NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

THESIS

QoE estimation for Adaptive Video Streaming over LTE Networks

Achilleas - M - Moustakis

Supervisors: **Lazaros Merakos,** Professor

ATHENS

FEBRUARY 2017

THESIS

QoE estimation for Adaptive Video Streaming over LTE Networks

Achilleas M. Moustakis
A.M.: 1115201000151

SUPERVISORS: **Lazaros Merakos,** Professor

ΠΕΡΙΛΗΨΗ

Η 4η γενιά (4G) κινητών επικοινωνιών, στην οποία ανήκει το σύστημα Long Term Evolution (LTE), παρέχει ευρυζωνική πρόσβαση σε κινητές συσκευές με ποιότητα και ταχύτητα που αγγίζουν τις ενσύρματες επικοινωνίες. Παρόλ'αυτά, η κινητικότητα εκ φύσεως εισάγει αστοχίες/διακυμάνσεις στην ασύρματη διεπαφή, γενόντας έτσι την ανάγκη για αντίστοιχη προσαρμογή της ροής μετάδοσης των δεδομένων. Η ανάγκη αυτή είναι ακόμη πιο έκδηλη για τις ροές δεδομένων βίντεο, που έχουν και τη μερίδα του λέοντος στην διαδικτυακή κίνηση. Καθώς, λοιπόν, η ροή βίντεο μέσω HTTP έχει γίνει ο κανόνας στη διανομήπεριεχομένου, η εφαρμογή ενός πρωτοκόλλου προσαρμογής βασισμένου στο HTTP είναι αναπόφευκτη. Το DASH (Dynamic Adaptive Streaming over HTTP) επιτρέπει μια ομαλή, αδιάκοπη ροή video εφαρμόζοντας αλγόριθμους προσαρμογής του bitrate στη μεριά του χρήστη αξιοποιώντας πλήρως την υπάρχουσα υποδομή. Έχοντας ως στόχο να τελειοποιήσουν την ποιότητα την οποία προσφέρει στους χρήστες το δίκτυο, οι ερευνητές συνεχώς αναπτύσσουν νέες φόρμουλες για την εκτίμηση της ποιότητας εμπειρίας του τελικού χρήστη, γνωστής υπο τον όρο Quality of Experience (QoE). Η παρούσα πτυχιακή αντιπροσωπεύει την προσπάθεια συγκερασμού των τριών ακόλουθων πυλώνων: της υποκείμενης υποδομής, του ελέγχου της ποιότητας υπηρεσίας με τη χρήση αλγορίθμων προσαρμογής και του επαναπροσδιορισμού του συστήματος με ανάλυση της ποιότητας και ανατροφοδότηση. Ανοίγει τη συζήτηση για τη χρήση προσαρμοζόμενης ροής μετάδοσης πάνω απο δίκτυα LTE και στοχεύει όχι μόνο να προσφέρει μια βαθιά βιβλιογραφική προσέγγιση των επιμέρους, αλλά και να περιγράψει πώς συνδέονται, πώς επικαλύπτονται, ή πώς αλληλεπιδρούν. Περιγράφει τα σημαντικότερα σύγχρονα μοντέλα μέτρησης QoE και πώς αυτά χρησιμεύουν στην αντικειμενική εκτίμηση της ποιότητας. Βασική συνεισφορά της εργασίας, είναι η ανάπτυξη μιάς πλήρης εκτελέσιμης οντότητας (module) για τον προσομοιωτή NS-3 συνδυάζοντας όλες τις έννοιες που αναφέρονται παραπάνω.Ο αναγνώστης μπορεί να βρεί ενα τυπικό παράδειγμα εκτέλεσης της εν λόγω οντότητας, με την συνοδεία μιας βήμα-βήμα εξήγησής του και και κάποιων διαγραμμάτων με αποτελέσματα. Το NS3 module αναπτύχθηκε με την ελπίδα να φανεί χρήσιμο σε κάθε ερευνητή τηλεπικοινωνιών που ασχολείται με θέματα παροχής ποιότητας εμπειρίας και αναζητά ένα εργαλείο προσομειώσεων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: LTE Δίκτυα 4ης Γενιάς

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: LTE, Ποιότητα Εμπειρίας, Ποιότητα Υπηρεσίας, Προσαρμοζόμενη Ροή Πολυμέσων

ABSTRACT

The ability to address an increasing need for mobility in work and entertainment has rendered LTE networks critically essential to our everyday environments. The promising 4th Generation (4G) of Long Term Evolution (LTE) provides ubiquitous broadband access to mobile devices matching land communications in speed and quality. However, the nature of mobility introduces a need for adaptivity in multimedia streaming, the largest part of mobile Internet traffic. As HTTP video streaming has become the de facto dominating solution to distribute media content, the implementation of an HTTP-based adaptive streaming protocol is inevitable. Dynamic Adaptive Streaming over HTTP (DASH) allows for smooth, uninterrupted video streaming by implementing bitrate adaptation algorithms on the client side, with complete utilization of the existing network infrastructure. In order to perfect the current quality served by the network, network researchers constantly develop new metrics to assess the end-user's Quality of Experience. This thesis represents an attempt to join these three pillars of mobile video streaming: the underlying infrastructure, the over-the-top algorithmic quality control, and the follow-up feedback measurement. It opens a discussion about the use of adaptive streaming in LTE networks, and aims to offer not only a deep down bibliographic approach of each individual concept, but also describe where they overlap, how they connect and interact with each other. It depicts the most important contemporary QoE models and metrics, explains their formulas, and outlines their uses as key performance indicators in objective quality estimation. Furthermore, within this work, we provide a complete, expandable NS-3 model combining all the concepts discussed. An HTTP Server-Client model within the LTE network architecture, with implemented adaptive streaming functionality. The tool was developed in the hope of becoming useful to any telecommunications researcher, supporting their research and introducing them to the NS-3 simulator. In the end, we present a typical execution of our example with a step by step explanation, followed by the plotting of some of the results using a C++ script we developed.

SUBJECT AREA: Adaptive Streaming in LTE Networks

KEYWORDS: LTE, DASH, QoE, QoS, HAS, Quality of Experience, Adaptive Streaming

To Michael and Melina, the dearest parents a man can have.
To my friends and family, for motivating and supporting me throughout my studies.

Human behavior flows from three main sources:
desire, emotion, and knowledge.
~ Plato

ACKNOWLEDGEMENTS

Upon completing this thesis, I would like to thank Professor Lazaros Merakos, for offering me the chance to get involved in a research program of major importance and introducing me to the most contemporary networking concepts. His networking wisdom, his teaching diligence and his professionalism will continue to inspire me throughout my engineering career.

I am more than grateful to Dr. Dimitris Tsolkas, for his invaluable guidance, his outstanding patience, and his positive attitude without which this thesis could not be completed. With tremendous respect to his academic potential, it was an honor to have him as my supervisor.

Lastly, I would also like to thank Ms. Eirini Liotou, for her support to our project and for devoting time to explain to me some of the most challenging QoE-related theories.

CONTENTS

PROLOGUE **11**

1. ADAPTIVE VIDEO STREAMING **12**

1.1 Non-adaptive Video Streaming **12**

1.1.1 Introduction 12

1.1.2 Brief History 12

1.2 Adaptive Bitrate Streaming **14**

1.2.1 Introduction 14

1.2.2 Adaptive Streaming: The Definition 14

1.2.3 On the Server side 16

1.2.4 On the Client side 17

1.3 The MPEG-DASH protocol **19**

1.3.1 Introduction 19

1.3.2 Brief History 19

1.3.3 The basic scenario 20

1.3.4 MPEG-DASH major advantages 22

2. LONG TERM EVOLUTION NETWORKS **23**

2.1 Brief History **23**

2.2 Objectives of the LTE system **25**

2.3 LTE Transmission Modes **27**

2.3.1 Multiple Antennas 27

2.3.2 Orthogonal Frequency-Division Multiple Access 29

2.4 LTE Architecture **31**

2.4.1 Overview 31

2.4.2 The Evolved Packet Core 34

2.4.3 The Non-Access Stratum (NAS) 36

2.4.4 The access network 37

2.4.5 Protocol Architecture 39

2.4.6 User plane 39

2.4.7 Control plane 41

2.5 Quality of Service in LTE **42**

2.5.1 EPS Bearers 42

2.5.2 QoS Class Identifiers 43
2.5.3 OTT Content Providers 46

3. QUALITY OF EXPERIENCE 48

3.1 Introduction 48

3.2 Influence Factors 49
3.2.1 Initial Delay 49
3.2.2 Stalling 50
3.2.3 Adaptation 51

3.3 QoS Metrics 51

3.4 QoE Metrics 53
3.4.1 Mean Opinion Score (MOS) 53
3.4.2 Using the PSNR 55
3.4.3 Network average 56

3.5 Service Providers and Applications 58

4. THE NS-3 NETWORK SIMULATOR 61

4.1 NS-3 Basics 61

4.2 Building on top of LENA Project 61
4.2.1 Introduction 61
4.2.2 A Simple Example 62
4.2.3 The Server-Client Model 63

4.3 Implementing DASH 67

4.4 Presenting the Results 69
4.4.1 Plots 69
4.4.2 Throughput Calculation 70
4.4.3 SINR Computation 71
4.4.4 QoE Metrics 73

ABBREVIATIONS-ACRONYMS 76

APPENDIX 79

REFERENCES 98

FIGURES

Figure 1.1: Progressive download architecture 14

Figure 1.2: Streaming example w/ Media Server 13

Figure 1.3 Adaptive streaming Web Service overview 14

Figure 1.4: Adaptive streaming system end-to-end overview 15

Figure 1.5: Adaptive streaming server function flow 16

Figure 1.6: General view of client functions 18

Figure 1.7: The MPEG-DASH process timeline 20

Figure 1.8: Example of the MPD file data model (by Dutch company TNO) 21

Figure 1.9: Main Adaptive-HTTP protocols 21

Figure 1.10: MPEG-DASH Server – Client Overview 22

Figure 2.1: History and evolution of cellular technologies 25

Figure 2.2: LTE Transmission modes tree (by 4G Americas) 27

Figure 2.3: MIMO release 8 scenarios 28

Figure 2.4: DL and UL in LTE 29

Figure 2.5: OFDMA and SC-FDMA in LTE 30

Figure 2.6: Cyclic Prefix Insertion in OFDM 30

Figure 2.7: LTE general architecture (by AIRCOM) 32

Figure 2.8: Elements forming the EPS 33

Figure 2.9: P-GW interaction with the network 34

Figure 2.10: S-GW interaction with the network 34

Figure 2.11: HSS in the center of the network 35

Figure 2.12: Access Stratum & Non-Access Stratum 36

Figure 2.13: E-UTRAN architecture 37

Figure 2.14: User Plane and Control Plane in the Protocol stack 39

Figure 2.15: RLC Sub Layer 40

Figure 2.16: EPS Bearers and Traffic Flows 43
Figure 2.17: Bearer Hierarchy 44
Figure 2.18: Standardized QCIs 45
Figure 2.19: OTT Content Flows 46
Figure 2.20: Default and Dedicated EPS bearers 47
Figure 3.1: HAS QOE key influence factors 48
Figure 3.2: Error concealment example 52
Figure 3.3: Relation of MOS and Video Quality 53
Figure 3.4: Relation of PSNR and Video Quality 55
Figure 4.1: Lena Example Execution 63
Figure 4.2: Sender's Dump File 66
Figure 4.3: Receiver's Dump File 66
Figure 4.4: DlRlcStats.txt Sample 70
Figure 4.5: Throughput, Node 1 71
Figure 4.6: DlRsprSinrStats.txt Sample 72
Figure 4.7: SINR, Node 1 73
Figure 4.8: Reception Ratio ρ, Node 1 74
Figure 4.9: MOS, Node 1 75

PROLOGUE

The present thesis is part of and written under the undergraduate program of the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens, Greece. The text presented below is organized in the following chapters:

In Chapter 1 we explain the concept of Adaptive Video Streaming and unveil its advantages. More specifically, MPEG-DASH is mentioned as an indicative option of the adaptive streaming functionality.

Chapter 2 is devoted to LTE networks as a whole, offering a deep analysis on their architecture, and every detail that an outsider should be aware of. Focus is especially place on the user-side equipment, and parts interacting with the application layer.

Our informative series concludes in Chapter 3 where the central topic is quality measurement and, more particularly, Quality of Experience. In this chapter, several QoS and QoE metrics are outlined and explained.

In Chapter 4, we analyze our NS-3 module along with two separate C++ scripts to be used with it. We present the results from a typical execution and attach a part of the code in the appendix.

Our code package was developed mainly in Eclipse running on OSX Yosemite 10.10.5 For our simulations, we used NS-3 version 3.24 on a Ubuntu 64-bit 14.04.3 Virtual Machine.

As a Network Engineer’s quest to perfect the global networking structure is surely never-ending, we hope to continue to improve the tools we created and openly invite others to help us to do so.

1. ADAPTIVE VIDEO STREAMING

1.1 Non-adaptive Video Streaming

1.1.1 Introduction

The process of delivering any multimedia content sent usually in a compressed form over the Internet and displayed by the viewer in real time is called “streaming”. Initially, any Internet peer who would choose to be delivered a media file would be required to download the full size of that media file and store it to his terminal’s hard disk drive before he could be able to display its multimedia content. The rapid increase in the quality of media files, and therefore their size prompted the need for the viewer to have the ability to access the content without waiting for the file to finish downloading. The key that makes video streaming revolutionary is undoubtedly the fact that it gives instead the client-user a sense of interactivity with the content by sending a continuous stream of data which is played as it arrives. From the user being able to shift the download index and display parts of the media file to online providers even serving or broadcasting live content real-time to a number of users with nothing more than a small delay, these are but a few of the countless possibilities provided by media streaming.

1.1.2 Brief History

Major streaming video and streaming media technologies included RealSystem G2 from RealNetworks, Microsoft Windows Media Technologies, and VDO. Progressive Networks (later renamed RealNetworks) is considered by many to have started the streaming media industry and deserves a lot of the credit being the primary company responsible for the wide adoption of audio and video streaming with content owners, and consumers, from 1995 to 2002. They dominated the market until 1999, but Microsoft was also working on video technology as early as 1993 [3]. Since then Microsoft and RealNetworks have provided numerous streaming solutions over the years with the addition of Apple in the years after 2005. Microsoft’s latest streaming viewer is Silverlight but most streaming websites use Flash, a component originally developed by Macromedia. Microsoft's approach uses the standard MPEG compression algorithm for video. [6] The other approaches use proprietary algorithms.

Nowadays, video streaming tends to split into two categories, according to how the incoming multimedia data flow is being handled on the client side: progressive download (or pseudo-streaming) and real-streaming [4]. Progressive download or pseudo-streaming is characterized by downloading an actual file or a part of it, even temporarily, and playing that file as it is being downloaded. Real streaming, on the other hand, is characterized by a data-buffering viewer (all data is kept in memory), with no file being saved on disk.

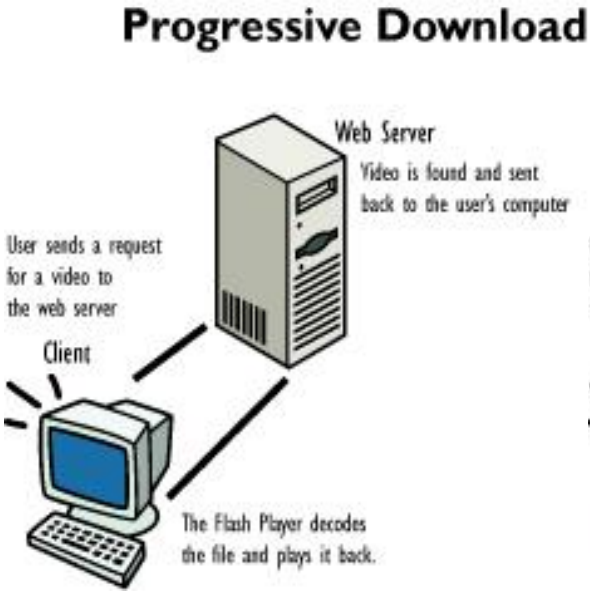

Figure 1.1: Progressive download architecture

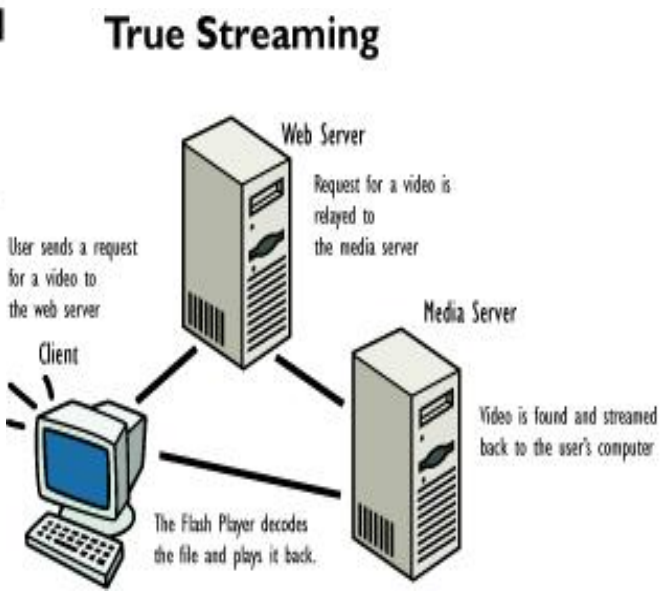

Figure 1.2: Streaming example w/ Media Server

From a server perspective, streaming video is usually sent from a collection of prerecorded video files, but can be distributed as part of a live broadcast "feed." In a live broadcast, the video signal is converted into a compressed digital signal and transmitted from a special Web server that is able to do multicast, sending the same file to multiple users at the same time.

The user needs a player, which is a special program, running on the client's terminal that decompresses and sends video and audio data to their respective output devices. A player can be either an integral part of a browser or a part of an external software package issued by the streaming source or by a third-party.

Overall, streaming technologies are rapidly gaining popularity as a way to deliver dynamic media content over the Internet. As bandwidth increases consistently and compression technologies mature, it becomes increasingly easier to deliver real-time, dynamic media, such as video, audio, animation, Java applications and 3D and vector graphic using streaming technologies. If used properly, streaming applications can add impressive capability to any service site.

1.2 Adaptive Bitrate StreamingIntroduction

Adaptive streaming technologies are a class of services able to optimize video viewing experience using a predefined set of connection speeds on a wide range of devices. Through the concept of adaptive streaming [5], web users are able to enjoy their favorite content with minimized delays and highly improved quality, suited to their Internet connection limits and their network conditions. This section describes the structure of adaptive streaming, explains the concept's architecture and identifies the main technology contenders in the market. It also addresses the most important factors that influence the adaptive streaming technology. As a consequence, adaptation has become a standard for all web organizations and enterprises offering streaming services.

1.2.2 Adaptive Streaming: The Definition

All Adaptive streaming technologies share a basic workflow executing a number of common steps. [5] They all start by encoding the source media file to produce multiple files to be distributed to viewers watching on different connection speeds, on different devices.

Then, they distribute these files in an adaptive manner, changing the stream which is being delivered according to effective changes in network values such as throughput and latency, or even sometimes adapting to the player needs.

In the end, they all manage to maintain a level of transparency towards the user since all stream switching is executed on the background, so that the viewer can avoid clicking multiple buttons. Users may often notice an extremely slight difference in their viewing quality during the switch (adaptation) of the streams, however no action needs to be taken on their end.

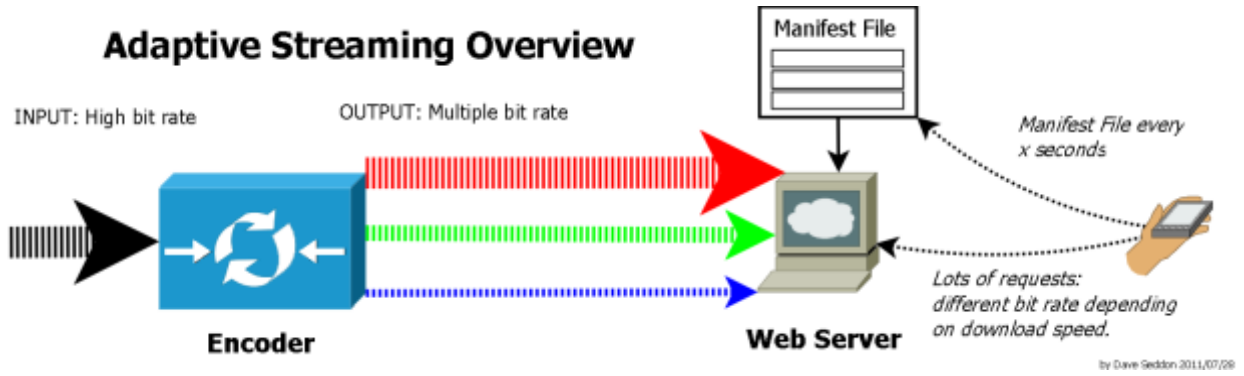

Figure 1.3 Adaptive streaming Web Service overview

Despite some main approach differences, adaptive technologies also have similar operating principles. [5] For instance, they all monitor important factors such as the user's effective throughput, the level of exhaustion of the player's media buffer, the delay and the dropped frames to assess the best suitable quality for the playback

terminal. This information is extracted in order to determine when it is preferable to switch streams and which stream to switch to.

In case the control process notices that the media buffer is overflowing and CPU utilization levels are low, the adaptive streaming process forces a switch to a stream of a higher quality to improve the user's quality of experience. In contrast, if CPU usage surpasses a specific threshold, or if the buffer's content drops below a predefined low, the technology may choose to force the user's player to make a switch to a lower quality stream.

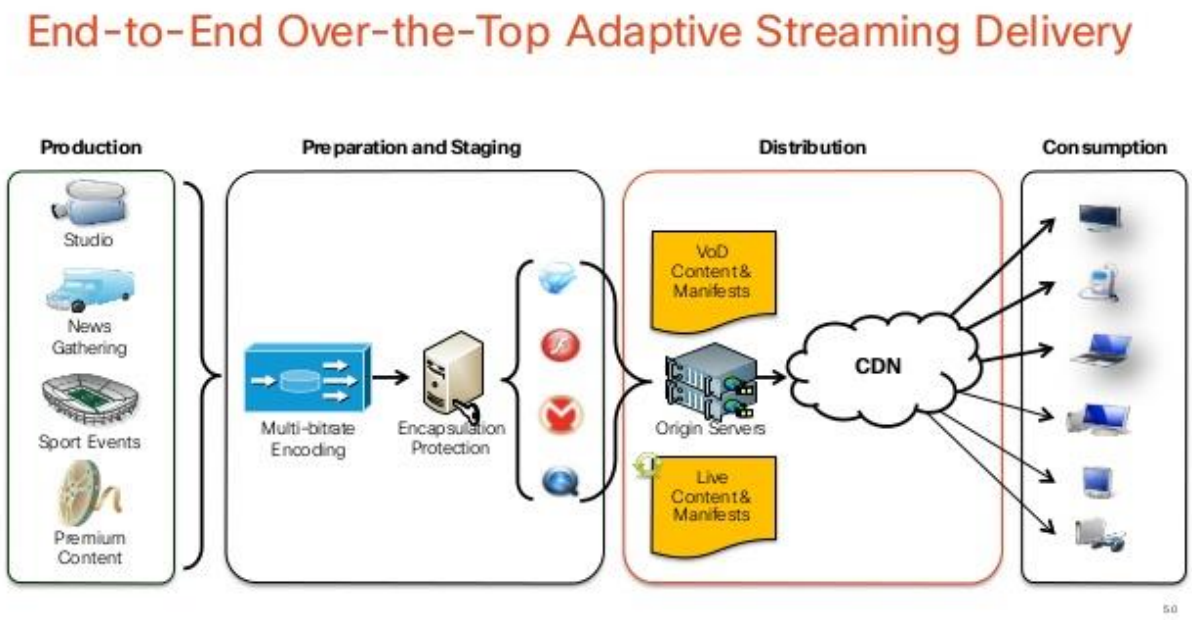

Figure 1.4: Adaptive streaming system end-to-end overview

The major difference between different adaptive technologies is in the implementation. More specifically, it relies in the involvement of a streaming server (as shown in figures 1.1 and 1.2). Some technologies are able to fully operate without a streaming server. The different quality streams are available at different URL addresses on a web server or across a network of web servers. After the player analyses operating metrics and utility factors to decide when a stream switch is preferable, it has the ability to execute a stream switch and initiates the adaptation procedure by retrieving data from a different stream than the one it was previously using.

On the other hand, there are adaptive technologies which definitely require a streaming server having constant communication with the player. In this case, the server is in charge of internally managing different streams and providing a data flow to the client. If a stream switch is required, the server implements it by sending a different stream to the viewer.

Either way, adaptive streaming implementations enable streaming services to deliver the highest possible quality streams serving both low and high throughput applications. For streaming web services, adaptive streaming is considered a must as without it most producers would force users to download different files for different quality standards or select a single quality for the duration of their viewing experience.

1.2.3 On the Server side

On the server side of an adaptive bitrate streaming client-server model the detailed perspective highly depends on the packaging approach of different technologies. Hereby we present a generic description of the basic processes executed on the server side.

Assuming that a high bitrate multimedia content already relies on the server's database as an input, in the beginning that content is passed through an encoder (as shown in figures 1.3 and 1.4). An encoding machine or a software component undertakes the task of producing two or more lower-bitrate versions of the input media content (figure 1.5). The multiple file versions are forwarded to the web server where depending on the approach can be stored or immediately delivered.

Upon client request for a number of media slices, the web server normally generates what is called a "media manifest file" often also referred to as Media Presentation File (MPD). Based on the location of the client device the web server indicates (in the media manifest file) one or more sources for each of the media slices and relevant information associated with the respective sources, most notably a bitrate threshold necessary for the client to maintain his playing rate of these media slices uninterrupted.

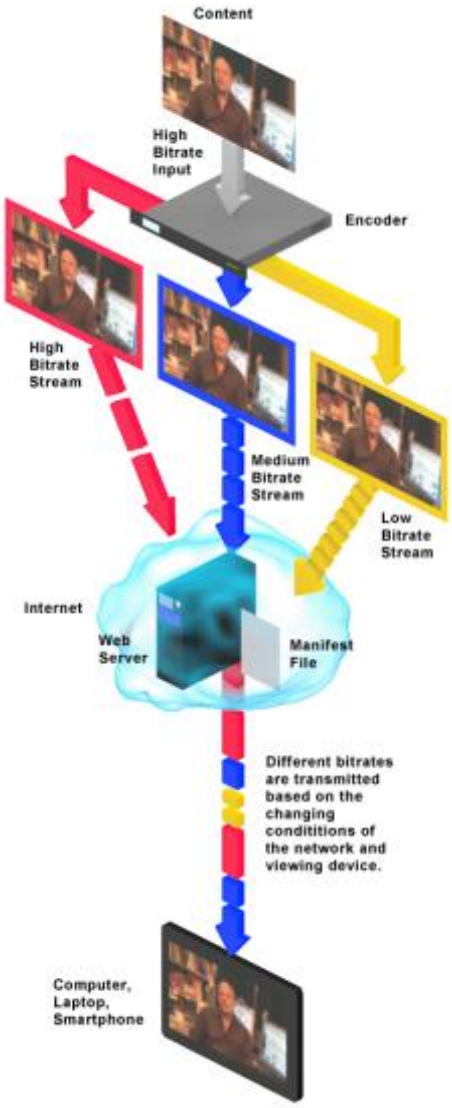

Figure 1.5: Adaptive streaming server function flow

At the start of communication between server and client the manifest described is sent to the client which is the event that triggers the streaming process. As the client requests a specific segment of the multimedia content from one of the available different bitrate sources, resources are located usually by HTTP URLs [1] in or out of the web server and the server transmits the correspondent multimedia segment through the internet.

Based on the constantly changing conditions of the network and the client's viewing device, the web server needs to be prepared to immediately change transmitting from one source to another with a lower or higher bitrate. That said, there is an ongoing discussion about what would appear to be the most efficient of ways to preload or cache parts of a media file on the delivery server [7].

The ability to switch to the stream that plays best at any time is ultimately what makes the service adaptive to client throughput. Consequently, an adaptive bitrate streaming web server guarantees the viewers will get the highest quality possible without buffering interrupts. This process vastly improves the client's Quality of Service which is considered an important objective for all modern applications.

1.2.4 On the Client side

In state-of-the-art adaptive streaming solutions, the client module installed on the viewer's side are highly intelligent. Here we demonstrate a general view of all the processes which are performed on the client side.

First of all, the client initiates communication with the server side by connecting to the server or making a request. If there is a media manifest file, the client requests and receives the Manifest file to extract the temporal Information of the media file included in it.

Once the client side is made aware of the different bitrates available by reading the manifest file, the receiving process makes a request to the server to be sent a chunk of data which represents the first segment of the media content, a video segment for instance. This segment contains multimedia content of an average length of 2 to 10 seconds [8], depending on the end-to-end implementation adopted by the content provider or the viewing tools.

After a segment is received, it is inserted in a fixed size buffer which guarantees that there will be enough content saved on the client's memory to avoid undesirable interruptions of the playing session. This initial buffering in most cases does not last long, with the segment passing through a decoder to be converted in a playable format and then fed directly to the client's player, usually frame by frame.

At the same time, another controlling process is monitoring network and video statistics to calculate the future download strategy. It measures the current bandwidth of the client's network so that the client is alerted in case of a low bandwidth. The client's throughput is recorded during the transmission of the last segment received along with an average throughput for a time period previously specified by the player software provider. It also keeps track of the current buffer level ensuring that buffer levels are kept to an average. Lastly, in many cases of modern players the controller is responsible for withdrawing content from the buffer and feeding it to the player after decoding.

When the receiver is ready to request another segment, it asks the controller for the average throughput and uses it as a threshold to decide, which is the highest quality-level that the client can support. This way, the next segment is requested from the respective stream of the decided quality level or a lower one in case the exact quality level does not exist as an option in the available streams. By checking with the manifest file it calculates the expected data chunk size, finds the source where the request must be sent to and makes the request. Once the request is processed by the server and the data chunk is being transmitted, the receiver process starts receiving and saving to the client's memory. All of the above processes can be seen at the diagram in figure 1.6 as well as reference [7].

As it becomes obvious to the reader, modern adaptive streaming clients adopt high independency principles in their implementation schema in order to fully adapt to the changes in network and displaying environments [10]. They primarily aim to utilize all resources available to the client, such as memory or bandwidth, at the maximum level to offer the best quality of experience possible.

