, and output results.
2. Design a new module in ns-3 that simulates behaviours of *ABR* clients and servers. Study and implement existing adaptation algorithms.
3. Obtain objective *QoE* and *QoS* metrics. Build new *LTE* scenarios and compare the performances of the implemented adaptation algorithms.

1.3 Structure of the thesis

Chapter 1. Presents the context, the motivations and the objectives of this thesis.

Chapter 2. The State of the Art. Includes an introduction to ABR and DASH. The architecture and video quality adaptation algorithms. Also, an brief explaination of LTE, its architecture and fundamentals.

Chapter 3. A starting guide to use *ns-3*. Brief introduction and usage of relevant *ns-3* modules for this thesis.

Chapter 4. Introduces a new module for *ns-3*, the *ABR* module. Describes components and models of the *ABR* module. Highlights the implemented adaptation algorithms.

Chapter 5. Goes through a set of testing scenarios. Analyses and compares the different implemented adaptation algorithms.

Chapter 6. Concludes the thesis and discusses possible future works.

Chapter 2 State of the Art

This chapter will introduce the main concepts and tools that will be used during the development of the project. The section 2.1 will explain the different methods of content distribution over *HTTP* and different types and implementations of adaptive streaming. The section 2.2 will make a introduction to the *DASH* standard, different types of adaptation algorithms and *QoE* and *QoS* metrics. The section 2.3 will describe basic architecture and fundamentals of 4G LTE, such as the radio interface, propagation loss model, fading model, antenna model, etc.

2.1 ABR Video Streaming

There are three ways of media delivery over *HTTP*. The first method is by **file download**, the media file is downloaded in its entirety in a local hard disk and then it can be played. The second method is called **progressive download**, this method is similar to the file download, but instead the download starts from the beginning and the media starts playing once enough data are playable. However, these two methods have disadvantages like waste of bandwidth or *DRM* issues and also requiring a reliable transmission. The last method is called **streaming**, contrary to the former two, the file itseft is not stored locally, smaller chunks of video are sent from the server and the client needs a data buffer to store the data that is being downloaded. The client plays the multimedia content from the buffer, and when the session is closed the data are deleted.

Streaming media also comes with some challenges. There are a lot of network variability and a big heterogeneity in video capable devices. Therefore, to overcome these shortcomings, *Adaptive bitrate streaming (ABR)* was created.

The basic idea of *Adaptive bitrate streaming* is to adapt the media content for the user by monitoring different parameters like estimated bandwidth, buffer level or *CPU load*, see Figure 2.1. There are many propietary adaptive streaming solutions:

- ***Apple HTTP Live Streaming (HLS)***: *HTTP Live Streaming HLS* is an implementation of an *ABR* protocol over *HTTP* developed by Apple [4] as part of the QuickTime software and the mobile operating system *iOS*. *HLS* supports live streaming and video on demand. *HLS* is proposed in 2009 as a standard to the *IETF* [17].

- ***Microsoft Smooth Streaming (MSS)***: *Smooth Streaming* is part of *Internet Information Services (IIS) Media Services* for delivering media over *HTTP* [22]. Their *MSS* technology was used for several sports events such a the Beijing Summer Olympic Games in 2008 and the 2010 Winter Olymplics in Vancouver [23].

- ***Adobe HTTP Dynamic Streaming (HDS)***: *HTTP Dynamic Streaming* is the implementation of adaptive streaming by Adobe. *HDS* enables high-quality, network efficient HTTP streaming for media delivery that is tightly integrated with Adobe software [3]. The solution is based in using *Open Source Media Framework (OSMF)* and Adobe Flash Player.

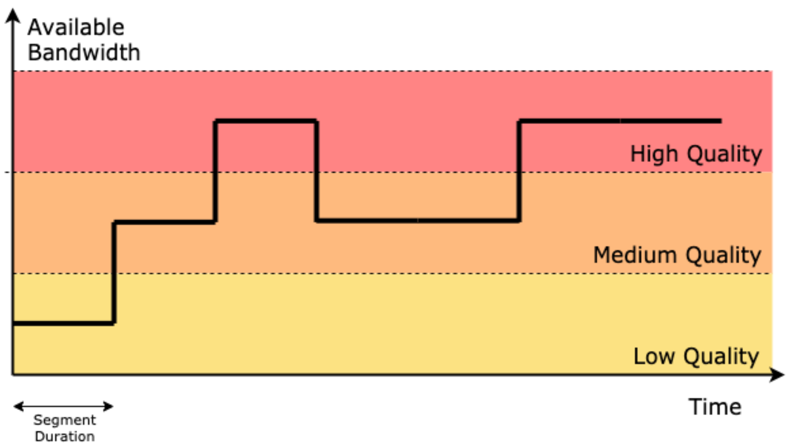

Figure 2.1: *Evolution of segment quality with time*

But there was no official standarization for adaptive video delivery over HTTP. For that reason, a new international stadard called *MPEG-DASH* was developed and published.

2.2 Dynamic Adaptive Streaming over HTTP

DASH was published in April 2012. The most recent revision of the standarization was released in 2019 as *MPEG-DASH ISO/IEC 23009-1:2019* [14]. *Moving Picture Experts Group* from *ISO/IEC* and the *3GPP* collaborated on the *DASH* standard. The *3rd Generation Partnership Project* defined the use of *DASH* as the standard of digital media delivery in mobile networks (4G *LTE*, 5G) in [2].

The objective of *DASH* was to create a unique standard that unifies the propietary solutions from Microsoft, Apple and Adobe. Also, it will offer the interoperability and the convergence needed for the expansion of large-scale video streaming solutions. Also, the *DASH Industry Forum (DASH-IF)* was created to promote and help the expansion of *DASH*. Microsoft, Apple, Netflix, Qualcomm, Ericsson and Samsung are some of the companies members of the *DASH-IF*.

One of the biggest advantages of *DASH* is the use of *HTTP* protocol. The use of *HTTP* means that reusing existing internet infrastructure and media content distribution tecniques using *CDN (Content Delivery Networks)* can be done. Another convenience of using *DASH* is, problems with passing through firewalls and the *Network Address Translation (NAT)* are avoided.

All the control of the media content delivery is located in the *DASH* client side. The standard does not define any web delivery mechanism nor the bitrate adaptation algorithm. What *DASH* does define in [14] is:

- ***The Media Presentation Description (MPD) File Format***: The *MPD* file uses the *eXtensible Markup Language (XML)* and contains the specifications of the media content and the *URL* of the segments in the *HTTP* video servers.
- **Segment format**: *DASH* defines the characteristics of the necessary codifications and the way that the media content is divided in small fragments called *segments*.

The Figure 2.2 presents a simple *DASH* architecture. The video and audio contents are processed and stored on an *HTTP* server. To access the content, the client sends *HTTP* requests to the server. But first, the client needs to download the *MPD* file, normally through *HTTP*. The client then does the parsing of the *MPD*, extract information such as the duration of a segment, the available representations, media types or resolutions.

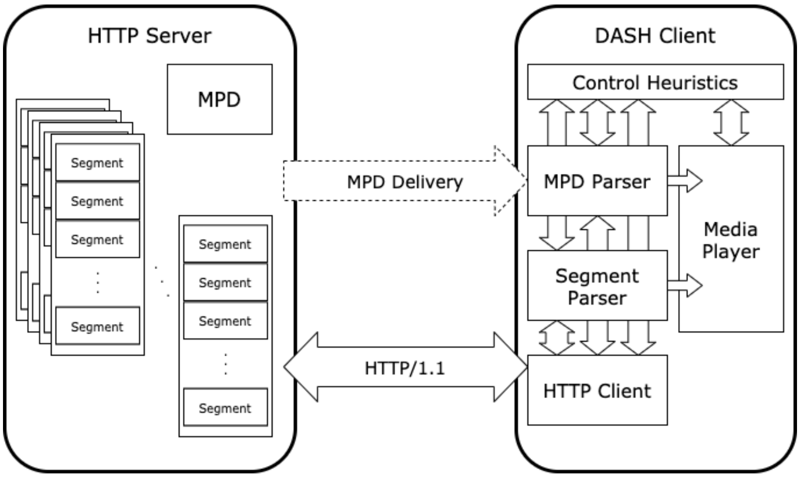

Figure 2.2: *DASH client-server architecture. Source: MPEG [28]*

Finally, the *DASH* client chooses the adequate quality and starts the streaming of the content using *HTTP GET* request to fetch the segments.

The *DASH* client stores the segments in a buffer and consumes the content. The adaptation algorithm selects the most appropiate representation, for example, basing on bandwidth estimations, to avoid problems like buffer underflow and maintain at least a set number of segments in the buffer.

2.2.1 MPD

The *MPD* file is an *XML* document that describes the characteristics of the different media components that composes the media content (e.g. video, audio, subtitles).

The structure of the *MPD* is hierarchical as illustrated in Figure 2.3. The media content is divided in a sequence of **periods**, where each period has a starting time and a duration. During a period, the set of characteristics of the media content, like the bitrates, languages or codecs, do not change.

Each period consists of one or multiple **adaptation sets**. A collection of interchangeable encoded versions of one or more media content components is referred to as an adaptation set. For instance, and adaptation set may contain the different bitrates of the video component of the same multimedia content and another adaptation set may contain the different bitrates of the audio component of the same multimedia content.

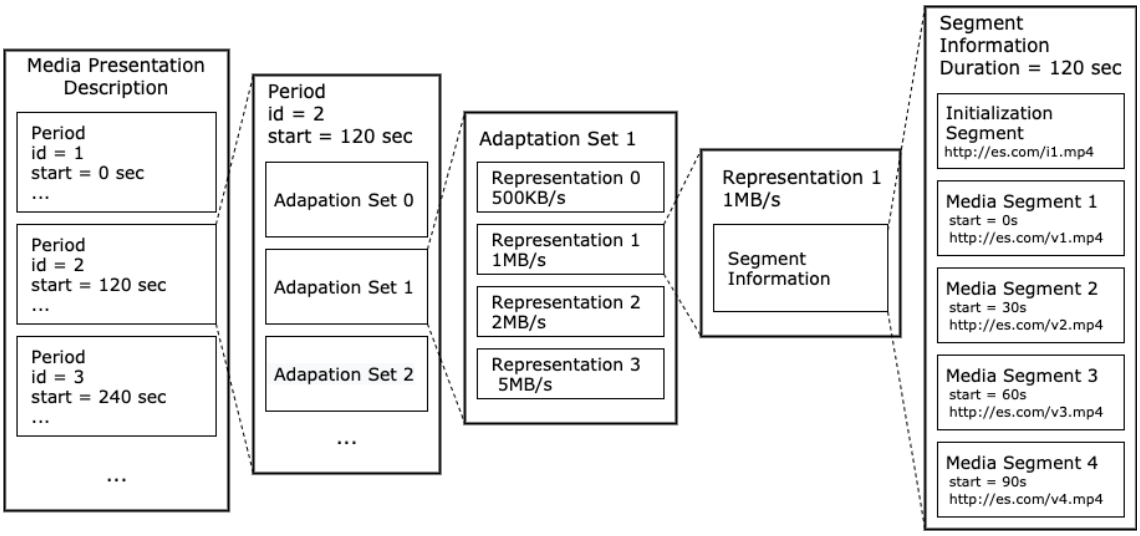

Figure 2.3: *The MPD hierarchical data model. Source: MPEG [28]*

An adaptation set contains a set of **representations**, where a representation can be defined as an enconded alternative of the same media component, representations are defined by parameters such as bitrate, resolution, framerates, codec, sampling rate or other characteristics.

Each representation consists of one or multiple **segments**. A segment is a fragment of the multimedia content. Each segment is univocately identified by a *URI*, the client sends *HTTP* requests by using the *URIs* to get the segments.

2.2.2 Adaptation Algorithms

In a video streaming service, factors such as the available bandwidth, delay or packet losses can make the buffer to starve. Rebuffering and interruptions lead to bad Quality of Experience. To solve these problems, different adaptation algorithms have been proposed in the literature.

An adaptation algorithm is a technique used in a multimedia streaming service to adjust the video quality in real-time according to different parameters. Some of the parameters are:

- **Client device**: The screen resolution, CPU capabilities, Buffer size, etc.
- **Network**: Type of access network (Mobile, Fixed), available bandwidth, etc.

The following subsections will explain different types of adaptation algorithms and the algorithms implemented for this thesis in *ns-3*.

2.2.2.1 Bandwidth throughput based algorithms

This group of algorithms uses bandwidth estimations to select the most adequate multimedia representation. The main difference between algorithms of this kind is the bandwidth estimation method and how the estimation influences on the representation selection.

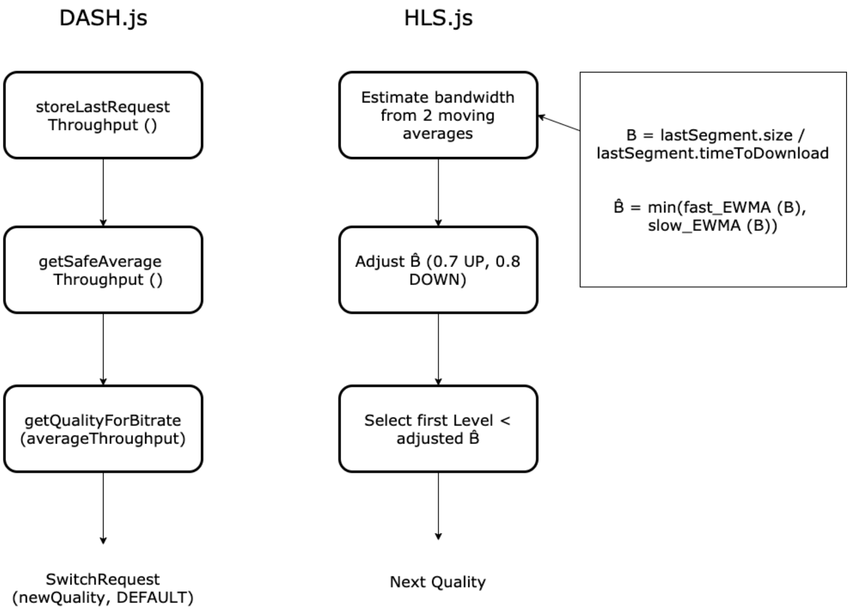

Figure 2.4: *Bandwidth based algorithms. Source: [12]*

- **HLS.js** [32]. The algorithm is called Bandwidth estimation.

  The algorithm processes two EWMA (Exponentially Weighted Moving Averages) and chooses the minimum of the two as the bandwidth estimation.

$$B_N = \frac{SegmentSize_N}{TimeToDownload_N} \tag{2.1}$$

$$FastEWMA_N = B_N \times \alpha_{fast} + FastEWMA_{N-1} \times (1 - \alpha_{fast}) \tag{2.2}$$

$$SlowEWMA_N = B_N \times \alpha_{slow} + SlowEWMA_{N-1} \times (1 - \alpha_{slow}) \tag{2.3}$$

$$\hat{B} = min\,(FastEWMA_N, SlowEWMA_N) \tag{2.4}$$

Then the bandwidth estimation is multiplied by a factor to reduce oscilation. Finally it selects the representation based on the adjusted bandwidth estimation.

- **DASH.js** [10]. Throughput Rule.

  This algorithm is basically the same as the Bandwidth estimation from HLS.js. It computes the average throughput, and uses an safety factor to avoid oscilations. And then chooses the quality based on the safe average and creates a new *SwitchRequest*.

2.2.2.2 Buffer based algorithms

This group of algorithms uses buffer occupancy information to try to choose the highest level of bitrate for the multimedia content.

- **BOLA**. Buffer Occupancy based Lyapunov Algorithm.

  The BOLA adaptation algorithm uses the Lyapunov optimization [29] to make decisions. This is an utility theory and it is configurable with a tradeoff parameter to choose between rebuffering potential and bitrate maximization.

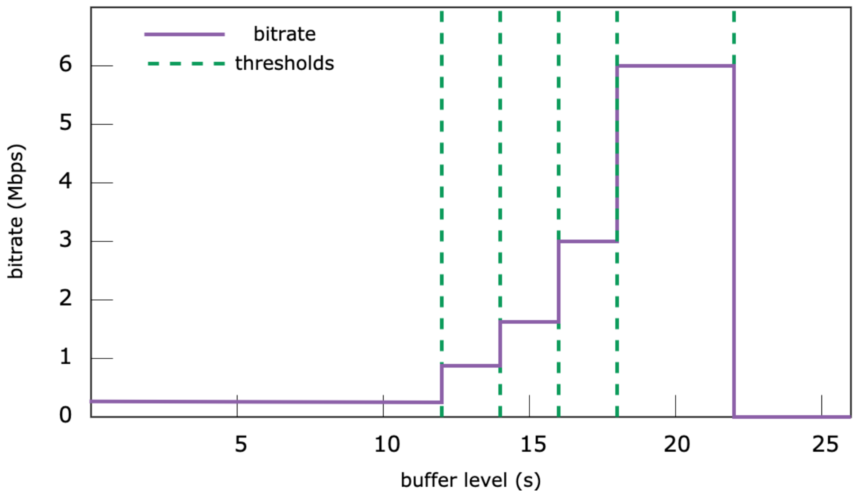

Figure 2.5: *BOLA's bitrate choice as function of buffer level. Source: [29]*

BOLA tries to maximize $V_n + \gamma S_n$. where:

- V_n is the bitrate utility.
- S_n is the playback smoothness.
- γ is the tradeoff weight parameter for rebuffering potential or bitrate maximization.

2.2.2.3 Control theory based or hybrid algorithms

This class of algorithms uses a combination of throughput estimation and buffer occupancy and tries to maximize the bitrate selection with decision-taking indicators calculated making use of control theory or stochastic optimal control equations.

2.2.3 QoS & QoE Metrics

The *Quality of Service (QoS)* is defined by the *ITU-T* in the document P.10/G.100 [20] 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". And the *Quality of Experience (QoE)* is defined as "The degree of delight or annoyance of the user of an application or service".

The standard *ISO/IEC 23009* defines a list of parameters for *Quality of Service (QoS)* and *Quality of Experience (QoE)* for the adaptation algorithms to base on. There parameters are also used to evaluate the overall quality in the multimedia distribution service.

Some of the metrics defined in [2] and [14] are as follows:

- **Average Throughput**: This is a *QoE* metric that defines a list in which the average throughput can be obtained that is observed in the client during a measuring period.
- **Initial Playout Delay**: This is a *QoE* metric that represents the initial delay in the reproduction of the media content.
- **Representation Switch Events**: This is a *QoS* metric for measuring the number of representation switch events of the multimedia content.
- **Buffer Level**: This is a *QoS* metric that monitors the level of occupancy of the buffer during the reproduction of the multimedia content.

2.3 LTE Fundamentals

Long Term Evolution (LTE) was first introduced in 2008 in the Release 8 of the *3GPP* specification [1]. The objective of *LTE* was to migrate the *3GPP* systems into a optimized system based on packet switching (all *IP*), with greater bitrates, lower latency y multiple radio access technologies support.

2.3.1 History

The first mobile phone call was made in 1973 [15]. New generations of mobile networks are developed almost every decade. The first generation 1G launched years later, but it was only capable of doing voice calls. In 1991, the second generation 2G *(GSM)* of mobile networks was introduced. *GSM* provided improved wireless capabilities and introduced by the first time multimedia content with *Multimedia Message Service (MMS).* But it was the third generation 3G, launched in 2001, that enabled new internet-driven services such as video conferencing and streaming. Later in 2009, the *LTE* 4G standard was commercially deployed. With theorical download bandwidth of almost 100Mbps made high-quality streaming into reality. 5G technologies will provide an improvement in bandwidth even more and brings video streaming in *UHD* and more.

The consumption of multimedia content on mobile networks is becoming increasingly relevant with the rise of bandwidth and ease of access. This section will provide a brief introduction to the basic concepts of mobile networks, their architecture and fundamentals.

2.3.2 Architecture

The design of the *LTE* architecture was done from the ground up. The goal was to build a flat, all *IP* architecture using packet-switching, well structured (separation of control plane and user plane) and with few elements.

The *Evolved Packet System (EPS)* is constituted by the following elements:

- ***User Equipment (UE)***: An *UE* is any device used by an end user to communicate in a mobile network.
- ***Evolved UMTS Terrestial Radio Access Network (E-UTRAN)***: The only elements in the *E-UTRAN* are the *e-NodeB*. An ***enhanced Node B (e-NodeB)*** works as a base station and a controller.

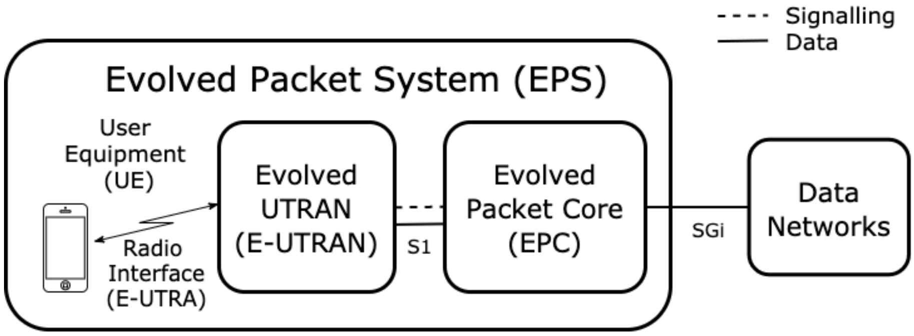

Figure 2.6: *LTE Architecture*

- ***Evolved Packet Core (EPC)***: The *EPC* is made up of a network of gateways, control servers, and databases linked by a *IP* backbone. The main elements of the *EPC* are:
    - ***Mobility Management Entity (MME)***: The *MME* is the main node for the control plane. It handles the signalling related to mobility and security for *E-UTRAN* access.
    - ***Serving Gateway (SGW)***: The *SGW* is the gateway used for communicating the access network *E-UTRAN* and the *PGW*.
    - ***Packet Data Network Gateway (PGW)***: The *PGW* is the gateway for the traffic between the core network and external packet data networks. It also performs functions such as IP address allocation or packet filtering.
    - ***Home Subscriber Server (HSS)***: The *HSS* is a database containing information about the *EPC* network users. It also provides support functions in mobility management, call and session setup, user authentication and access authorization.
    - ***Policy Charging and Rule Function (PCRF)***: The *PCRF* is used for *QoS*, policy and charging management.

2.3.3 Wireless Fundamentals

Large-scale wireless networks, such as LTE, are fundamentally inefficient and prone to interference. Supporting mobility while also obtaining high levels of power efficiency, such as through directional antennas, can be really challenging. Base stations must be selectively

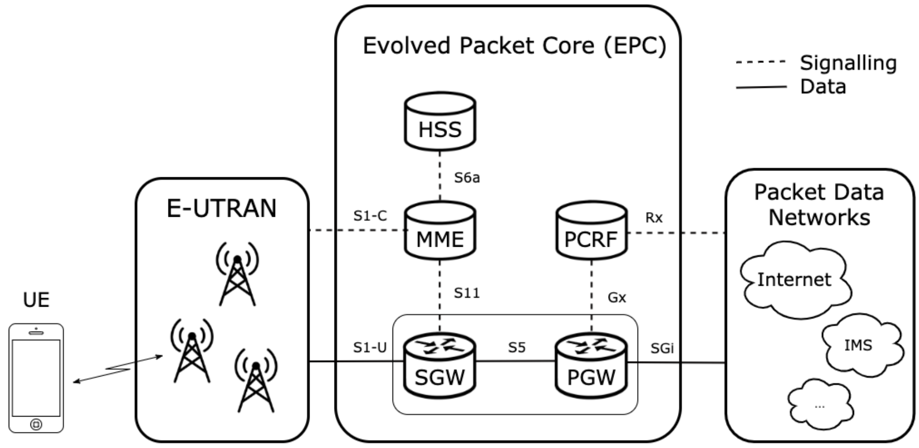

Figure 2.7: *Evolved Packet Core (EPC) Architecture*

installed but accommodate vast user populations in order to be cost-effective, resulting in a significant amount of self-interference. As a result, achieving high coverage, capacity, and dependability at low cost and used power is extremely difficult, if not impossible.

The following list highlights the main parameters affecting the received signal in a wireless system.

2.3.3.1 Propagation loss

The amount of transmitted power that actually reaches the receiver is the first visible difference between wired and wireless channels. The transmitted signal energy extends along a spherical wavefront if an isotropic antenna is utilized, hence the energy received at an antenna d distant is inversely proportional to the sphere surface area, $4\pi d^2$. However, in reality the propagation environment is not free space, we may also take into account other factors such as reflections.

2.3.3.2 Shadowing

Obstacles such as trees and buildings, as shown in Figure 2.8, may be situated between the transmitter and receiver, causing temporary signal degradation, whereas a temporary line-of-sight transmission path would result in abnormally high received power.

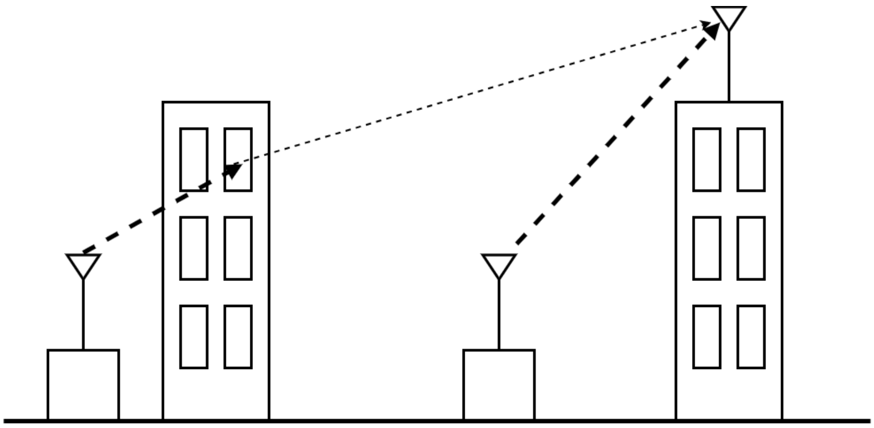

Figure 2.8: *Shadowing effect. Source: [24]*

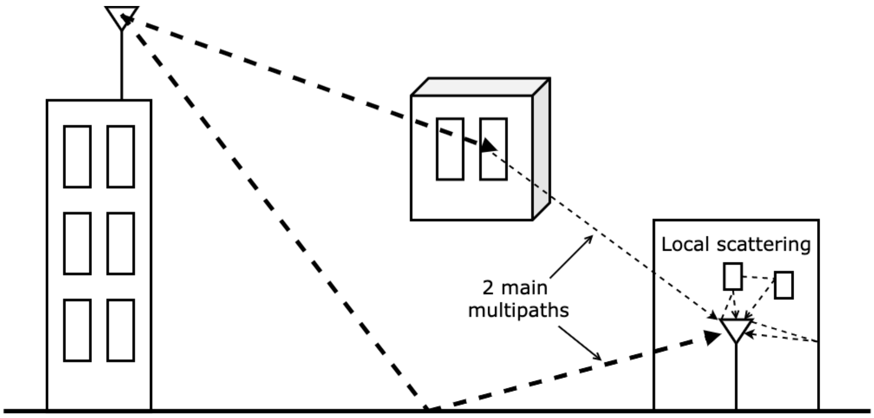

Figure 2.9: *Fading loss effect. Source: [24]*

2.3.3.3 Fading loss

The fading effect is another aspect of wireless channels. Fading is generated by the receiving of multiple versions of the same signal (multipath), unlike path loss or shadowing, which are large-scale attenuation effects induced by distance or obstacles.

The reflections may arrive at very short intervals. For example, if there is local dispersion around the receiver, or they may arrive at relatively longer intervals, for instance, if the transmitter and receiver are on multiple pathways. Figure 2.9 illustrates this.

2.3.4 Antennas & MIMO

An antenna is a device that uses electromagnetic waves to transmit or receive information. The transmitting antenna turns electrical currents into electromagnetic waves, and vice versa (receiving antenna).

Multiple Input, Multiple Output (MIMO) is a technique for increasing the capacity of a radio link by employing multiple transmitting and receiving antennas to take advantage of multipath propagation. MIMO has become a key component of wireless communication technologies such as LTE.

There are several implementations of MIMO in LTE:

- ***Single antenna***: Most simple wireless links employ this type of radio transmission. One antenna transmits a single data stream, which is received by one or more antennas. It is also know as SISO.

- ***Transmit diversity***: This type of LTE MIMO method makes use of several antennas to transmit the same data stream.

- ***Open loop spatial multiplexing***: This type of MIMO involves delivering two information streams through two or more antennas.

- ***Close loop spatial multiplexing***: Similar to the above but with a close loop feedback.

- ***Clesed loop with pre-coding***: This type of MIMO transmits a single code word over a single spatial layer.

- ***Multi-User MIMO***: Single-user SU-higher MIMO's per-user throughput is better suited to more sophisticated user devices with more antennas, whereas MU-MIMO is more practical for low-complexity mobile phones with a small number of reception antennas.

- ***Beam-forming & MIMO***: This is the most advanced MIMO mode. It allows the antenna to focus on a specific location.

2.3.5 Physical Layer

2.3.5.1 OFDMA and SC-FDMA

The cellular communication systems needs to have a strategy for multiple access. In LTE, the *Orthogonal Frequency Division Multiple Access (OFDMA)* is used for downlink and the *Single- Carrier Frequency Division Multiple Access (SC-FDMA)* is used for uplink. Both are very similar, consisting in allocating each subscriber some portion of the subcarriers for certain amount of time.

In the Figure 2.10, a transmission structure of LTE is presented. The two dimentions of the plane are time and frequency. Two important concepts are defined as:

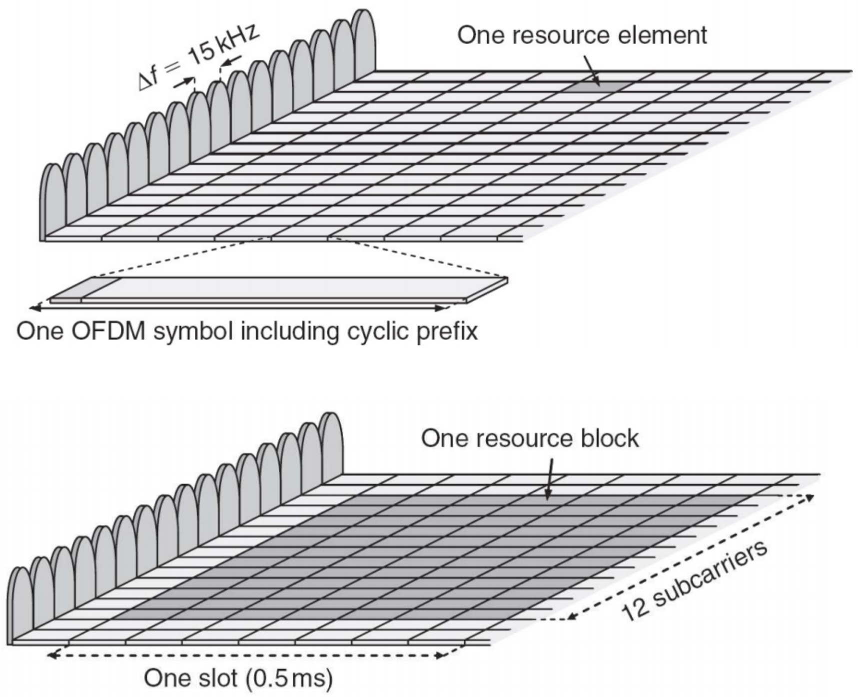

Figure 2.10: *LTE Time-Frequency Grid. Source: [31]*

- ***Resource Element (RE)***: A Resource Element is the basic element of resouce, it is defined as one subcarrier in a symbol period.
- ***Resource Block (RB)***: A Resource Block is composed by twelve subcarriers (180 kHz) in a time interval of 0.5 ms (7 OFDM symbols).

Users are assigned resources in resource blocks across a subframe, i.e., 12 subcarriers over $2 \times 7 = 14$ OFDM symbols for a total of 168 Resource Elements. Because some of the 168 resource components are utilized for various layer 1 and layer 2 control messages, not all of them can be used for data.

The number of Resource Blocks available for each channel bandwidth is given by the Table 2.1.