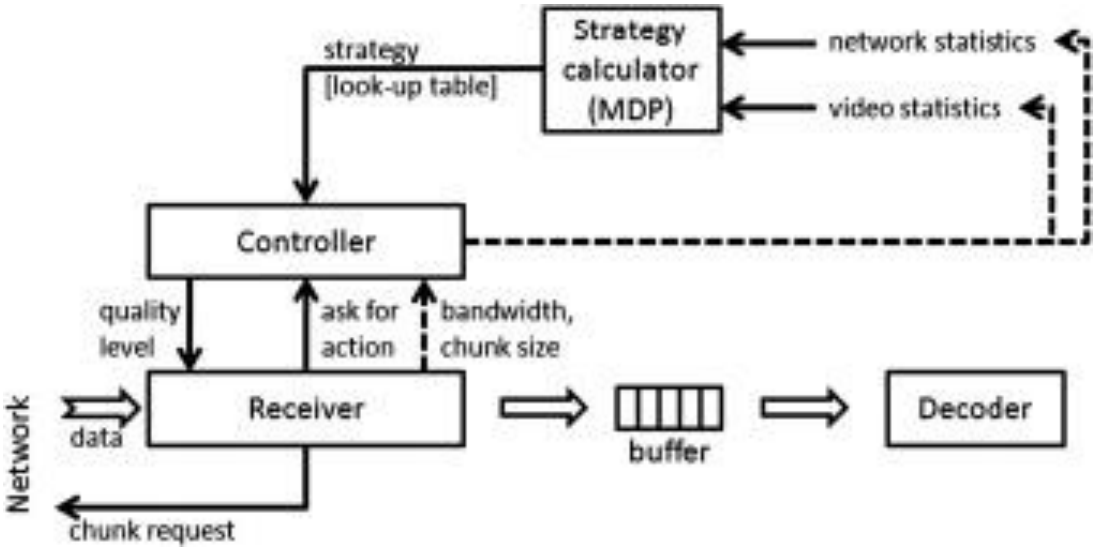

Figure 1.6: General view of client functions

1.3 The MPEG-DASH protocol

1.3.1 Introduction

The Moving Picture Expert Group (MPEG) is a working group of ISO/IEC with the mission to develop standards for coded representation of digital audio and video and related data, according to [9]. It has developed several widely used multimedia standards addressing the need for content creators to reach multiple platforms and devices in an efficient and cost-effective way. Protocols such as MPEG-2, MPEG-4, MPEG-7, and MPEG-21 are well known and widely appreciated. MPEG-DASH (Dynamic Adaptive Streaming over HTTP) is MPEG's proposed solution to the complex problems of HTTP adaptive streaming services that content delivery providers face in different devices.

The first major trial of the MPEG-DASH protocol was the coverage of the 2012 London Summer Olympics. From London, Belgian public broadcaster Vlaamse Radio- en Televisieomroeporganisatie (VRT) offered its viewers the experience of broadcasting the Olympic Games on their personal devices using the MPEG-DASH protocol providing a display of the strengths of the adaptive streaming [11]. As a result, major web media delivery contenders have adopted MPEG's new standard or have shown interest in adopting it in the near future.

1.3.2 Brief History

In recent years, adaptive streaming video is growing in popularity as the media content delivery standard for several user devices and electronics [12]. Adaptive streaming is comprised of a server and client software which interact with each other to measure a client's throughput capacity and adjusts the quality level of the projected video accordingly.

Delivering a media file with no dynamic quality adjustment was not enough. It could not support the midstream switching of a video stream to a number of available resolutions depending on the client's network conditions and connection speed. On top of that, the interruptions and buffering delays during playtime are unavoidable when a client's internet connection could not support the quality of the selected video or when it presents fluctuations.

MPEG responded to the clear need described above by issuing an official call for proposing an HTTP adaptive streaming standard in April 2009 [13]. It developed the MPEG-DASH specification after cooperating with several expert groups and accepting collaboration from other standard organizations such as the Third Generation Partnership Project (3GPP). MPEG's project was coordinated with other industry organizations such as the Digital Entertainment Content Ecosystem (DECE LLC), the Open IPTV Forum (OIPF), and the World Wide Web Consortium (W3C). Also, a big number of global multinational companies were involved, most notably Microsoft, Netflix, and Adobe. All this resulted in the MPEG-DASH standard [15] being developed as timelined in reference [13].

The fact that it uses the standard HTTP port, thus avoiding firewalls, proxies and cache, has increased its popularity and efficiency. Since its establishment, the main protocols implementing the HTTP adaptive delivery have been: Microsoft Smooth Streaming, HTTP Dynamic Streaming and HTTP Live Streaming [5] [14]. They are all presented in the table in figure 1.9. In order to be served media content from any server, user devices must support all of the previous, since every one of these protocols utilizes different formats varying in structure. Theoretically, the MPEG-DASH standard bridges the operating differences of servers and clients created by different vendors, allowing for a client that supports the standard to stream media content from any standard-based server.

1.3.3 The basic scenario

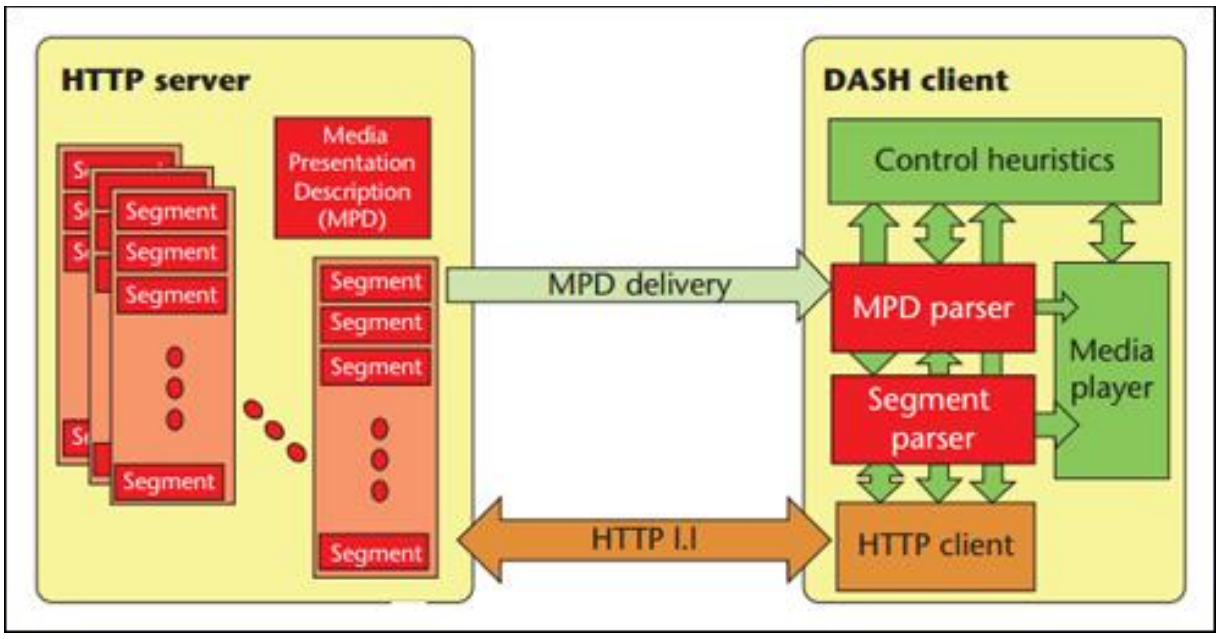

Figure 1.7: The MPEG-DASH process timeline

Reference [13] quotes a simplistic but descriptive example of using MPEG-DASH. The architecture described in the example is the following: An HTTP server undertakes to deliver stored content through HTTP. The media content consists of a segmented file and a media manifest file. The segments combined form media streams which deliver parts of the requested file, while the MPD manifest file gives information on the available content and its primary characteristics in order for the procedure to be coordinated.

Obtaining that MPD file is the first action of an MPEG-DASH client. The transmission method of the MPD file can either be an HTTP link or an email request, a broadcast or other. The MPEG-DASH client receives the MPD file (well presented in [12]) to extract indispensable information about the media content such as timing, availability, formats or encoding alternatives. The MPD file can also notify the client for accessibility options, different resolutions available or even digital rights.

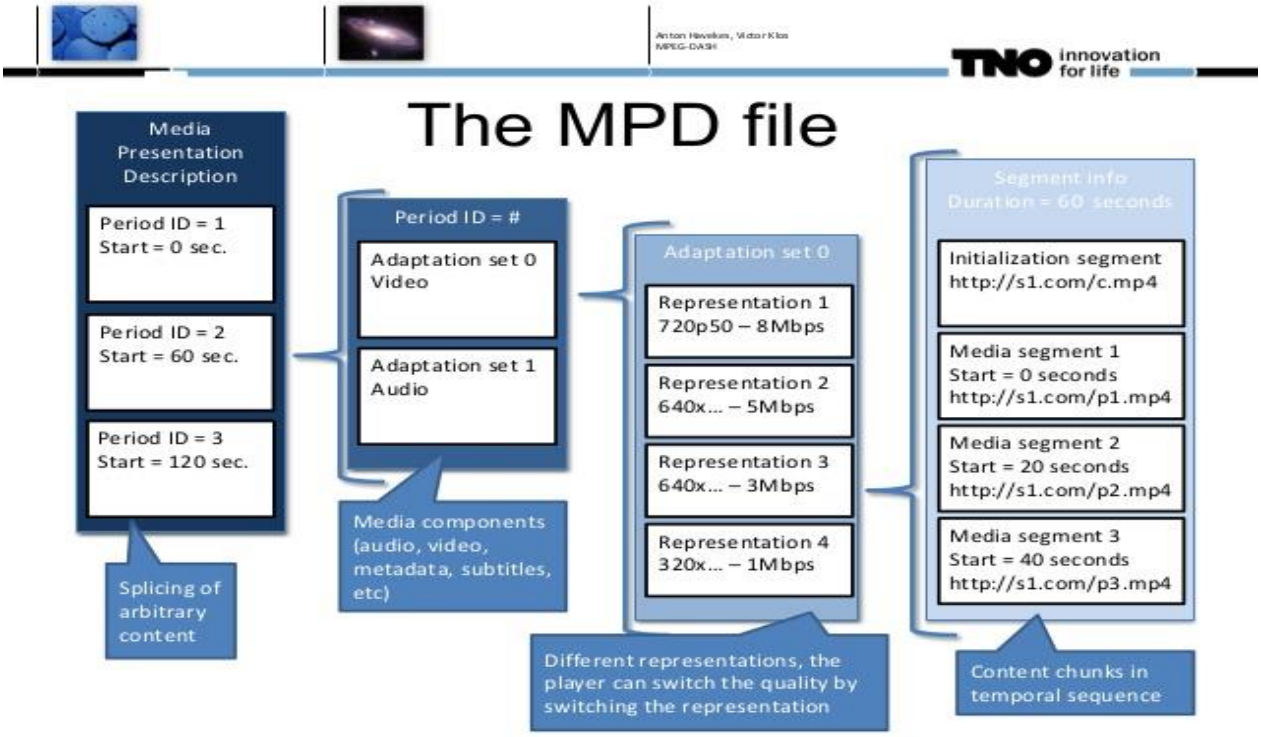

Figure 1.8: Example of the MPD file data model (by Dutch company TNO)

In the described example, the client has control over the choice of streams, and so chooses one by requesting and fetching some of the available media segments. The first segments that are received by the player do not directly feed the player since there is a need for buffering and thus they are directed to feed the buffer.

After this need is satisfied, the client continues requesting the next segments of its choice while analyzing network traffic to detect any connection instabilities. If a drop on connection speed is detected, the client adapts to it by requesting a segment of a smaller size, or lower resolution until the controller concludes that the player can maintain a stable buffering level.

However, as we previously mentioned, the client's level of control, the adaptation decision-making and the player's behavior over time are not strictly defined in the MPEG-DASH specification [15]. It is only the above basic structure and the generic purpose of the segments and MPD files that remain unchanged between different media delivery applications. Values such as segment size or duration vary from application to application depending on the implemented switching logic [16]. A basic overview of the MPEG-DASH Client-Server model is shown in figure 1.10.

Proprietary solution	Silverlight Smooth Streaming – MSS	HTTP Live Streaming – HLS	HTTP Dynamic Streaming – HDS
Owner	Microsoft	Apple	Adobe
Data description	Manifest (XML) [44]	Playlist file (M3U8) [35]	Manifest (F4M) [45]
Video codec	H.264, VC-1	H.264	H.264, VP6
Audio codec	AAC, WMA	AAC (HE, LC), MP3, AC3	MP3, AAC
Format	fMP4 [34] *.ismv + *.isma files	M2TS [35] *.ts files	fMP4 [37] *.f4f files
Segment length (typical)*	2 s [34]	10 s [35]**	2 – 5 s [37], [46]***

Figure 1.9: Main Adaptive-HTTP protocols

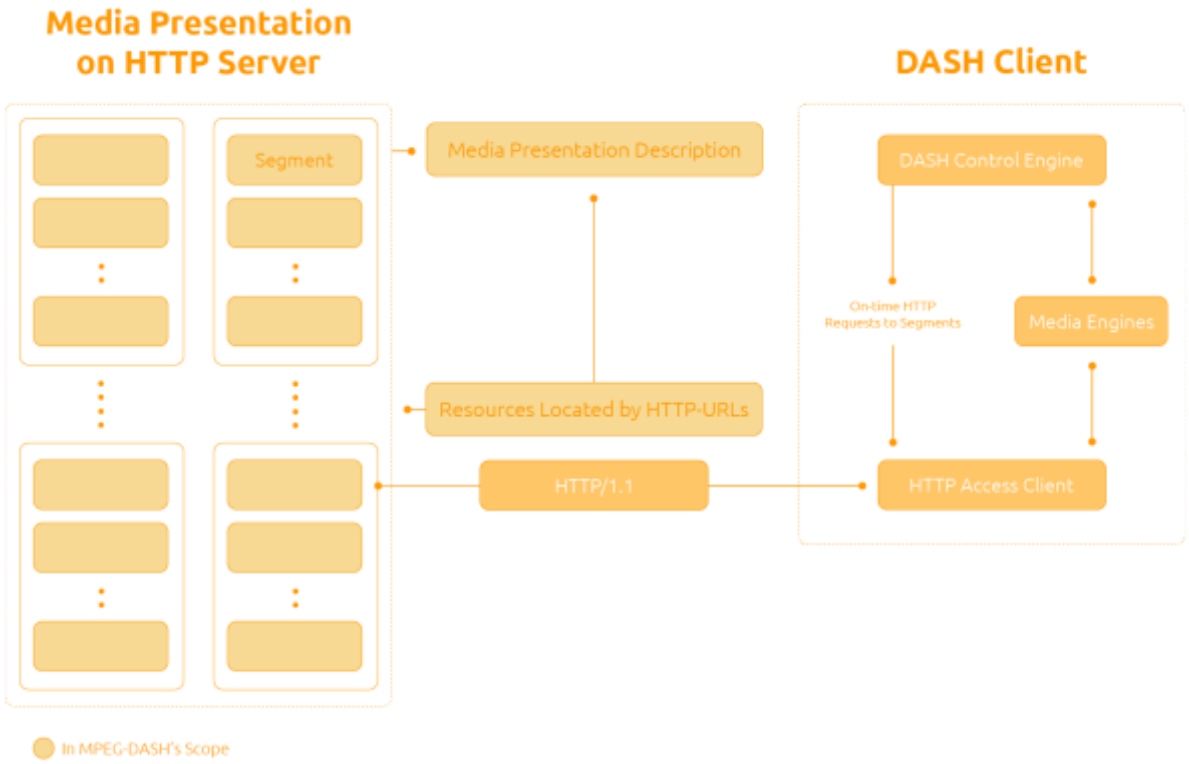

Figure 1.10: MPEG-DASH Server – Client Overview

1.3.4 MPEG-DASH major advantages

Adopting the MPEG's new standard leads to numerous major advantages [13]. Essentially, it is widely supported by all platforms and technologies which was a primary goal of its design process, thus eliminating incompatibilities. This allows for users to switch devices without suffering viewing restrictions and therefore increases the audience of every application using the MPEG-DASH model.

Secondly, being backed up by leading players of the media delivery market [12], MPEG-DASH has minimized problems concerning delivery or different forms of compression. Consequently, it allows for competitive delivery of media content to desktop and mobile applications [17] ensuring that users have universal coverage regardless the viewing means on their end. It is also guaranteed to receive regular support and evolvement, giving the impression of a standard that is here to stay.

As per content delivery providers, they rest assured since they are not required to create several file versions which would be the case for other, format-specific standards [13]. That idea of allowing control to anyone interfering with it and at the same time bringing platforms of different origin together constitutes the main philosophy of the MPEG-DASH development group.

2. LONG TERM EVOLUTION NETWORKS

2.1 Brief History

LTE is the latest step of a series of evolution in mobile network technology towards the advancement of next generation telecommunication networks [19]. During this evolution, mobile network design and modelling gradually stopped being a matter of domestic dispute and passed into the hands of international standardization organizations like ITU.

Stages in evolution of mobile technology can be divided in generations, starting from the invention of cellular networking for land networks, in 1947 by AT&T. Although cellular networks could make use of a different frequency per different cell, expensive equipment and high power requirements initially limited their availability only to an in-vehicle use.

Vast development in mobile telecommunication networks started during the 1980s, while mobile communications began to attract International interest. This generation's network, named “First Generation Network”, essentially consisted of a number of independent networks with different characteristics and names around different areas: AMPS in America [22], TACS in Western Europe [23], NMT in Scandinavia or J-TACS in Japan and Hong Kong [24]. It was mainly analog and only able to carry voice or related services. At the same time, it could not be considered undoubtedly reliable let alone that the supporting devices remained enormous.

The advent of digital technology during the 1980s, however, created the need of developing a digital system for mobile communications. The “2nd Generation Networks” (2G) were the result of a cooperation between several international telecommunication agencies. In Europe, under the supervision of the European Telecommunications Standards Institute (ETSI), emerged the Global System for Mobile communications (GSM) which rapidly became the global standard.

Digital technology produced devices with higher capacity batteries and smaller size, improved network capacity and more reliable service. While at first the available service was limited to voice, data transfer services such as the “Short Message Service” (SMS) [25] were quickly added. The evolved forms of GSM that followed, GPRS (General Packet Radio Service) and EDGE (Enhanced Data rates for GSM Evolution) [26] introduced the concept of packet switching in mobile networks and were able to provide more advanced services, paving the way for next generation’s (more complex) mobile networks.

The development of “3rd Generation Networks” (3G) [21], which could utilize a bigger bandwidth and made use of the radio interface known as “Universal Terrestrial Radio Access” (UTRA) [20], highlighted the need for unified version of Internet and Mobile services. The key objective was the ability to provide service “anywhere” and “anytime” to a user, meaning that any mobile user would be able to start moving and still enjoy the same services even if those were provided by other systems and not directly by 3rd generation systems.

Even though all development of 3G networks is currently managed by 3GPP, it had started almost at the same time as that of 2nd generation networks, long before its existence. The globalization of mobile standards played an important part in this process. Already since late 1980s the International Telecommunication Union (ITU) had been working on a 3rd generation network called “IMT-2000” [27]. Meanwhile in Europe and Japan operations were carried out towards the development of a multiple access prototype based on Wideband CDMA. Until the establishment of 3GPP in 1998, the solutions proposed by Europe and Japan merged into a single prototype called Universal Mobile Telecommunication Services (UMTS) [28], as a result of the standardization process of ETSI started in 1996.

Responsible for coordinating the development of 3rd generation networks and their evolution into 4th generation networks is the ITU Radio Communication Sector (ITU-R), organized respectively under the name of IMT-2000 and IMT-Advanced. Their primary goal is the classification of new technologies in “families of standards” while drafting proposals for their advancement and development as well as coordinating all standardization sectors responsible for implementing these proposals, with 3GPP posing as the most significant one. Another important obligation of ITU-R is setting the part of the bandwidth that may be utilized in a new technology and when that spectrum will be used paired and not paired.

However, it all comes down to 3GPP being the most important body within the scope of ITU. Under 3GPP’s responsibility fall the 2G technologies (GSM, GPRS, and EDGE) which are based on Time-Division Multiple Access (TDMA) and Frequency-Division Multiple Access (FDMA) [29] as well as the 3G technologies: UMTS which uses Code Division Multiple Access (CDMA) and LTE which is built on Orthogonal Frequency-Division Multiplexing (OFDM).

The advancement of the rest of 3GPP’s technologies continues in parallel with the LTE’s. The transition of UMTS to HSDPA (High Speed Downlink Packet Access) and HSUPA (High Speed Uplink Packet Access) was continued in Release 5 and 6 respectively, and is known as High Speed Packet Access (HSPA), as described in reference [21].

HSPA’s expansion went on with the newer HSPA+ which added higher grades of modulation and the use of Multiple Input Multiple Output (MIMO) antennas [30]. LTE is benefiting from the improvements in 3G networks but also adds obligations to these networks such as the requirement to work with Frequency-Division Duplex (FDD) and Time Division-Duplex (TDD) methods or even Time-Division Synchronous Code Division Multiple Access (TD-SCDMA) [31].

It is obvious from mobile networks’ vast development that they are heading towards more flexible, packet-switched systems that will be able to offer multiple service packages with a quality of service comparable to wired networks.

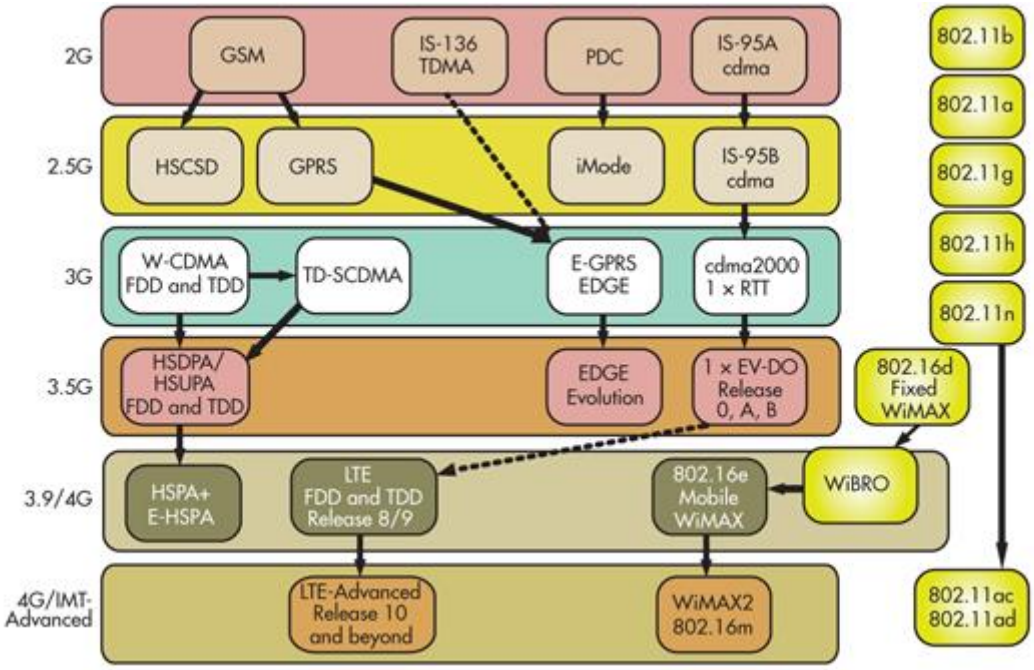

Figure 2.1: History and evolution of cellular technologies

2.2 Objectives of the LTE system

Rapid growth of mobile devices over the past 20 years in conjunction with the number of available services around the Internet have been the main reasons of transfer to 4th generation networks. The need for Internet services in mobile devices and the evolution of mobile technology systems featured as Mobile Broadband was aiming to provide services on top of the IP protocol.

The start came with GPRS which provided data transferring capabilities through packet-switched systems and continued with HSPA. LTE is designed from the beginning to only use packet-switched networks but with a flatter architecture. The Long Term Evolution is completed by the evolution of the core network under the name System Architecture Evolution (SAE) which includes the Evolved Packet Core (EPC). LTE along with SAE form the Evolved Packet System (EPS).

The main objectives that have been set for LTE development can be briefly listed as such:

- On duplex mode: LTE must support both FDD and TDD duplexing modes.
- On Bandwidth: a tiered spectrum use with a bandwidth set of 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, and 20 MHz

- On Throughput: a 3-4 times better user throughput average per MHz in downlink and uplink than HSDPA and HSUPA respectively.
- On peak transmission rate: it has a theoretical net bitrate capacity of up to 100 Mbit/s in the downlink and 50 Mbit/s in the uplink if a 20 MHz channel is used – or even more if a Multiple-Input Multiple-Output (MIMO) antenna array, is used instead.
- On peak spectral efficiency: 3 to 5 times more spectrally efficient than HSPA assuming there is no MIMO in HSPA.
- On Mobility: supports user high speed movement allowing for terminals moving at up to 350 km/h (220 mph) or 500 km/h (310 mph) although LTE is optimized for low mobile speed from 0 to 15 km/h.
- On Latency: a user-level requirement of reducing round-trip latency from the user to the base station down to 5-10ms while on system-level connection establishment should be as short as possible to optimize battery consumption. This time is defined as the elapsed time to shift between an idle state (RRC_IDLE) to a connected status (RRC_CONNECTED) and is required to be less than 100ms. User-level latency might possibly vary depending on the volume of data being transmitted and the broadcasting conditions.
- Cost effectiveness and interoperability: LTE is required to have the ability to work in cooperation with the existing UTRAN/GERAN systems as well as other non-3GPP systems. It must be able to support a handover procedure to and from these systems. LTE must also contain simple functions for user devices in order to ensure power saving for the user.
- Quality of Service: an end-to-end support to maximize quality of service even for users at the edge of a cell, even for demanding services like VoIP.

2.3 LTE Transmission Modes

To achieve the above goals, LTE introduces a number of physical layer transmission modes which reduce complexity in the system and user equipment and are categorized as shown in figure 2.2. These techniques also allow for a flexible development of the existing spectrum, especially in the case of LTE Advanced (LTE-A), and a good use of and coordination with other 3GPP technologies.

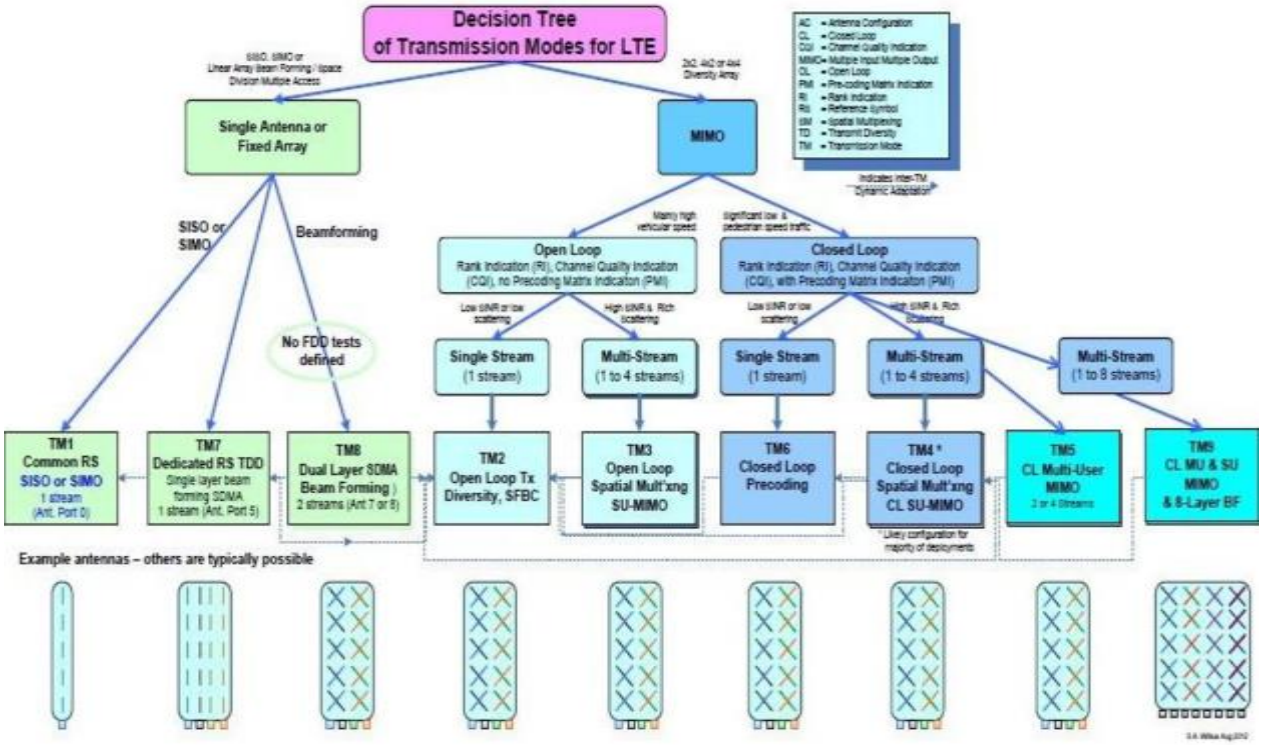

Figure 2.2: LTE Transmission modes tree (by 4G Americas)

2.3.1 Multiple Antennas

The use of multiple antennas in LTE networks is prominent because it grants the ability to exploit the spatial domain, along with the frequency domain, achieving a higher spectral efficiency. The realized (by the use of multiple antennas) spectral efficiency is, under normal circumstances [32], linearly dependent on the minimum number of transmitting and receiving antennas.

The application of multiple antennas can be implemented through different ways and, although most have some theoretical advantages against the others, not all of them can be easily applied in practice. Usually, the aforementioned ways consist of the ones described below [33]:

- Diversity Gain: Use of spatial diversity in signals for a more reliable transmission by increasing the durability of the channel against multipath fading.
- Array Gain: High energy concentration from one or more directions to a specific user or a number of different users close to each other is called beamforming, as explained in reference [35]. Beamforming techniques improve noise tolerance which results in significantly better coverage and a wider range.
- Spatial Multiplexing: The transmission of multiple data flows to one or more receiving users in different space levels. Spatial Multiplexing, shown in figure 2.3 below, may improve LTE transmission rates or channel capacity if data flows are directed to a single user or multiple users respectively.

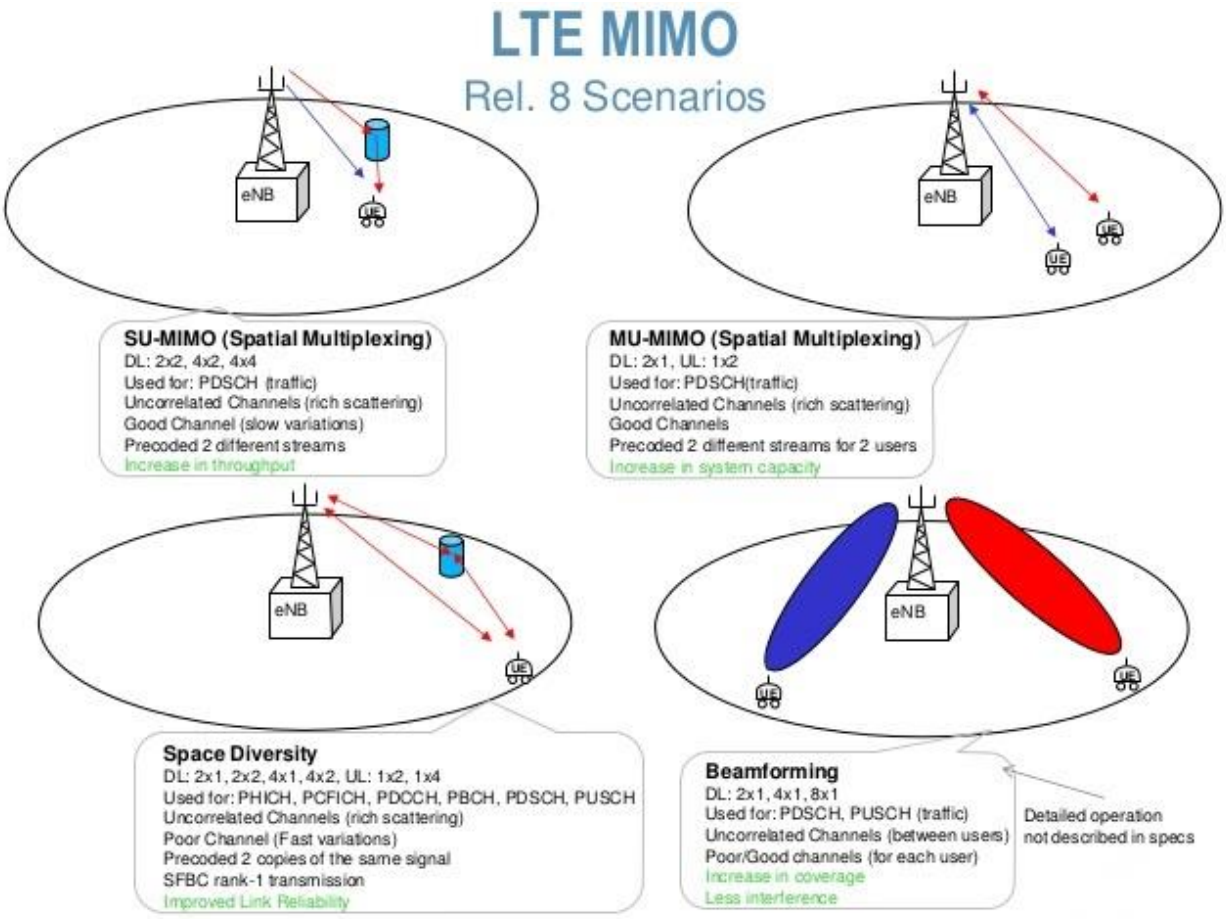

Figure 2.3: MIMO release 8 scenarios

2.3.2 Orthogonal Frequency-Division Multiple Access

As in figure 2.4, LTE networks make use of Orthogonal Frequency-Division Multiple Access (OFDMA) technology on the downlink and Single Carrier - Frequency Division Multiple Access (SC-FDMA) on the uplink. OFDMA relies on OFDM (Orthogonal Frequency-Division Multiplexing) to provide multiple access.

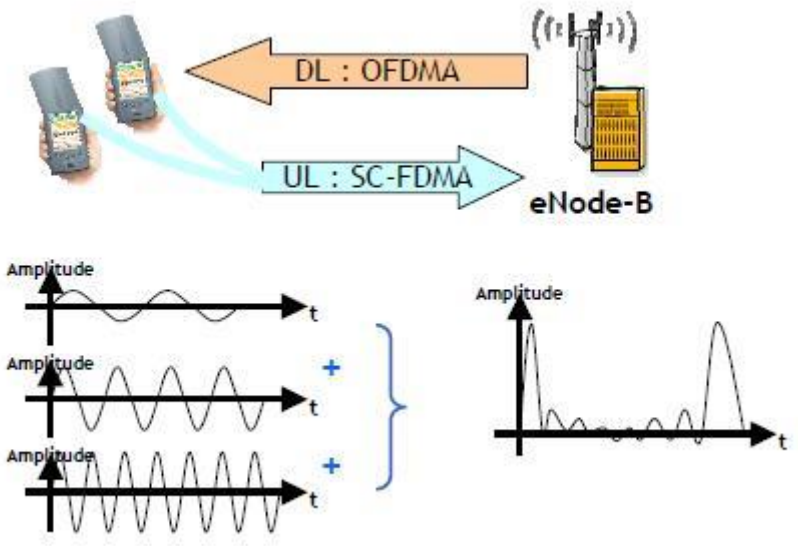

Figure 2.4: DL and UL in LTE

OFDM divides any given channel into many narrower subcarriers of 15 kHz, in an orthogonal way that either one by one or in groups they form independent data transmission streams (see figure 2.5 below). In OFDMA subcarriers may be shared among users allowing for simultaneous data transfer to different users along with control channels and pilot symbols with minimum interference. The advantages of the technique described above are numerous. It offers:

i. The ability to utilize different parts of the spectrum without a need to change system parameters or user equipment.
ii. Resource reservation from different parts of the spectrum to different users and an independent form of scheduling for all parts which resembles that of a singular spectrum approach.
iii. Extended flexibility in fractional frequency reuse as well as interference suppression and management.
iv. Robustness in mitigating inter-cell interference and temporal dispersion of a signal.
v. The construction of modern receivers with low complexity and high efficiency at a minimized cost.

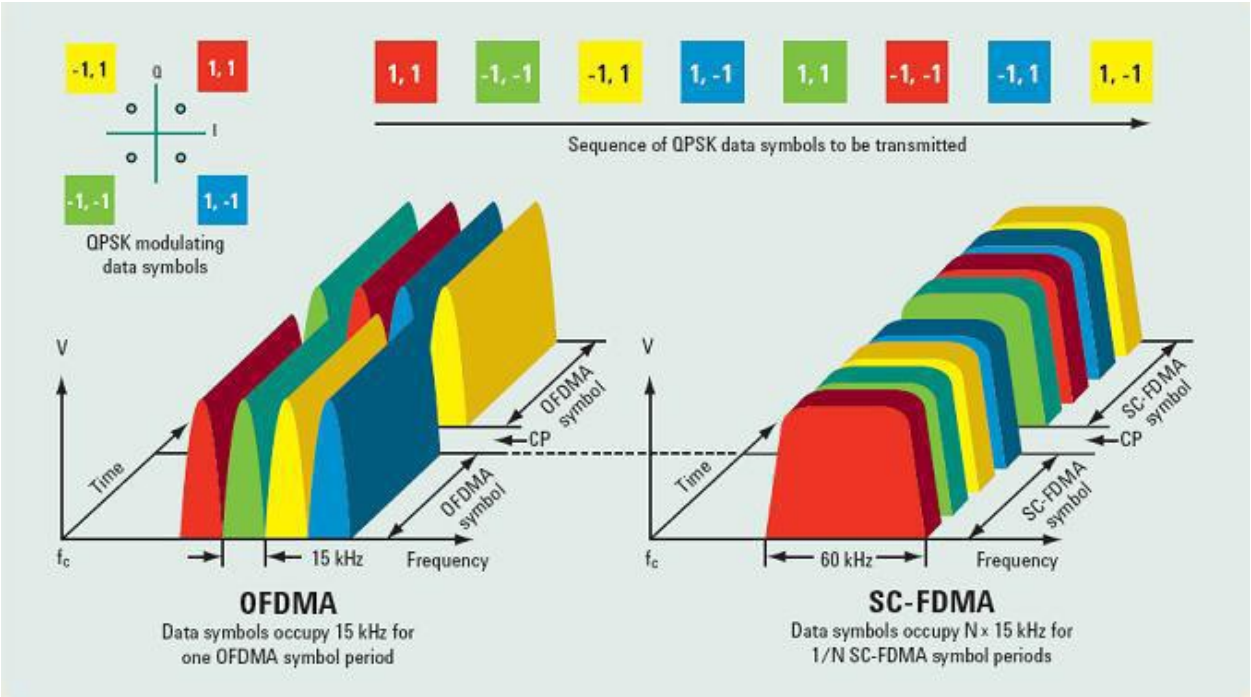

Figure 2.5: OFDMA and SC-FDMA in LTE

The fact that OFDMA can exploit better the available channel bandwidth compared to FDM is of great value. The principle of orthogonality in subcarriers contributes to a finer exploitation of the spectrum and relieves the system administrators from using guard bands for splitting the subcarriers.

To its disadvantage and despite its many benefits, OFDMA unfortunately requires high power consumption. Transmitters, in contrast to the receivers, have a higher Peak-to-Average Power Ratio (PAPR) in an OFDM signal and that heavily increases their cost. Because of this, OFDMA is primarily used in downlink, since the development cost for pricey transmitters in base stations is less important to network carriers than the mobile equipment which is offered to their users.

In contrast, high PAPR in uplink processes may be covered by the needs of mobile devices in emitting power and battery capacity. For this reason, uplink uses SC-FDMA (as in figure 2.5) which offers the same flexibility in frequency management but with a significantly lower demand for power consumption. [33][34]

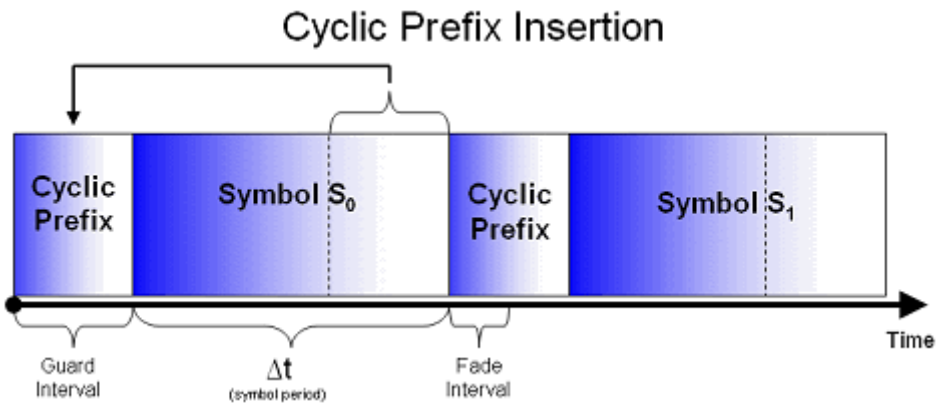

Figure 2.6: Cyclic Prefix Insertion in OFDM

Every transmission to the radio interface is potentially subject to corruption and a reason for that is the signal's temporal dispersion. As a result, the appearance of interference is evident not only in OFDM symbols but also between subcarriers. For their validity against temporal dispersion, OFDM symbols are protected by the Cyclic-Prefix Insertion (CPI) (figure 2.6) during transmission.

In this insertion technique, the last piece of every OFDM symbol is copied and appended to its beginning increasing the size of the symbol and decreasing its spectral efficiency. Validity against temporal dispersion is achieved if the duration of that dispersion is less than or equal to the cyclic prefix duration. The cyclic prefix duration can be defined as normal or extended, depending on current transmission circumstances.

2.4 LTE Architecture

2.4.1 Overview

The LTE standard has been specifically designed to support only packet-switched services unlike all circuit-switched models of previous cellular systems. It aims to provide a reliable service including seamless IP connectivity between User Equipment (UE) (specified in [36]) and the Packet Data Network (PDN), without any major disruption to the end users' applications during mobility.

LTE poses as the evolved version of the Universal Mobile Telecommunications System (UMTS) radio access utilizing the Evolved UTRAN (E-UTRAN) [37]. It is followed by an evolution of the non-radio aspects known as "System Architecture Evolution" (SAE) [38], including the Evolved Packet Core (EPC) network [39]. Together SAE and LTE comprise what is known as the Evolved Packet System (EPS).

EPS is built on the concept of EPS bearers that route all IP traffic from a gateway in the PDN to the UE. An IP packet flow in the EPS with a predefined Quality of Service (QoS) between the gateway and the User Equipment is called a bearer. The EPC along with the E-UTRAN prepare, release and manage bearers as it is required by applications being used.

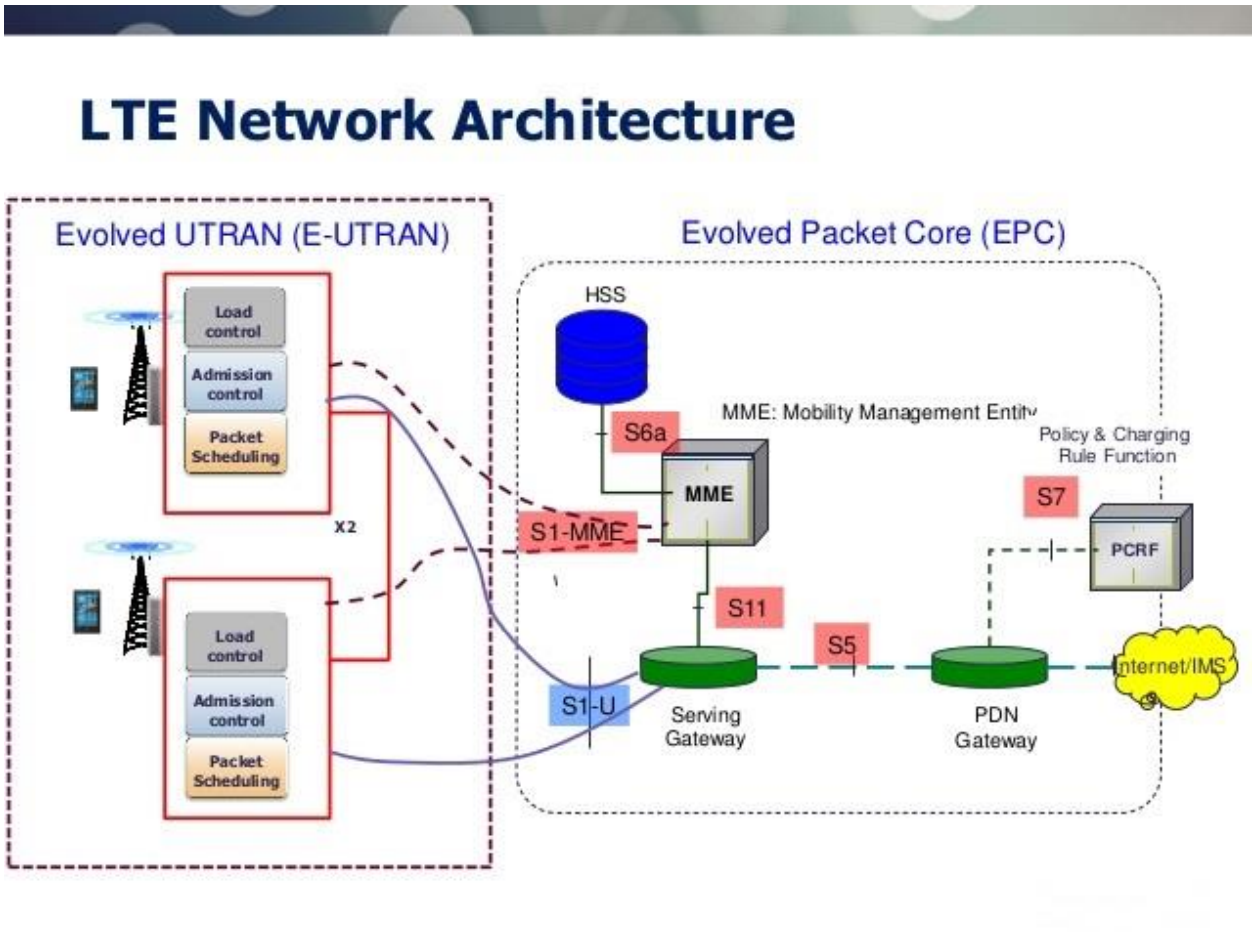

Figure 2.7: LTE general architecture (by AIRCOM)

The overall EPS network architecture which gives an overview of all the necessary functions provided by E-UTRAN and the Core Network (CN) are comprehensively described below. The bearer path as well as QoS aspects are outlined from end to end, including the process of establishing an EPS bearer. The LTE protocol stack across the different interfaces is detailed, giving an overview of functions offered by different layers in the protocol stack. All network interfaces are presented in high detail focusing especially on the E-UTRAN interfaces and the common processes used across them, most notably all the procedures supporting user mobility.

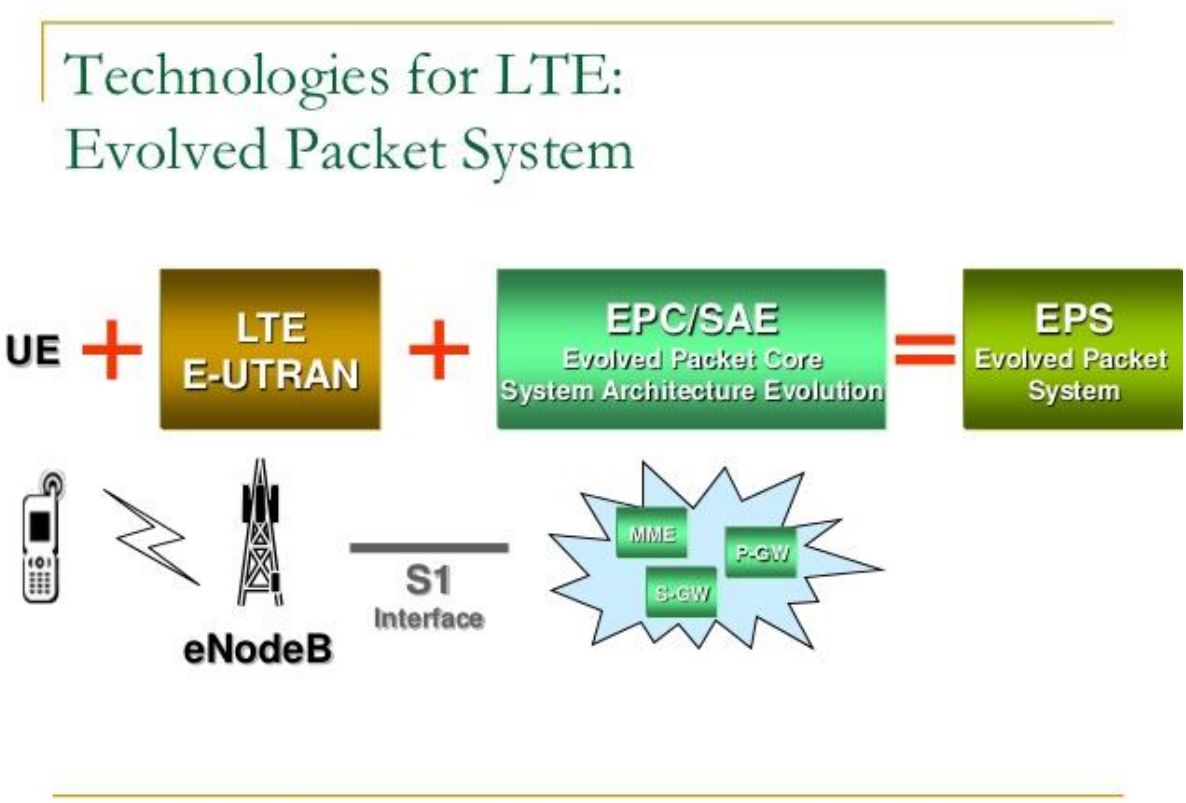

Figure 2.8: Elements forming the EPS

EPS (figure 2.8) ensures that a user is connected to a PDN over IP to run services like Voice over IP (VoIP) or access the Internet, while EPS bearers are generally linked with a pre-specified QoS guarantee. In order to provide several QoS streams or connectivity for a number of PDNs more than one bearers are usually formed. In a typical case where a random user is browsing the web, constantly downloading content while participating in an IP based voice call at the same time, it would be up to the EPS bearer established for the VoIP call to provide the guaranteed QoS for the voice call. Meanwhile, most suitable for non-real-time services (i.e. FTP downloading, browsing) would be a best-effort bearer. Moreover, a fact worth noting is that, as described in [40], the EPS network is configured to protect the user offering privacy during his connection time and securing the network against any kind of fraudulent and elusive behavior.

In overview, E-UTRAN as the access network and EPC as the core network together comprise our whole networking concept. Although the core network includes numerous logical nodes, the access network consists, in essence, of just a single node known as the evolved NodeB (or eNodeB) [41], which in turn connects to the user equipment. However, all these network elements share a careful interconnection through standardized interfaces so that they are able to provide interoperability to multiple vendors. This offers the opportunity for network administrators to piece together a variety of network elements manufactured by different individual vendors. In practice, depending on commercial factors network engineers may combine or split these logical network parts to build their implementations on the physical layer. Both the EPC and E-UTRAN network elements and their functional branching are explained in higher detail in the parts below.

2.4.2 The Evolved Packet Core

The Evolved Packet Core (EPC) network is the part of the network responsible for the master control of the User Equipment as well as the establishment of the EPS bearers [39]. The most notable nodes of the Evolved Packet Core are: The Packet Data Network Gateway (P-GW), the Serving Gateway (S-GW) (figures 2.9, 2.10) and the Mobility Management Entity (MME) [42].

Additionally, EPC includes other logical parts and functions, most notably the PCRF (Policy Control and Charging Rules Function) and the HSS (Home Subscriber Server) [43]. Control of VoIP or other media applications is handled by a system outside the EPS called IMS (IP Multimedia Subsystem), specified in [44], mainly because of EPS's responsibility to provide strictly only a bearer path of a specific QoS level. The logical core network nodes are discussed in every detail below:

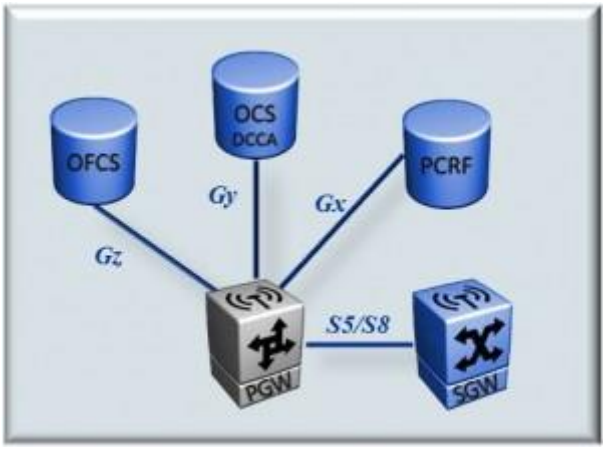

Figure 2.9: P-GW interaction with the network

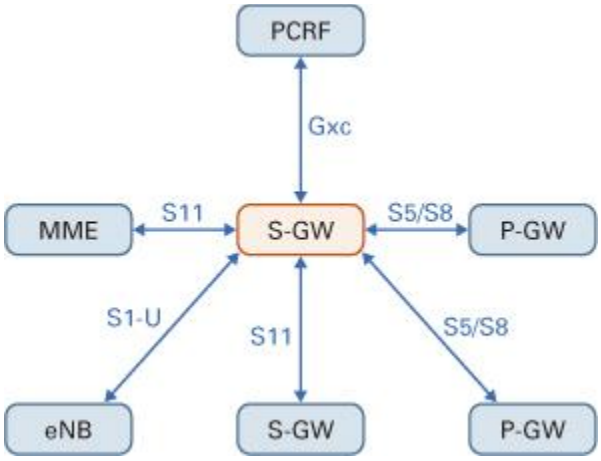

Figure 2.10: S-GW interaction with the network

- P-GW: The main responsibility of the P-GW is to allocate an IP address for the user (UE) and enforce the QoS level required. It filters downlink user IP packets into the different QoS-based bearers based on Traffic Flow Templates (TFTs). The PDN Gateway enforces a QoS for Guaranteed Bitrate Bearers (GBR) and serves as a mobility cushion when communicating with non-3GPP technologies.

- S-GW: The Serving Gateway is the mobility “anchor” helping the bearers, responsible for transferring all the IP packets of a user moving between eNodeBs. It maintains all information on bearers while the User Equipment is in an EPS Connection Management - IDLE state (ECM-IDLE), temporarily buffering downlink data until the MME reestablishes the EPS bearers through paging. The S-GW also collects information for data charging, lawful interception and other network administrative tasks. What is more, it provides interconnection with other 3GPP technologies like GPRS and UMTS.
- MME: The Mobility Management Entity [45] is a control node that handles the signaling between users (the UEs) and the Core Network (CN) using the Non Access Stratum (NAS) protocols. The most important functions supported by MMEs may be listed as:
    - Functions related to connection management, including the establishment of a secure connection between the UE and the network which is usually managed by the mobility management of the NAS protocol layer.
    - Functions related to bearer management, including the release and maintenance of EPS bearers handled by the session management in the NAS protocol.

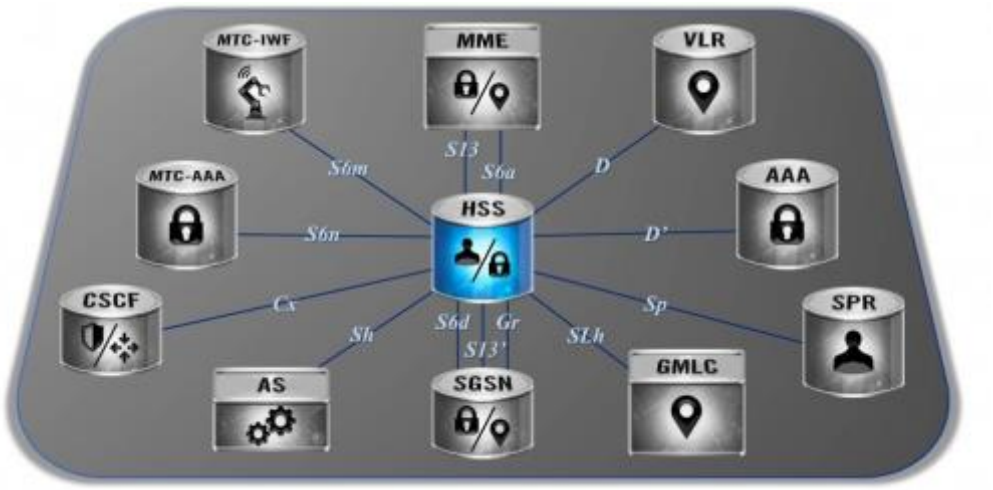

Figure 2.11: HSS in the center of the network

- PCRF: The Policy Control and Charging Rules Function implements the Policy Control Enforcement Function (PCEF) of the P-GW and makes all important decisions concerning policy control using flow-based charging functionalities [43]. The PCRF is also in charge of the QoS authorization process consisting of the current QoS class identifier (QCI) and the respective bitrates, which decides the way a specific data flow will be handled by the PCEF making sure it does not oppose to the user’s subscription.
- HSS: The Home Subscriber Server, shown in figure 2.11, holds information about users’ subscription data, most notably their subscribed QoS profile, the MME where they are currently connected and the PDNs to which they are authorized to connect in forms of an Access Point Name (APN) or a PDN address. [46] Additionally, the Home Subscriber Server possibly integrates the AuC (Authentication Center), in charge of generating security and authentication vectors.

2.4.3 The Non-Access Stratum (NAS)

The Non-Access Stratum corresponding to the UMTS, differs in that it allows faster establishment of the bearers acting as a form of cached information [47]. That is because when a piece of User Equipment attaches to the LTE network, the MME builds a UE context by downloading subscription information from the HSS and assigns an SAE Temporary Mobile Subscriber Identity (S-TMSI) to it. Additionally, this UE context holds a list of the established bearers and the capabilities of a terminal.

The MME keeps track of the UE's location using a procedure known as tracking area update, reducing processing in the UE and decreasing the overhead of the E-UTRAN. Every UE informs the network of a new location when exiting its current TA (Tracking Area) and while the UE is in the ECM-IDLE state, UE context information is retained by the MME.

In case the MME needs to wake an idle UE, it contacts eNodeBs in the UE's TA [45]. The eNodeBs in turn page the UE over the radio interface which performs a Service Request switching to the ECM-CONNECTED state and the E-UTRAN (through the MME) creates the information needed to reestablish the radio bearers.

During all the above signaling and data-sending procedures, security is managed completely by the MME which triggers a mutual authentication between the UE and the network and establishes security keys used for the encryption of the bearers [40].

In most cases the EPS intentionally allows Non-AS and AS procedures to run in combination to accelerate this idle-to-active transition and bearer establishment by executing necessary procedures in parallel, contrary to the UMTS.

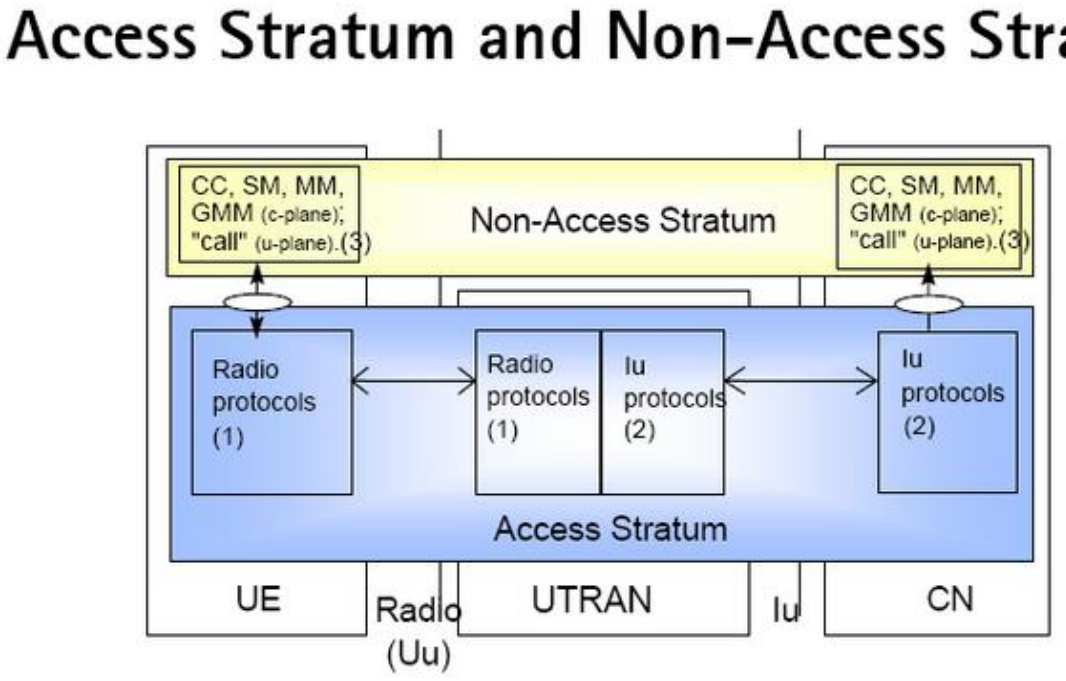

Figure 2.12: Access Stratum & Non-Access Stratum

2.4.4 The access network

E-UTRAN is the access network of LTE with a particularly flat architecture (shown in figure 2.13 below) consisting of a number of interconnected eNodeBs [20] [37]. Therefore, it can be said that under normal traffic circumstances the E-UTRAN has a decentralized control system.