Bandwidth	1.4 MHz	3 MHz	5 MHz	10 MHz	15 MHz	20 MHz
Number of RBs available	6	15	25	50	75	100

Table 2.1: *Number of Resource Blocks against each channel bandwidth. Source: [30]*

2.3.5.2 AMC & CQI

AMC stands for Adaptive Modulation and Coding, is a terminology used in LTE to describe how modulation and coding are matched to the radio link's conditions.

The eNB applies AMC by selecting the appropriate MCS based on quality estimations supplied by the UE mobile terminal via the *Channel Quality Indication (CQI)* parameter.

MCS

2.3.5.3 EARFCN

The *E-UTRA Absolute Radio Frequency Channel Number (EARFCN)* is a number between 0-65535 used for desginating uplink and downlink carrier frequencies.

$$F_{downlink} = FDL_{Low} + 0.1(NDL - NDL_{Offset}) \tag{2.5}$$
$$F_{uplink} = FUL_{Low} + 0.1(NUL - NUL_{Offset}) \tag{2.6}$$

Where NDL is the downlink EARFCN, NUL is the uplink EARFCN.

CQI Index	Modulation	Code Rate \times 1024	Efficiency
0		out of range	
1	QPSK	78	0.1523
2	QPSK	120	0.2344
3	QPSK	193	0.3770
4	QPSK	308	0.6016
5	QPSK	449	0.8770
6	QPSK	602	1.1758
7	16QAM	378	1.4766
8	16QAM	490	1.9141
9	16QAM	616	2.4063
10	64QAM	466	2.7305
11	64QAM	567	3.3223
12	64QAM	666	3.9023
13	64QAM	772	4.5234
14	64QAM	873	5.1152
15	64QAM	948	5.5547

Table 2.2: *4-Bit CQI Table*

2.3.5.4 Souding Reference Signal

Souding Reference Signal (SRS) are wideband reference signals used by the eNode-B to determine uplink channel quality information in order to allocate uplink resources. There are three types of SRS transmissions, single SRS, periodic SRS and aperiodic SRS.

2.3.6 Medium Access Control Layer

The *Medium Access Controll (MAC)* layer essentially provides the higher layer with radio resource allocation and data transfer services and connects the RLC layer and the PHY layer. The MAC layer executes procedures such as logical channel priority, power headroom reporting, UL grant and DL assignment, and so on as part of the radio resource allocation service. The MAC layer performs functions like scheduling requests, buffer status reporting, random access, and HARQ as part of the data transmission service.

2.3.7 Radio Link Control Layer

The RLC layers's key functions include data unit segmentation and concatenation, error correction via the ARQ protocol, and packet delivery in sequence to higher levels. It has three modes of operation:

- **Transparent Mode (TM)** is the most basic mode, with no RLC header, data segmentation, or concatenation, and is used for specialized applications like random access.
- **Unacknowledged Mode (UM)** The UM mode detects packet loss and allows for packet reordering and reassembly, but does not require the missing protocol data units to be retransmitted (PDUs).
- **Acknowledged Mode (AM)** is set up to request retransmission of missing PDUs in addition to the UM mode's features.

2.3.8 Packet Data Convergence Protocol Layer

The PDCP layer's main features are IP packet header compression and decompression based on the *RObust Header Compression (ROHC)* protocol, data and signaling ciphering, and signaling integrity protection. Per bearer, there is only one PDCP entity at the eNode-B and the UE.

Chapter 3 | Network Simulator 3

The *ns-3* simulator is an open, extensible discrete-event network simulator designed primarily for educational and network research purposes [25].

In summary, *ns-3* provides models of how packet data networks work and operate, as well as a simulation engine that allows users to run simulation experiments. To do research that are more difficult or impossible to do with real systems, to examine system behavior in a highly controlled, reproducible setting, and to understand about how networks work.

ns-3 is a collection of modules that can be used together as well as with other software libraries. This tool works mainly at the command line on *Linux* or *MacOS* and with *C++* and *Python* programming languages and development tools.

3.1 ns-3 Concepts

This section will go over several networking concepts that have a specific meaning in ns-3.

Node: A `Node` in *ns-3* is the basic computing device abstraction. The `Node` class has methods for managing computing device representations in simulations.

Application: A *ns-3* application run on *ns-3* `Nodes`. An `Applicacion` is the basic abstraction for a user program that generates some simulated activity. The `Application` class provides functions for controlling the representations of the simulated version of user-level applications.

Channel: A `Channel` in *ns-3* is an abstraction of the basic communication subnetwork in which `Nodes` are connected in. It can be as simple as a wire or as complicated as a Ethernet switch.

Net Device: A `NetDevice` in *ns-3* simulates a *Network Interface Card (NIC)* and the software controlling the *NIC*. A `NetDevice` is installed in a `Node` to allow it to communicate over `Channels` with other `Nodes` in the simulation.

Helpers: Helper objects are created to make some commun tasks easier. Such as connecting `NetDevices` to `Nodes`, `NetDevices` to `Channels`, assigning IP addresses, etc.

The Figure 3.1 shows a high level node architecture.

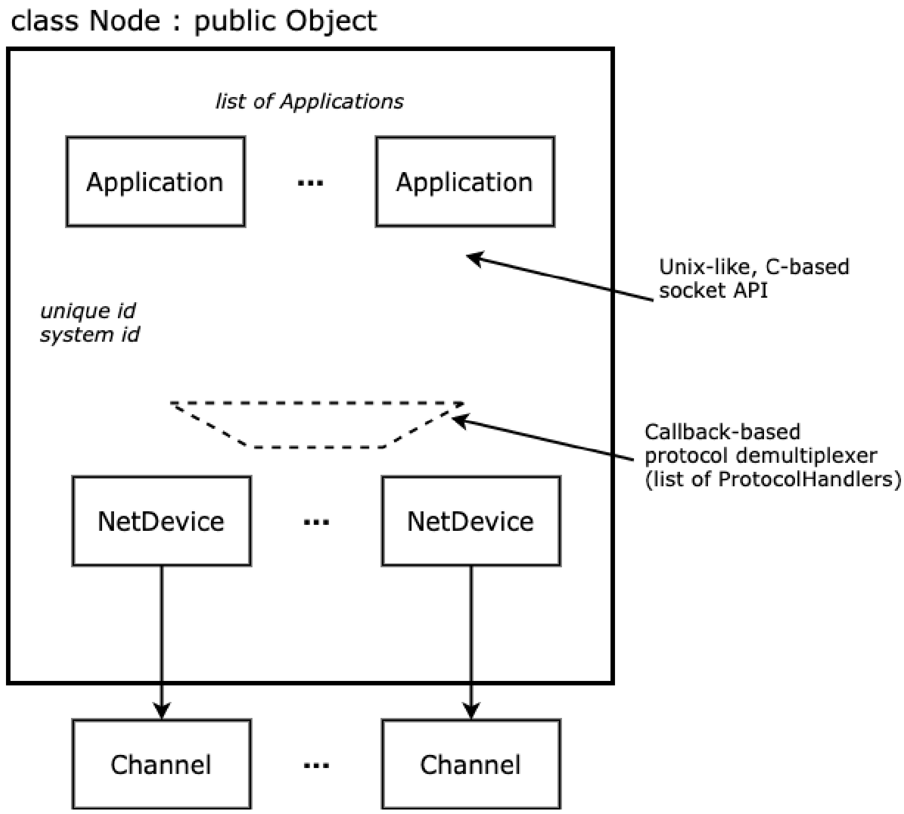

Figure 3.1: *ns-3 High-level node architecture. Source: [25]*

3.2 Logging Module

Message logging is a basic feature for large softwares, and *ns-3* is no different. *ns-3* offer a complete module for message logging with configurable verbosity levels. This means that logging functions of specific components can be enabled and other can be disabled completely.

There are different levels of log messages of ascending verbosity defined in *ns-3*:

- **`LOG_ERROR`**: For error messages (associated function: `NS_LOG_ERROR`).
- **`LOG_WARN`**: For warning messages (associated function: `NS_LOG_WARN`).
- **`LOG_DEBUG`**: For relatively rare, ad-hoc debugging messages (associated function: `NS_LOG_DEBUG`).
- **`LOG_INFO`**: For informational messages about program progress (associated function: `NS_LOG_INFO`).
- **`LOG_FUNCTION`**: For messages describing each function called (two associated function: `NS_LOG_FUNCTION` used for member functions, and `NS_LOG_FUNCTION_NOARGS`, used for static functions)).
- **`LOG_LOGIC`**: For messages describing logical flow within a function (associated function: `NS_LOG_LOGIC`).
- **`LOG_ALL`**: Log everything mentioned above (no associated function).

To enable all logs, it is as simple as modifying a shell variable. In the next example the logging for the class `UdpEchoClientApplication` and `UdpEchoServerApplication` is enabled with all levels, the time and the function prefixes:

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func|
    prefix_time:UdpEchoServerApplication=level_all|prefix_func|
    prefix_time'
```

Listing 3.1: *Enabling logging in ns-3*

For more information with the logging modure see [25].

3.3 Command Line Arguments

There are local and global variables that can be changed in the command line without editing the scripts. To be able to know what variables are available, the option `-PrintHelp` is used.

3.4 Tracing System

The main goal of the simulations is to extract and generate output, and *ns-3* offers two mechanisms for this. Also, since *ns-3* is a C++ software, using `std::cout` for output is also available.

3.4.1 ASCII Tracing

ns-3 includes helper function that encapsulates the low-level tracing system and guides you through the technicalities of establishing some simple packet traces. If you enable this feature, the output will be in ASCII files, hence the name.

To enable ASCII Tracing, right before the call to `Simulator::Run ()`, create an `AsciiTraceHelper` and call the function `EnableAsciiAll`. This will generate the output into the home directory of *ns-3*.

3.4.2 PCAP Tracing

The *ns-3* device helpers can also create `.pcap` trace files. The *pcap* file contains the packets captured during the simulation. *Wireshark* or *tcpdump* are programs capable of reading and visualizing *pcap* files.

To enable *pcap* tracing simply add the `EnablePcapAll` function. And it will create various `.pcap` files in the format `"myfirst-0-0.pcap"`, meaning the trace file for node 0 and device 0.

3.5 ns-3 Modules & Models

In this section, modules used in this thesis will be presented, based on the official manual from [25]. But first, It is essential to understand the difference between modules and models:

- **Modules** are the different libraries that form *ns-3*.
- **Models** are the simulated, abstract representations of real-life objects, protocols, devices, etc.

As the reader may already know, *ns-3* is modular. A new module will be introduced in the chapter 4 as a result of this Master final project.

3.5.1 Antenna Module

The Antenna module provides a `AntennaModel` base class as an interface for radiation pattern modelling of an antenna. Also, there are a set of classes derived from this base class that implements types of antennas with differente radiation patterns.

3.5.1.1 AntennaModel

The `AntennaModel` uses a coordinate system as shown in the Figure 3.2. This model uses, for a point p in the space with Cartesian coordinates used by the `MobilityModel`, the coordinates (x, y, z) and transforms into spherical coordinates (r, θ, ϕ).

The radiation pattern is express as a mathematical function $g(\theta, \phi) \to \mathbb{R}$

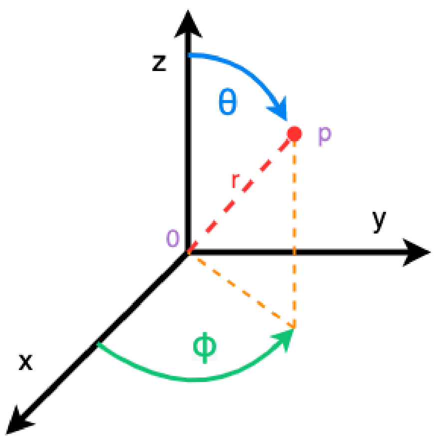

Figure 3.2: *Coordinate system of the AntennaModel. Source: nsnam [25]*

- **IsotropicAntennaModel**

The `IsotropicAntennaModel` is onmidirectional, this means that the radiation pattern have a 0dB gain for all direction.

- **CosineAntennaModel**

  The antenna gain of the `CosineAntennaModel` is defined as:

$$g(\phi, \theta) = cos^n\left(\frac{\phi - \phi_0}{2}\right) \tag{3.1}$$

  with ϕ_0 as the antenna's azimuthal orientation, this is, the direction of maximum gain. And the exponential

$$n = -\frac{3}{20\, log_{10}\left(cos\frac{\phi_{3dB}}{4}\right)} \tag{3.2}$$

  determines the wanted 3db beamwidth ϕ_{3dB}.

- **ParabolicAntennaModel**

  In the `ParabolicAntennaModel`, the antenna gain is determined as:

$$g(\phi, \theta) = -min\left(12\left(\frac{\phi - \phi_0}{\phi_{3dB}}^2\right), A_{max}\right) \tag{3.3}$$

  where A_{max} is the maximum attenuation in dB of the antenna.

3.5.2 Application Module

The `Application` class is a base class for *ns-3* applications. Nodes can have one or more applications. Each node maintains a list of smart pointers (references) to all its applications. The `Applications` are the responsibles to create the sockets, if needed.

There are a few implementations of `Applications` in *ns-3*:

- `BulkSendApplication`. This traffic generator basically sends data as quickly as possible until MaxBytes is reached or the application is terminated (if MaxBytes is zero)
- `OnOffApplication`. After `StartApplication` is called, this traffic generator alternates between "On" and "Off" states.
- `PacketSink`. This application is for receiving packets from any other applications, for example, from `BulkSendApplication` or `OnOffApplication`.

3.5.3 Buildings Module

The Buildings module provide various models, but the following are the most relevant for this thesis:

3.5.3.1 Building class

The `Building` model implements and tries to simulate real-life buildings, which affects wireless communications in different ways.

A `Building` can be residential, office or commertial and has different types of external wall (wood, concrete with/out windows, stone blocks), has a number of floors and rooms in each floor.

Some limitations have to be made:

- A `Building` is represented as a rectangular parallelepiped.
- The walls needs to be parallel to the cardinal coordinates.
- A `Building` is a grid of rooms, with z axis as the floor number and the x and y room indexes start from 1 and increses along the x and y axis.
- All the rooms are of the same size.

3.5.3.2 MobilityBuildingInfo class

The `MobilityBuildingInfo` keeps track of the mobility and positional information of the nodes with respect to buildings in a simulation. A node can be inside or outside of a building and if the node is indoors, this class knows in which building and in which room the node is positioned.

3.5.3.3 ItuR1238PropagationLossModel

This class provides an ITU P.1238-based building-dependent indoor propagation loss model that includes losses owing to building type (i.e., residential, office and commercial). The following is the analytical expression:

$$L_{total} = 20\,log\,f + N\,log\,d + L_f(n) - 28[dB] \tag{3.4}$$

where N is the power loss coefficient, L_f is the loss depending of type of building, f is the frequency [MHz] and d is de distance [m].

3.5.3.4 BuildingPropagationLossModel

The `BuildingsPropagationLossModel` adds a set of pathloss model elements that are building-dependent and can be used to design various pathloss logics. The elements of the pathloss model are discussed in the subsections below.

- **External Wall Loss**

  This component simulates the loss of communication from indoors to outdoors and vice versa through walls.

- **Internal Wall Loss**

  This component simulates the loss of penetration in indoor-to-indoor communications within a single building.

- **Height Gain Model**

  This component simulates the gain caused by the transmitting equipment being on a higher floor than the ground.

- **Shadowing Model**

  The shadowing is represented using a log-normal distribution with a variable standard deviation as a function of the `MobilityModel` instances' relative position (inside or outdoor). For each pair of `MobilityModels`, a single random value is generated and remains constant during the simulation. As a result, the model is only suitable for static nodes.

3.5.3.5 Pathloss logics

The pathloss logic provided by inheriting from `BuildingsPropagationLossModel` is described in the following sections.

- **HybridBuildingsPropagationLossModel**

  In order to imitate multiple outdoor and interior circumstances, as well as indoor-to-outdoor and outdoor-to-indoor scenarios, the `HybridBuildingsPropagationLossModel` was created by combining various well-known pathloss models. In particular, this class combines the pathloss models listed below:

- **OhBuildingPropagationLossModel** This is a simpler propagation loss model. It uses the `OkumuraHataPropagationLossModel` and also taking account the pathloss elements of the `BuildingPropagationLossModel`.

3.5.4 Internet Module

This module includes the implementations of TCP/IP related components like IPv4, ARP, UDP, TCP and so on. A `Node` with the Internet Stack installed is called a Internet Node.

In order to use the Internet Protocol, a node should be assigned an IP address. It can be done manually or through the *Dynamic Host Configuration Protocol (DHCP)*.

Full bidirectional TCP with connection setup and close logic is supported by the native *ns-3* TCP model. Various TCP congestion algorithms are also available, such as New Reno, Cubic, HighSpeed, etc. The *TCP* congestion algortihms affects on the adaptation algorithms, but it will not be the focus on this thesis.

3.5.5 Mobility Module

The mobility module in *ns-3* includes model to keep track the position and movement of the nodes and objects in cartesian coordinates and also a number of helper classes used for placing nodes and set up mobility models.

The `MobilityModel` is the base class for all the subclasses for different moving paths or behaviours. The class `PositionAllocator` is typically used for seting the initial position of objects. `MobilityHelper` combines a mobility model and position allocator used for adding mobility capabilities for a set of nodes.

Some useful subclasses of `MobilityModel` are:

- `ConstantPositionMobilityModel` for stationary nodes.
- `ConstantVelocityMobilityModel` for contant velocity moving nodes.
- `RandomWalk2SMobilityModel` for random walking in a 2D plane.

3.5.6 Network Module

The Network Module includes implementations of network related classes like `Packet`, `NetDevice`, TCP and UDP Sockets, etc.

3.5.6.1 Packets

A network packet is compesed by a byte buffer, a group of tags and metadata. The serialized content of the headers and trailers added to a packet is stored in the byte buffer. The serialized form of these headers is expected to match the serialized representation of real network packets bit for bit, implying that the content of a packet buffer is supposed to be the same as that of a real packet.

3.5.6.2 Sockets

A socket is an abstraction that enables applications to communicate with other Internet hosts, among other services, and exchange reliable byte streams and unreliable datagrams.

- **ns-3 Sockets API**

  The native sockets API for *ns-3* provides an interface to TCP and UDP. Although, the `ns3::Socket` have some differences compared to real sockets.

- **Using Sockets**

  In *ns-3*, if an application wants to use sockets must create one first. By calling `CreateSocket`, *ns-3* creates a smart pointer to a `Socket` object. *ns-3* sockets have all the functions of a real socket, including bind, connect, send, receive and close.

  In addition, the `Socket` class can set up events to make callbacks. For example, `SetConnectCallback` is called when a connection is made, whether it succeeded of failed, `SetSendCallback` is invoked when the send buffer is available and `SetRecvCallback` will notify when the data is received.

3.5.7 PointToPoint NetDevice

The *ns-3* point-to-point model simulates a very basic point-to-point data link that connects two `PointToPointNetDevice` devices across a `PointToPointChannel`. This can be compared to a full duplex RS-232 or RS-422 connection with no handshaking and no null modem.

The create point-to-point net devices and channels, `PointToPointHelper` is used. To connect two nodes, simply call the `Install` function.

3.5.8 LTE Module

There are two main components in the LTE-EPC simulation model.

- **LTE Model**. Includes models for the UE and the eNodeB nodes. Also the LTE Radio Protocol Stack (PHY, MAC, RLC, etc.).
- **EPC Model**. Includes models for the entities, interfaces and protocols in the Evolved Packet Core.

3.5.8.1 LteHelper

The `LteHelper` is a helper which manages the LTE radio access network's configuration as well as the setup and release of EPS bearers. The API definition and implementation are both provided by the `LteHelper` class.

A code snipper to create UEs and eNodeBs with `LteHelper` is found in section C.2

- **Network Attachment**

  To connect an UE to the network, the UE needs to be attached to an eNodeB. This is done by calling the `LteHelper::Attach` function. The are two possible ways for network attachment.

- **Automatic Attachment**

  This method uses the strengh of the received signal as the criteria to choose, in the initial cell selection process, which eNodeB to connect to.
- **Manual Attachment** Alternatively, selecting the eNodeB at the beginning of the simulation is also possible.

- **Simulation Output**

  The LTE module offer PHY, MAC, RLC, and PDCP level *Key Performance Indicators (KPI)* that can be enabled using **LteHelper**:

```
lteHelper->EnablePhyTraces ();
lteHelper->EnableMacTraces ();
lteHelper->EnableRlcTraces ();
lteHelper->EnablePdcpTraces ();
```

Listing 3.2: *Enable LTE trace outputs*

3.5.8.2 EpcHelper

The `EpcHelper` allows the simulation of the Evolve Packet Core. The usage of EPC with LTE devices allows for IPv4 and IPv6 networking. To put it another way,it is possible to use standard *ns-3* apps and sockets across IPv4 and IPv6 via LTE, as well as connect an LTE network to any other IPv4 and IPv6 network in the simulation.

It is possible to access the SGW and PGW nodes by calling the `GetSgwNode` and the `GetPgwNode` respectively.

3.5.8.3 MAC

- **MAC Scheduler**

  In *ns-3*, there are several types of MAC schedulers available. User can choose which one to use with the `LteHelper`:

  - FD-MT (Frequency Domain Maximum Throughput Scheduler)
  - TD-MT (Time Domain Maximum Throughput Scheduler)
  - TTA (Throughput to Average Scheduler)
  - FD-BET (Frequency Domain Blind Average Throughput Scheduler)
  - TD-BET (Time Domain Blind Average Throughput Scheduler)

  - FD-TBFQ (Frequency Domain Token Bank Fair Queue Scheduler)
  - TD-TBFQ (Time Domain Token Bank Fair Queue Scheduler)
  - PSS (Priority Set Scheduler Scheduler)

- **AMC & CQI**

  In terms of selecting MCSs and generating CQIs, the simulator offers two options. The first is the implementation by [16] and operates on a per-RB basis, and the other is based on the physical error mode.

3.5.8.4 Mobility Model with Buildings

The propagation model to be used with the LTE module is defined in the Buildings module.

This creates a residential building, concrete with windows, with 3 floors and 6 rooms each floor. It is set that all UEs are in a constant position, but other mobility models are also possible.

The LTE module is also compatible with existing propagation loss models. Only the propagation from the UEs to the base stations are computed.

3.5.8.5 AntennaModel & MIMO Model

Any model of `AntennaModel` is supported, by default, the `IsotropicAntennaModel` is used for both eNBs and UEs. In case of using multiple antennas, *ns-3* offers different MIMO operation modes.

3.5.8.6 Radio Environment Maps

With this class is possible to create a Radio Environment Map (REM), which is a uniform 2D grid of values that reflect the signal-to-noise ratio in the downlink with regard to the eNB with the strongest signal at each point, to a file by using the class `RadioEnvironmentMapHelper`.

Using a software like gnuplot[1], the output file can be visualized.

Figure 3.3 shows an example of a Radio Environment Map.

[1] http://www.gnuplot.info/

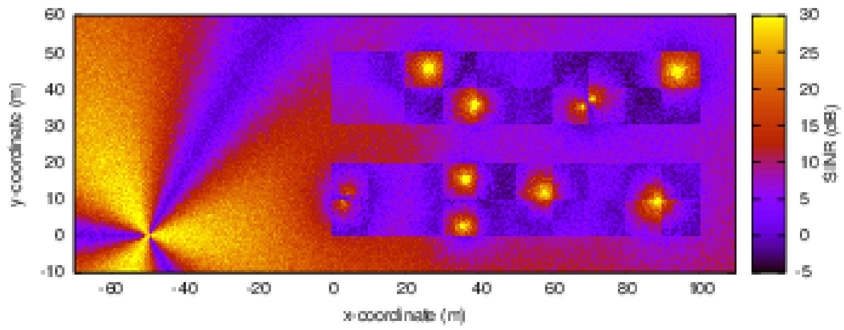

Figure 3.3: *Example Radio Environment Map. Source: [25]*

3.6 Parameter Configuration

The *ns-3* attribute system is the entity that manages all the parameters. It is possible to use input files using `ConfigStore` and set initial values for default and global paramaters.

It is important to include `#include "ns3/config-store.h"` in the script. Then create a text file named as defined before and specify the new default values to be used.

Chapter 4 | ABR Module for ns-3

This chapter will introduce a new module for *ns-3* for ABR streaming simulation. The section 4.1 will set the objective and the scope of the design. The section 4.2 will present the architecture of the module. The section 4.3 will go over the models the module is composed of. Finally, the section 4.4 will explain the adaptation algorithms implemented in this module.

4.1 Design Objectives

The main objective of this chapter is to design and implement a *ns-3* module able to simulate the behavior of video streaming devices in mobile network scenarios. To build a framework capable of testing new adaptation algorithms and be possible to extract quality of service and quality of experience metrics.

4.2 Architecture

The ABR module provides:

- **`AbrClient`**. This class mimics a video streaming player. It has an instance of `AbrAlgorithm`, which is responsible of deciding which is the multimedia representation that best fits the available bandwidth to download from the `AbrServer`. It is an implementation of `ns3::Application`.
- **`AbrServer`**. This class simulates a video streaming HTTP server. It receives requests from clients and sends the multimedia segments requested. It is an implementation of `ns3::Application`.

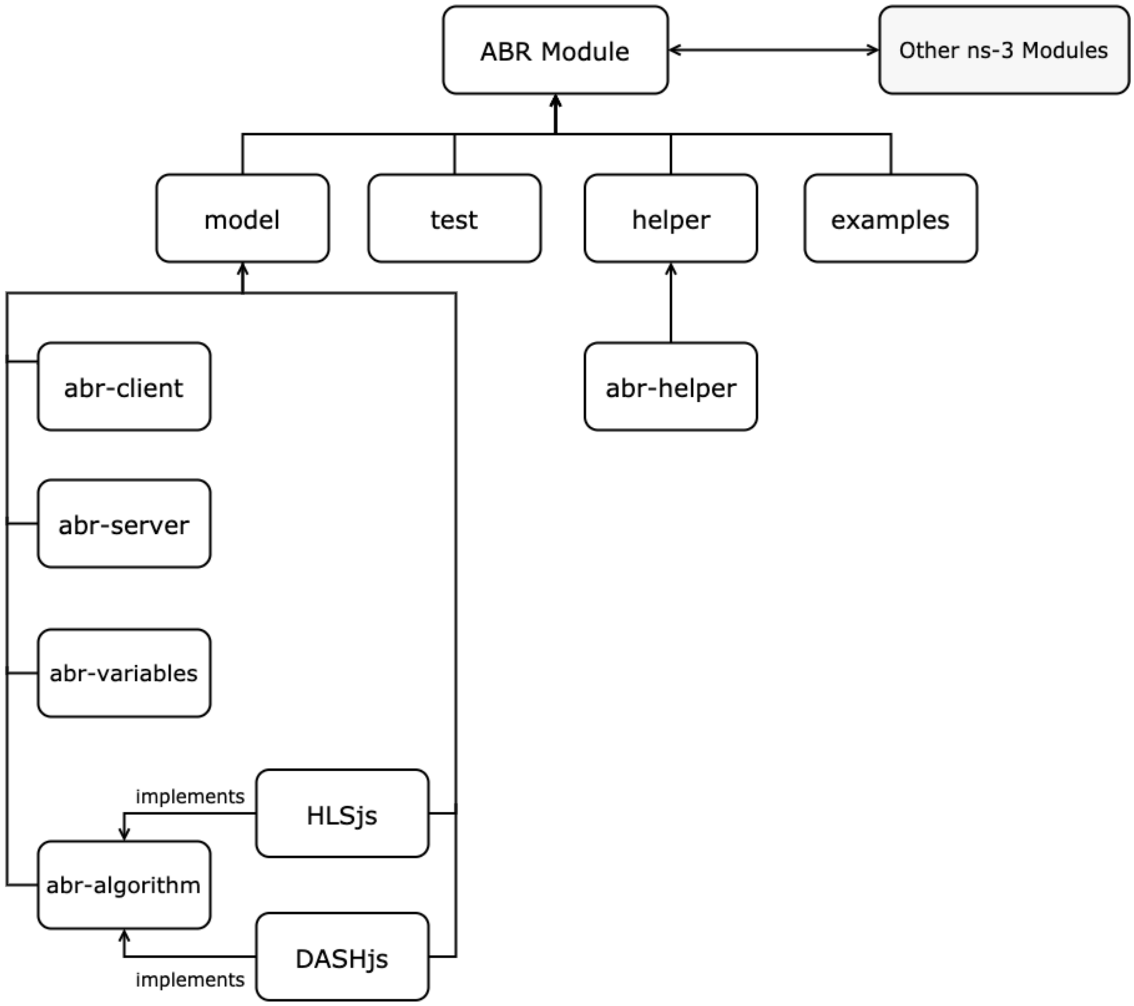

Figure 4.1: *ABR Module architecture*

- **`AbrVariables`**. This class is used for storing common variables between the *DASH* clients and the servers. It contains the definition of Segment, Representation, etc.
- **`AbrHelper`**. This is a Helper class for the ABR module. It is the responsible for managing the instances of the ABR clients and servers. In addition, `AbrHelper` can be used for extracting QoS and QoE metrics.
- **`AbrAlgorithm`**. This is a base class as an interface for the different adaptation algorithms implementations.
- **`HLSjs`**. This is an implementation of `AbrAlgorithm` based on *hls.js* [32].
- **`DASHjs`**. This is an implementation of `AbrAlgorithm` based on *dash.js* [10].
- **`abr-example.cc`**. An basic example script with two nodes linked with a `PointTo-Point` connection and a unstable connection.

The Figure 4.1 shows the architecture of the ABR module. Although this module was designed to be used in mobile environments, it can be used with any other `Application` class in *ns-3*, meaning that the ABR clients and servers can be installed in any `Node` and work with other *ns-3* modules and models.

4.3 Models

This section will go through all the models, classes and helpers in the ABR module and how they work together.

4.3.1 AbrClient

The `AbrClient` is an implementation of `ns3::Application`. This class uses an implementation of `AbrAlgorithm` to create HTTP-like requests to the `AbrServer` and mimics the playing of the media content.

The `AbrClient` is created with the `AbrHelper` and the server address and listening port as parameters. The client application needs to be installed on the client `Nodes`. When the simulation starts, the function `StartApplication` is called and the simulator is scheduled to call the client's `HandlePlay` function to simulate video watching. The client will create a new socket, in this case a TCP socket, to connect with the server. The socket is set with various callback functions:

- `ConnectionSucceded`. is called if the connection succeeded. Then it calls the `CheckAlgorithm` function.
- `ConnectionFailed`. is called if the connection failed. This should not happen if the simulation script is correctly written.
- `HandleRead`. is called when new packets are received. It stores the segments to the segment buffer, and checks the adaptation algorithms after one entire segment is downloaded.

The `CheckAlgorithm` method asks the `AbrAlgorithm` and returns one or more `Abr-Tasks`. The client will call the scheduled functions depending on the designated task and delay.

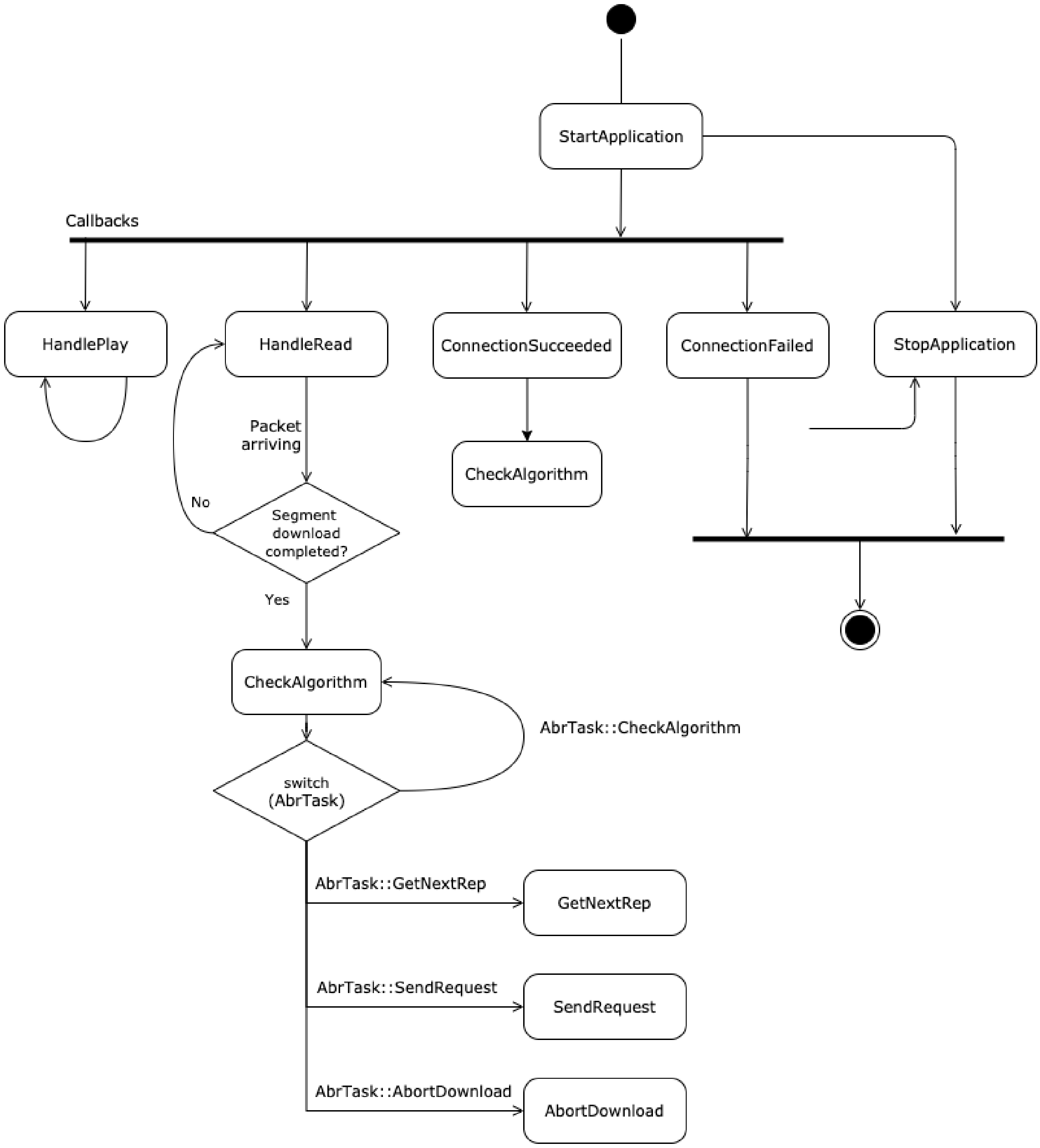

Figure 4.2: *ABR Client*

4.3.2 AbrServer

The `AbrServer` is an implementation of `ns3::Application`. This class receives HTTP-like requests from the `AbrClient` and sends the requested segment.

The request is in the format:

`GET qualityIndex startSegment`

For example, "`GET 4 3`" means "GET 1 segment of quality index 4 starting from the 3^{rd} segment".

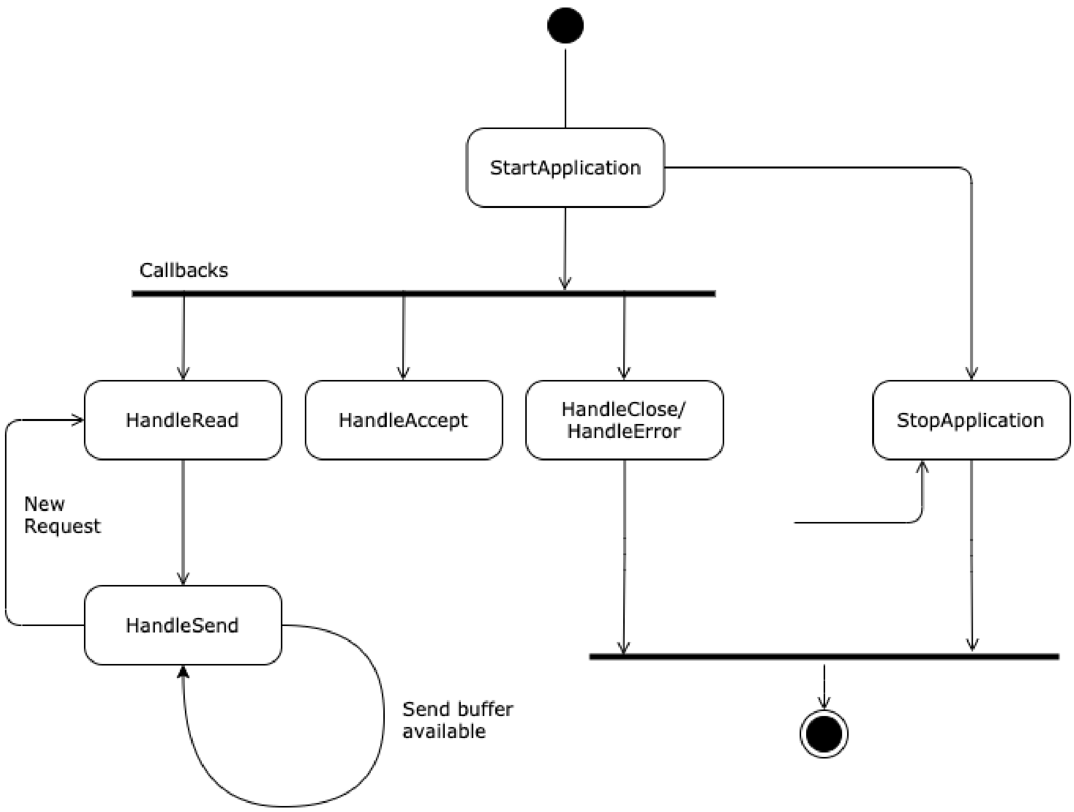

Figure 4.3: *ABR Server*

The `AbrServer` is created with the `AbrHelper` and the listening port as the parameter. The server application needs to be installed on the server `Node`. When the simulation starts, the function `StartApplication` is called. The server will create and bind a socket, and starts to listen. The socket is set will callback functions. When the sockets are connected, the server will schedule the callbacks to handle reading requests and sending data.

In the `HandleRead` function, the server reads the requests from the clients. As an example, for a certain request, the `AbrServer` gets the segment size from `AbrVariables` and suppose that the segment size is 2 megabytes, the `AbrServer` sends 2MB of dummy data, since there is not actual multimedia data, to the `AbrClient`.

4.3.3 AbrVariables

The `AbrVariables` class contains variables and functions used by the `AbrClient` and `AbrServer`, including the definition of a set of essential data structures. These data structures are:

- **`Segment`**. It is an abstraction of a media segment. A `Segment` has a size (in bytes), a start time and a duration.
- **`Representation`**. Describes a certain version of a encoded media. A `Representation` include the resolution, the frames per second and the encoding bitrate.
- **`SegmentInfo`**. This is a aditional data structure for `Segment`. A `SegmentInfo` contains information about download start/finish time, playback start/finish time, the bandwidth estimation used to download that segment and the quality index of the segment.
- **`PlayerStates`**. This class keeps track on the player status.
- **`AbrTask`**. An `AbrTask` is a used to schedule tasks for `AbrClients`.

`AbrVariables` has these variables:

- **`m_segments`**. It is a two-dimentional vector containing all the segments for the simulation. Each row has de the same quality and the higher the row index, the higher the quality. The segments are ordered in time in the columns.
- **`m_representations`**. It is a vector containing information about the resolution, framerate and bitrates of the `Representations`. The row index also means the quality level.
- **`m_segmentDuration`**. The duration of the segment in milliseconds. By default, it is $2000ms$.

Before the simulation starts, the `AbrVariables` class initializes the variables. Starting with the representations, there are a predefined set of `Representations` by default, but they can be changed in the source file. Continuing with the segments, their sizes are calculated based on the resolution, framerate and the encoding bitrate for each representation.

Also, the possibility of creating a MPD file parser has been considered, but it can be done in the future as an improvement.

4.3.4 AbrHelper

`AbrHelper` are helper classes providing the functionality of managing the ABR clients and servers (creating, setting attribute, etc.). There are two classes, `AbrServerHelper` and `AbrClientHelper`. The `AbrClientHelper` have methods to extract QoE metrics after the simulation ends.

4.3.5 AbrAlgorithm

`AbrAlgorithm` serves as the base class for the implementations of adaptation algorithms. In the next section, two implementations of `AbrAlgorithm` are presented.

4.4 Adaptation Algorithms

This section will present two implementation of `AbrAlgorithm`. The first one is based on the JavaScript library implementation of *HTTP Live Streaming (HLS) hls.js*[1] client [32]. The second implementation is based on the *dash.js* [2] from the *DASH Industry Forum* [10].

4.4.1 HLSjs.cc

This class is based on the implementation from a open-source JavaScript-based project called *hls.js*. The `HLSjs.cc` class has some simplifications compared to the original library.

[1] *hls.js* will refer to the original JavaScript Library while `HLSjs.cc` will refer to the *ns-3* implementation

[2] *dash.js* will refer to the original DASH implementation while `DASHjs.cc` will refer to the *ns-3* implementation

hls.js has two main rules and some aditional secondary rules. These rules are:

- Main Rules
  - **Bandwidth Estimation**. This is the main rule, which is an ABR adaptation algorithm rule explained in the subsection 2.2.2.
  - **Abort Rules**. These are a set of rules to abort a segment download depending on some coditions, for example, a timeout for a segment to download.
- Secondary Rules
  - **Screen & player size cap level**. This rule is used at the beginning to cap the highest level of representation to the device capabilities. For instance, there is no need for a FHD device to play 4K videos in most cases.
  - **Dropped frames per second**. This rule is triggered if the cpu cannot handle the decoding of the multimedia content and produces too much dropped frames.

`HLSjs.cc` will focus only on the Bandwidth Estimation rule.

- **`BandwidthRule`**. This is the implementation of a EWMA based adaptation algorithm. The Listing 4.1 show a pseudocode of the algorithm.

4.4.2 DASHjs.cc

This class is based on the implementation build by the *DASH Industry Forums* with the *dash.js* name. `DASHjs.cc` is a simplified version of *dash.js*. See section C.3 for more details.

dash.js works with a combination of rules. Each rule returns a `SwitchRequest`. A `SwitchRequest` is an object that indicates the next representation to request, the request priority, etc. The priorities of the `SwitchRequest` can be `NO_CHANGE`, `DEFAULT`, `STRONG` or `WEAK`.

If more than one `SwitchRequest` is created, the `GetMinSwitchRequest` is called. It always considers the request with the highest priority and the quality with the minimum difference compared to the current representation.

`DASHjs.cc` has two rules implemented:

- **`ThroughputRule`**. This is the implementation of a EWMA based adaptation algorithm. It is very similar to the *hls.js* Bandwidth estimation rule.