These eNodeBs form a network with each other using an interface called X2, described in [48] and are externally connected to the EPC. ENodeBs connect to the S-GW through the use of the S1-U interface and to the MME by means of the S1-MME interface. At the same time, User Equipment connects to the eNodeBs by running the Access Stratum (AS) protocols.

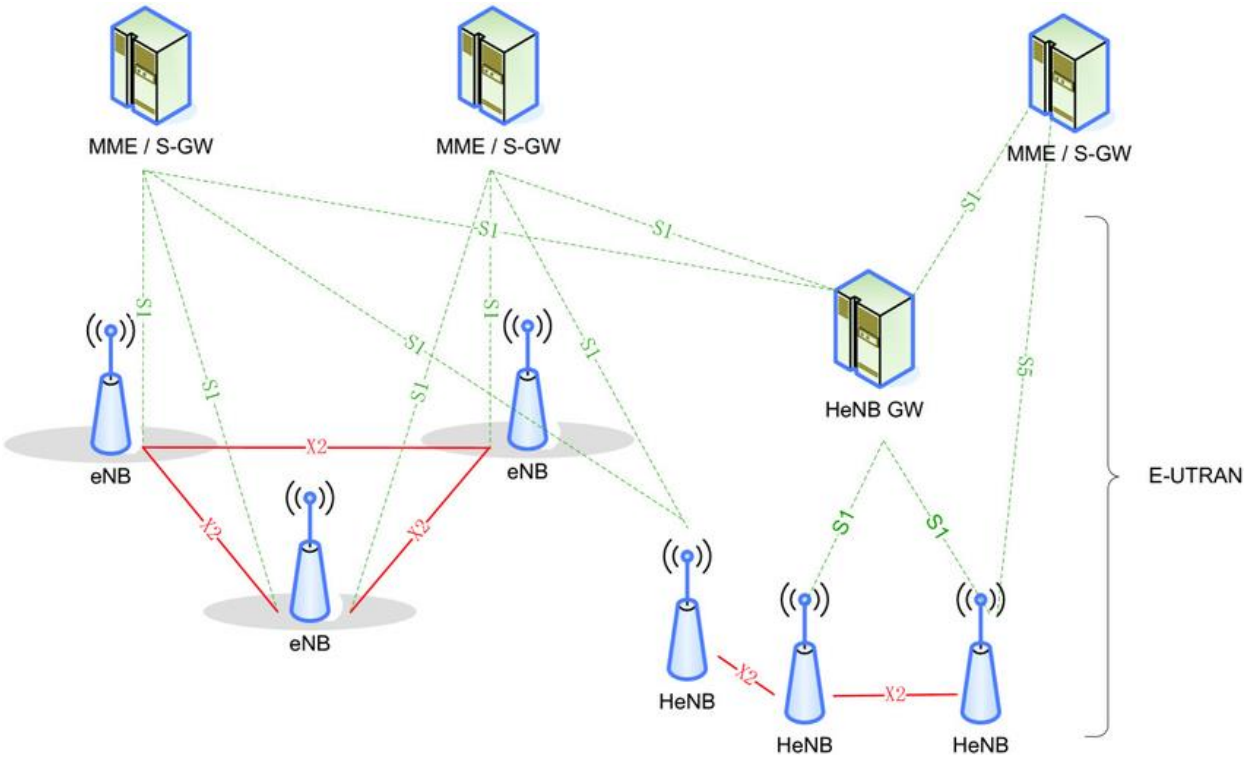

Figure 2.13: E-UTRAN architecture

The basic functionality operated in the E-UTRAN includes all radio-related tasks described below:

- EPC connection, which consists of establishing the bearer path for communicating with the S-GW and sending signals to the MME.
- Compression of IP packet headers to avoid the unnecessary overhead in small multimedia packets and make certain that the network is used in the most efficient manner.

- The RRM (Radio Resource Management) which includes radio-bearers-related functionality, most notably radio admission and radio bearer control, radio mobility control and scheduling as well as the dynamic allocation of resources for both downlink and uplink in the UEs.
- Data security is attained by ensuring all data that is sent over the radio interface is encrypted.

Having a distributed control system means that each eNodeB manages and executes tasks like the above for several cells, providing close coordination of protocol layers of the RAN. With current eNodeB design [41], the need for an exclusively engaged controller is rendered unnecessary making the system hard to fail with low latency and further more cost-effective.

However, due to the fact that E-UTRAN does not support a soft handover, but rather passes all user information from an eNodeB to another, additional functionality is needed to avoid data loss. This functionality is one of the tasks carried out by the X2 interface.

As per the connection between eNodeBs and the EPC, a number of MME and S-GW pairs form a pool that provides service to multiple eNodeBs located in a specific area. This concept is part of the S1 interface and guarantees a fast service to the eNodeBs while eliminating the existence of a single point of failure.

2.4.5 Protocol Architecture

From the protocol stack perspective there is a certain number of different protocols being used in the LTE, but the protocol architecture can be separated into the User plane architecture and the Control plane architecture. Therefore, all protocols are being described below according to their use in each of these two pillars of LTE's protocol architecture.

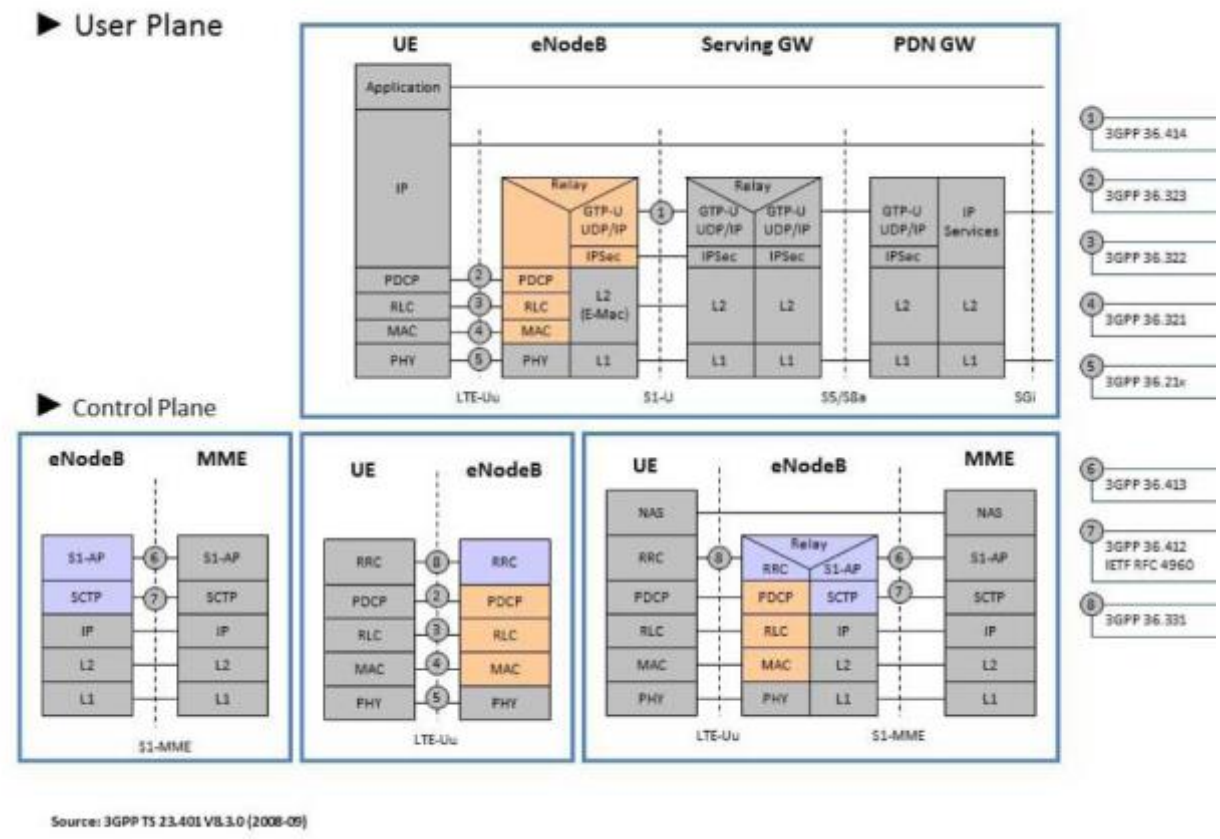

Figure 2.14: User Plane and Control Plane in the Protocol stack

2.4.6 User plane

The protocol that is used for tunneling over the Core Network interfaces S1 and S5/S8.1 is 3GPP-specific and is known as GTP (GPRS Tunneling Protocol) [50]. When an IP packet is tunneled from the P-GW to the eNodeB, encapsulated in an EPC protocol, and sent to the UE, different protocols are used by different interfaces.

The access network user plane protocol stack [49] (figure 2.14) includes the following layers that are terminated in the eNodeB:

- Physical (PHY) layer:

  Connecting the eNodeB with a UE, the physical layer in LTE networks supports the HARQ (a hybrid combination of high-rate forward error-

correcting coding and ARQ error-control) ensuring uplink power control and multi-stream transmission and reception.

- Media Access Control (MAC) layer:

  The MAC sublayer [51] provides (de)multiplexing of different RLC layers, priority management and error correction on UEs or across different logical channels of a UE, and reports traffic volume.

- Radio Link Control (RLC) layer:

  Apart from transferring upper-layer PDUs, the RLC reassembles or concatenates the packets, detects possible duplicates, controls the traffic flow and performs error-correction through HARQ. (see figure 2.15)

- Packet Data Convergence Protocol (PDCP) layer:

  The PDCP layer is primarily responsible for header compression and ciphering on the user plane [52]. Data handling such as protection and buffering during a handover is also assigned to PDCP, while MAC and RLC are both designed to start in a new cell after handover.

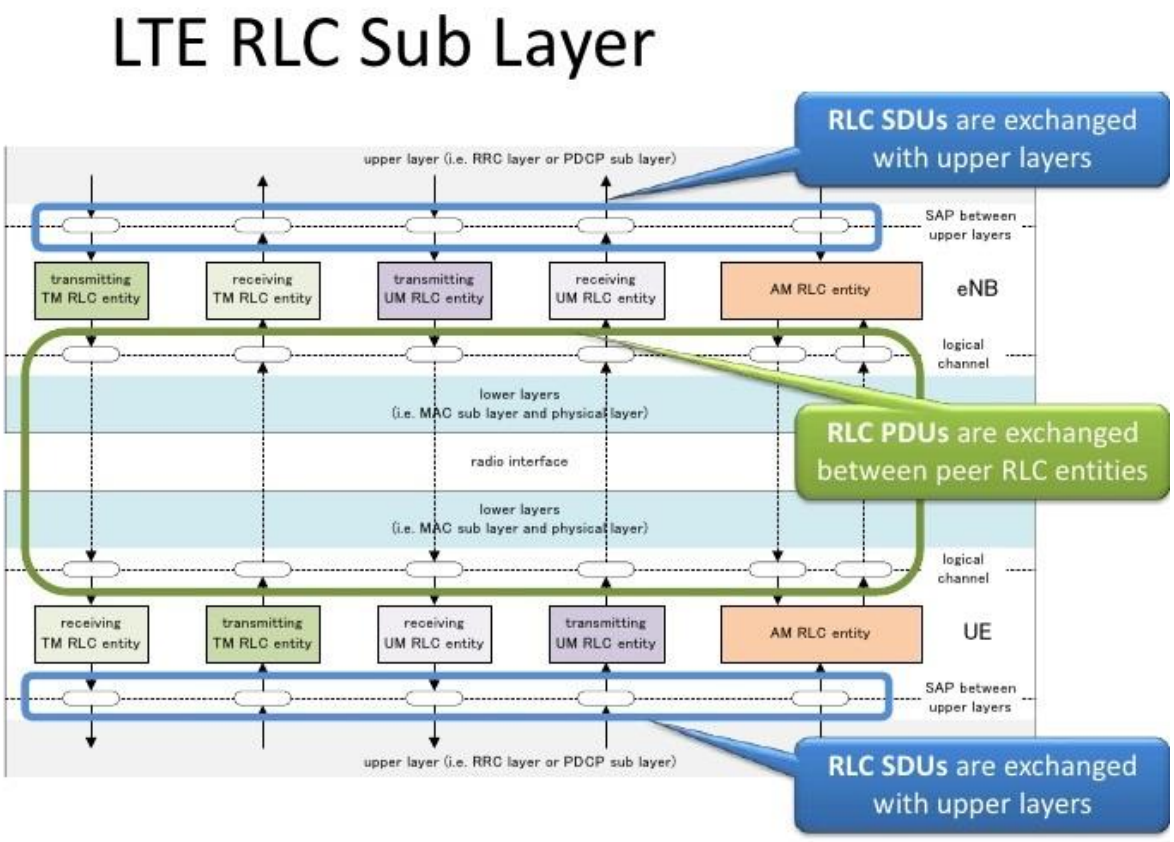

Figure 2.15: RLC Sub Layer

2.4.7 Control plane

The control plane protocol stack (figure 2.14) includes more or less the same protocols which are used in the user plane [49]. Their objective here is to successfully interconnect an MME to a UE.

The lower layers perform the same functionality as previously said for the user plane with the exception of a header compression function for the control plane and the addition of a NAS sublayer. The NAS sublayer works strictly between the MME and the UEs providing authentication procedures, idle-mode paging origination, idle-mode mobility handling, and overall security control.

The Radio Resource Control (RRC) protocol, known as layer-3 in the AS protocol stack, works on the Control plane as the main controlling function in the AS [53]. The RRC is responsible for configuring all the lower layers using RRC-specific signaling between the UEs and the eNodeBs, as well as for establishing EPS radio bearers. It essentially provides all broadcasting, paging and connection management, along with radio bearer control and mobility functions.

2.5 Quality of Service in LTE

2.5.1 EPS Bearers

An Evolved Packet System Bearer is defined as a transmission channel through an EPS network with a strictly specified set of data transmission characteristics such as QoS data rate and data flow control [54].

At any given time, a UE is probably handling several web applications with varying Quality of Service requirements. For example, VoIP applications are highly more demanding in their requirements for QoS in terms of avoiding delay and jittering. On the other hand, non-real-time applications like simple web browsing or FTP downloading value a much lower packet-loss rate without prioritizing a low delay. In order to support this difference in QoS demands among different applications, the EPS establishes a number of EPS bearers, each being associated with a different QoS level standard, researched in reference [55].

Consequently, EPS bearers are divided into the following two basic categories given the type of QoS they are intended to offer:

1) *Guaranteed Bit Rate* (GBR) bearers are the ones used for real-time demanding applications such as VoIP. During establishment, these are assigned a minimum GBR value by a control function according to the application's needs. To match this minimum GBR value, dedicated channel resources are allocated for and used by the EPS bearer.

   In case of a need for a higher bitrate than the GBR value, the bitrate needed can be provided by the EPS bearer as long as the resources available allow it. However, Guaranteed Bit Rate bearers also hold a Maximum Bit Rate (MBR) value, which effectively defines a ceiling on the maximum bit rate that can be supported by a GBR bearer.

2) *Non-Guaranteed Bit Rate* (non-GBR) bearers are the EPS bearers which do not guarantee any specific bit rate. The resource allocation for non-GBR bearers has a more flexible, more volatile nature allowing for variations at will and on the spot. These can be used for downloading and browsing applications. For such bearers, there is no obligation that bandwidth resources are necessarily allocated permanently to the EPS bearer.

As per the ARP (Allocation/Retention Priority) value of an EPS bearer, it plays a key role in call admission control when a bearer is requested and it should be decided if the radio is too congested for that bearer to be established. Moreover, it determines the prioritization of an establishment request for a new or an existing bearer. After the establishment of an EPS bearer, its ARP can no longer affect characteristics such as the rate control or scheduling of a bearer. Packet forwarding behavior will be entirely specified by QoS parameters of bearers like the abovementioned MBR, GBR or the QCI.

EPS bearers are paths which need to go through several different interfaces. A typical bearer will use the LTE-U interface between the eNodeB and the UE. The S1 interface

will be the one connecting the eNodeB to the S-GW, and after that the S5/S8 interface should come in between the P-GW and the S-GW.

Crossing these different interfaces, an EPS bearer is bound to a sublayer bearer with a separate ID on each interface. Therefore, network nodes are obliged to oversee and record the mapping to these different bearer IDs while the EPS bearer is on its path between interfaces.

An S5/S8 interface bearer transfers the packets of an EPS bearer from an S-GW to a P-GW and the opposite in the following way [54]:

First, the S-GW is storing a one-to-one mapping from that S5/S8 interface bearer to an S1 interface bearer. Then, the original bearer can always be identified by the GTP tunnel-ID across both the S1 and the S5/S8 interfaces. Between an eNodeB and an S-GW the IP packets of an EPS bearer are transported by an S1 bearer when the eNodeB stores an identical with the above one-to-one mapping between the two, as is the case for the radio bearer connecting the UEs to an eNodeB.

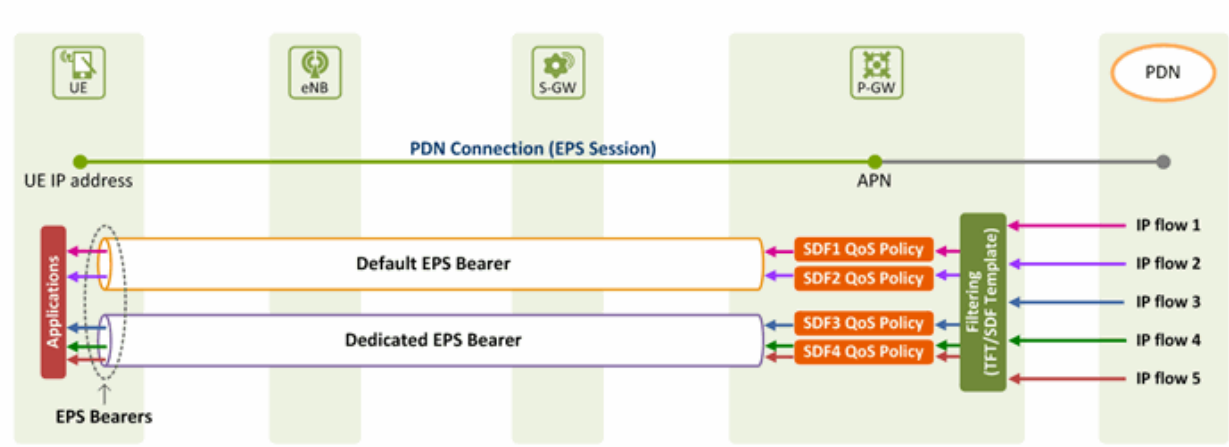

Figure 2.16: EPS Bearers and Traffic Flows

2.5.2 QoS Class Identifiers

Specifically in the E-UTRAN, the eNodeBs are in charge of providing the necessary QoS for an EPS bearer over the radio interface. Each bearer is associated with a specific QoS Class Identifier (QCI) [55]. QCIs include a priority value, a packet delay budget, a maximum acceptable packet loss rate and a QCI label providing the way it should be handled by the eNodeB. Not many QCIs have been standardized, giving possible vendors a sufficient understanding of the fundamentals of the service but leaving them the flexibility of selecting a custom queue management, QoS level policy and priority handling.

This way it is guaranteed to LTE network operators that regardless who is the vendor of the E-UTRAN equipment, traffic handling is highly consistent all over the area of the LTE network. Therefore, it is of little importance which of the standardized QCIs will the PCRF select for an EPS flow since the QCIs follow the same logic and are different only in the details.

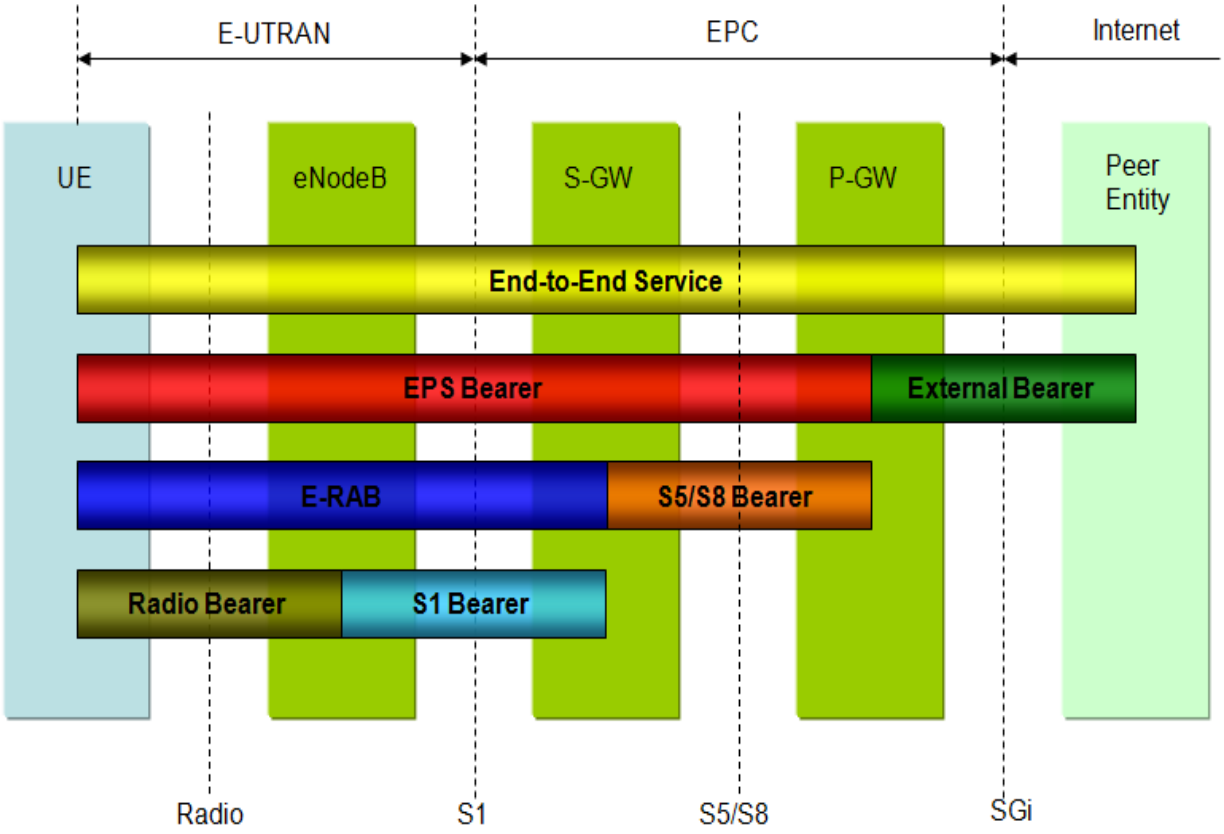

Figure 2.17: Bearer Hierarchy

The priority, the acceptable packet loss rate, and the packet delay budget described in the QCI label define how the scheduling policy of the MAC protocol that handles IP packets sent over the EPS bearer, its rate modification policy, its queue management and most notably how the RLC mode is configured. For instance, the scheduler might choose to send a packet with a higher priority first and leave a packet with lower priority back at the queue. The EPS bearers which have a significantly lower acceptable packet loss rate use what is called an Acknowledged Mode (AM) within the RLC protocol layer to make sure that all packets are delivered with success across the network's radio interface.

According to which EPS bearer an IP packet is bound to, it will be given the same treatment in terms of RLC, queue management and QoS configuration across all interfaces which that EPS bearer is entitled to [55]. In other words, if a need for a different QoS flow appears, the network must be inclined to provide and establish a separate EPS bearer in order to serve the QoS level upgrade. Once it does so, all user IP packets which required a more favorable treatment are channeled into the new EPS bearer.

This packet filtering system which guides IP packets into different bearers and was mentioned above is primarily built over on Traffic Flow Templates (TFTs). TFTs use IP header information to distinguish and divide packets which belong to different QoS categories, as for instance would be parts of a VoIP connection compared to parts of FTP downloading.

Information such as source/destination IP address or TCP port are used so that each packet can be guided in the respective bearer. In particular, a TFT associated with each

bearer in the P-GW which filters IP packets to EPS bearers in the downlink direction would be called a Downlink TFT (DL TFT), while a TFT which is found on the User Equipment with different bearers and filters packets to the uplink is an Uplink TFT (UL TFT).

QCI	Bearer Type	Priority	Packet Delay	Packet Loss	Example
1	GBR	2	100 ms	10^{-2}	VoIP call
2		4	150 ms	10^{-3}	Video call
3		3	50 ms		Online Gaming (Real Time)
4		5	300 ms	10^{-6}	Video streaming
5	Non-GBR	1	100 ms		IMS Signaling
6		6	300 ms		Video, TCP based services e.g. email, chat, ftp etc
7		7	100 ms	10^{-3}	Voice, Video, Interactive gaming
8		8	300 ms	10^{-6}	Video, TCP based services e.g. email, chat, ftp etc
9		9			

Figure 2.18: Standardized QCIs

When a UE first attaches to the LTE network, after the P-GW assigns it an IP address the EPC establishes a non-GBR bearer for the UE called the “Default Bearer” (see figure 2.17). The MME subsequently retrieves user subscription parameters such as QoS default values from the HSS and assigns them to the default bearer [55]. These values can later be changed through the cooperation of the PCEF and the PCRF. This is the minimum establishment that can happen during the process of a UE attachment to the network. The default bearer remains established through the whole lifecycle of a PDN connection ensuring that IP connectivity between the UE and the PDN will under no circumstances be interrupted.

All bearers additionally created after the default one are called “dedicated” bearers and their establishment may happen during or after the completion of attaching the UE to the network. Dedicated bearers can be both GBR and non-GBR bearers (though obviously only one or the other) and through identification they become clearly separated from the default as far as the E-UTRAN is concerned. Dedicated bearers in a UE may have been provided by more than one P-GWs and are usually established by the EPC network, either upon a UE bearer establishment request or following a trigger from the IMS domain.

As previously pointed, each EPS bearer has an associated QoS level and, consequently, each bearer should also be associated with the respective TFTs in case there are more than one bearers running on a UE. The bearer QoS-level parameters for every dedicated bearer are forwarded to the S-GW after being sent from the PCRF to the P-GW [55]. The MME is responsible to receive those values from the S-GW and pass them over to inform the E-UTRAN.

2.5.3 OTT Content Providers

As mentioned previously, video is poised to become the predominant data type flowing through networks, and this includes video tied to entertainment and video that is added to many non-entertainment services that we use today. As a result, handling video QoS is of imperative value to all Over-The-Top (OTT) content providers, and cannot always be left to a best-effort IP delivery method. We described above how this process is applied on Radio bearer, S1 bearer and S5/S8 bearer, collectively called as EPS bearer.

According to the standardized QCIs in LTE (figure 2.18), TCP-based progressive video streaming is assigned primarily to non-GBR type bearers. In the case of YouTube, for example, which uses the (TCP-based) Adobe HTTP Adaptive Streaming protocol, the IP flow would be channeled through a Default Bearer with QCI 6, 8, or 9. This non-GBR bearer would indicate a 300ms packet delay tolerance and a 10^{-6} acceptable packet loss error rate. The same would be the case for the on-demand service of the American premium cable network HBO, HBO GO.

It is worth noting that, although non-GBR bearers do not provide guaranteed bitrates, they still essentially manage QoS using parameters like A-AMBR and UE-AMBR. A-AMBR indicates the maximum possible bitrate for all best-effort services on an APN, while UE-AMBR is a value that limits the possible bitrate of best effort flows on a particular client. These parameters prevent the client from taking over all the available bandwidth of the interface and preserve control over traffic flows within the PDN.

On the other hand, Netflix, which offers a similar on-demand video streaming service, but is using the Dynamic Adaptive Streaming over HTTP (DASH) protocol would request to allocate a Dedicated bearer in order to guarantee a minimum bitrate to the user. Therefore, the application's IP flow would use a bearer with QCI 4 which allows for a 300ms delay tolerance, but would hold a priority of 5 having still a packet loss rate of 10^{-6}.

For services like VoLTE, we need to provide better user experience and this is where Dedicated bearer would really come handy. VoIP pioneer Skype, for example, would also use a Dedicated GBR bearer for calls made using the app, with a QCI of 1 which reduces delay to 100ms and acceptable packet loss to 10^{-2}. A bearer with QCI 1 has a priority of 2, second only to QCI 5 which is used for IMS signaling.

However, Skype video calls would surely drop to a QCI 2 bearer, along with all other video calling apps such as Facebook Messenger, Viber and Facetime to adopt a priority of 4. The tolerance is slightly increased in QCI 2 bearers, having a packet delat limitation of 150ms and a packet loss rate of 10^{-3}.

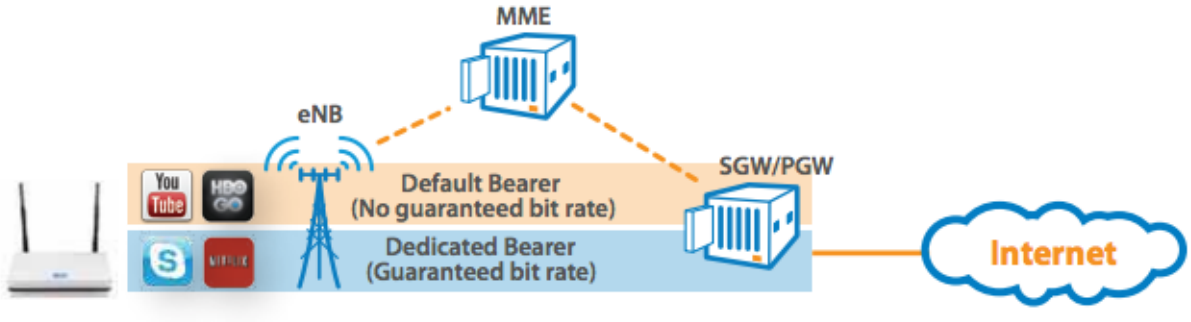

Figure 2.19: OTT Content Flows

It is important to note that, as more than one Default bearers may exist, if the QCI requested for an application is not satisfied by one of the existing bearers, then the UE will establish a new Default bearer and if necessary a Dedicated bearer over the Default one. What is more, the new bearer might potentially connect the UE to a different P-GW and therefore the UE be given a different IP address within the network of the second gateway.