```
if First Segment then
  nextQuality ← 0;
  return true;
end if
if Enough segments in buffer then
  newSample ← estimation of last segment;
  if fastEWMA is 0 or slowEWMA is 0 then
    fastEWMA ← newSample;
    slowEWMA ← newSample;
  else
    fastEWMA ← newSample × α_fast + fastEWMA × (1 − α_fast);
    slowEWMA ← newSample × α_slow + slowEWMA × (1 − α_slow);
  end if
  averageBw ← min(slowEWMA, fastEWMA);
else
  averageBw ← current estimation;
end if
for i = representations.size - 1 → 0 do
  if i < current quality then
    adjustedBw ← bwFactor × averageBw;
  else
    adjustedBw ← bwUpFactor × averageBw;
  end if
  if adjustedBw > representations[i].bitrate then
    nextQuality ← i;
    return true;
  end if
end for
return false;
```

Listing 4.1: *HLSjs.cc Bandwidth Rule*

- **`BolaRule`**. This is a simplified implementation of the buffer based algorithm BOLA introduced in subsubsection 2.2.2.2.

  BOLA has three states:

  - `BOLA_STATE_ONE_BITRATE`. This is the state when there is only one bitrate available.
  - `BOLA_STATE_STARTUP`. This is the initial state of BOLA.
  - `BOLA_STATE_STEADY`. This is the state when the buffer is really for using BOLA.

The main methods of BOLA is `BolaRule` and `GetQualityFromBufferLevel`. This last method uses a score calculated, using BOLA's parameters such as playback utility and playback smoothness, for each representation and chooses the representation with the highest score.

Chapter 5 | Simulations and Results

This chapter will test the adaptation algorithms implemented in the *ABR* module from the chapter 4 in various simulated scenarios. The section 5.1 will present the metrics used for the comparisons. The section 5.2 will go through all the simulation scripts and its results, analyse the performance and fairness of the adaptation algorithms. Finally, the section 5.3, will discuss the conclusions and limitations of the *ABR* module.

5.1 Comparison Metrics

For the comparison, the metrics introduced in the subsection 2.2.3 will be used. In addition, other metrics will be used that may also be of interest. And they are as follows:

- **Average Throughput**. The average of the network throughput.
- **Playback start time**. The time the client takes to start the playback.
- **Total time watched**. Total time watched by the client.
- **Quality switches**. The number of times the representation changed.
- **Paused times**. The number of times the playback paused.
- **Time at each quality**. The time spent at each quality.
- **Buffer status**. The buffer status in milliseconds of content as a function of time.
- **QoE Score**. This is a score based on various metrics.
  The QoE score is defined for this thesis in order to compare the different algorithms. It ranges from 0 to 1, the greater the score the better the performance. The formula is as follows:

$$QoE\ score = \frac{t_w - pb_s - \sum_{i=1}^{M} \frac{t_i}{2} \cdot (\frac{1}{2} - \frac{i+1}{M}) - \frac{1}{2} \cdot (qs + pt)}{simulation\ time} \tag{5.1}$$

  with

- t_w Time watched
- pb_s Playback start time
- $t_{i\ q=\{0,1,...,M\}}$ Time at each quality
- qs Quality switches
- pt Paused times

5.2 Scenarios

In the next sections, the throughput rule of *dash.js* will be refered as *DASH throughput*, the BOLA rule as *BOLA*, the bandwidth estimation of *hls.js* as *HLS* and the combination of the BOLA and the throughput rule of *dash.js* simply as *DASH*.

5.2.1 Scenario 1

This section will take a look a basic network scenario. To be as simply as possible, the will be only two nodes linked with a `PointToPoint` connection. The simulation time is 50 seconds and the datarate will vary in time. In this scenario, seven bitrates are used.

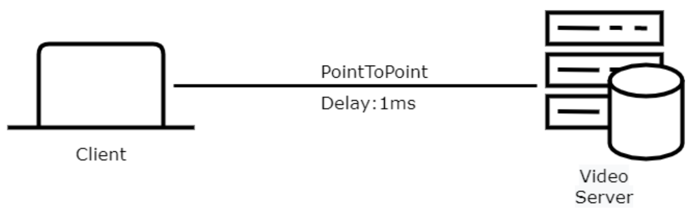

Figure 5.1: *Scenario 1*

By looking at the Figure 5.2, the first thing noticeable is that the *HLS* and the *DASH throughput*'s quality functions are almost the same line. *BOLA* reached a little bit earier to the quality index 5 and sustained a good overall performance. The combination of both rules in *DASH* seems to work pretty nicely, with one exception of going down to quality index 2 almost at the end of the simulation.

The Figure 5.3 show a *Cumulative Distribution Function (CDF)*. In this case, the *CDF* shows the video quality choice statistics in the form of percentages. Both *HLS* and *DASH*

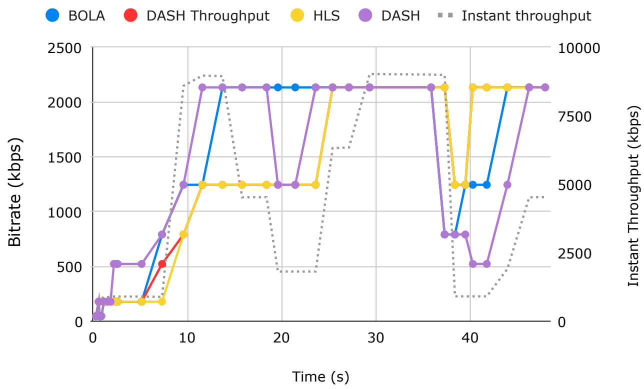

Figure 5.2: *Scenario 1. Quality vs time*

throughput obteined a higher quality during most of the simulation time and never played segments from the lowest quality.

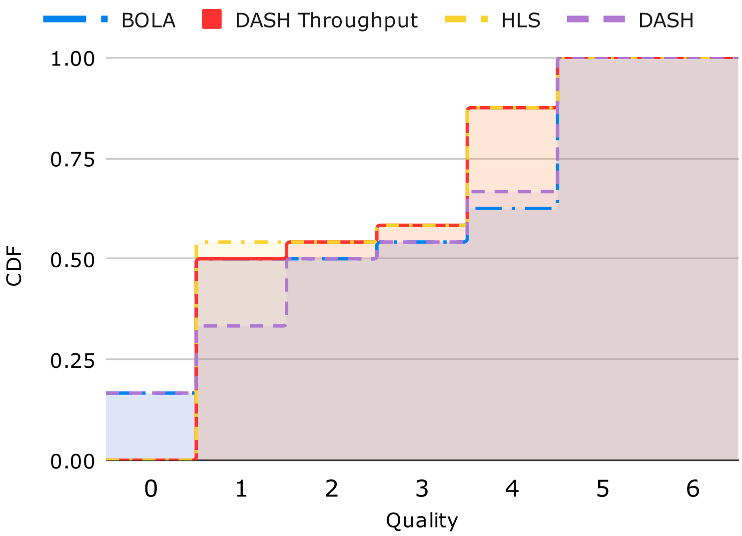

Figure 5.3: *Scenario 1. CDF quality*

5.2.2 Scenario 2

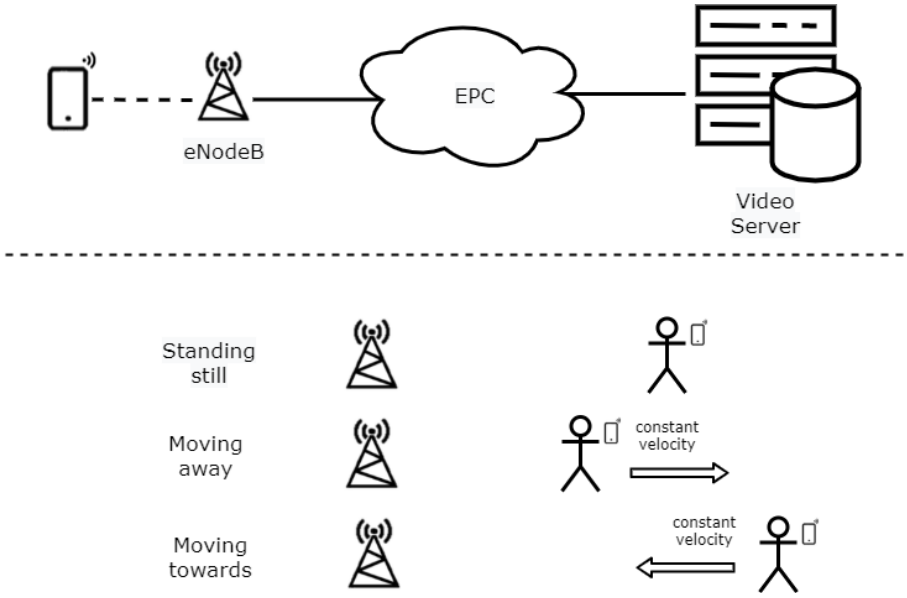

Figure 5.4: *Scenario 2*

This scenario will find out what will happen if an *UE* stands still, or if it is moving away or towards the *eNodeB*. The only changing factor is the position of the *UE*, the rest of the parameter remains the same. There are only one *eNodeB*, meaning there is no possibility to handover.

This setup will be using only the *HLS* algorithm. In the first simulation, the *UE* will be placed at a reasonable distance from the *eNodeB*. In the second one, the *UE* is moving with a constant velocity away from the *eNodeB*. Finally, the *UE* is placed a little bit further and walks towards the *eNodeB*.

The Figure 5.5 shows the buffer status as a function of time. What moving away from the *eNodeB* really does is to decrease the available bandwidth and finally the *UE* will be unable to request a new segment.

Within the wireshark statistics tools, by selecting the throughput option within *TCP* flows, it is possible to display throughput graphs of the *UEs*. In the Figure 5.6, the first graph show that the *UE* approaching the *eNodeB* performs relatively well. By contrast, the throughput of the second *UE* drops significantly every time it goes further from the *eNodeB*. All these behaviours are reflected in the Table 5.1 with some *QoE* metrics.

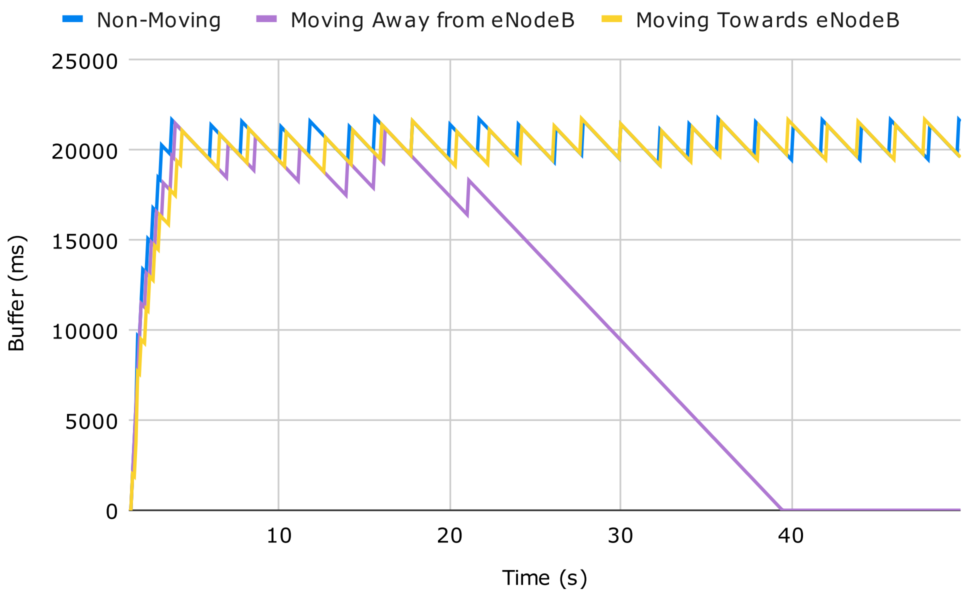

Figure 5.5: *Scenario 2. Buffer Status*

	Non-Moving	Moving Away	Moving Towards
QoE Score	0.812078	0.535326	0.701826
Average throughput (Mbps)	23.0487	8.1720	12.7028
Time watched (s)	48.9	38.0	48.485
Quality switches	7	12	13

Table 5.1: *Scenario 2. Metrics Comparison*

5.2.3 Scenario 3

This section provides an analysis of a *LTE* scenario with poor conditions. There will be six *UEs* watching video at the same time. three of them remain stationary and the remaining three are randomly moving. In this scenario, fifteen bitrates are used. The only changing factor is the position of some *UEs*, the rest of the parameter remains the same. There are only one *eNodeB*, meaning there is no possibility to handover.

The scenario has one *eNodeB*, with one antenna, and a residential building. The Figure 5.7 shows a radio enviroment map of the scenario.

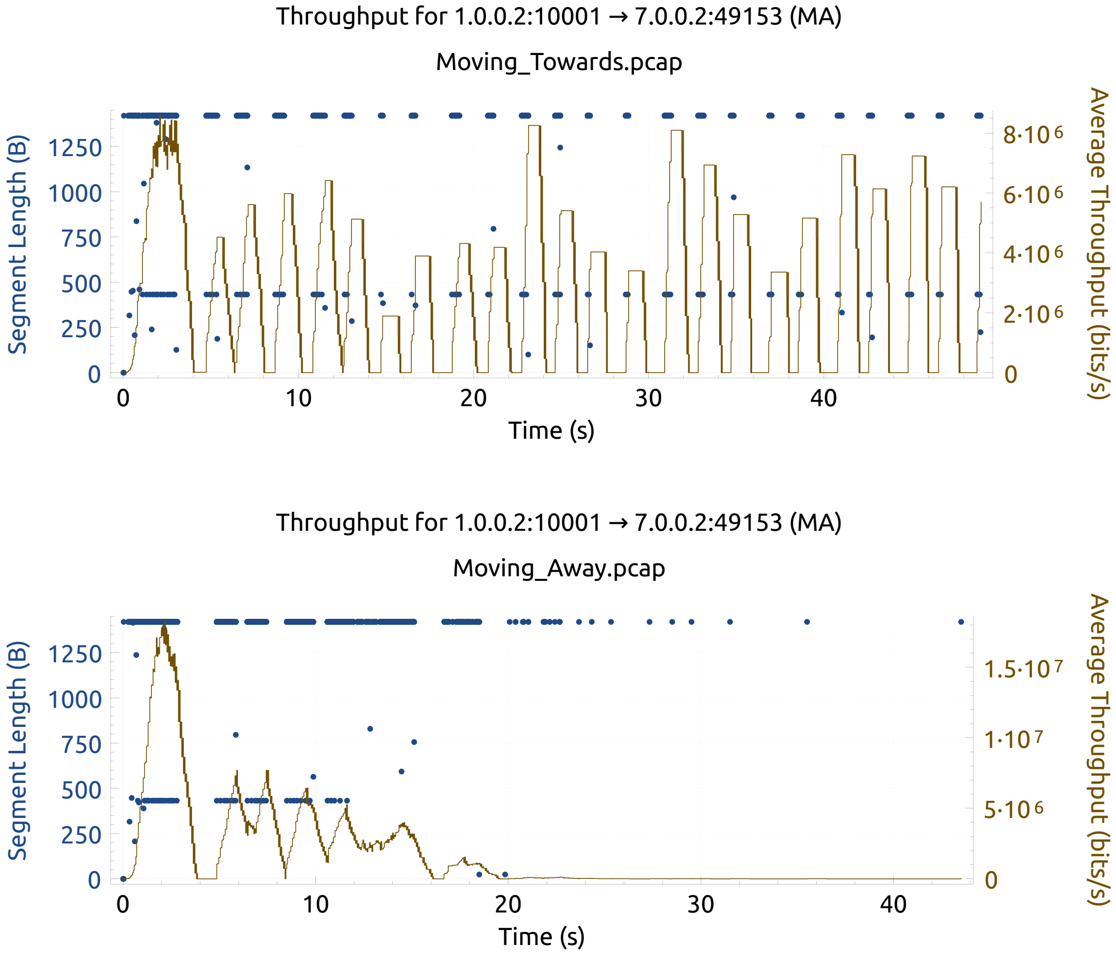

Figure 5.6: *Scenario 2. Throughput*

With the results shown in the Table 5.2, we conclude that on average, the bandwidth based algorithms have higher QoE scores. But the other two algorithms have less difference between their highest and lowest scores.