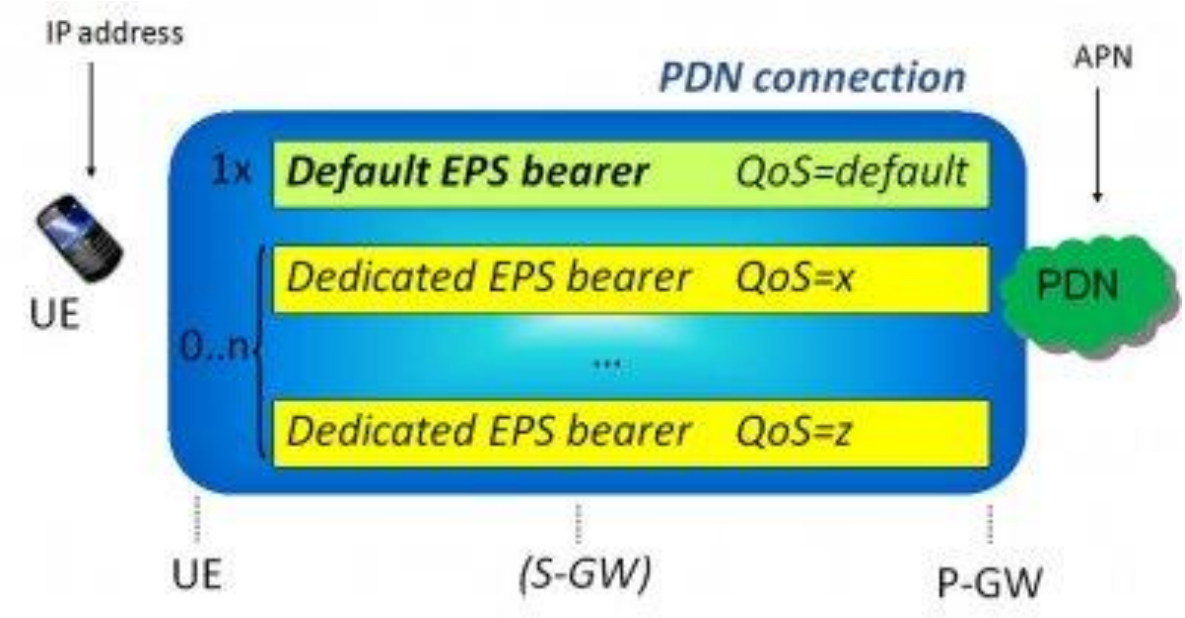

Figure 2.20: Default and Dedicated EPS bearers

Concluding this chapter, we addressed the mechanisms by which the Evolved Packet System provides user equipment with IP connectivity to the Packet Data Network in an extensively detailed manner [54]. We analyzed the long term goals of the LTE standard and explained the ways modern LTE networks work to achieve them. We outlined the key LTE parts and their role in the LTE edifice.

As high-speed wireless communication has become the norm for information-transfer transactions, it is considered useful for any individual to have an idea of the LTE architecture. Therefore, with the above we provide a concise, full, and accurate summary of the principles of LTE functionality for further academic study.

3. QUALITY OF EXPERIENCE

3.1 Introduction

When an on-demand video is streamed over an IP network using TCP, the client receives a theoretically intact copy of the media file. That is the case however according to the network's technical specifications. In any realistic scenario, the situation would be quite the contrary. Established communications in any wired network are restricted in quality to what the network infrastructure can offer. In most cases, they present instabilities due to latency and throughput changes, which possibly challenge the buffer of an active Internet application. Mobile or wireless networks also suffer from signal issues such as cell interference, noise, and fading which are major causes of delays in multimedia transmission. Having an unstable connection with fluctuating bandwidth is confusing for the buffering process of any multimedia application, and inevitably causes interruptions which can be catastrophic to the user's streaming experience.

Consequently, the success of telecommunication networks relies and is assessed on their Quality of Service (QoS). Quality of Service refers to the overall performance of a network, based on objective network metrics and parameters such as packet loss rate, average jitter and delay, availability, maximum bitrates, and others. A set of guaranteed values for such network parameters by a service provider constitute a QoS standard offered to connected users.

However, offering a high QoS standard does not always reflect proportionally on user experience. Therefore, as the need to move away from service-oriented quality assessment and towards a user-centric system emerged, a concept of subjectively perceived quality, Quality of Experience (or QoE) was introduced.

Reference [56] accurately defines Quality of Experience as "the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state." meaning that QoE evaluates how customers perceive the overall value of a service. Thus, it becomes clear that QoE estimation relies on subjective criteria which also differ among different types of services.

Figure 3.1: HAS QOE key influence factors

3.2 Influence Factors

An initial classification of QoE influence factors could be their division into perceptual and technical factors, as it is shown on the tree in figure 3.1. Technical factors are not perceived directly by the end user but heavily influence other, perceptual factors and indirectly the QoE. On the other hand, a perceptual factor such as the overall waiting time of a user may be the result of a number of technical factors.

For HTTP video streaming, initial delay and stalling [57] are the key influence factors of QoE, with the addition of adaptation in the case of HTTP adaptive streaming systems which introduce a new perceptual dimension. Hereby we present and analyze all common QoE influence factors as suggested by related research sources.

3.2.1 Initial Delay

A few seconds of initial delay are inevitable before the playback of any multimedia application. Delaying reasons begin with the establishment of connection between the user and the media server, continue with the transmission of information data about the media structure, and end with buffering and decoding. Transmission of the first two fully depends on the available bandwidth and the encoding used rather than the media application itself. Buffering, on the other hand, is a way to eliminate future playback interruptions. By delaying the beginning of the playback, incoming multimedia data is stored on a buffer to be used as a safety source of playtime in case the incoming data rate drops lower than the player bitrate.

However, the size of this buffer and the length of the additional delay to fill it are not fixed and vary from application to application. That being said, there is an evident tradeoff to be handled by the application's manufacturer. With a longer delay comes a lower risk of buffer depletion, while a shorter delay is associated with a smaller buffer and a higher risk for stalling during playtime because of buffer depletion.

Research confirms initial delay being a matter of different application-specific approaches, even though it includes connection establishment and loading. Furthermore, it suggests the existence of a logarithmic relationship between a Mean Opinion Score (MOS) and initial delay length [57]. However, it needs to be noted that users in their vast majority prefer a slightly longer initial delay than take the risk of stalling playtime. And that is because the impact of an initial delay on perceived quality is insignificant and its duration does not vary based on media clip duration. Initial delay is more or less expected and the users know when it starts - having the same experience in all their applications. On the other hand, stalling appears to the user as a sudden interruption of the media streaming service. Therefore, stalling is commonly perceived much worse by human sensing due to its unexpectedness. The same applies on mobile applications where users consider parameters like stalling or the technical video quality to be more crucial for their experience than the buffering delay. Ultimately, a delay of 15 seconds or less does not severely affect the user's perceived QoE and is typically considered acceptable according to research results.

As media service users are familiar with a minimal delay before the start of the playback, they normally tolerate it assuming they intend to watch the full length of the video. However, recent QoE research indicates the appearance of a new user behavior

especially for user-generated contents. Lately, users tend to browse through videos searching for some contents which they are interested in. Initial delays in such cases should be lower to be accepted by the user. Although the QoE of users browsing videos has not been deeply investigated yet, it is only subsequent that short delays might be desirable for user-generated content since users often just want to just peek into the video.

It follows for video service implementations in general, like for any service, that initial delays should be kept as short as possible, but initial delays are definitely not a major performance issue for the users' QoE. As we previously noted, even longer delays up to several seconds will be tolerated, especially if users intend to watch a video. Overall, they remain a key factor but a factor being traded off nonetheless.

3.2.2 Stalling

When the current throughput of a video streaming application is lower than the video player bitrate, the buffer occupancy will start to reduce. Eventually, data provided by the buffer will be insufficient, forcing the video playback to stop. At that point, playback is interrupted until the buffer loads again to contain a minimum amount of video data. This stopping of video playback because of playout buffer underrun is known as "stalling". Once interrupted, the application decides to what extent it will wait for the reload of the buffer, taking into account that a longer wait might ensure more buffered playtime and eliminate the possibility of another interruption, but also means a longer duration of the initial interruption. So, application developers are once again presented with the challenge of a performance tradeoff.

Research around stalling and its influence on QoE has been prominent. Interesting findings suggest that users are more patient during a single long stalling than a number of short, frequent ones. At worst, if multiple interruptions cannot be avoided, being as periodic as possible makes them more tolerable to clients. It is also believed that moments of interruption have a varying impact to streaming client users depending on their position in time and the importance of the –then- current part of the video to the user.

Another interesting aspect of stalling is its relation to quantization and frame rate. [58] Results prove that users by majority prefer an increased quantization in the encoding of a video if they are to avoid a more extensive stalling period during playtime. As per frame rate reduction, research suggests that it is also considered more tolerable than stalling. Frame rate reduction still appears to cause a drop of the user's QoE but subjective studies show users' opinion to be more positive towards it.

To summarize, stalling is probably the most key factor degrading user perceived QoE and should be avoided by all media streaming services whenever possible since users in most cases will not tolerate more than one interruption per video clip. Several models have been proposed for mapping stalling patterns and duration to an indicative MOS. All models agree on the existence of exponential relationship between them, meaning that extended stalling results in high dissatisfaction. Precisely for this reason, adaptive streaming techniques have gone a long way in adjusting play rate to the current throughput, effectively minimizing stalling limitations and ultimately offering a more attractive video service.

3.2.3 Adaptation

As we previously discussed, modern multimedia streaming services implement a method of adaptive streaming to make it possible for the video quality to adapt to the current throughput of the application. The client needs to acquire the ability to control the data rate depending on network fluctuations. The server side also has to be changed accordingly to encode the video in different quality levels and then split the media files into segments to deliver them upon request. Adaptation is designed precisely to improve perceived quality and forestall adverse interruptions, and as such it is considered another key factor influencing the user's QoE.

Multiple studies have shown that, in comparison with classical streaming applications, adaptive streaming concepts can effectively reduce stalling by large numbers when bandwidth decreases in mobility models. In the same way, adaptive streaming is capable of better utilizing the available bandwidth when the user moves and bandwidth increases. In stable, non-mobile environments, adaptive streaming provides an efficient way to guarantee a standard QoE and eliminate interruptions. Reference [59] proves that if presented with such a dilemma, users would choose to enjoy a sense of control in regards to their QoE. They appreciate knowing what to expect of the service they are being offered, thus preferring to deduct on video quality than experience sudden pauses. When objective results of stalling and resolution reduction are mapped to QoE, it is found that uncontrolled interruptions have a more disturbing effect than this of a deliberate quality change in resolution.

It becomes obvious to any researcher that adaptation introduces yet another aspect in QoE measurement and study. Being relatively new and constantly evolving, adaptation is in need of more intense and extensive research. This need constitutes in brief the aim of our research within the present text as we focus in explaining the interaction between streaming adaptation and quality of experience especially in mobile situations. Forms of adaptivity on content delivery network structure or traffic management also exist but are considered to be out of the scope of the present research since end users in such cases have minimal participation and essentially no control.

3.3 QoS Metrics

According to Wikipedia, Quality of Service (QoS) is defined as "the overall performance of a telephony or computer network, particularly the performance seen by the users of the network. To quantitatively measure quality of service, several related aspects of the network service are often considered, such as error rates, bit rate, throughput, transmission delay, availability, jitter, etc.". Or as [63] suggests, "QoS refers to a network's ability to achieve maximum bandwidth and deal with other network performance elements like latency, error rate and uptime." QoS also involves controlling and managing network resources by setting priorities for specific types of data (video, audio, files) on the network. Quality of Service is exclusively applied to network traffic generated for video on demand, IPTV, VoIP, streaming media, videoconferencing and online gaming.

To measure QoS, according to [60], a formula widely used is that of the mean-squared error loss distortion. It describes the effect of lost frames to the Quality of Service

provided to the user during a wireless transmission. The mean-squared error distortion can be computed as follows:

$$D(f,\underline{f}) = \frac{1}{N_1 N_2}\sum_{n_1=0}^{N_1-1}\ \sum_{n_2=0}^{N_2-1}\ (f(n_1,n_2) - \underline{f}(n_1,n_2))^2,$$

Where $\underline{f}$ is an estimated frame, created to replace a frame f of dimensions $N_1 \times N_2$ pixels and $f(n_1,n_2)$ (or $\underline{f}(n_1,n_2)$) indicate the pixel value at position (n_1,n_2) of the frame f (or $\underline{f}$). In this case, to calculate the PSNR, one could use the following metric:

$$P_{SNR}(f,\underline{f}) = \frac{(2^B-1)^2}{D(f,\underline{f})}, \text{ or in } dB \text{ as proposed in [62] for } B = 8:$$

$$P_{SNR_{dB}}(f,\underline{f}) = 10log_{10}\frac{(255)^2}{D(f,\underline{f})} = 20log_{10}(\frac{255}{\sqrt{D(f,\underline{f})}}),$$

Where B is the number of bits used in a pixel's encoding. The researchers in [60] accurately note that the closer a frame f is to the beginning of a Group of Pictures (GOP) the higher the distortion value will be, depending also on error concealment (figure 3.2) and encoding. Notably, a PSNR value ranging from 30 to 40 characterizes a medium to high quality video.

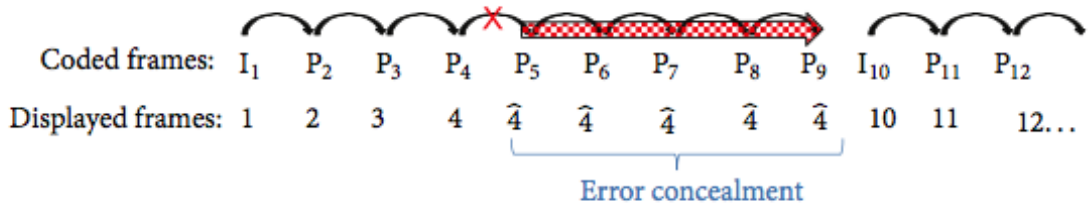

Figure 3.2: Error concealment example

Error concealment is usually performed using the "previous frame" concealment approach. This approach indicates that if a P-frame is lost in a GOP, then the previous frame is repeated until the end of that GOP or until the next I-frame is received. As proposed, the total loss distortion in a GOP of N_G total frames, one I-frame and $N_G - 1$ P-frames where a frame f is lost would be:

$$D^{(f)} = \sum_{y=f}^{N_G}\ D(y, f-1),$$

Since frame f and all following frames are replaced by frame $f-1$. In the case of an I-frame, it is replaced by the last frame of the previous GOP.

3.4 QoE Metrics

Quality of experience (QoE) is a complex concept, with conflicting aspects in confluent domains. It tries to measure the QoS as it is perceived by the end user and one could argue over its consistently growing interest as the best method to quantify the multimedia experience of mobile users. Traditionally, QoE is obtained from subjective tests, where human viewers evaluate the quality of tested videos under a laboratory environment.

The relationship between QoE and QoS (such as coding parameters and network statistics) is complicated because users' perceptual video quality is subjective and diversified in different environments. In [62], researchers note that the two featured approaches mapping QoS to QoE are the stimulus-centric and the perception-centric approaches. The stimulus-centric one is based, as stated, on the “WQL hypothesis” which defines that the relationship between waiting time t and its QoE evaluation on a linear ACR scale is logarithmic. What derives from this law is the fact that a change in perceived quality can be seen as surpassing a hardly detectable margin. Thus, a lot of studies have initiated a search to determine this tiniest noticeable difference between two consecutive levels of QoE. On the other hand, the perception-centric approach is reflected by the “IQX hypothesis” according to which the relationship between a QoS parameter and QoE is negative exponential. Moreover, apart from providing an equation between a QoS impairment and the perceived stimulus, it also indicates that a user’s QoE sensitivity is highly dependent on his currently provided QoE level.

However, it is common secret among researchers that despite QoE’s significant advancement as a scientific field over the past years, its analysis methodology has not always kept pace due to the complexity of its multidimensional nature. In this section, we devote a part to outline and explain the definitions of known and commonly used metrics regarding QoE evaluation.

3.4.1 Mean Opinion Score (MOS)

As it is correctly stated in [62], the Mean Opinion Score (MOS) is arguably the most important QoE indicator. MOS is usually a 5-point scale, originally used to measure the subjective quality of real time multimedia data. Most are familiar with the post-service evaluation tests used by several web applications. A mapping of MOS to video quality is shown in figure 3.3 below.

MOS	QUALITY
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Figure 3.3: Relation of MOS and Video Quality

For a specific subjective experiment, we can define a random variable U to represent the quality ratings (i.e. the 5-point scale). In order to simplify our calculations, we can assume that conditions are the same throughout the experiment and that the identity of a subject is irrelevant to the calculation. Given that U is discrete or continuous with a probability mass function f_u or a probability density function $f(u)$ in each case respectively, the expected, or "mean" value of the random variable U is given by the formula:

$E[U] = \sum_{u=U^-}^{U^+} uf_u$, when U is discrete and $E[U] = \int_{u=U^-}^{U^+} uf(u)du$,

when U is continuous.

Based on this, we can consider the Mean Opinion Score to be an estimate $\widehat{U}$ of $E[U]$ and, for a number of quality classes N, to be given by [61]:

$$MOS = \sum_{u=1}^{N} u\widehat{f_u}$$

Now, because for a given number of test subjects R the estimated probability of opinion score u is $\widehat{f_u} = \frac{1}{R}\sum_{i=1}^{R} \delta_{U_i,u}$ with $\delta_{i,j} = 1$, if $i = j$ or 0 otherwise, we have:

$\sum_{u=1}^{N} u\widehat{f_u} = \sum_{u=1}^{N} u \frac{1}{R}\sum_{i=1}^{R} \delta_{U_i,u} = \frac{1}{R}\sum_{i=1}^{R} \sum_{u=1}^{N} u\delta_{U_i,u} = \frac{1}{R}\sum_{i=1}^{R} U_i$, and so

$MOS = \frac{1}{R}\sum_{i=1}^{R} U_i$,

which practically means $MOS = \frac{set\ of\ samples\ \ for\ \ R\ test\ subjects}{number\ of\ test\ subjects}$ and is, in essence,

taking us back to the definition of MOS.

To measure the uncertainty of MOS we can use its standard deviation σ_U or SOS (Standard deviation of Opinion Score) which is an estimate of the standard deviation and converges to σ_U for very large numbers. The SOS hypothesis formulates the relationship between MOS and SOS as such:

$$SOS = \sqrt{a(-MOS^2 + (U^- + U^+)MOS - U^- \cdot U^+)},$$

where the SOS parameter $a \in [0;1]$ and depends on the application and the test conditions, derived from subjective tests.

More on a can be found in Section $II - B$ of [61].

3.4.2 Using the PSNR

Figure 3.4 shows a relation between the PSNR and video quality. According to reference [60], we can also calculate the overall QoE through the PSNR by using the following exponential relation:

$$Q = \frac{1}{1 + e^{b1(P_{SNR} - b2)}},$$

Where b1 and b2 are parameters depending on video characteristics and PSNR is expressed in dB. The authors note that 0 indicates the best quality while 1 indicates the worst. It is useful to point out that the previous and the following derivations are associated with QoE estimation in general and not specifically in mobile or wireless networking.

PSNR	QUALITY
>37	Excellent
31-37	Good
25-31	Fair
20-25	Poor
<20	Bad

Figure 3.4: Relation of PSNR and Video Quality

Another metric suggested by an alternate source following a subjective video quality assessment is:

$$Q_m = Q_{max}\left(\frac{1}{1 + e^{b1(P_{SNR} - b2)}}\right) \cdot \frac{1 - e^{-b3(f/fmax)}}{1 - e^{b3}},$$

Where Q_{max} is a constant representing maximum quality (usually 100 so that $Q_m$is on a 0-100 scale), f and f_{max} are the current and maximum frame rates respectively and b3 is another parameter of the video. However, the above equation is often written simplified as:

$$Q_m = Q_{max}\left(\frac{1}{1 + e^{b1(P_{SNR} - b2)}}\right),$$

Since error concealment can maintain a frame rate f so that $f = f_{max}$.

The video characteristics used above can be calculated offline in the server where the video is stored and then used in the QoE equation. In case of a live broadcast there are alternative forms of the QoE estimation formula to approach dynamically changing content. Similarly, the SSIM (Structural Similarity Index) metric is calculated on various windows of an image. The measure between two image windows x and y of size $N \times N$ is given by [62]:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x{}^2 + \mu_y{}^2 + c_1)(\sigma_x{}^2 + \sigma_y{}^2 + c_2)},$$

Where μ is the average, σ^2 is the variance, σ_{xy} is the covariance and c_1, c_2 are constant values. SSIM along with PSNR have been important metrics used to derive many complex equations for quality assessment, such as the VQM (Video Quality Metric) and MPQM (Moving Pictures Quality Metric).

3.4.3 Network average

QoE estimation is indeed based on video characteristics as shown above. However, recent studies focus on the relation between QoE and network attributes. Performance in the Radio Resource Management apparently has an instant reflection on perceived quality by all users in the network. Thus, we present some metrics which widely appear in literature describing the aforementioned effect. Overall, the average QoE in the network is given by:

$$Q_m^{(avg)} = \frac{1}{K}\sum_{k=1}^{K} Q_{m,k},$$

Where $Q_{m,k}$ is the QoE of a specific, single user. This formula describes the average QoE in an accurate way, but could still hide the existence of a number of users experiencing a very low quality when there are others enjoying the opposite therefore counterbalancing them. The solution proposed by researchers is to try to maximize the minimum QoE in the network, calculated below:

$$Q_m^{(min)} = min_k\ Q_{m,k}$$

This formula allows network designers and internet providers to focus on improving the lowest QoE encountered in the network. Since favoring low QoE users would upset the connection of other subscribers and is unfair, an accurate settlement would be to use the geometric mean ($Q_m^{(gm)}$) QoE, whose formula is presented below:

$$Q_m^{(gm)} = \left(\prod_{k=1}^{K} Q_{m,k}\right)^{1/K}$$

The above metric will attend users with high QoE potential who will increase the product, but it will also look to avoid low QoE in other users since that would drop the product asymptotically to 0.

Now, in order to optimize QoE in the network, the general equation given is:

$$\max_{a_{k_l,i,l}^{(DL)}, a_{k_l,i,l}^{(UL)}, P_l^{(DL)}, P_{k_l}^{(UL)}} Q_m^{(net)},$$

where $Q_m^{(net)}$ is any of the metrics described before and is subject to the following:

$$P_{k_l}^{(UL)} \leq P_{k_l,max}^{(UL)};\ \forall k_l = 1,\ldots,K_l;\ \forall l = 1,\ldots,N_{BS},$$

$$P_l^{(DL)} \leq P_{l,max}^{(DL)};\ \forall l = 1,\ldots,N_{BS},$$

$$\Sigma_{k_l}^{K_l}\ a_{k_l,i,l}^{(UL)} \leq 1;\ \forall i = 1,\ldots,N_{sub}^{(UL)};\ \forall l = 1,\ldots,N_{BS},$$

$$\Sigma_{k_l}^{K_l}\ a_{k_l,i,l}^{(DL)} \leq 1;\ \forall i = 1,\ldots,N_{sub}^{(DL)};\ \forall l = 1,\ldots,N_{BS},$$

Where:

K_l	is the number of users located in the range of cell l
N_{BS}	is the number of Base Stations
$P_{k_l}^{(UL)}$	is the UpLink transmit power of user k_l in cell l (with $P_{k_l,max}^{(UL)}$ its maximum)
$P_l^{(DL)}$	is the DownLink transmit power of BaseStation l (with $P_{l,max}^{(DL)}$its maximum)
$N_{sub}^{(UL)}$	is the UpLink number of OFDMA subcarriers
$N_{sub}^{(DL)}$	is the DownLink number of OFDMA subcarriers
$a_{k_l,i,l}^{(UL)}$	is an indicator variable for the UpLink
$a_{k_l,i,l}^{(DL)}$	is an indicator variable for the DownLink

(indicators' value: 1 if subcarrier i is assigned to user k_l and 0 otherwise)

The above constraints suggest that transmit power cannot exceed maximum as well as that each subcarrier is allocated exclusively to a unique user in each cell for a specific scheduling instant. Reference [60] points out that the same laws apply to QoS, and thus the same formulas can be used for QoS if we replace the QoE parameters with the respective QoS values.

Regarding multimedia quality, reference [69] analyzes a parametric multimedia quality integration function using a video quality estimation function V_q and a speech quality estimation function S_q. If T_s is the time delay in speech and T_v Is the time delay in video, the multimedia quality can be calculated as:

$$MM_q = m_1 \cdot MM_{SV} + m_2 \cdot MM_T + m_3 \cdot MM_{SV} \cdot MM_T + m_4$$

where:

$$MM_{SV} = m_5 \cdot S_q + m_6 \cdot V_q + m_7 \cdot S_q \cdot V_q + m_8$$

$$MM_T = \max\{m_9 \cdot (T_S + T_V) + m_{10} + MS, 1\}$$

$$MS = \begin{cases} min\{m_{11} \cdot (T_S - T_V) + m_{12}, 0\}, & if\ T_S > T_V \\ min\{m_{13} \cdot (T_V - T_S) + m_{14}, 0\}, & if\ T_V > T_S \end{cases}$$

and m_i, i = 1,2,…14 are coefficients.

All coefficient values are displayed in [69] for different video display cases, and V_q,S_q functions involving speech and video quality are explained in a consise, granulated format.

3.5 Service Providers and Applications

In order for service providers to plan, scale and operate their service packages there is increased need for metrics that represent best the customer's opinion. Thus, they tend to resort to not just subjective experiments but also behavioral measurements. The θ-$acceptability$ is the probability that the opinion score will surpass a certain threshold θ and can be estimated by [61]:

$$A_\theta = \int_{s=\theta}^{U^+} \widehat{f_s} ds = \frac{1}{R} |\{U_i \geq \theta : i = 1, \ldots, R\}|$$

Providers also used subjective and behavioral percentages such as the "poor or worse" (%PoW), the "good or better" (%GoB), or the "terminate early" (%TME) percentages to measure the users' opinion. A model that could theoretically map such percentages to MOS is known as the E-model [61]. This relationship is depicted in figure 3.5. Through an intermediary random variable called "the Transmission Rate" $R \in [0; 100]$ as the RV U used above and assuming U follows normal distribution $N(0,1)$, the E-model defined these measures as:

$$GoB(u) = F_U(\frac{u-60}{16}) = P_U(U \geq 60)$$

$$PoW(u) = F_U(\frac{45-u}{16}) = P_U(U \leq 45)$$

$$TEM(u) = F_U(\frac{36-u}{16}) = P_U(U \leq 36)$$

and the transformation of U to a continuous MOS scale as:

$$MOS(u) = 7u(u-60)(100-u)\times 10^{-6} + 0.035u + 1, MOS \in [1; 4.5]$$

It should be noted that the quantiles used for GoB, PoW, and TEM come from a number of subjective tests, part of E-model's development. These measures are estimated by using $\theta\text{-}acceptability$ in the equations:

$$\%\widehat{GoB} = A_{\theta_{gb}}$$

$$\%\widehat{PoW} = 1 - A_{\theta_{pw}}$$

$$\%\widehat{TEM} = 1 - A_{\theta_{te}}$$

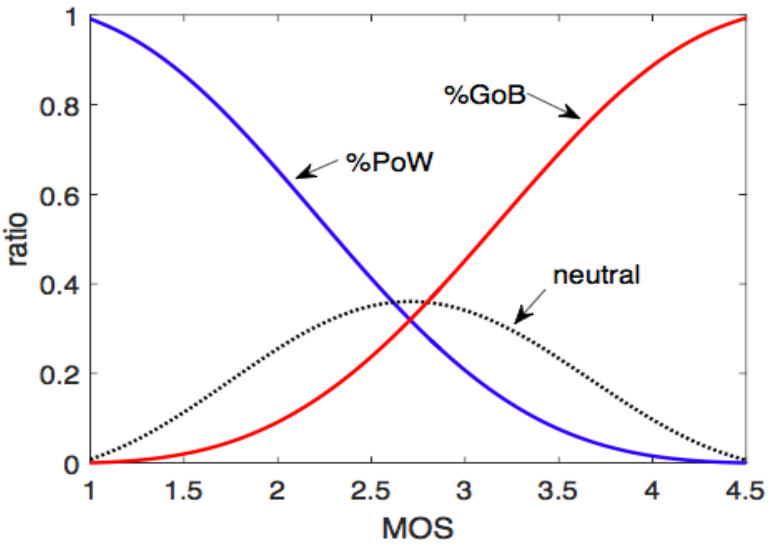

Figure 3.5: Relationship between MOS and %GoB-%PoW

if $\theta_{gb} = 60,\ \theta_{pw} = 45, \theta_{te} = 36$ for $U \in [0; 100]$ or

if $\theta_{gb} = 3.1, \theta_{pw} = 2.3, \theta_{te} = 1.9$ for $U \in [1; 5]$

We extensively analyzed several QoE metrics, with the most significant listed above. However, it should be noted that many other metrics which are used are application-specific. An example of that case is the following, known as reception ratio calculated using YouTube [62]:

$$\rho = \frac{Download\ throughput\ or\ Bottleneck\ capacity}{Video\ encoding\ rate},$$

which is a good indicator considering that a video has good quality if $\rho > 1$, as is the "fraction of the video downloaded" which indicates the user's behavior towards the video being downloaded.

Video delivery quality on the other hand can be shown by the rate λ where:

$$\lambda = \frac{total\ stalling\ time}{total\ video\ elapsed\ time}.$$

In [69], researchers present a simple QoE model which is truly indicative and applies not only to YouTube but to any HTTP based Adaptive Streaming service. It is based on the valid assumption that the QoE provided by HAS applications is highly dependable on the percentage of viewing not degraded from the highest available quality. If *t* is the percentage of time that the player spends on the highest layer then MOS is given by:

$$MOS_{HAS} = 0.003 * e^{0.064*t} + 2.498$$

A second metric presented, in regards to HAS and especially YouTube adaptation is the activity factor *a* which shows whether the client is is able to fluently download each video segment taking into account the available bandwidth. Per the researchers, if this indicator is approaching 1 then the client will present difficulty in downloading the video segments in question. The activity factor is defined as:

$$a = \frac{total\ time\ of\ actual\ data\ download}{total\ time\ elapsed\ for\ complete\ video\ download}$$

For Skype video application, they mention a practical MOS formula proposed in literature for different screen resolutions. If *F* is the Frame Rate, with a maximum of 35fps and *I* is the Image Quality ranging from 0 to a perfect 1, MOS for resolutions 160x120, 320x240, and 640x480 would be:

$$MOS_{skype} = \begin{cases} 1 & res = 160x120 \\ 2 & res = 320x240 \\ 3 + \frac{F}{35fps} + (2I - 1) & res = 640x480 \end{cases}$$

4. THE NS-3 NETWORK SIMULATOR

4.1 NS-3 Basics

Our engagement in the functionality of LTE networks and, later, the measurement of users' QoE upon the addition of adaptive streaming on their media all join in one at this point: a real-world scenario. Since communication networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, communication researchers have turned to network simulation, a technique where a program models the behavior of a network based on mathematical formulas allowing the users to draw realistic conclusions. One of the most popular network simulators available on the market is NS-3.

NS-3 is a discrete event network simulator, developed for network research and education. Its modules are written in C++ with Python bindings allowing the user to create C++ executable files making use of the existing modules. It uses the Waf build system, a build automation tool which compiles and builds all necessary modules included in the .waf file of the directory of the user-created files.

Individual modules, compatible with each other, provide almost every necessary functionality so that the researcher can create a full scale realistic network model of their choice. In most cases, helpers are also provided in order to avoid a complex interaction with low-level programming modules.

4.2 Building on top of LENA Project

4.2.1 Introduction

NS-3 owes its LTE functionality to the LENA project, an LTE simulation module developed by the Technologic Center of Telecommunications in Catalonia [64]. LENA focuses mainly on modeling the E-UTRA part of the system, with a particular attention on the aspects related to the channel, PHY and MAC layers following in detail the architecture of the LTE networks.

By using LENA in NS-3, we were able to reproduce a basic, realistic, real-world example, which we are proud to include in the file "lenaexample.cc" and describe below. Our initial goal was to simulate a basic media server-client model, where a video is being streamed from a remote host, a content-delivery server, to one or more mobile devices of an LTE network (UEs). On top of that, our ambition was to implement an adaptive streaming functionality on the video streaming process of the server. Following is a brief explanation of the example program we have created.

4.2.2 A Simple Example

Going through the code, we start by enabling some necessary log components in order to track the behavior of both the server and the client at execution time. Then we declare the variables which will be used by the simulator, among others giving the user the ability to change the number of mobile nodes and the distance between them. Before we can create our nodes, we need to create our helpers, and set a node as our PDN gateway (P-GW).

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

It is time to create the nodes, starting from our remote host. We install the internet stack in the remote host container and create a point to point link between the remote host that will be our server and the P-GW. In our example, we use a 100Gbps link with 10ms of latency, attributes that can be easily changed by the user. Then we can finally create our mobile nodes and install LTE functionality in our eNodeB and our UEs. To provide UE mobility, LENA offers various mobility models, the simplest being the Constant Position model, with the initial positions of the nodes being set by a position allocator vector. The distance step of the vector can be set at the beginning of our code. We can also assign static IPs to our remote host and our UEs, and set their default gateway. Lastly, we attach the UEs to the eNodeB.

```
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);
Ipv4AddressHelper ipv4h;
ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);
```

The core of our simulation, however, are the server and client applications. We install a Server application on our remote host, and a Client application on our UE nodes. The server port and IP are pre-specified and passed to the client as a parameter. Every application in LENA should have a start and stop time, ideally with the clients starting after and finishing before the server, giving the impression of a continuous service, at least from the client's perspective.

```
apps = server.Install(remoteHostContainer.Get(0));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));
apps = client.Install (ueNodes.Get(i));
apps.Start (Seconds (1.1));
apps.Stop (Seconds (9.9));
```

A dump file is used here to indicate and record the activity of each of the clients, as well as the server. LENA also offers the ability to generate trace statistics for PHY, MAC, RLC and PDCP layers. The above is very helpful for the user to draw conclusions and present the simulation findings. It is worth noting that even though the simulator starts from the command line execution (see figure 4.1), it is advised that we set the simulation to end after a user-specified time.

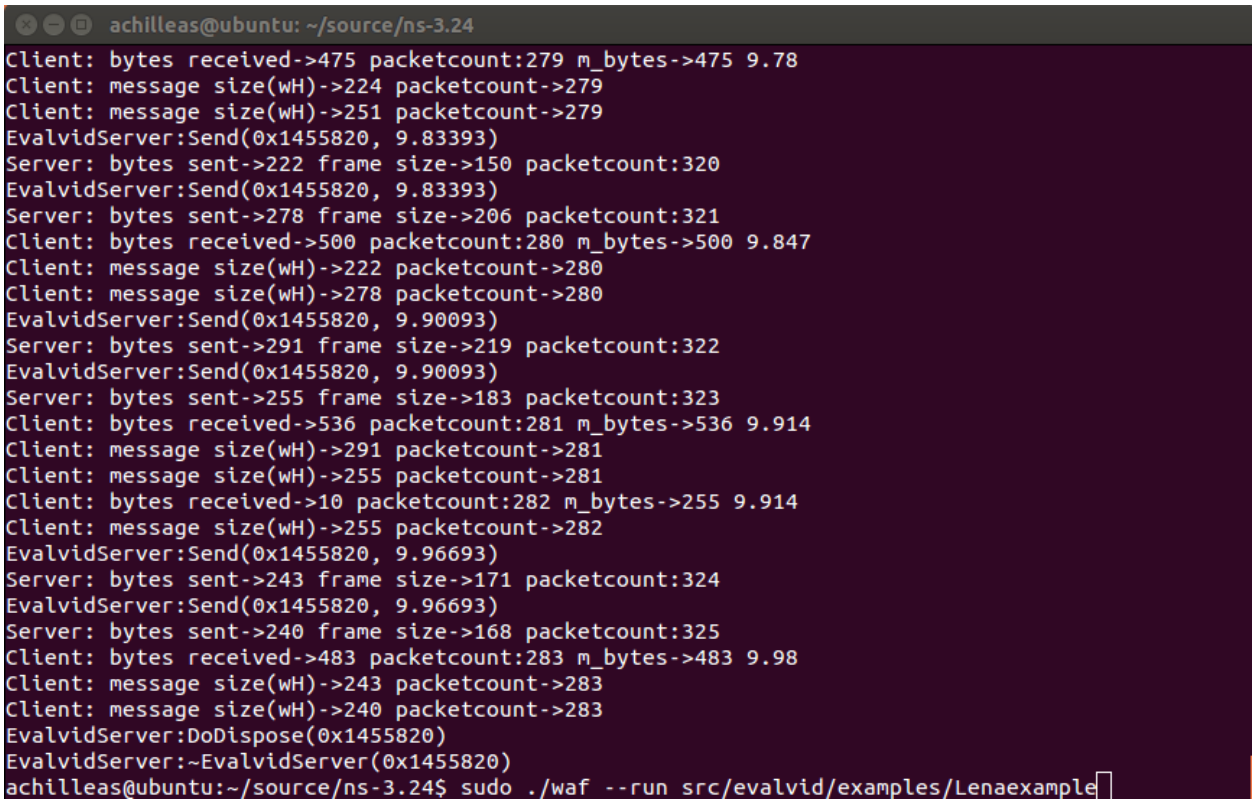

Figure 4.1: Lena Example Execution

4.2.3 The Server-Client Model

Our quest has been to experiment on a simple scenario of a video streaming server, similar to those of modern content delivery networks. An HTTP request for a video file from the client to the server is followed by a stream of data segments sent directly to the client by the server. However, on-demand content demands in turn for adapting quality and for that reason, as explained in previous chapters, we were in need of an adaptive streaming mechanism.

As a basis to build on, we worked with the excellent open source NS-3 Evalvid module, a project initially created by GERCOM [65] and destined for video evaluation simulations. On this foundation, we built our Client-Server module following the previous structure as well as the NS-3 project guidelines (limit user access to low-level modules through helpers, offer installation and attribute-setting functions). We redesigned the model to support a TCP connection to the client instead of a UDP stream, developing a socket allocation and binding process. Our server model, which is HTTP based, uses information encapsulated into the HTTP header to identify client requests aiming to better simulate the functionality of a video server offering a variety of videos for the client to request in several different resolutions.

Taking a brief look at file “evalvid-server.cc”, we can see that our video streaming server follows a typical TCP server process. After all attributes have been set using function *EvalvidServer::GetTypeId* in during the *EvalvidServerHelper* construction, the server creates and binds a TCP socket in function *EvalvidServer::StartApplication* listening

then on the socket for any incoming connections. In the same function, the video is setup and the server is ready to start sending upon request.

```
TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");
socket = Socket::CreateSocket (GetNode (), tid);
InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), m_port);
socket->Bind (local);
socket->Listen();
socket->SetRecvCallback (MakeCallback (&EvalvidServer::HandleRead, this));
socket->SetAcceptCallback(
        MakeNullCallback<bool, Ptr<Socket>, const Address &>(),
        MakeCallback(&EvalvidServer::HandleAccept, this));
```

Most NS-3 streaming evaluation simulations avoid transferring the actual file between the server and the client. Instead, they use a video trace file which contains the frame type, the frame size and the number of packets needed for each frame, as well as the time when each frame should be played, relative to the play time of the first frame. Having this info, the server can create and transfer the same amount of data that it would if it was sending the actual file, but without having to load a real video on the simulator's memory, which would introduce the factor of individual device performance, negatively affecting the credibility of the experiment.

```
while (videoTraceFile >> frameId >> frameType >> frameSize >> numOfTcpPackets >> sendTime)
{
    videoInfoStruct = new m_videoInfoStruct_t;
    videoInfoStruct->frameType = frameType;
    videoInfoStruct->frameSize = frameSize;
    videoInfoStruct->numOfTcpPackets = frameSize/m_packetPayload;
    videoInfoStruct->packetInterval = Seconds(sendTime - lastSendTime);
    m_videoInfoMap.insert (pair<uint32_t, m_videoInfoStruct_t*>(frameId, videoInfoStruct))
```

After all, for the evaluation to be accurate, a researcher would only need the percentage of lost information per video frame, but not the exact bytes lost. Once the replicated data is sent over to the client, a new version of the video can be reproduced using the packets that reached the destination as a mask on the original video, showing the effect of the lost information on the video quality.

For the production of a video trace file extracted from an original .MP4 video file we used the MP4 trace tool included in the file "mp4trace.c" of the EvalVid framework. EvalVid, different than the previously mentioned Evalvid NS-3 module, is a video transmission and quality evaluation framework written in C [66] by the Telecommunications Networks Group of the Technische Universität Berlin. The MP4 trace tool from EvalVid is able to send a hinted mp4-file to a specified destination host recording the packet flow which it separates from other traffic on the interface. Other network monitoring tools such as tcpdump can be used to produce the above trace file, but would require an application of manual filtering.

The mp4trace tool from EvalVid is able to send a hinted mp4-file

```
mp4trace -f -s 192.168.0.2 12346 a01.mp4
```

Once the trace file is loaded in the *EvalvidServer::Setup* function, the server preserves a mapping of the frame ID to a struct that holds all that frame's necessary information and then waits for a client's request. When the client initializes its connection to the socket, it creates an HTTP request packet and sends it to the server, in function *EvalvidClient::Send*. This HTTP request always contains the ID of the video requested by the client, along with the ID of the video segment to be sent next and the resolution chosen for it.

```
httpHeader.SetSeq(1);
httpHeader.SetMessageType(HTTP_REQUEST);
httpHeader.SetVideoId(m_videoId);
httpHeader.SetResolution(m_bitRate);
httpHeader.SetSegmentId(m_segmentId++);
p->AddHeader(httpHeader);
m_socket->Send (p);
```

After the server receives this client request, in function *EvalvidServer::HandleRead* it extracts the video ID, the segment ID and the preferred resolution bitrate and starts sending media to the client. Using the *EvalvidServer::Send* function, the server creates HTTP response packets, addressing the client's HTTP request with the same video ID, segment ID, and segment resolution included in the HTTP header. This HTTP response packet also contains a Sequence header which shows the ID of the packet allowing our simulation to be modified to identify lost packets if needed.

```
packet->RemoveHeader(header);
videoId = header.GetVideoId();
segmentId = header.GetSegmentId();
clientResolution = header.GetResolution();

http_header.SetMessageType(HTTP_RESPONSE);
http_header.SetVideoId(videoId);
http_header.SetResolution(clientResolution);
http_header.SetSegmentId(segmentId);
```

However, encapsulated in the HTTP response is the most important component of our server-client communication. That is the MPEG header containing the current frame ID, the play time of each frame (relative to the first frame), the frame type, and the frame size. Depending on the size of the frame and according to our preset Maximum Transmission Unit, the server might need to send multiple packets in order to transfer the frame as a whole. Thus, the server is responsible to calculate the number and the size of packets, a process which is executed in every call of the function *EvalvidServer::Send.* Based on that information, the client can effectively anticipate packets until the frame is completely sent. After all the packets required for the frame are sent, the *EvalvidServer::Send* recalls itself until the last frame of the segment is sent.

```
mpeg_header.SetFrameId(m_videoInfoMapIt->first);
mpeg_header.SetPlaybackTime(MilliSeconds(
  (m_videoInfoMapIt->first + (segmentId * MPEG_FRAMES_PER_SEGMENT))
      * MPEG_TIME_BETWEEN_FRAMES)); //50 fps
mpeg_header.SetType(m_videoInfoMapIt->second->frameType);
mpeg_header.SetSize(m_videoInfoMapIt->second->frameSize);
```

It is important to note that, following the sending/receiving of packets by the two parties, both the server and the client mark down all traffic transferred as detected from their respective ends. The server writes a packet's size, its ID and the time sent in the sender's trace file. On the other end, the client adds the same information to its receiver dump file, in function *EvalvidClient::MessageReceived* (see figures 4.2 and 4.3 below).

sd_a01_lte

```
1.1569          id 26           tcp 460
1.1569          id 27           tcp 78      frame id 1
1.1909          id 28           tcp 460
1.1909          id 29           tcp 435      frame id 2
1.2239          id 30           tcp 460
1.2239          id 31           tcp 318      frame id 3
1.2559          id 32           tcp 380      frame id 4
1.2889          id 33           tcp 373      frame id 5
1.3229          id 34           tcp 264      frame id 6
1.3559          id 35           tcp 348      frame id 7
1.3889          id 36           tcp 288      frame id 8
1.4209          id 37           tcp 146      frame id 9
1.4549          id 38           tcp 250      frame id 10
1.4879          id 39           tcp 154      frame id 11
1.5199          id 40           tcp 220      frame id 12
1.5539          id 41           tcp 167      frame id 13
1.5869          id 42           tcp 160      frame id 14
1.6189          id 43           tcp 245      frame id 15
1.6519          id 44           tcp 212      frame id 16
1.6859          id 45           tcp 123      frame id 17
```

Figure 4.2: Sender's Dump File

rd_a00_lte

```
1.5800          id 1            tcp 13910
1.5810          id 28            tcp 967
1.5810          id 30            tcp 850
1.5810          id 32            tcp 380
1.5820          id 33            tcp 373
1.5820          id 34            tcp 264
1.5820          id 35            tcp 348
1.5820          id 36            tcp 288
1.5820          id 37            tcp 146
1.5820          id 38            tcp 250
1.5820          id 39            tcp 154
1.5830          id 40            tcp 220
1.5830          id 41            tcp 167
1.6150          id 42            tcp 160
1.6320          id 43            tcp 245
1.6650          id 44            tcp 212
1.7000          id 45            tcp 123
1.7310          id 46            tcp 196
1.7660          id 47            tcp 269
1.7970          id 48            tcp 140
1.8320          id 49            tcp 189
```

Figure 4.3: Receiver's Dump File

4.3 Implementing DASH

One of the main reasons for us to move away from the original Evalvid NS-3 module implementation was the need to rebuild the module's foundation to support HTTP adaptive streaming using a TCP interconnection. For the same reason we chose to follow the MPEG/DASH client-server module [67] and combine its methods with our version of the Evalvid server-client application. Trying to follow the same path, we list here the most important of our interventions, and present the code.

The idea of a thoroughly detailed simulation of an MPEG player's infrastructure is certainly very useful. Based on that, the researcher needs to add a fully functional MPEG video player in order to simulate not only the physical transfer of media, but also the playing process. The process of reproducing the media on the receiving end, which the media file is destined for leads to a more accurate calculation of the client resolution to be requested in the consecutive segment HTTP requests, therefore making the simulation more realistic.

Going through the code, the client, upon receiving a frame, it sends it to the MPEG player for playing, and then adds the frame size to the ongoing segment size and to the total bytes sent. The MPEG player then adds the frame to the queue and in case the player is found paused, it adds the time of the interruption of play to the total interruption, and resumes playing in function *MpegPlayer::ReceiveFrame*, after clearing the interruption counters.

```
m_state = MPEG_PLAYER_PLAYING;
m_interruption_time += (Simulator::Now() - m_lastpaused);
PlayFrame();
```

As long as the queue is not empty, the MPEG player continuously plays the frames available one by one in function *MpegPlayer::PlayFrame*, which calls itself every 20 milliseconds until it finds the queue empty. In that case, it switches the player to an idle state, measuring the time since then as interruption, keeping a record of the moment that the last interruption took place.

```
m_state = MPEG_PLAYER_PAUSED;
m_lastpaused = Simulator::Now();
m_interrruptions++;
```

From the client's perspective, we calculate the time when a frame that just arrived is expected to be played, taking into account the start of play time (when the first frame was played), the frame's supposed play time relative to the first frame, as well as any potential interruption time recorded by the player. The MPEG player then saves the current bitrate as the minimum transmitted bitrate in case it is lower than the previously recorded minimum.

On a segment level, the client is responsible for holding timing information, imperative to calculating when a bitrate adaptation is necessary. When the time comes for a new segment to be requested, the client takes note of the time it contacted the server, at the moment when that initial HTTP request is sent. Therefore, when the last frame of the segment is received, the client immediately calculates the elapsed time between the request and the full receipt of the media segment.

After that, and before requesting the next segment, the client calculates the average bitrate during the transmission of the last segment by dividing that segment's bits and the fetch time from the previous computation. The resulting bitrate is added to the list of recorded bitrates, mapped to the current time of that addition. Checking the list of recorded bitrates, the client produces a bitrate estimate by averaging the last bitrates that were recorded within our predefined time window.

```
m_segmentFetchTime = Simulator::Now() - m_requestTime;
AddBitRate(Simulator::Now(),8*m_segment_bytes/m_segmentFetchTime.GetSeconds());
Time currDt = m_player.GetRealPlayTime(mpegHeader.GetPlaybackTime());
```

Once the client refreshes its bitrate estimate in function *EvalvidClient::AddBitRate*, it uses function *EvalvidClient::CalcNextSegment* to change the active bitrate variable that will be injected in the client's next segment request. In case the previous bitrate requested does not match to the current bitrate variable, the client also keeps count of the rate changes, so that we can provide a statistic to the researcher of how stable our video connection has been.

```
uint32_t prevBitrate = m_bitRate;
uint32_t nextRate = 8 * m_segment_bytes / m_segmentFetchTime.GetSeconds();
CalcNextSegment(nextRate, m_bitRate, bufferDelay);
```

What is more, the DASH application offers a function to schedule a delayed wakeup to request the next segment, based on the buffer levels which the client can have monitored at all times. In function *EvalvidClient::SchduleBufferWakeup*, the structure sets a buffer delay which causes the client to hold the request until the MPEG player allows for it to be initiated through function *MpegPlayer::PlayFrame*.

```
LogBufferLevel(currDt);
m_player.SchduleBufferWakeup(bufferDelay, this);
```

Among the most fundamental components of any NS3 application, is the function responsible for reading from the socket created by the client. As our example aims to simulate a server-client application that uses layer 7 intelligence such as HTTP messaging, the above functionality is part of our HTTP parser. The HTTP parser is therefore responsible for parsing data coming out of the client's socket as HTTP messages.

In function *HttpParser::ReadSocket*, which is called by *EvalvidClient::HandleRead*, the HTTP parser receives the incoming bytes from the socket, limited by the preset maximum size of the MPEG message, and copies them to its buffer. It counts all received packets which allows the researcher to match the counter to the sequence tag included in the sequence header of the packet.

```
Address from;
int bytes = socket->RecvFrom(&m_buffer[m_bytes], MPEG_MAX_MESSAGE - m_bytes, 0, from)
packetcount++;
m_bytes += bytes;
```

Upon receiving a new batch of data through the socket, the HTTP parser is in charge of creating a new packet for the client, as well as adding the proper headers to it. By serially parsing the raw incoming data, it correctly populates the MPEG, HTTP, and Sequence headers it previously created. This is done in accordance with the transport layer, with different maximum transmission unit values applied specifically for TCP applications like ours.

```
Packet headerPacket(m_buffer, headersize);
headerPacket.RemoveHeader(mpeg_header);
message_size = mpeg_header.GetSize() + (mpeg_header.GetSize()/m_app->m_packetPayload)*headersize
if (mpeg_header.GetSize()%m_app->m_packetPayload > 0)
    message_size+=headersize;
```

This part of receiving data is of critical importance to the client, since any incoming data may easily become unreadable in case the parsing function *HttpParser::ReadSocket* is poorly implemented. Although most of low-level TCP communications are internally handled by the native NS3 functionalities, the server needs to correctly send the data that the client is expecting and the client always must verify that this information is received. For that reason, the http parser is instructed to calculate the message size with and without the above-mentioned headers.

```
Packet message(m_buffer, message_size);
memmove(m_buffer, &m_buffer[message_size], m_bytes - message_size)
m_bytes -= message_size;
m_app->MessageReceived(message);
```

Receiving the first bytes, the parser can determine the size of the MPEG message it is waiting for. So, knowing that, it waits until the buffer content exceeds the expected message size, and when that happens, it forwards the received message to the upper level of the client application and to function *EvalvidClient::MessageReceived.* In the end, it moves the buffer index to deallocate the memory that the message was occupying.

4.4 Presenting the Results

4.4.1 Plots

As mentioned in our previous chapter, one of the greatest advantages of using the LENA project for an LTE network simulation is the ability to generate output for further study. This is achieved through a group of trace files following the end of the network simulation. The generated trace files contribute to our network research providing useful information regarding the PHY, MAC, RLC and PDCP layers.

Researchers agree that the best way to assess a simulation's output information is through plotted statistics. Plots can easily be evaluated and compared showing an overall view of the nodes and their network behavior. In our case, the tool which we used to plot our research statistics, in an Ubuntu terminal environment, is Gnuplot.

According to its documentation, Gnuplot is a command-driven interactive function plotting program. It can be used to plot functions and data points in both two- and three-dimensional plots in many different formats. It is designed primarily for the visual display of scientific data such as network statistics. Although Gnuplot is copyrighted, it is freely distributable.

Gnuplot needs to be installed in order to be used in a research program, and is usually run externally through the command line when supplied with a .plt file containing the points to be plotted in the correct format.

```
plot.GenerateOutput (plotFile);
system(("gnuplot < " + plotFileName).c_str());
```

4.4.2 Throughput Calculation

Since the scenario implemented in our LENA example included the use of bitrate adaptation in the streaming server-client application, probably the most crucial stat provided by the LENA trace files is the ability for a throughput measurement. The RLC downlink stats included in output file “DlRlcStats.txt” contain the bytes received per network node and the start and end time for each node’s client application, as in figure 4.4. This is all we need to calculate the throughput and create a throughput plot for every node in our simulation.

DlRlcStats.txt

% start	end	CellId	IMSI	RNTI	LCID	nTxPDUs	TxBytes	nRxPDUs	RxBytes
1	1.25	1	1	1	3	2	708	2	708
1.25	1.5	1	1	1	3	3	4734	3	4734
1.5	1.75	1	1	1	3	16	18630	16	18630
1.75	2	1	1	1	3	12	4921	12	4921
2	2.25	1	1	1	3	9	5858	9	5858
2.25	2.5	1	1	1	3	12	4616	12	4616
2.5	2.75	1	1	1	3	10	3425	10	3425
2.75	3	1	1	1	3	12	4376	12	4376
3	3.25	1	1	1	3	10	5749	10	5749
3.25	3.5	1	1	1	3	11	4227	11	4227
3.5	3.75	1	1	1	3	12	4371	12	4371

Figure 4.4: DlRlcStats.txt Sample

In file “plotmaker.cc”, the CalculateThroughputperNode function creates an array of Gnuplot 2D datasets, one for each node, and holds a pointer to the array. The function then reads the respective stats file and parses every line as tokens. From the extracted line tokens, it reads the bytes received and divides them by the elapsed time between the start and the end time which will give the node’s throughput at the time of the measurement. After it calculates the throughput in Kbps, it sends the measurement time and then-current throughput as a pair of coordinates to the dataset of the respective node. Following that process, the points are saved and ready to be plotted.

```
Gnuplot2dDataset** datasetArray = new Gnuplot2dDataset*[nUe];
for (num = 0; num < nUe; num++)
    datasetArray[num] = new Gnuplot2dDataset();

datasetArray[atoi(tokens.at(3).c_str())-1]->Add(atof(tokens.at(1).c_str()),
  atof(tokens.at(9).c_str())*8/1000/
  (atof(tokens.at(1).c_str())-atof(tokens.at(0).c_str()))); //(End time, Kbps)
```

The pointer returned by the CalculateThroughputperNode function is forwarded to function PlotStatistics, which in turn creates a separate plot file for every node named “Throughput-<node number>” and passes the dataset to function Create2DPlotFile which will populate the files by producing the necessary plots from the datasets.

```
sstm << "Throughput-" << num+1;
plotName = sstm.str();
Create2DPlotFile(plotName, simTime, "Seconds", "Kbps", *datasets[num])
```

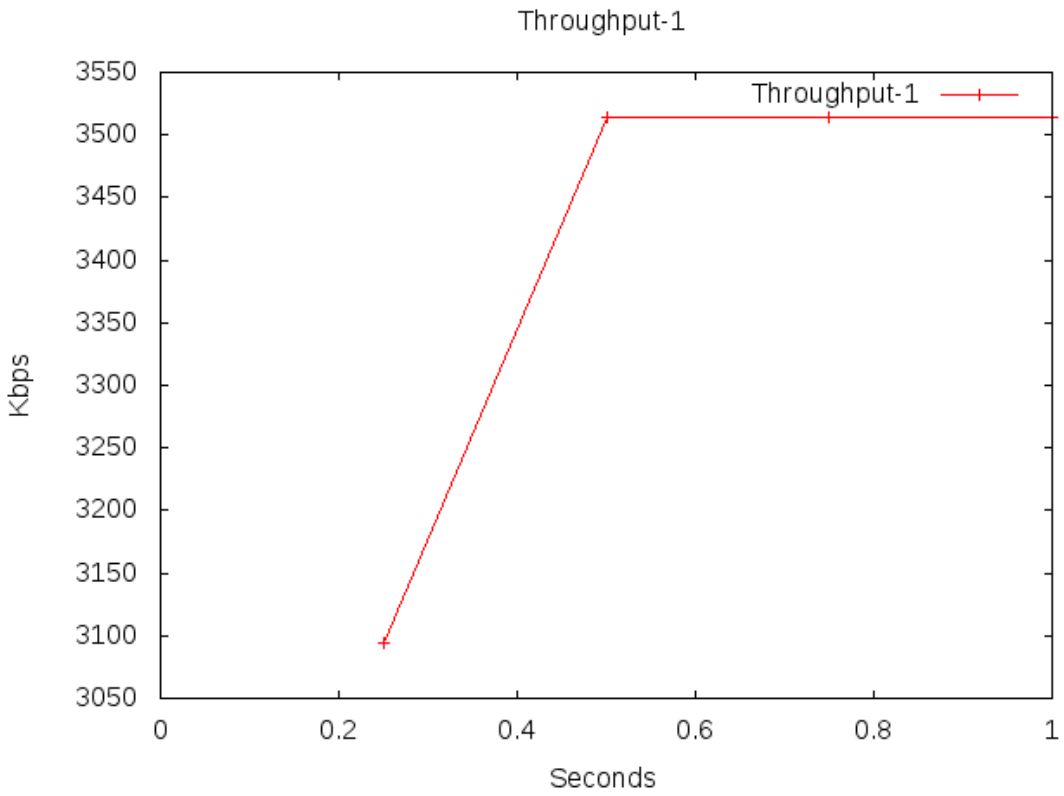

Figure 4.5: Throughput, Node 1

In *Create2DPlotFile,* we chose our output to be a .png image, with the range being equal to the simulation time, and our axis named “Kbps” and “Seconds”. The dataset is then instantiated, its title is set to the file’s name and the dataset points are plotted along with connecting lines in order to show the continuous evolution and possible fluctuation of the node’s throughput. Lastly, we execute the external gnuplot command, to generate our final output.

4.4.3 SINR Computation

The role of multisource interference is thoroughly discussed in some of the previous chapters of this thesis. The metric which introduces the effect of interference in telecommunications engineering is the Signal-to-Interference-plus-Noise Ratio (SINR). The SINR is generally used to measure the quality of wireless connections, the path loss, and set an upper limit on the wireless channel’s capacity.

Due to its importance, the physical trace in all NS3 LENA applications outputs a file named “DlRsprSinrStats.txt”. This file contains the following in columns:

- Simulation time in seconds at which the allocation is indicated by the scheduler
- Cell ID
- unique UE ID (IMSI)
- RSRP
- Linear average over all RBs of the downlink SINR in linear units

DlRsrpSinrStats.txt

% time	cellId	IMSI	RNTI	rsrp	sinr	
0.000214285	1	1	0	0.00333333	7.02728e+12	
0.00121429	1	1	0	0.00333333	7.02728e+12	
0.00221428	1	1	0	0.00333333	7.02728e+12	
0.00321428	1	1	0	0.00333333	7.02728e+12	
0.00421428	1	1	0	0.00333333	7.02728e+12	
0.00521428	1	1	0	0.00333333	7.02728e+12	
0.00621428	1	1	0	0.00333333	7.02728e+12	
0.00721428	1	1	0	0.00333333	7.02728e+12	
0.00821428	1	1	0	0.00333333	7.02728e+12	
0.00921428	1	1	0	0.00333333	7.02728e+12	
0.0102143	1	1	0	0.00333333	7.02728e+12	
0.0112143	1	1	0	0.00333333	7.02728e+12	
0.0122143	1	1	0	0.00333333	7.02728e+12	
0.0132143	1	1	0	0.00333333	7.02728e+12	
0.0142143	1	1	0	0.00333333	7.02728e+12	
0.0152143	1	1	0	0.00333333	7.02728e+12	
0.0162143	1	1	0	0.00333333	7.02728e+12	
0.0172143	1	1	0	0.00333333	7.02728e+12	
0.0182143	1	1	0	0.00333333	7.02728e+12	
0.0192143	1	1	0	0.00333333	7.02728e+12	
0.0202143	1	1	0	0.00333333	7.02728e+12	

Figure 4.6: DlRsprSinrStats.txt Sample

It is a file which is widely used by NS3 researchers as it provides useful feedback on the quality of a UE's wireless capability and experience in the network. As our goal remains to offer an expandable NS3 scenario and a decent feedback toolkit, we tried to build the foundation for more complex LTE network simulations. The SINR computation is a vital part of the later and that is why we chose to implement it.