5.2.4 Scenario 4

5.2.4.1 Jain Fairness Index

Fairness metrics are used to determine whether the adaptation algorithm is able to deliver an equitable share of the network bandwidth to different clients. In this analysis, the *Jain Fairness Index (JFI)* [21] is used. The *JFI* is calculated by this formula:

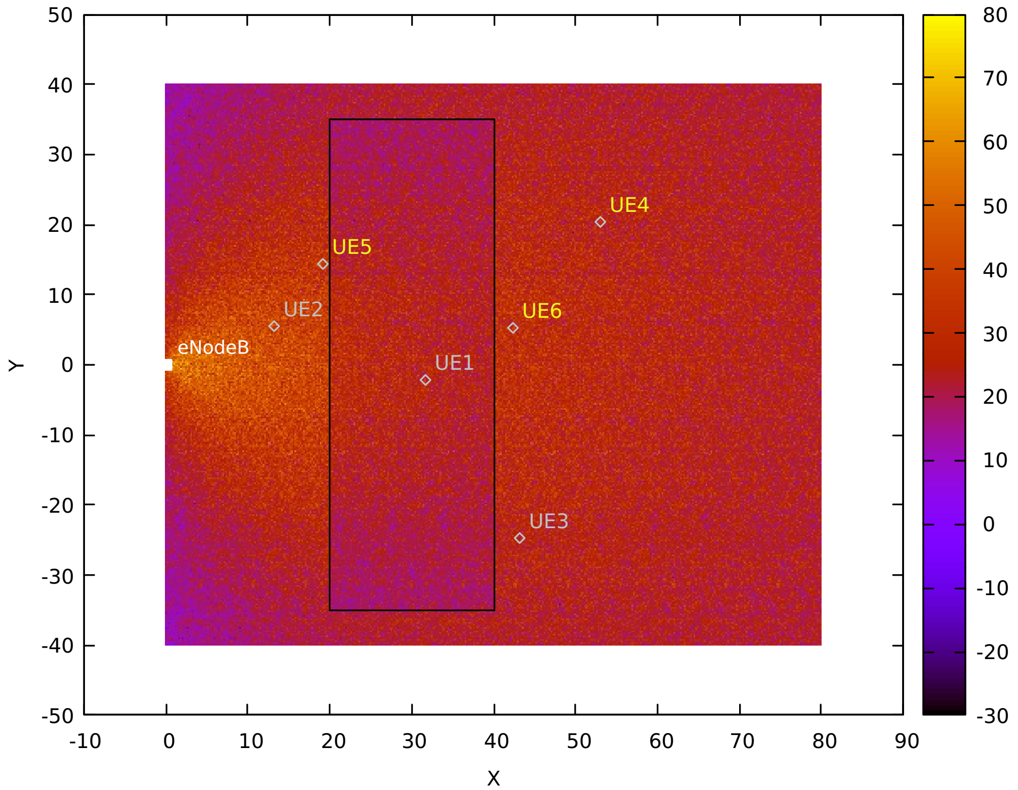

Figure 5.7: *Scenario 3. Radio Environment Map*

$$JFI = \frac{\left(\sum\limits_{c \in C} \bar{B}\right)^2}{|C| \sum\limits_{c \in C} (\bar{B})^2} \tag{5.2}$$

where C are the Clients and $\bar{B}$ are the Average Throughputs

This testing scenario uses ten *UEs* and one *eNodeB* at a time in a *LTE* environment.

Based on the results presented in the Table 5.3, the throughput based algorithms have the least fairness index. That is as expected seeing the results of the last scenario because they had greater differences of performance between *UEs*. That is also true in this case, with *HLS* having the single highest throughput. Nevertheless, all the the algorithms are in a very resonable range, greater than 0.9.

		BOLA	DASH	DASH throughput	HLS
	Average	0.3222501667	0.3389108333	0.3891433333	0.3908161667
QoE Score	Max	0.47895	0.568955	0.621531	0.631055
	Min	0.245517	0.245722	0.220542	0.231055

Table 5.2: *Scenerio 3. QoE Score*

	BOLA	DASH	DASH throughput	HLS
JFI	**0.972185193**	0.9716453768	0.9271291449	0.929252551
MAX	10289.8	10289.8	19953.9	**20006.4**

Table 5.3: *Scenerio 4. Fairness Comparison*

5.3 Conclusions

These testing scenarios are only a small part of what the *ABR* module can do with *ns-3*. All the implemented adaptation algorithms are compered in some interesting cases. But more importantly is to show the potential of this module and also the improvement that can be made.

Due to limitations with *ns-3*, the BOLA implementation is not complete. There some bugs like enabling REM will crush the simulation but the output of the REM is done without problems. Or sometimes the simulations stops caused by some internal data structure corruption.

The implementation of BOLA was a really challenging task for this project, and its complexity translates, in some cases, in better performance.

Chapter 6 | Conclusions and Future Work

6.1 Conclusions

6.2 Future Work

Bibliography

[1] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Technical Specifications and Technical Reports for a GERAN-based 3GPP system (Release 8). Technical Report 3GPP TS 41.101, 2009.

[2] 3GPP. *Transparent End-to-End Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH).* 3GPP TS 26.247 v16.4.1, 2020.

[3] Adobe. Live Streaming. `https://www.adobe.com/es/products/hds-dynamic-streaming.html`.

[4] Apple. HTTP Live Streaming. `https://developer.apple.com/streaming`.

[5] Benny Bing. *Next-generation video coding and streaming.* Wiley, 1st edition, 2015.

[6] Julián Cabrera. *Apuntes de Sistemas y Servicios de Multimedia (MUIT).* Universidad Politécnica de Madrid, 2020.

[7] Lyn Cantor. The global internet phenomena report covid-19 spotlight. Technical report, Sandvine, 2020.

[8] Cisco. Cisco predicts more ip traffic in the next five years than in the history of the internet. `https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935`, 11 2018.

[9] DASH-IF. DASH Industry Forum. `https://dashif.org/`.

[10] DASH-IF. dash.js. `https://github.com/Dash-Industry-Forum/dash.js`.

[11] Universidad Politécnica de Valencia. Vídeo adaptativo a través de MPEG-DASH. `https://www.comm.upv.es/es/dash/`.

[12] Streaming Media East. Adapting your player logic to your use case: how to use ABR to your advantage. `https://es.slideshare.net/EricaBeavers/abr-algorithms-2017-streaming-media-east`, 2017.

[13] electronicsnotes. LTE EARFCN Radio Channel Numbers. `https://www.electronics-notes.com/articles/connectivity/4g-lte-long-term-evolution/lte-earfcn-radio-channel-numbers.php`.

[14] International Organization for Standardization. *Information technology — Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats.* International Organization for Standardization, ISO/IEC 23009-1:2019(E) edition, 2019.

[15] William. E. Gibson. First Cellular Phone Call Was Made 45 Years Ago. `https://www.aarp.org/politics-society/history/info-2018/first-cell-phone-call.html`, 2018.

[16] Giuseppe Piro, Nicola Baldo, Marco Miozzo. *An LTE Module for the ns-3 network simulator.* CTTC Catalan Telecommunications Technology Centre, 2011.

[17] IETF. RFC 8216 - HTTP Live Streaming - IETF Tools. `https://tools.ietf.org/html/rfc8216`, August 2017.

[18] ISO. MEPG-DASH. `http://www.iso.org/iso/home/standards.htm`.

[19] ITU-T. *Recommendation E.800 : Definitions of terms related to quality of service.* ITU, 2017.

[20] ITU-T. *Recommendation P.10/G.100: Vocabulary for performance, quality of service and quality of experience.* ITU, 2017.

[21] Raj Jain, Dah-Ming Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *CoRR*, cs.NI/9809099, 1998.

[22] Microsoft. Silverlight. `https://www.microsoft.com/silverlight/smoothstreaming/`.

[23] Christopher Mueller. Microsoft smooth streaming. `https://bitmovin.com/microsoft-smooth-streaming-mss/`, 2015.

[24] Rias Muhamed, Arunabha Ghosh, Jun Zhang, and Jeffrey G Andrews. *Fundamentals of LTE.* Prentice Hall Communications Engineering and Emerging Technologies Series from Ted Rappaport. Pearson, 2010.

[25] ns 3. A Discrete-Event Network Simulator. `https://www.nsnam.org/`.

[26] Miguel Ángel Aguayo Ortuño. Contribución a los mecanismos de adaptación dinámica para servicios de distribución multimedia sobre redes móviles. December 2020.

[27] Jane Pitkethly. Cumulative distribution functions. `http://amsi.org.au/ESA_Senior_Years/SeniorTopic4/4e/4e_2content_2.html`.

[28] Iraj Sodagar. White paper on MPEG-DASH Standard. MPEG-DASH: The Standard for Multimedia Streaming Over Internet. Technical report, ISO/IEC, 2012.

[29] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[30] Techplayon. LTE FDD System Capacity and Throughput Calculation. `https://www.techplayon.com/lte-fdd-system-capacity-and-throughput-calculation/`.

[31] Luis Mendo Tomas. *Apuntes de Comunicaciones Móviles (GITST)*. Universidad Politécnica de Madrid, 2018.

[32] video dev. HLS.js. `https://github.com/video-dev/hls.js`.

[33] Hubregt J. Visser. *Antenna theory and applications*. 2012.

[34] SeungJune Yi, SungDuck Chun, YoungDae Lee, SungJun Park, and SungHoon Jung. *Radio Protocols for LTE and LTE-Advanced*. O'Reilly Media, Inc., 2021.

Appendix A | Impact

A.1 Social Impact

A.2 Economic Impact

A.3 Ambiental Impact

A.4 Ethic Impact

Appendix B | Budget

Appendix C | ns-3

C.1 Getting Started

The prerequisites for the *ns-3* release version 3.32 are the following tools:

Prerequisite	Package/version
C++ compiler	`clang++` or `g++` (`g++` version 4.9 or greater)
Python	`python3` version >= 3.5
Git	any recent version
tar	any recent version
bunzip2	any recent version

Table C.1: *Prerequisites for ns-3*

Start by downloading the source archive from nsnam or gitlab. Then build *ns-3* with `build.py`:

```
# Download from nsnam
$ cd
$ mkdir workspace
$ cd workspace
$ wget https://www.nsnam.org/release/ns-allinone-3.32.tar.bz2
$ |\color{myblue}tar| xjf ns-allinone-3.32.tar.bz2
$ cd ns-allinone-3.32
# Building ns-3
$ ./build.py --enable-examples --enable-tests
# Running a script
# Create or copy a script to the scratch directory
```

```
$ cp examples/tutorial/first.cc scratch/myfirst.cc
$ ./waf --run scratch/myfirst
```

Listing C.1: Download and installation of ns-3

Logging Module

Enable logging:

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func|
    prefix_time:UdpEchoServerApplication=level_all|prefix_func|
    prefix_time'
```

Listing C.2: Enabling logging in ns-3

To disable logging simply type:

```
$ export NS_LOG=
```

Listing C.3: Disabling logging in ns-3

Command Line Arguments

An example of a command could be like this:

```
# To check help
$ ./waf --run "scratch/myfirst --PrintHelp"
# To check variables for PointToPointNetDevice
$ ./waf --run "scratch/myfirst --PrintAttributes=ns3::
    PointToPointNetDevice"
# We set the Datarate to 5Mbps
$ ./waf --run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5
    Mbps"
```

Listing C.4: Command line arguments

ASCII Tracing

To enable ASCII Tracing, right before the call to `Simulator::Run ()`, add the following lines of code:

```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("out.tr"));
```

Listing C.5: ASCII tracing

This will generate the output from `pointToPoint` to a file named `out.tr`.

PCAP Tracing

To enable *pcap* tracing simply add:

```
pointToPoint.EnablePcapAll ("myfirst");
```

Listing C.6: *PCAP tracing*

Sockets

Creating a socket:

```
Ptr<Node> node;
// Create a TCP socket
Ptr<Socket> mySocket = Socket::CreateSocket (node, TcpSocketFactory::
    GetTypeId ());
// Functions
mySocket->Bind();
mySocket->Connect( ... );
mySocket->Send( ... );
mySocket->Recv( ... );
mySocket->Close();
```

Listing C.7: *ns-3 Socket programming*

For callbacks:

```
mySocket->SetConnectCallback (
  MakeCallback (&MyClass::ConnectionSucceeded, this),
  MakeCallback (&MyClass::ConnectionFailed, this)
);
mySocket->SetSendCallback (MakeCallback (
  &MyClass::HandleSend, this));
mySocket->SetRecvCallback (MakeCallback (
  &MyClass::HandleRead, this));
```

Listing C.8: *Socket callbacks*

PointToPoint NetDevice

```
NodeContainer n;
n.Create (2);
PointToPointHelper p2ph;
```

```
p2ph.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
p2ph.SetChannelAttribute ("Delay", StringValue ("5ms"));
NetDeviceContainer devs = p2ph.Install (n);
```

Listing C.9: *PointToPointHelper*

C.2 LTE Module

LteHelper

```
// Create LteHelper and the nodes
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
NodeContainer enbNodes;
enbNodes.Create (1);
NodeContainer ueNodes;
ueNodes.Create (2);

// Set the mobility model
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (enbNodes);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (ueNodes);

// Install NetDevices to the nodes
NetDeviceContainer enbDevs;
enbDevs = lteHelper->InstallEnbDevice (enbNodes);
NetDeviceContainer ueDevs;
ueDevs = lteHelper->InstallUeDevice (ueNodes);

// Attach UEs to the eNodeB
lteHelper->Attach (ueDevs, enbDevs.Get (0));
```

Listing C.10: *LteHelper usage*

Network Attachment

```
lteHelper->Attach (ueDevs); // attach one or more UEs to a strongest
    cell
lteHelper->Attach (ueDevs, enbDev); // attach one or more UEs to a
```

```
    single eNodeB
```

Listing C.11: UE Automatic Attachment

EpcHelper

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
Ptr<PointToPointEpcHelper> epcHelper = CreateObject<
    PointToPointEpcHelper> ();
lteHelper->SetEpcHelper (epcHelper);

// To access PGW or SGW
Ptr<Node> pgw = epcHelper->GetPgwNode ();
Ptr<Node> sgw = epcHelper->GetSgwNode ();
```

Listing C.12: Enable Evolved Packet Core

MAC

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
lteHelper->SetSchedulerType ("ns3::FdMtFfMacScheduler");    // FD-MT
    scheduler
lteHelper->SetSchedulerType ("ns3::TdMtFfMacScheduler");    // TD-MT
    scheduler
lteHelper->SetSchedulerType ("ns3::TtaFfMacScheduler");     // TTA
    scheduler
lteHelper->SetSchedulerType ("ns3::FdBetFfMacScheduler");   // FD-BET
    scheduler
lteHelper->SetSchedulerType ("ns3::TdBetFfMacScheduler");   // TD-BET
    scheduler
lteHelper->SetSchedulerType ("ns3::FdTbfqFfMacScheduler");  // FD-TBFQ
    scheduler
lteHelper->SetSchedulerType ("ns3::TdTbfqFfMacScheduler");  // TD-TBFQ
    scheduler
lteHelper->SetSchedulerType ("ns3::PssFfMacScheduler");     //PSS
    schedulerUIntegerValue(yourvalue));
```

Listing C.13: MAC Scheduler

AMC & CQI

```
// Piro
Config::SetDefault ("ns3::LteAmc::AmcModel", EnumValue (LteAmc::
    PiroEW2010));
// Physical error model
Config::SetDefault ("ns3::LteAmc::AmcModel", EnumValue (LteAmc::
    MiErrorModel));
```

Listing C.14: AMC Model

Building

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

Ptr<Building> b = CreateObject <Building> ();
// Box (xmin, xmax, ymin, ymax, zmin, zmax)
b->SetBoundaries (Box (0.0, 10.0, 0.0, 20.0, 0.0, 20.0));
b->SetBuildingType (Building::Residential);
b->SetExtWallsType (Building::ConcreteWithWindows);
b->SetNFloors (3);
b->SetNRoomsX (3);
b->SetNRoomsY (2);

mobility.Install (ueNodes);
mobility.Install (enbNodes);
BuildingsHelper::Install (ueNodes);
BuildingsHelper::Install (enbNodes);
```

Listing C.15: Mobility Model

```
Ptr<HybridBuildingsPropagationLossModel> propagationLossModel =
    CreateObject<HybridBuildingsPropagationLossModel> ();
lteHelper->SetAttribute ("PathlossModel", StringValue ("ns3::
    HybridBuildingsPropagationLossModel"));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaExtWalls",
    DoubleValue (1));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaOutdoor",
    DoubleValue (0));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaIndoor",
    DoubleValue (0));
```

Listing C.16: Pathloss Model

AntennaModel & MIMO

```
lteHelper->SetEnbAntennaModelType ("ns3::CosineAntennaModel");
lteHelper->SetEnbAntennaModelAttribute ("Orientation",
    DoubleValue (0.0));
```

```
lteHelper->SetEnbAntennaModelAttribute ("Beamwidth",
    DoubleValue (70));
// MaxGain in dBs
lteHelper->SetEnbAntennaModelAttribute ("MaxGain",
    DoubleValue (0.0));
// Set the MIMO transmission mode
Config::SetDefault ("ns3::LteEnbRrc::DefaultTransmissionMode",
    UintegerValue (2)); // MIMO Spatial Multiplexity (2
    layers)
```

Listing C.17: Antenna & MIMO Model

Radio Environment Maps

```
Ptr<RadioEnvironmentMapHelper> remHelper = CreateObject<
    RadioEnvironmentMapHelper> ();
remHelper->SetAttribute ("Channel", PointerValue (lteHelper->
    GetDownlinkSpectrumChannel ()));
remHelper->SetAttribute ("OutputFile", StringValue ("rem.out"));
remHelper->SetAttribute ("XMin", DoubleValue (-400.0));
remHelper->SetAttribute ("XMax", DoubleValue (400.0));
remHelper->SetAttribute ("XRes", UintegerValue (100));
remHelper->SetAttribute ("YMin", DoubleValue (-300.0));
remHelper->SetAttribute ("YMax", DoubleValue (300.0));
remHelper->SetAttribute ("YRes", UintegerValue (75));
remHelper->SetAttribute ("Z", DoubleValue (0.0));
remHelper->SetAttribute ("UseDataChannel", BooleanValue (true));
remHelper->SetAttribute ("RbId", IntegerValue (10));
remHelper->Install ();
```

Listing C.18: Radio Environment Maps helper

Parameter Configuration

At the beginning of the `main` function, include:

```
Config::SetDefault ("ns3::ConfigStore::Filename", StringValue ("sim-
    input.txt"));
Config::SetDefault ("ns3::ConfigStore::Mode", StringValue ("Load"));
CommandLine cmd (__FILE__);
cmd.Parse (argc, argv);
ConfigStore inputConfig;
inputConfig.ConfigureDefaults ();
```

```
cmd.Parse (argc, argv);
```

Listing C.19: *Configuration parameters*

Example input file

```
default ns3::LteHelper::Scheduler "ns3::PfFfMacScheduler"
default ns3::LteHelper::PathlossModel "ns3::
    FriisSpectrumPropagationLossModel"
default ns3::LteEnbNetDevice::DlBandwidth "25"
default ns3::LteEnbNetDevice::DlEarfcn "100"
default ns3::LteEnbNetDevice::UlEarfcn "18100"
default ns3::LteUePhy::TxPower "10"
default ns3::LteEnbPhy::TxPower "30"
default ns3::LteEnbRrc::SrsPeriodicity "40"
default ns3::TcpSocket::SndBufSize "524280"
default ns3::TcpSocket::RcvBufSize "524280"
global RngSeed "24"
global simTime "10.0"
global nRB "100"
```

Listing C.20: *Configuration parameters*

C.3 DASHjs

```
#ifndef DASH_JS_H
#define DASH_JS_H

#include "abr-algorithm.h"
#include "abr-variables.h"
#include "abr-client.h"

namespace ns3 {

// 0 one bitrate
// 1 set placeholder buffer such that we download fragments at most
    recently measured throughput.
// 2 buffer is ready for using BOLA
constexpr uint16_t BOLA_STATE_ONE_BITRATE = 0;
constexpr uint16_t BOLA_STATE_STARTUP = 1;
constexpr uint16_t BOLA_STATE_STEADY = 2;

constexpr int32_t NO_CHANGE = -1;

namespace PRIORITY {
  // The priority can have these values
  // 0.5 default priority
  // 1 strong priority
  // 0 weak priority
  constexpr double DEFAULT = 0.5;
  constexpr double STRONG = 1;
  constexpr double WEAK = 0;
}

struct BolaState {
  uint16_t              state;
  uint32_t              stableBufferTime;
  uint32_t              lastQuality;
  double                Vp;
  double                gp;
  std::vector<double>   utilities;
  std::vector<double>   bitrates;
```

```
};

struct SwitchRequest {
  double    priority;
  int32_t   quality;
};

class DASHjs : public AbrAlgorithm
{
public:
  DASHjs ();
  DASHjs (uint32_t bufferSize);
  /**
  * \return the next quality
  */
  uint16_t GetNextQlty ();
  /**
  * \brief    check de DASH.js rules, similar to ABRRulesCollection.js
  * \return   a list of tasks for the client to schedule
  */
  std::vector<AbrTask>  CheckRules (uint16_t     currentQlty,
                                    uint32_t     segmentDuration,
                                    uint32_t     segIndex,
                                    double       currentBw,
                                    Time         dlStartTS,
                                    PlayerStates state,
                                    std::vector<SegmentInfo> buffer);

  Representation GetNextRep ();

  // Auxiliary Functions
  SwitchRequest GetMinSwitchRequest (std::vector<SwitchRequest> requests
      );
  SwitchRequest CreateSwitchRequest (double priority, int32_t quality);
  SwitchRequest CreateSwitchRequest (int32_t quality);

private:
  // Rule
  SwitchRequest ThroughputRule ();

```

```
void      UpdateAverageEwma ();
double    GetSafeAverageThroughput ();
uint32_t  GetQualityForBitrate (double bitrate);