Going through the code in file "plotmaker.cc" it becomes obvious that our NS3 application has all the means necessary to track and locate potential interference issues by plotting the SINR statistics for every node. At the end of the simulation, the output file "DlRsprSinrStats.txt" which is produced, is used by function CalculateSINRperNode of file "plotmaker.cc".

```
datasets = CalculateSINRperNode("DlRsrpSinrStats.txt", nUe);
for (num = 0; num < nUe; num++)
```

The above function creates an array of Gnuplot two-dimensional datasets, one for each node, holding a pointer to the array. Then, exactly the same way we previously described for the CalculateThroughputperNode function, it reads the statistics file line by line tokenizing every line.

```
datasetArray[atoi(tokens.at(2).c_str())-1]->Add(atof(tokens.at(0).c_str()),
        atof(tokens.at(5).c_str())); //(timestamp, SINR)
```

Since the SINR statistics file holds the measurement of the timestamp in column 1, the unique UE ID in column 3 and the SINR value in column 6, our script only holds the tokens located in the respective positions, in every line. Once this information is parsed from the file, it is inserted in each node's dataset in coordinates pairing the SINR value and the timestamp of the measurement moment.

```
sstm << "SINR-" << num+1;
plotName = sstm.str();
Create2DPlotFile(plotName, simTime, "Seconds", "Linear SINR", *datasets[num])
```

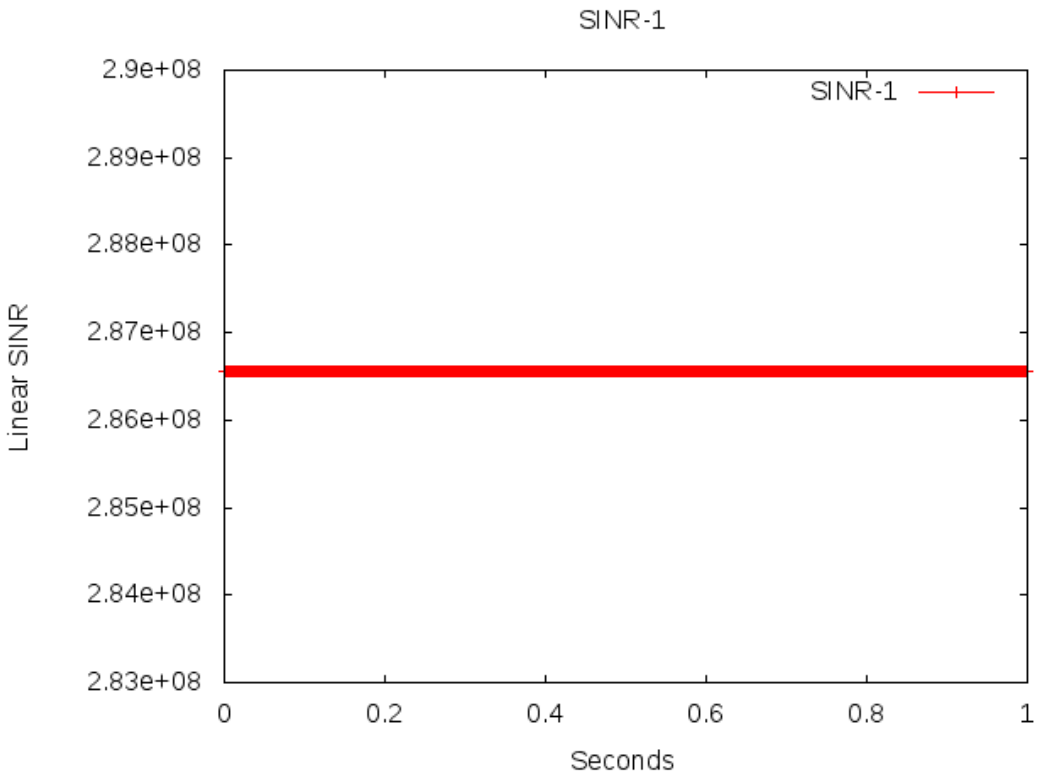

Figure 4.7: SINR, Node 1

The datasets that are generated are forwarded to the *PlotStatistics* function, which in turn calls function *Create2DPlotFile*. That function sets the axis names to Linear SINR/Seconds and populates a .png image with the points included in the datasets. The range is again equal to the duration of our simulation, stated in variable simTime. All the points are connected with lines, showing the progressive improvement or deterioration of the SINR value for each node.

4.4.4 QoE Metrics

Concluding our efforts to provide a complete executable instance of our module, we have implemented some of the most used QoE metrics in our code. This allows us to extract diagrams that give researchers an impression of the nodes' QoE. Since the throughput value heavily impacts a node's quality of experience, as explained also in Chapter 3, we are using the same file that we used to calculate throughput, "DlRlcStats.txt" (figure 4.4).

Reading from file DlRlcStats.txt, we calculate the throughput and we divide it by the video encoding rate, in our case the mean encoding rate which is 77 kbps. What we get is a plot of the reception rate ρ presented in chapter 3.5. As mentioned there, a good

indication of the user's QoE is whether the reception rate is equal to or higher than 1. To show this comparison, we have also added a straight line where y=1 on the graph.

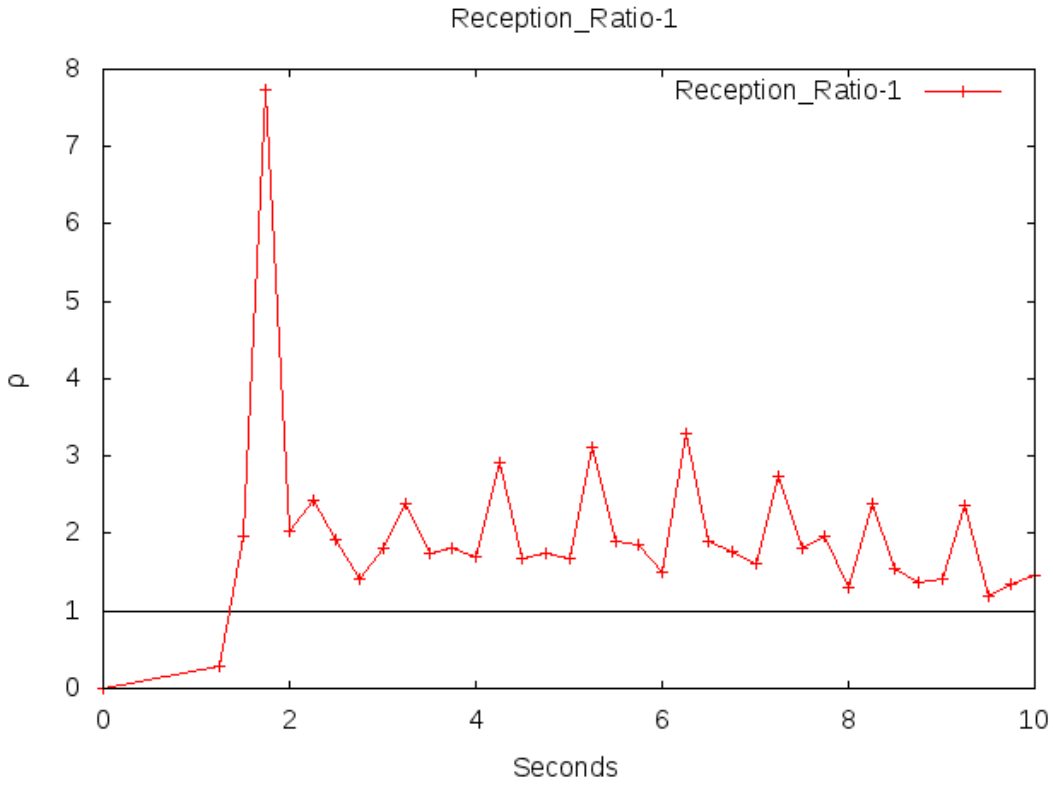

Figure 4.8: Reception Ratio ρ, Node 1

Function *CalculateReceptionRatioperNode* gets the encoding rate as argument from *PlotStatistics* and uses the formula to calculate the points and enter them in the reception ratio dataset. After that, it returns the dataset to *PlotStatistics* which creates the plot file having time as x-axis and *ρ* as y-axis.

```
datasetArray[atoi(tokens.at(3).c_str())-1]->Add(atof(tokens.at(1).c_str()),
        atof(tokens.at(9).c_str())*8/1000/(atof(tokens.at(1).c_str())-atof(
                tokens.at(0).c_str()))/encodingRate); //(End time, ρ)

sstm << "Reception_Ratio-" << num+1;
plotName = sstm.str();
Create2DPlotFile(plotName, simTime, "Seconds", "ρ", *datasets[num]);
```

In addition to the reception ratio figure, we are using a formula to also export a MOS graph, using the formula proposed in [68]. This formula is simply based on the percentage of time playing the video in the highest quality level, in an HTTP adaptive streaming system.

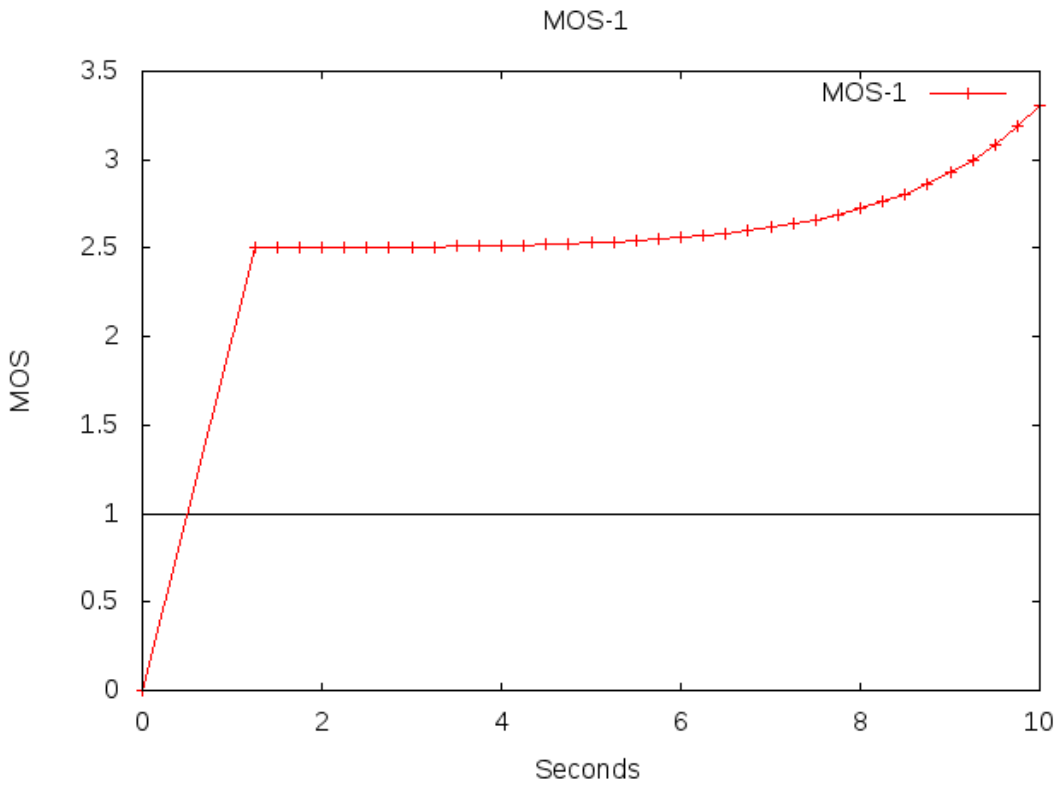

Figure 4.9: MOS, Node 1

```
datasetArray[atoi(tokens.at(3).c_str())-1]->Add(atof(tokens.at(1).c_str()),
    0.003*exp(0.064*100*timeAtBestRate[atoi(tokens.at(3).c_str())-1]/simTime)+2.498);
```

Going through the code, we first use file “DlRlcStats.txt” to calculate the throughput in function *CalculateMOSperNode*. For every time window where we find the node’s throughput to be greater than or equal to the highest video bitrate available, we add this time window to our counter and then divide this counter with the total simulation time up to this point. Using the resulting value in the formula, we get an accurate estimation of the user’s MOS, shown in figure 4.9.

```
sstm << "MOS-" << num+1;
plotName = sstm.str();
Create2DPlotFile(plotName, simTime, "Seconds", "MOS", *datasets[num]);
```

Function *CalculateMOSperNode* populate the dataset by calculating the points mentioned above, and then returns the dataset to calling function *PlotStatistics.* In *PlotStatistics,* the plot file is created with y-axis representing the estimated MOS value at time x.

ABBREVIATIONS-ACRONYMS

3GPP	3rd Generation Partnership Project
AAC	Advanced Audio Coding
ACR	Absolute Category Rating
AM	Acknowledged Mode
AMPS	Advanced Mobile Phone Service
APN	Access Point Name
ARP	Allocation/Retention Priority
AS	Access Stratum
AuC	Authentication Center
CDMA	Code Division Multiple Access
CN	Core Network
CPI	Cyclic Prefix Insertion
DASH	Dynamic Adaptive Streaming over HTTP
DECE LLC	Digital Entertainment Content Ecosystem
EDGE	Enhanced Data rates for GSM Evolution
eNodeB	Evolved Node B
EPC	Evolved Packet Core
EPS	Evolved Packet System
ETSI	European Telecommunications Standards Institute
F4V	Open container format for delivering synchronized audio/video streams
FDMA	Frequency Division Multiple Access
FTP	File Transfer Protocol
GBR	Guaranteed Bit Rate (Bearers)
GERAN	GSM EDGE Radio Access Network
GoB	Good or Better
GoP	Group of Pictures
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
GTP	GPRS Tunneling Protocol
HARQ	Hybrid Automatic Repeat Request

HAS	HTTP Adaptive Streaming
HSDPA	High Speed Downlink Packet Access
HSPA	High Speed Packet Access
HSS	Home Subscriber Server
HSUPA	High Speed Uplink Packet Access
HTTP	HyperText Transfer Protocol
IMS	IP Multimedia Subsystem
IMT	International Mobile Telecommunications
IPTV	IP Television
ITU	International Telecommunication Union
J-TACS	Japan-Total Access Communication System
LTE-A	Long Term Evolution-Advanced
MAC	Medium Access Control
MBR	Maximum Bit Rate
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MPD	Media Presentation Description
MPEG	Moving Picture Expert Group
MPQM	Moving Pictures Quality Metric
NMT	Nordic Mobile Telephone
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	OFDM Access
OIPF	Open IPTV Forum
PAPR	Peak-to-Average Power Ratio
PCEF	Policy Control Enforcement Function
PCRF	Policy Control and Charging Rules Function
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PDU	Protocol Data Unit
P-GW	PDN-Gateway
PoW	Poor or Worse
PSNR	Peak Signal-to-Noise Ratio

QCI	QoS Class Identifier
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
RTP	Real-time Transport Protocol
RV	Random Variable
SAE	System Architecture Evolution
SC-FDMA	Single Carrier-Frequency-Division Multiple Access
S-GW	Serving-Gateway
SOS	Standard deviation of Opinion Score
SSIM	Structural Similarity Index Metric
S-TMSI	SAE-Temporary Mobile Subscriber Identity
SVC	Scalable Video Coding
TA	Tracking Area
TACS	Total Access Communication System
TCP	Transmission Control Protocol
TDD	Time Division-Duplex
TDMA	Time Division Multiple Access
TD-SCDMA	Time-Division Synchronous Code Division Multiple Access
TFT	Traffic Flow Templates
TME	Terminate Early
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunication
URL	Uniform Resource Locator
UTRAN	Universal Terrestrial Radio Access Network
VoIP	Voice-over-IP
VQM	Video Quality Metric
W3C	World Wide Web Consortium