// BOLA
SwitchRequest BolaRule ();
uint32_t      MinBufferLevelForQuality (uint32_t quality);
uint32_t      GetQualityFromBufferLevel ();
void          GetBolaState ();
void          GetInitialBolaState ();
double        GetAverageThroughput ();
std::vector<double>   CalculateBolaParameters (uint32_t
    stableBufferTime, std::vector<double> bitrates, std::vector<double>
     utilities);
std::vector<double>   UtilityFromBitrates (std::vector<double>
    bitrates);
std::vector<double>   NormalizeUtility (std::vector<double> utilities)
    ;

// Variables
uint16_t m_currentQlty; // current buffer Quality
uint16_t m_nextQlty; // next quality to request
uint32_t m_segmentDuration; // segment duration in ms
uint32_t m_segIndex; // index of the buffer playing
uint32_t m_bufferSize; // maximum buffer size
double   m_timeWatched; // in milliseconds
double   m_currentBw;  // current bandwidth
double   m_averageBw; // estimation of average bandwidth
double   m_slowEWMA; // slow Exponentially Weighted Moving Average
double   m_fastEWMA; // fast Exponentially Weighted Moving Average
double   m_slowAlpha; // alpha factor for slow EWMA
double   m_fastAlpha; // alpha factor for fast EWMA
double   m_bandwidthSafetyFactor; // safety factor
Time     m_dlStartTS; // time stamp of one segment starting to
    download
bool     m_firstSegment; // if it is the first segment
PlayerStates m_state; // actual state of the player
std::vector <SegmentInfo> m_buffer; // buffer of the segments
    downloaded
std::vector <Representation> m_representations;
```

```
    // BOLA variables
    BolaState m_bolaState;
    uint32_t  m_placeHolderBuffer;
    double    m_delay;
  };

  } // namespace ns3

  #endif /* DASH_ALGO_H */
```

Listing C.21: *DASHjs.h*

```
#include "DASHjs.h"
#include <math.h>
#include <cmath>
#include <limits>
#include <algorithm>

namespace ns3 {
NS_LOG_COMPONENT_DEFINE ("DASHjs");

NS_OBJECT_ENSURE_REGISTERED (DASHjs);

DASHjs::DASHjs (uint32_t bufferSize)
{
  m_bufferSize = bufferSize;
  m_averageBw = 0.0;
  m_slowEWMA = 0.0;
  m_fastEWMA = 0.0;
  m_slowAlpha = 0.1;
  m_fastAlpha = 0.5;
  m_bandwidthSafetyFactor = 0.7;
  m_firstSegment = true;
  m_bolaState.state = -1;
  m_placeHolderBuffer = 0;
  m_delay = 0.0;
  m_representations = AbrVariables::GetRepresentations ();
}

uint16_t
DASHjs::GetNextQlty ()
```

```
{
  return m_nextQlty;
}

std::vector<AbrTask>
DASHjs::CheckRules (uint16_t     currentQlty,
                    uint32_t     segmentDuration,
                    uint32_t     segIndex,
                    double       currentBw,
                    Time         dlStartTS,
                    PlayerStates state,
                    std::vector<SegmentInfo> buffer)
{
  NS_LOG_FUNCTION (this);
  m_currentQlty = currentQlty;
  m_segmentDuration = segmentDuration;
  m_segIndex = segIndex;
  m_dlStartTS = dlStartTS;
  m_currentBw = currentBw;
  m_buffer = buffer;
  m_state = state;

  std::vector<SwitchRequest> requests;
  std::vector<AbrTask> tasks;
  requests.push_back (ThroughputRule ());
  requests.push_back (BolaRule ());

  SwitchRequest request = GetMinSwitchRequest (requests);
  if (request.quality != NO_CHANGE) {
    m_nextQlty = request.quality;

    NS_LOG_INFO ("Change Quality to " << AbrVariables::GetQuality (
        m_nextQlty));

    tasks.push_back (AbrVariables::CreateTask (Seconds (m_delay),
        AbrTask::GetNextRep));
    tasks.push_back (AbrVariables::CreateTask (Seconds (m_delay +
        0.0001), AbrTask::SendRequest));

    m_delay = 0.0;
```

```
  }
  return tasks;
}

SwitchRequest
DASHjs::ThroughputRule ()
{
  SwitchRequest switchRequest = CreateSwitchRequest (NO_CHANGE);
  if (m_firstSegment && m_segIndex == 0)
  {
    NS_LOG_INFO ("First Segment");
    m_firstSegment = false;
    switchRequest.quality = 0;
    return switchRequest;
  }
  UpdateAverageEwma ();
  double average = GetSafeAverageThroughput ();
  switchRequest.quality = GetQualityForBitrate (average);
  return switchRequest;
}

SwitchRequest
DASHjs::BolaRule ()
{
  SwitchRequest switchRequest = CreateSwitchRequest (NO_CHANGE);
  GetBolaState ();
  if (m_bolaState.state == BOLA_STATE_ONE_BITRATE) {
    NS_LOG_INFO ("BOLA_STATE_ONE_BITRATE");
    switchRequest.quality = NO_CHANGE;
    return switchRequest;
  }

  // First segment
  if (m_firstSegment && m_segIndex == 0)
  {
    NS_LOG_INFO ("First Segment");
    m_firstSegment = false;
    switchRequest.quality = 0;
    return switchRequest;
  }
```

```
    uint32_t quality = 0;
    uint32_t bufferLevel = (m_buffer.size () - m_segIndex) *
        m_segmentDuration;
    uint32_t qualityForThroughput = 0;
    switchRequest.quality = 0;

    UpdateAverageEwma ();
    double safeThroughput = GetSafeAverageThroughput ();

    switch (m_bolaState.state) {
      case BOLA_STATE_STARTUP:
        NS_LOG_INFO ("BOLA_STATE_STARTUP");
        quality = GetQualityForBitrate(safeThroughput);

        switchRequest.quality = quality;
        m_bolaState.lastQuality = quality;

        if (bufferLevel >= 1)
        {
          m_bolaState.state = BOLA_STATE_STEADY;
        }

        break;

      case BOLA_STATE_STEADY:
        NS_LOG_INFO ("BOLA_STATE_STEADY");

        quality = GetQualityFromBufferLevel ();

        // BOLA-O variant
        qualityForThroughput = GetQualityForBitrate (safeThroughput);
        if (quality > m_bolaState.lastQuality && quality >
            qualityForThroughput)
        {
          // to avoid oscillations
          quality = std::max (qualityForThroughput, m_bolaState.
              lastQuality);
        }

```

```
      switchRequest.quality = quality;
      m_bolaState.lastQuality = quality;

      break;

    default:
      NS_LOG_INFO ("BOLA ABR Rule Bad State");
      quality = GetQualityForBitrate(safeThroughput);
      m_bolaState.state = BOLA_STATE_STARTUP;

      break;
  }
  return switchRequest;
}

void
DASHjs::GetBolaState ()
{
  if (m_bolaState.state > 2)
  {
    GetInitialBolaState ();
  }
}

void
DASHjs::GetInitialBolaState ()
{
  NS_LOG_INFO ("Initial BOLA state");
  std::vector<double> bitrates;
  bitrates = AbrVariables::GetBitratesInKbps ();

  std::vector<double> utilities = UtilityFromBitrates (bitrates);
  std::vector<double> normalizedUtilities = NormalizeUtility (utilities)
      ;
  uint32_t stableBufferTime = 12; // DEFAULT_MIN_BUFFER_TIME;
  // uint32_t stableBufferTime = 20; //
      DEFAULT_MIN_BUFFER_TIME_FAST_SWITCH;
  std::vector<double> params = CalculateBolaParameters (stableBufferTime
      , bitrates, normalizedUtilities);

  if (params.size () <= 0)
```

```
  {
    m_bolaState.state = BOLA_STATE_ONE_BITRATE;
  }
  else
  {
    m_bolaState.state = BOLA_STATE_STARTUP;

    m_bolaState.bitrates = bitrates;
    m_bolaState.utilities = normalizedUtilities;
    m_bolaState.stableBufferTime = stableBufferTime;
    m_bolaState.Vp = params[0];
    m_bolaState.gp = params[1];

    m_bolaState.lastQuality = 0;
  }
}

std::vector<double>
DASHjs::NormalizeUtility (std::vector<double> utilities)
{
  std::vector<double> normalized;;
  double offset = -utilities[0];
  for (std::vector<double>::iterator it = utilities.begin();
       it != utilities.end(); ++it) {

    double n = *it + offset;
    normalized.push_back (n);
    NS_LOG_INFO (n);
  }
  return normalized;
}

std::vector<double>
DASHjs::UtilityFromBitrates (std::vector<double> bitrates)
{
  std::vector<double> utilities;
  for (std::vector<double>::iterator it = bitrates.begin();
       it != bitrates.end(); ++it) {
    double u = log(*it);
    utilities.push_back (u);
```

```
      NS_LOG_INFO (this << " " << log(*it));
    }
    return utilities;
  }

  std::vector<double>
  DASHjs::CalculateBolaParameters (uint32_t stableBufferTime, std::vector<double> bitrates, std::vector<double> utilities)
  {
    std::vector<double> params;

    const uint32_t MINIMUM_BUFFER_S = 10000;
    const uint32_t MINIMUM_BUFFER_PER_BITRATE_LEVEL_S = 2000;
    uint32_t nBitrates = bitrates.size ();
    uint32_t bufferTime = std::max (stableBufferTime,
      MINIMUM_BUFFER_S + MINIMUM_BUFFER_PER_BITRATE_LEVEL_S * nBitrates);

    double gp = (utilities.back () - 1) / (bufferTime / MINIMUM_BUFFER_S - 1);
    double Vp = MINIMUM_BUFFER_S / gp;
    NS_LOG_INFO ("gp: " <<gp << " u: " << utilities.back() <<" buf: "<< bufferTime);

    params.insert (params.begin (), Vp);
    params.insert (params.begin () + 1, gp);
    return params;
  }

  uint32_t
  DASHjs::MinBufferLevelForQuality (uint32_t quality)
  {
    uint32_t qBitrate = m_bolaState.bitrates[quality];
    uint32_t qUtility = m_bolaState.utilities[quality];

    uint32_t min = 0;

    for (uint16_t i = quality - 1; i > 0; --i)
    {
      NS_LOG_INFO (i);
      if (m_bolaState.utilities[i] < m_bolaState.utilities[quality])
```

```
      {
        uint32_t iBitrate = m_bolaState.bitrates[i];
        uint32_t iUtility = m_bolaState.utilities[i];

        uint32_t level = m_bolaState.Vp * (m_bolaState.gp + (qBitrate *
            iUtility - iBitrate * qUtility) / (qBitrate - iBitrate));
        min = std::max (min, level);
      }
    }
    return min;
  }

  // Main function
  uint32_t
  DASHjs::GetQualityFromBufferLevel ()
  {
    uint32_t bitrateCount = m_bolaState.bitrates.size ();
    uint32_t quality = 0;
    uint32_t bufferLevel = (m_buffer.size () - m_segIndex) *
        m_segmentDuration;
    double score = NAN;
    for (uint16_t i = 0; i < bitrateCount; ++i)
    {
      double s = (m_bolaState.Vp * (m_bolaState.utilities[i] + m_bolaState
          .gp)
      - bufferLevel) / (m_bolaState.bitrates[i]);
      if (std::isnan(score) || s > score)
      {
        NS_LOG_INFO ("s: " << s << " Vp: "<< m_bolaState.Vp << " u: "
        << m_bolaState.utilities[i]
        << " gp: " << m_bolaState.gp << " level: " << bufferLevel
        << " bitrate: " << m_bolaState.bitrates[i]);
        score = s;
        quality = i;
      }
    }
    NS_LOG_INFO (quality << " " << AbrVariables::GetQuality(quality) << "
        "
    << m_representations[quality].bitrate << "Buffer Level: " <<
        bufferLevel);
```

```
  return quality;
}

double
DASHjs::GetAverageThroughput ()
{
  return m_averageBw;
}

SwitchRequest
DASHjs::GetMinSwitchRequest (std::vector<SwitchRequest> requests)
{
  SwitchRequest newSwitchReq = CreateSwitchRequest (NO_CHANGE);
  int32_t newQuality = -1;
  std::map<double, int32_t> values;

  if (requests.size() == 0)
  {
    return newSwitchReq;
  }
  else if (requests.size() == 1)
  {
    return requests.back ();
  }

  values.insert (std::pair<double, int32_t>(PRIORITY::STRONG, NO_CHANGE)
      );
  values.insert (std::pair<double, int32_t>(PRIORITY::WEAK, NO_CHANGE));
  values.insert (std::pair<double, int32_t>(PRIORITY::DEFAULT, NO_CHANGE
      ));

  for (std::vector<SwitchRequest>::iterator it = requests.begin ();
        it != requests.end (); ++it) {
    SwitchRequest req = *it;
    if (req.quality != NO_CHANGE) {
      if (values.at (req.priority) == NO_CHANGE ||
          values.at (req.priority) > req.quality) {
        NS_LOG_INFO (req.quality);
        values.at (req.priority) = req.quality;
      }
```

```
      }
    }

    if (values.at (PRIORITY::WEAK) != NO_CHANGE) {
      newQuality = values.at (PRIORITY::WEAK);
    }

    if (values.at (PRIORITY::DEFAULT) != NO_CHANGE) {
      newQuality = values.at (PRIORITY::DEFAULT);
    }

    if (values.at (PRIORITY::STRONG) != NO_CHANGE) {
      newQuality = values.at (PRIORITY::STRONG);
    }

    if (newQuality > -1) {
      newSwitchReq = CreateSwitchRequest (newQuality);
      NS_LOG_INFO (newQuality);
      NS_LOG_INFO ("SwitchRequest to quality" << AbrVariables::GetQuality
          (m_nextQlty));
    }

    return newSwitchReq;
  }

  SwitchRequest
  DASHjs::CreateSwitchRequest (double priority, int32_t quality) {
    SwitchRequest req;
    if (priority != 0 || priority != 0.5 || priority != 1) {
       // priority by default
       std::cout << priority << std::endl;
       req.priority = PRIORITY::DEFAULT;
     } else {
       req.priority = priority;
     }
    req.priority = priority;
    req.quality = quality;
    NS\_LOG\_INFO (req.priority << " " << req.quality);
    return req;
  }
```

```

SwitchRequest
DASHjs::CreateSwitchRequest (int32_t quality) {
  SwitchRequest req;
  req.priority = PRIORITY::DEFAULT;
  req.quality = quality;
  NS_LOG_INFO (req.priority << " " << req.quality);
  return req;
}

uint32_t
DASHjs::GetQualityForBitrate (double bitrate)
{
  Representation rep;
  if (m_representations.size () < 2) return 0;

  for (uint16_t j=m_representations.size () - 1; j>0; j--)
  {
    rep = m_representations[j];
    // bitrates are in bps
    if (bitrate > rep.bitrate) {
      NS_LOG_INFO (j << " " << AbrVariables::GetQuality(j) << " " <<
          m_representations[j].bitrate << " " << bitrate);
      return j;
    }
  }
  return 0;
}

void
DASHjs::UpdateAverageEwma ()
{
  double newSample = 0;
  if (m_buffer.size () != 0)
  {
    newSample = m_buffer.back ().dlBandwidth;
  }

  if (m_fastEWMA == 0 || m_slowEWMA == 0)
  {
```

```
      m_fastEWMA = newSample;
      m_slowEWMA = newSample;
    }
    else
    {
      m_fastEWMA = newSample * m_fastAlpha + m_fastEWMA * (1 - m_fastAlpha
          );
      m_slowEWMA = newSample * m_slowAlpha + m_slowEWMA * (1 - m_slowAlpha
          );
    }
    m_averageBw = std::min (m_slowEWMA, m_fastEWMA);
  }

  double
  DASHjs::GetSafeAverageThroughput ()
  {
    double average = m_averageBw;
    if (average != 0) {
      average *= m_bandwidthSafetyFactor;
    }
    return average;
  }

  Representation
  DASHjs::GetNextRep ()
  {
    Representation rep = AbrVariables::GetRep (m_currentQlty);
    return rep;
  }

  } // namespace ns3
```

Listing C.22: *DASHjs.cc*