APPENDIX

```
/*
 * File: Lenaexample.cc
 * Author: Achilleas Moustakis
 * Copyright (c) 2016 National and Kapodistrian University of Athens, Greece
 */

#include "ns3/csma-helper.h"
#include "ns3/evalvid-client-server-helper.h"
#include "ns3/lte-helper.h"
#include "ns3/epc-helper.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-helper.h"
#include "ns3/config-store.h"


using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Lena_Example");

int main (int argc, char *argv[])
{


/*       Enable log components.*/

         LogComponentEnable ("EvalvidClient", LOG_LEVEL_INFO);
         LogComponentEnable ("EvalvidServer", LOG_LEVEL_ALL);

/*       Declare variables.*/

         uint16_t numberOfNodes = 1;
         double distance = 60.0;
         uint16_t serverPort = 1124;

/*       Create Helpers*/

         Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
         Ptr<PointToPointEpcHelper>  epcHelper = CreateObject<PointToPointEpcHelper>
();
         lteHelper->SetEpcHelper (epcHelper);
         Ptr<Node> pgw = epcHelper->GetPgwNode ();

/*       Create our remote host. Install the Internet stack.*/

         NodeContainer remoteHostContainer;
         remoteHostContainer.Create (1);
         Ptr<Node> remoteHost = remoteHostContainer.Get (0);
         InternetStackHelper internet;
         internet.Install (remoteHostContainer);

/*       Create a P2P connection between the P-GW and our remote host. Assign IP
addresses.*/

         PointToPointHelper p2ph;
         p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
         p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
         p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
         NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);
         Ipv4AddressHelper ipv4h;
         ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
         Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);
```

```
/*      Add a static network route for our remote host.*/

        Ipv4StaticRoutingHelper ipv4RoutingHelper;
        Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
ipv4RoutingHelper.GetStaticRouting (remoteHost->GetObject<Ipv4> ());
        remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address ("7.0.0.0"), Ipv4Mask
("255.0.0.0"), 1);

/*      Create our UE and eNodeB nodes. Install LTE functionality, Internet stack.*/

        NodeContainer ueNodes, enbNodes;
        enbNodes.Create(1);
        ueNodes.Create(numberOfNodes);
        NetDeviceContainer enbLteDevs = lteHelper->InstallEnbDevice (enbNodes);
        NetDeviceContainer ueLteDevs = lteHelper->InstallUeDevice (ueNodes);
        internet.Install (ueNodes);

/*      Create a mobility model and install our nodes in it. (Initial position is
defined by ListPositionAllocator.)*/

        Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>
();
        for (uint16_t i = 0; i < numberOfNodes; i++)
        {
                positionAlloc->Add (Vector(distance * i, 0, 0));
        }
        MobilityHelper mobility;
        mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
        mobility.SetPositionAllocator(positionAlloc);
        mobility.Install(enbNodes);
        mobility.Install(ueNodes);

/*      1.Assign IP addresses to our UE nodes.
 *      2.Add a static network route for each one.
 *      3.Set P-GW as the Default Gateway.
 *      4.Attach to eNodeB.*/

        Ipv4InterfaceContainer ueIpIface;
        ueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueLteDevs));
        for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
        {
              Ptr<Node> ueNode = ueNodes.Get (u);
              Ptr<Ipv4StaticRouting> ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4> ());
              ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress
(), 1);
        }
        for (uint16_t i = 0; i < numberOfNodes; i++)
        {
                lteHelper->Attach (ueLteDevs.Get(i), enbLteDevs.Get(0));
                // side effect: the default EPS bearer will be activated
        }


        NS_LOG_INFO ("Create Applications.");

/*      Create Server application. Install on our remote host.*/

        EvalvidServerHelper server(serverPort);
        server.SetAttribute ("SenderTraceFilename", StringValue("st_highway_cif.st"));
        server.SetAttribute ("SenderDumpFilename" , StringValue("sd_a01_lte"));
        ApplicationContainer apps = server.Install(remoteHostContainer.Get(0));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (10.0));

/*      Create Client applications. Install on our UE nodes.*/
        for (uint16_t i = 0; i < numberOfNodes; i++)
        {
                EvalvidClientHelper client (internetIpIfaces.GetAddress
(1),serverPort);//serverIP,serverPort
```

```
                std::ostringstream ReceiverDumpFilename;
                ReceiverDumpFilename <<"rd_a0" <<i <<"_lte";
                client.SetAttribute ("ReceiverDumpFilename",
StringValue(ReceiverDumpFilename.str()));
                client.SetAttribute ("VideoId", UintegerValue(1));
                apps = client.Install (ueNodes.Get(i));
                apps.Start (Seconds (1.1));
                apps.Stop (Seconds (9.9)); //Note: Clients start after and finish
before the Server.
        }

        lteHelper->EnableTraces (); //Enables trace sinks for PHY, MAC, RLC and PDCP.


        NS_LOG_INFO ("Run Simulation.");

        Simulator::Stop(Seconds(10));//Set Simulation stop time.
        Simulator::Run();                          //Start Simulation.
        Simulator::Destroy();
        return 0;
}
```

```
/*
 * File: plotmaker.cc
 * Author: Achilleas Moustakis
 * Copyright (c) 2016 National and Kapodistrian University of Athens, Greece
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/config-store.h"
#include <ns3/buildings-module.h>
#include <ns3/netanim-module.h>
#include "ns3/gnuplot.h"
#include <iomanip>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <sstream>
#include <cstdlib>

using namespace ns3;
using namespace std;

Gnuplot2dDataset** CalculateThroughputperNode(string statsFileName, uint32_t nUe)
{
/*The stats file we are using contains the start/end time in columns 1/2,
 * the IMSI in column 4 and Bytes Received in column 10. Therefore, we
 * extract the according tokens from every line of the file.*/
        uint32_t num;
        //Create an array of datasets (one for each node) and hold a pointer of the
array (DYNAMIC ALLOCATION)
        Gnuplot2dDataset** datasetArray = new Gnuplot2dDataset*[nUe];
        for (num = 0; num < nUe; num++)
                datasetArray[num] = new Gnuplot2dDataset();
                datasetArray[num]->Add(0,0);

        //Read stats file line by line and tokenize
        string line;
        ifstream myfile(statsFileName.c_str());
        if (myfile)
        {
                getline(myfile,line);
                vector<string> tokens;
                while (getline(myfile,line)) {
                        string delimiter = "     ";
                        size_t pos = 0;
                        string token;
                        while ((pos = line.find(delimiter)) != string::npos) {
                                token = line.substr(0, pos);
                                tokens.push_back(token);
                                line.erase(0, pos + delimiter.length());
                        }
                        datasetArray[atoi(tokens.at(3).c_str())-1]-
>Add(atof(tokens.at(1).c_str()),

atof(tokens.at(9).c_str())*8/1000/(atof(tokens.at(1).c_str())-
atof(tokens.at(0).c_str()))); //(End time, Kbps)
                        tokens.clear();
                }
                myfile.close();
        }
        return datasetArray;
}

double highestBitrate;
double statsInterval;
```

```
Gnuplot2dDataset** CalculateMOSperNode(string statsFileName, uint32_t nUe, double
simTime)
{
/*The stats file we are using contains the start/end time in columns 1/2,
 * the IMSI in column 4 and Bytes Received in column 10. Therefore, we
 * extract the according tokens from every line of the file.*/

        uint32_t num;
        double throughput;

        //Create an array of datasets (one for each node) and hold a pointer of the
array (DYNAMIC ALLOCATION)
        Gnuplot2dDataset** datasetArray = new Gnuplot2dDataset*[nUe];
        double timeAtBestRate[nUe];
        for (num = 0; num < nUe; num++){
                datasetArray[num] = new Gnuplot2dDataset();
                datasetArray[num]->Add(0,0);
                timeAtBestRate[num]=0;
        }
        //Read stats file line by line and tokenize
        string line;
        ifstream myfile(statsFileName.c_str());
        if (myfile)
        {
                getline(myfile,line);
                vector<string> tokens;
                while (getline(myfile,line)) {
                        string delimiter = "      ";
                        size_t pos = 0;
                        string token;
                        while ((pos = line.find(delimiter)) != string::npos) {
                                token = line.substr(0, pos);
                                tokens.push_back(token);
                                line.erase(0, pos + delimiter.length());
                        }
                        throughput =
atof(tokens.at(9).c_str())*8/1000/(atof(tokens.at(1).c_str())-
atof(tokens.at(0).c_str()));
                        /* If Throughput within this time window equals or exceeds
the highest Bitrate
                         * then add this window to the aggregate time being on the
highest layer */
                        if (throughput >= highestBitrate) {
                                timeAtBestRate[atoi(tokens.at(3).c_str())-1] +=
statsInterval;
                        }
                        datasetArray[atoi(tokens.at(3).c_str())-1]-
>Add(atof(tokens.at(1).c_str()),
0.003*exp(0.064*100*timeAtBestRate[atoi(tokens.at(3).c_str())-1]/simTime)+2.498);
//(End time, MOS)
                        tokens.clear();
                }
                myfile.close();
        }
        return datasetArray;
}

Gnuplot2dDataset** CalculateReceptionRatioperNode(string statsFileName, uint32_t nUe,
double encodingRate)
{
/*The stats file we are using contains the start/end time in columns 1/2,
 * the IMSI in column 4 and Bytes Received in column 10. Therefore, we
 * extract the according tokens from every line of the file.*/

        uint32_t num;

        //Create an array of datasets (one for each node) and hold a pointer of the
array (DYNAMIC ALLOCATION)
        Gnuplot2dDataset** datasetArray = new Gnuplot2dDataset*[nUe];
        for (num = 0; num < nUe; num++){
```

```
                datasetArray[num] = new Gnuplot2dDataset();
                datasetArray[num]->Add(0,0);
        }

        //Read stats file line by line and tokenize
        string line;
        ifstream myfile(statsFileName.c_str());
        if (myfile)
        {
                getline(myfile,line);
                vector<string> tokens;
                while (getline(myfile,line)) {
                        string delimiter = "     ";
                        size_t pos = 0;
                        string token;
                        while ((pos = line.find(delimiter)) != string::npos) {
                                token = line.substr(0, pos);
                                tokens.push_back(token);
                                line.erase(0, pos + delimiter.length());
                        }
                        datasetArray[atoi(tokens.at(3).c_str())-1]-
>Add(atof(tokens.at(1).c_str()),
atof(tokens.at(9).c_str())*8/1000/(atof(tokens.at(1).c_str())-
atof(tokens.at(0).c_str()))/encodingRate); //(End time, ρ)
                        tokens.clear();
                }
                myfile.close();
        }
        return datasetArray;
}


Gnuplot2dDataset** CalculateSINRperNode(string statsFileName, uint32_t nUe)
{
/*The stats file we are using contains the measurement timestamp in column 1,
 * the IMSI in column 3 and the SINR in column 6. Therefore, we
 * extract the according tokens from every line of the file.*/
        uint32_t num;
        //Create an array of datasets (one for each node) and hold a pointer of the
array (DYNAMIC ALLOCATION)
        Gnuplot2dDataset** datasetArray = new Gnuplot2dDataset*[nUe];
        for (num = 0; num < nUe; num++)
                datasetArray[num] = new Gnuplot2dDataset();
                datasetArray[num]->Add(0,0);

        //Read stats file line by line and tokenize
        string line;
        ifstream myfile(statsFileName.c_str());
        if (myfile)
        {
                getline(myfile,line);
                vector<string> tokens;
                while (getline(myfile,line)) {
                        string delimiter = "     ";
                        size_t pos = 0;
                        string token;
                        while ((pos = line.find(delimiter)) != string::npos) {
                                token = line.substr(0, pos);
                                tokens.push_back(token);
                                line.erase(0, pos + delimiter.length());
                        }
                        tokens.push_back(line);
                        line.clear();
                        datasetArray[atoi(tokens.at(2).c_str())-1]-
>Add(atof(tokens.at(0).c_str()), atof(tokens.at(5).c_str())); //(timestamp, SINR)
                        tokens.clear();
                }
                myfile.close();
        }
        return datasetArray;
```

```
}

void Create2DPlotFile (string fileName, double simTime, string axisXname, string
axisYname, Gnuplot2dDataset dataset)
{
  std::string fileNameWithNoExtension = fileName;
  std::string graphicsFileName        = fileNameWithNoExtension + ".png";
  std::string plotFileName            = fileNameWithNoExtension + ".plt";
  std::string plotTitle               = fileName;
  std::string dataTitle               = fileName;

  // Instantiate the plot and set its title.
  Gnuplot plot (graphicsFileName);
  plot.SetTitle (plotTitle);

  // Make the graphics file, which the plot file will create when it
  // is used with Gnuplot, be a PNG file.
  plot.SetTerminal ("png");

  // Set the labels for each axis.
  plot.SetLegend (axisXname, axisYname);

  // Set the range for the x axis
  std::stringstream sstm;
  sstm << simTime;
  plot.AppendExtra ("set xrange [0:" + sstm.str() + "]");

  // Instantiate the dataset, set its title, and make the points be
  // plotted along with connecting lines.
  dataset.SetTitle (dataTitle);
  dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);

  // Add the dataset to the plot.
  plot.AddDataset (dataset);

  // Open the plot file.
  std::ofstream plotFile (plotFileName.c_str());

  // Write the plot file.
  plot.GenerateOutput (plotFile);
  system(("gnuplot < " + plotFileName).c_str());

  // Close the plot file.
  plotFile.close ();
}

void PlotStatistics(uint32_t nUe, double simTime){
        //Create throughput datasets for every node and plot statistics
        Gnuplot2dDataset** datasets = CalculateThroughputperNode("DlRlcStats.txt",
nUe);
        string plotName;
        uint32_t num;
        for (num = 0; num < nUe; num++)
        {
                std::stringstream sstm;
                sstm << "Throughput-" << num+1;
                plotName = sstm.str();
                Create2DPlotFile(plotName, simTime, "Seconds", "Kbps",
*datasets[num]);
                plotName.clear();
                delete datasets[num];
        }
        delete datasets;

        //Create MOS datasets for every node and plot statistics
        datasets = CalculateMOSperNode("DlRlcStats.txt", nUe, simTime);
        for (num = 0; num < nUe; num++)
        {
                std::stringstream sstm;
                sstm << "MOS-" << num+1;
```

```
                plotName = sstm.str();
                Create2DPlotFile(plotName, simTime, "Seconds", "MOS",
*datasets[num]);
                plotName.clear();
                delete datasets[num];
        }
        delete datasets;

        //Create reception ratio for every node and plot statistics
        datasets = CalculateReceptionRatioperNode("DlRlcStats.txt", nUe,
encodingRate);
        for (num = 0; num < nUe; num++)
        {
                std::stringstream sstm;
                sstm << "Reception_Ratio-" << num+1;
                plotName = sstm.str();
                Create2DPlotFile(plotName, simTime, "Seconds", "ρ", *datasets[num]);
                plotName.clear();
                delete datasets[num];
        }
        delete datasets;

        //Create SINR datasets for every node and plot statistics
        datasets = CalculateSINRperNode("DlRsrpSinrStats.txt", nUe);
        for (num = 0; num < nUe; num++)
        {
                std::stringstream sstm;
                sstm << "SINR-" << num+1;
                plotName = sstm.str();
                Create2DPlotFile(plotName, simTime, "Seconds", "Linear SINR",
*datasets[num]);
                plotName.clear();
                delete datasets[num];
        }
delete datasets;
}
int main (int argc, char *argv[])
{
/*In order to run properly, please provide the number of User Equipments (nUe)
 * and the simulation time (simTime) before execution.*/

        uint32_t nUe = 1;
        const double simTime = 10.0;
        const double encodingRate = 77.09; //kbps
        highestBitrate = 80; //kbps
        statsInterval = 0.25; //secs
        PlotStatistics(nUe, simTime, encodingRate);}
```

```
/*
 * File: evalvid-client-server-helper.cc
 */

#include "../../evalvid/helper/evalvid-client-server-helper.h"
#include "ns3/evalvid-client.h"
#include "ns3/evalvid-server.h"
#include "ns3/uinteger.h"
#include "ns3/string.h"
namespace ns3 {

         EvalvidServerHelper::EvalvidServerHelper (){}

         EvalvidServerHelper::EvalvidServerHelper (uint16_t port)
         {
                 m_factory.SetTypeId (EvalvidServer::GetTypeId ());
                 SetAttribute ("Port", UintegerValue (port));
         }

         void
         EvalvidServerHelper::SetAttribute(std::string name,const AttributeValue&value)
         {
                 m_factory.Set (name, value);
         }

         ApplicationContainer EvalvidServerHelper::Install (NodeContainer c)
         {
                 ApplicationContainer apps;
                 for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
                 {
                         Ptr<Node> node = *i;
                         m_server = m_factory.Create<EvalvidServer> ();
                         node->AddApplication (m_server);
                         apps.Add (m_server);
                 }
                 return apps;
         }

         Ptr<EvalvidServer> EvalvidServerHelper::GetServer (void)
         {
                 return m_server;
         }

         EvalvidClientHelper::EvalvidClientHelper () {}
         EvalvidClientHelper::EvalvidClientHelper (Ipv4Address ip,uint16_t port)
         {
                 m_factory.SetTypeId (EvalvidClient::GetTypeId ());
                 SetAttribute ("RemoteAddress", Ipv4AddressValue (ip));
                 SetAttribute ("RemotePort", UintegerValue (port));
         }

        void
        EvalvidClientHelper::SetAttribute(std::string name,const AttributeValue &value)
         {
                 m_factory.Set (name, value);
         }

         ApplicationContainer EvalvidClientHelper::Install (NodeContainer c)
         {
                 ApplicationContainer apps;
                 for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
                 {
                      Ptr<Node> node = *i;
                      Ptr<EvalvidClient> client = m_factory.Create<EvalvidClient>();
                      node->AddApplication (client);
                      apps.Add (client);
                 }
                 return apps;
         }
}
```

```
/*
 * File: evalvid-server.cc
 */

#include "../../evalvid/model/evalvid-server.h"
#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/string.h"
#include <ns3/tcp-socket.h>
#include "mpeg-header.h"
#include "http-header.h"
using namespace std;

namespace ns3 {
NS_LOG_COMPONENT_DEFINE ("EvalvidServer");
NS_OBJECT_ENSURE_REGISTERED (EvalvidServer);
TypeId
EvalvidServer::GetTypeId (void)
{
        static TypeId tid = TypeId ("ns3::EvalvidServer")
                .SetParent<Application> ()
                .AddConstructor<EvalvidServer> ()
                .AddAttribute ("Port",
                        "Port on which we listen for incoming packets.",
                        UintegerValue (100),
                        MakeUintegerAccessor (&EvalvidServer::m_port),
                        MakeUintegerChecker<uint16_t> ())
                .AddAttribute ("SenderDumpFilename",
                        "Sender Dump Filename",
                        StringValue(""),
                        MakeStringAccessor(&EvalvidServer::m_senderTraceFileName),
                        MakeStringChecker())
                .AddAttribute ("SenderTraceFilename",
                        "Sender trace Filename",
                        StringValue(""),
                        MakeStringAccessor(&EvalvidServer::m_videoTraceFileName),
                        MakeStringChecker())
                .AddAttribute ("PacketPayload",
                        "MTU",
/*In our case:   MTU       - (SEQ_HEADER + TCP_HEADER + IP_HEADER + HTTP HEADER + MPEG
HEADER) so: 1500 - (12 + 20 + 20 + 28 + 32) = 1388
But for now we use 460 to avoid TCP truncation.*
*/
                        UintegerValue (460),
                        MakeUintegerAccessor (&EvalvidServer::m_packetPayload),
                        MakeUintegerChecker<uint16_t> ())
                ;
        return tid;
}

EvalvidServer::EvalvidServer ()
{
        m_socket = 0;
        m_port = 0;
        m_numOfFrames = 0;
        m_packetPayload = 0;
        m_packetId = 0;
        m_sendEvent = EventId ();
        packetcount = 0;
        m_totalRx = 0;
        videoId = -1;
}
```

```
EvalvidServer::~EvalvidServer ()
{
        NS_LOG_FUNCTION (this);
}

void
EvalvidServer::DoDispose (void)
{
        NS_LOG_FUNCTION (this);
        Application::DoDispose ();
}

void
EvalvidServer::StartApplication (void)
{
        NS_LOG_FUNCTION_NOARGS();

        Ptr<Socket> socket = 0;
        if (socket == 0)
        {
                TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");
                socket = Socket::CreateSocket (GetNode (), tid);
                InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (),
m_port);
                socket->Bind (local);
                socket->Listen();
                socket->SetRecvCallback (MakeCallback (&EvalvidServer::HandleRead,
this));
                socket->SetAcceptCallback(
                                MakeNullCallback<bool, Ptr<Socket>, const Address
&>(),
                                MakeCallback(&EvalvidServer::HandleAccept, this));
        }


/*      //For IPv6 only:

        Ptr<Socket> socket6 = 0;
        if (socket6 == 0)
        {
                TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");//A
                socket6 = Socket::CreateSocket (GetNode (), tid);
                Inet6SocketAddress local = Inet6SocketAddress (Ipv6Address::GetAny
(), m_port);
                socket6->Bind (local);
                socket6->Listen();
                socket6->SetRecvCallback (MakeCallback (&EvalvidServer::HandleRead,
this));
        }*/

        Setup(); //Setup the video(s) for distribution.
}

void
EvalvidServer::StopApplication ()
{
        NS_LOG_FUNCTION_NOARGS();
        //Simulator::Cancel (m_sendEvent);
}

void
EvalvidServer::Setup()
{
        NS_LOG_FUNCTION_NOARGS();

        m_videoInfoStruct_t *videoInfoStruct;
        uint32_t frameId;
        string frameType;
        uint32_t frameSize;
        uint16_t numOfTcpPackets;
```

```
        double sendTime;
        double lastSendTime = 0.0;

        ifstream videoTraceFile(m_videoTraceFileName.c_str(), ios::in);
        if (videoTraceFile.fail())
        {
                NS_FATAL_ERROR(">> EvalvidServer: Error while opening video trace
file: " << m_videoTraceFileName.c_str());
                return;
        }

        while (videoTraceFile >> frameId >> frameType >> frameSize >> numOfTcpPackets
>> sendTime)
        {
                videoInfoStruct = new m_videoInfoStruct_t;
                videoInfoStruct->frameType = frameType;
                videoInfoStruct->frameSize = frameSize;
                videoInfoStruct->numOfTcpPackets = frameSize/m_packetPayload;
                videoInfoStruct->packetInterval = Seconds(sendTime - lastSendTime);
                m_videoInfoMap.insert (pair<uint32_t, m_videoInfoStruct_t*>(frameId,
videoInfoStruct));
                //NS_LOG_LOGIC(">> EvalvidServer: " << frameId << "\t" << frameType
<< "\t" <<frameSize << "\t" << numOfTcpPackets << "\t" << sendTime);
                lastSendTime = sendTime;
        }

        m_numOfFrames = frameId;
        m_videoInfoMapIt = m_videoInfoMap.begin();

        m_senderTraceFile.open(m_senderTraceFileName.c_str(), ios::out);
        if (m_senderTraceFile.fail())
        {
                NS_FATAL_ERROR(">> EvalvidServer: Error while opening sender trace
file: " << m_senderTraceFileName.c_str());
                return;
        }
}

void
EvalvidServer::Send () /*Sends one frame at a time! Calls itself until the last frame
of the segment is sent.*/
{
  NS_LOG_FUNCTION( this << Simulator::Now().GetSeconds());

  if (m_videoInfoMapIt != m_videoInfoMap.end())
    {
      for(int i=0; i<m_videoInfoMapIt->second->numOfTcpPackets; i++)
        {
          Ptr<Packet> p = Create<Packet> (m_packetPayload);//originally 1460 bytes
          m_packetId++;
          m_senderTraceFile << std::fixed << std::setprecision(4) <<
Simulator::Now().ToDouble(Time::S)
                            << std::setfill(' ') << std::setw(16) <<  "id " <<
m_packetId
                            << std::setfill(' ') <<  std::setw(16) <<  "tcp " << p-
>GetSize()
                            << std::endl;

          /* Add headers to the packet. */
          SeqTsHeader seqTs;
          seqTs.SetSeq (m_packetId);
          HTTPHeader http_header;
          http_header.SetMessageType(HTTP_RESPONSE);
          http_header.SetVideoId(videoId);
          http_header.SetResolution(clientResolution);
          http_header.SetSegmentId(segmentId);
          MPEGHeader mpeg_header;
          mpeg_header.SetFrameId(m_videoInfoMapIt->first);
          mpeg_header.SetPlaybackTime(
                   MilliSeconds(
```

```
                                    (m_videoInfoMapIt->first - 1)
                                    * MPEG_TIME_BETWEEN_FRAMES)); //50 FPS
          mpeg_header.SetType(m_videoInfoMapIt->second->frameType);
          mpeg_header.SetSize(m_videoInfoMapIt->second->frameSize);

          p->AddHeader (seqTs);
          p->AddHeader(http_header);
          p->AddHeader(mpeg_header);
          packetcount++;
          std::cout <<"Server: bytes sent->"<<p->GetSize() <<" frame size->"
<<mpeg_header.GetSize() <<" packetcount:"<<packetcount <<std::endl;
          m_socket->SendTo(p, 0, m_peerAddress);
        }
      Ptr<Packet> p = Create<Packet> (m_videoInfoMapIt->second->frameSize %
m_packetPayload);
      m_packetId++;
      m_senderTraceFile  << std::fixed << std::setprecision(4)
                << Simulator::Now().ToDouble(Time::S)
                << std::setfill(' ') << std::setw(16) <<  "id " << m_packetId
                << std::setfill(' ') <<  std::setw(16) <<  "tcp " << p>GetSize()
                << std::setfill(' ') << std::setw(16) <<  "frame id "
                << m_videoInfoMapIt->first << std::endl;

      /* Add headers to the packet. */
      SeqTsHeader seqTs;
      seqTs.SetSeq (m_packetId);

      HTTPHeader http_header;
      http_header.SetMessageType(HTTP_RESPONSE);
      http_header.SetVideoId(videoId);
      http_header.SetResolution(clientResolution);
      http_header.SetSegmentId(segmentId);

      MPEGHeader mpeg_header;
      mpeg_header.SetFrameId(m_videoInfoMapIt->first);
      mpeg_header.SetPlaybackTime(
                  MilliSeconds((m_videoInfoMapIt->first + (segmentId *
MPEG_FRAMES_PER_SEGMENT))
                                    * MPEG_TIME_BETWEEN_FRAMES)); //50 fps
      mpeg_header.SetType(m_videoInfoMapIt->second->frameType);
      mpeg_header.SetSize(m_videoInfoMapIt->second->frameSize);
      p->AddHeader (seqTs);
      p->AddHeader(http_header);
      p->AddHeader(mpeg_header);
      packetcount++;
      std::cout <<"Server: bytes sent->"<<p->GetSize() <<" frame size->"
<<mpeg_header.GetSize() <<" packetcount:"<<packetcount <<std::endl;
      m_socket->SendTo(p, 0, m_peerAddress);
      m_videoInfoMapIt++;
      if (m_videoInfoMapIt == m_videoInfoMap.end())
        {
          NS_LOG_INFO(">> EvalvidServer: Video streaming successfully completed!");
        }
      else if ((m_videoInfoMapIt->first - 1) % MPEG_FRAMES_PER_SEGMENT == 0)
      {
           NS_LOG_INFO(">> EvalvidServer: Sending segment " <<segmentId <<"
complete!");
      }
      else
        {
          if (m_videoInfoMapIt->second->packetInterval.GetSeconds() == 0)
            {
              m_sendEvent = Simulator::ScheduleNow (&EvalvidServer::Send, this);
            }
          else
            {
              m_sendEvent = Simulator::Schedule (m_videoInfoMapIt->second->
                 packetInterval, &EvalvidServer::Send, this);
            }
        }
```

```
    }
  else
    {
      NS_FATAL_ERROR(">> EvalvidServer: Frame does not exist!");
    }
}

void
EvalvidServer::HandleRead (Ptr<Socket> socket)
{
  //NS_LOG_FUNCTION_NOARGS();
  Ptr<Packet> packet;
  Address from;
  m_socket = socket;

  while ((packet = socket->RecvFrom (from)))
    {
      m_peerAddress = from;
      m_totalRx += packet->GetSize();

      /*Extract info from received client request packet. */
      HTTPHeader header;
      packet->RemoveHeader(header);

      videoId = header.GetVideoId();
      segmentId = header.GetSegmentId();
      clientResolution = header.GetResolution();

      if (InetSocketAddress::IsMatchingType (from))
        {
          NS_LOG_INFO (">> EvalvidServer: Client at " <<
InetSocketAddress::ConvertFrom (from).GetIpv4 ()
                       << " is requesting a video streaming.");
        }
      /*//For IPv6 only:

      else if (Inet6SocketAddress::IsMatchingType (from))
        {
             NS_LOG_INFO (">> EvalvidServer: Client at " <<
Inet6SocketAddress::ConvertFrom (from).GetIpv6 ()
                          << " is requesting a video streaming.");
        }*/
      if (m_videoInfoMapIt != m_videoInfoMap.end())
        {
          NS_LOG_INFO(">> EvalvidServer: Starting video streaming...");
          if (m_videoInfoMapIt->second->packetInterval.GetSeconds() == 0)
            {
              m_sendEvent = Simulator::ScheduleNow (&EvalvidServer::Send, this);
            }
          else
            {
                m_sendEvent = Simulator::Schedule(m_videoInfoMapIt->second->
                packetInterval, &EvalvidServer::Send, this);
            }
        }
      else
        {
          NS_FATAL_ERROR(">> EvalvidServer: Frame does not exist!");
        }
      //m_rxTrace(packet, from);
    }
}

void
  EvalvidServer::HandleAccept(Ptr<Socket> s, const Address& from)
  {
    NS_LOG_FUNCTION(this << s << from);
    s->SetRecvCallback(MakeCallback(&EvalvidServer::HandleRead, this));
  }
}
```

```
/*
 * File: evalvid-client.cc
 */

#include "../../evalvid/model/evalvid-client.h"
#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include <stdlib.h>
#include <stdio.h>
#include "ns3/string.h"
#include "http-header.h"

namespace ns3 {

    NS_LOG_COMPONENT_DEFINE ("EvalvidClient");
    NS_OBJECT_ENSURE_REGISTERED (EvalvidClient);

    TypeId
    EvalvidClient::GetTypeId (void)
    {
        static TypeId tid = TypeId ("ns3::EvalvidClient")
            .SetParent<Application> ()
            .AddConstructor<EvalvidClient> ()
            .AddAttribute("VideoId", "The Id of the video that is played.",
                          UintegerValue(0),
                          MakeUintegerAccessor(&EvalvidClient::m_videoId),
                          MakeUintegerChecker<uint32_t>(1))
            .AddAttribute ("RemoteAddress",
                           "The destination Ipv4Address of the outbound packets",
                           Ipv4AddressValue (),
                           MakeIpv4AddressAccessor (&EvalvidClient::m_peerAddress),
                           MakeIpv4AddressChecker ())
            .AddAttribute ("RemotePort", "The destination port of the outbound
packets",
                           UintegerValue (100),
                           MakeUintegerAccessor (&EvalvidClient::m_peerPort),
                           MakeUintegerChecker<uint16_t> ())
            .AddAttribute ("ReceiverDumpFilename",
                           "Receiver Dump Filename",
                           StringValue(""),
                           MakeStringAccessor(&EvalvidClient::receiverDumpFileName),
                           MakeStringChecker())
            .AddAttribute ("PacketPayload",
                           "Packet Payload, i.e. MTU - (SEQ_HEADER + UDP_HEADER +
IP_HEADER). "
                           "This is the same value used to hint video with MP4Box.
Default: 1460.",
                          /*In our case:  MTU      - (SEQ_HEADER + TCP_HEADER +
IP_HEADER + HTTP HEADER + MPEG HEADER)
                           *              so:             1500 - (12       +
        20      +       20      +       28      +       32)      = 1388
                           * But for now we use 460 to avoid TCP truncation.
                           * */
                           UintegerValue (460),
                           MakeUintegerAccessor (&EvalvidClient::m_packetPayload),
                           MakeUintegerChecker<uint16_t> ())
            ;
        return tid;
    }

    EvalvidClient::EvalvidClient () : m_bitRate(80000), // default bitrate in bps
                                      m_segmentId(0) // seems to start with 0
    {
```

```
        NS_LOG_FUNCTION_NOARGS ();
        m_sendEvent = EventId ();
        m_parser.SetApp(this); // So the parser knows where to send the received
messages
    }

    EvalvidClient::~EvalvidClient ()
    {
        NS_LOG_FUNCTION_NOARGS ();
    }

    void
    EvalvidClient::SetRemote (Ipv4Address ip, uint16_t port)
    {
        m_peerAddress = ip;
        m_peerPort = port;
    }

    MpegPlayer&
    EvalvidClient::GetPlayer(){
        return m_player;
    }

    void
    EvalvidClient::DoDispose (void)
    {
        NS_LOG_FUNCTION_NOARGS ();
        Application::DoDispose ();
    }

    double
    EvalvidClient::GetBitRateEstimate()
    {
        return m_bitrateEstimate;
    }

    void
    EvalvidClient::StartApplication (void)
    {
        NS_LOG_FUNCTION_NOARGS();
        if (m_socket == 0)
        {
            TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");
            m_socket = Socket::CreateSocket (GetNode (), tid);

            if (m_socket->GetSocketType() != Socket::NS3_SOCK_STREAM
                      && m_socket->GetSocketType() != Socket::NS3_SOCK_SEQPACKET)
            NS_FATAL_ERROR ("Using HTTP with an incompatible socket type. "
                            "HTTP requires SOCK_STREAM or SOCK_SEQPACKET. "
                            "In other words, use TCP instead of UDP.");
            m_socket->Bind ();
            m_socket->Connect (InetSocketAddress (m_peerAddress, m_peerPort));

        }


        receiverDumpFile.open(receiverDumpFileName.c_str(), ios::out);
        if (receiverDumpFile.fail())
        {
            NS_FATAL_ERROR(">> EvalvidClient: Error while opening output file: " <<
receiverDumpFileName.c_str());
            return;
        }

        m_socket->SetRecvCallback (MakeCallback (&EvalvidClient::HandleRead, this));
        m_socket->SetConnectCallback(
                  MakeCallback(&EvalvidClient::ConnectionSucceeded, this),
                  MakeCallback(&EvalvidClient::ConnectionFailed, this));

    }
```

```
    void
    EvalvidClient::ConnectionSucceeded(Ptr<Socket> socket)
    {
        NS_LOG_FUNCTION(this << socket);
        NS_LOG_LOGIC("Connection succeeded!");
        //m_connected = true;
        Send(); //Request Segment
    }

    void
    EvalvidClient::ConnectionFailed(Ptr<Socket> socket)
    {
        NS_LOG_FUNCTION(this << socket);
        NS_LOG_LOGIC("Connection Failed");
    }

    void
    EvalvidClient::Send (void) /* Request segment. */
    {
        NS_LOG_FUNCTION_NOARGS ();

        Ptr<Packet> p = Create<Packet> (100);

        SeqTsHeader seqTs;
        seqTs.SetSeq (0);
        p->AddHeader (seqTs);

        HTTPHeader httpHeader;//Achilleas
        httpHeader.SetSeq(1);
        httpHeader.SetMessageType(HTTP_REQUEST);
        httpHeader.SetVideoId(m_videoId);
        httpHeader.SetResolution(m_bitRate);
        httpHeader.SetSegmentId(m_segmentId++);
        p->AddHeader(httpHeader);
        m_socket->Send (p);
        m_requestTime = Simulator::Now();
        m_segment_bytes = 0;
        NS_LOG_INFO (">> EvalvidClient: Sending request for video streaming to
EvalvidServer at "
                     << m_peerAddress << ":" << m_peerPort);
    }


    void
    EvalvidClient::StopApplication ()
    {
        NS_LOG_FUNCTION_NOARGS ();
        receiverDumpFile.close();
        Simulator::Cancel (m_sendEvent);
    }

    void
    EvalvidClient::HandleRead (Ptr<Socket> socket)
    {
        NS_LOG_FUNCTION (this << socket);
        m_parser.ReadSocket(socket);
    }

    void
    EvalvidClient::MessageReceived(Packet message)
    {
        NS_LOG_FUNCTION(this << message);

        MPEGHeader mpegHeader;
        HTTPHeader httpHeader;
        SeqTsHeader seqTs;

        // Send the frame to the player
```

```
        m_player.ReceiveFrame(&message); //TODO: In case frame consists of more than 1
packets, play them all together.
        m_segment_bytes += message.GetSize();
        m_totBytes += message.GetSize();

        message.RemoveHeader(mpegHeader);
        message.RemoveHeader(httpHeader);
        message.RemoveHeader(seqTs);

        receiverDumpFile << std::fixed << std::setprecision(4) <<
Simulator::Now().ToDouble(ns3::Time::S)
                                    << std::setfill(' ') << std::setw(16) <<  "id "
<< seqTs.GetSeq()
                                    << std::setfill(' ') <<  std::setw(16) <<  "tcp
" << message.GetSize()
                                    << std::endl;

        // Calculate the buffering time
        switch (m_player.m_state)
          {
        case MPEG_PLAYER_PLAYING:
          m_sumDt += m_player.GetRealPlayTime(mpegHeader.GetPlaybackTime());
          break;
        case MPEG_PLAYER_PAUSED:
          break;
        case MPEG_PLAYER_DONE:
          return;
        default:
          NS_FATAL_ERROR("WRONG STATE");
          }

        // If we received the last frame of the segment
        if (mpegHeader.GetFrameId()!= 0 && mpegHeader.GetFrameId() %
MPEG_FRAMES_PER_SEGMENT == 0)
        {
            m_segmentFetchTime = Simulator::Now() - m_requestTime;

            NS_LOG_INFO(Simulator::Now().GetSeconds() << " bytes: " << m_segment_bytes
                        <<" segmentTime: " << m_segmentFetchTime.GetSeconds()
                        <<" segmentRate: " << 8 * m_segment_bytes /
m_segmentFetchTime.GetSeconds());

            // Feed the bitrate info to the player
            AddBitRate(Simulator::Now(),
                8 * m_segment_bytes / m_segmentFetchTime.GetSeconds());

            Time currDt = m_player.GetRealPlayTime(mpegHeader.GetPlaybackTime());
            // And tell the player to monitor the buffer level
            LogBufferLevel(currDt);
            Time bufferDelay;

            uint32_t prevBitrate = m_bitRate;
            uint32_t nextRate = 8 * m_segment_bytes / m_segmentFetchTime.GetSeconds();
            CalcNextSegment(nextRate, m_bitRate, bufferDelay);

            if (prevBitrate != m_bitRate)
            {
                m_rateChanges++;
            }

            if (bufferDelay == Seconds(0))
            {
                std::cout <<"Buffer delay!" <<std::endl;
                Send();
            }
            else
            {
                m_player.SchduleBufferWakeup(bufferDelay, this); //if we want to
schedule a segment request after delaying the buffer.
            }
```

```
            std::cout   << Simulator::Now().GetSeconds() << " Node: " << m_id
                        << " newBitRate: " << m_bitRate << " oldBitRate: " <<
prevBitrate
                        << " estBitRate: " << GetBitRateEstimate() << " interTime: "
                        << m_player.m_interruption_time.GetSeconds() << " T: "
                        << currDt.GetSeconds() << " dT: "
                        << (m_lastDt >= 0 ? (currDt - m_lastDt).GetSeconds() : 0)
                        << " del: " << bufferDelay << std::endl;

            NS_LOG_INFO("==== Last frame received. Requesting segment " <<
m_segmentId);
            m_lastDt = currDt;
        }
    }

    void
    EvalvidClient::CalcNextSegment(uint32_t currRate, uint32_t & nextRate, Time &
delay)
    {
        nextRate = currRate;
        delay = Seconds(0);
    }

    void
    EvalvidClient::AddBitRate(Time time, double bitrate)
    {
        m_bitrates[time] = bitrate;
        double sum = 0;
        int count = 0;
        for (std::map<Time, double>::iterator it = m_bitrates.begin();
        it != m_bitrates.end(); ++it)
        {
            if (it->first < (Simulator::Now() - m_window))
            {
                m_bitrates.erase(it->first);
            }
            else
            {
                sum += it->second;
                count++;
            }
        }
        m_bitrateEstimate = sum / count;
    }

    void
    EvalvidClient::LogBufferLevel(Time t)
    {
        m_bufferState[Simulator::Now()] = t;
        for (std::map<Time, Time>::iterator it = m_bufferState.begin(); it !=
m_bufferState.end(); ++it)
        {
            if (it->first < (Simulator::Now() - m_window))
            {
                m_bufferState.erase(it->first);
            }
        }
    }

}
```

REFERENCES

[1] IETF RFC 2616: "Hypertext Transfer Protocol – HTTP/1.1", Fielding R. et al., June 1999.
[2] Stockhammer, T.: Dynamic Adaptive Streaming over HTTP { standards and design principles. In: Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, MMSys '11, pp.133{144. ACM, New York, NY, USA (2011)
[3] https://www.theguardian.com/media-network/media-network-blog/2013/mar/01/history-streaming-future-connected-tv
[4] http://www.onlinevideo.net/2011/05/streaming-vs-progressive-download-vs-adaptive-streaming/
[5] http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-Adaptive-Streaming-75195.aspx
[6] https://tech.ebu.ch/docs/techreview/trev_2011-Q1_adaptive-streaming_laukens.pdf
[7] Stockhammer, T.: Dynamic Adaptive Streaming over HTTP { standards and design principles.In: Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, MMSys '11, pp.133{144. ACM, New York, NY, USA (2011)
[8] J.-M. Jeong and J.-D. Kim. Effective bandwidth measurement for dynamic adaptive streaming over http. In Information Networking (ICOIN), 2015International Conference on, pages 375{378. IEEE,2015.
[9] http://mpeg.chiariglione.org/about
[10] Dmitri Jarnikov, and Tanır Özçelebi: Client intelligence for adaptive streaming solutions, Signal Processing: Image Communication Volume 26, Issue 7, August 2011
[11] https://www.harmonicinc.com/news-events/press-releases/read/harmonic-powers-first-public-mpeg-dash-trial-during-london-olympics/
[12] I. Sodagar, “The mpeg-dash standard for multimedia streaming over the internet,” IEEE Multimedia, vol. 18, no. 4, pp. 62–67, Oct. 2011.
[13] https://www.encoding.com/mpeg-dash/
[14] MPEG-DASH vs. Apple HLS vs. Microsoft Smooth Streaming vs. Adobe HDS". https://bitmovin.com/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/ 2015-03-29.
[15] MPEG-DASH, Reference number ISO/IEC 23009-1:2014, ISO International Standard, 2nd edition 2014-05-15
[16] Sideris, Markakis, Zotos, Pallis, Skiaris: MPEG-DASH users' QoE: The segment duration effect, DOI: 10.1109/QoMEX.2015.7148117
[17] Aloman, Iosif, Ciotirnae, Cano: Performance evaluation of video streaming using MPEG DASH, RTSP, and RTMP in mobile networks, DOI: 10.1109/WMNC.2015.12, Conference: 8th IFIP Wireless and Mobile Networking Conference (WMNC 2015)
[18] George, Kaiser, Pham and Krauss: Internet-Delivered Television using MPEG DASH: Opportunities and Challenges
[19] D.Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom,S.Parkvall, “LTE: the evolution of mobile broadband”,IEEE Communications Magazine, vol. 4, pp. 44–51, 2009.[Online].Available:http://dx.doi.org/10.1109/MCOM.2009.4907406
[20] 3GPP, “Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN),” Tech. rep. 25.913, http://www.3gpp.org
[21] E. Dahlman, S. Parkvall, J. Sk• old, and P. Beming,3G Evolution { HSPA and LTE for Mobile Broadband, 1st ed. Aca-demic Press, 2007
[22] AT&T Tech Channel (2011-06-13). "AT&T Archives : Testing the First Public Cell Phone Network". Techchannel.att.com
[23] Barnes, David M (May 1985). The Introduction of Cellular Radio to the United Kingdom. Vehicular Technology Conference, 1985. 35th. pp. 147–152
[24] Japanese Total Access Communication (JTAC), mobiledia.com
[25] GSM Doc 28/85 "Services and Facilities to be provided in the GSM System" rev2, June 1985
[26] ITU: 3G Definition, International Mobile Telecommunications (IMT) Cellular and Mobile Broadband Access for the 21st Century
[27] https://www.itu.int/osg/spu/imt-2000/technology.html#Cellular Standards for the Third Generation
[28] E. Dahlman, B. Gudmundson, M. Nilsson,and A. Skold,“UMTS/IMT-2000 based on wideband CDMA,”IEEE Communications Magazine,vol.36, no. 9, pp. 70-80, September 1998
[29] Guowang Miao; Jens Zander; Ki Won Sung; Ben Slimane (2016). Fundamentals of Mobile Data Networks. Cambridge University Press. ISBN 1107143217
[30] J. G. Proakis, Digital Communications, 3rd ed. New York: McGraw Hill, 1995

[31] Siemens (2004-06-10). "TD-SCDMA Whitepaper: the Solution for TDD bands" (PDF). TD Forum. pp. 6–9
[32] Junse Lee, Namyoon Lee, Francois Baccelli: Scaling Laws for Ergodic Spectral Efficiency in MIMO Poisson Networks, Cornell University Library, Submitted to IEEE Transactions on Information Theory, https://arxiv.org/abs/1608.06065v1
[33] S. Sesia, I. Toufik and M. Baker, Eds., LTE - The UMTS Long Term Evolution, 2nd ed., Wiley, 2011
[34] R. Kwan and C. Leung, "A Survey of Scheduling and Interference Mitigation in LTE," Journal of Electrical and Computer Engineering - Special issue on LTE/LTE advanced cellular communication networks, vol. 2010, no. 1, 2010
[35] Van Veen, B.D.; Buckley, K.M. (1988). "Beamforming: A versatile approach to spatial filtering" (PDF). IEEE ASSP Magazine. 5 (2): 4. doi:10.1109/53.665.
[36] 3GPP: Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception, Technical specification (TS) 36.101, Release 8
[37] 3GPP: UTRAN overall description (Release 11); Technical Specification Group Radio Access Network, December 2012
[38] 3GPP: General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access, Technical specification (TS) 23.404, Release 8
[39] 3GPP: “The Evolved Packet Core”, Frédéric Firmin, 3GPP MCC, http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core
[40] Koien, G.M., “Mutual entity authentication for LTE”, Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International, 4-8 July 2011, pp. 689-694, Istanbul, Turkey.
[41] 3GPP:Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2, Technical specification (TS) 36.300, Release 8
[42] https://www.tutorialspoint.com/lte/lte_network_architecture.htm
[43] 3GPP: Policy and charging control architecture, Technical specification (TS) 23.203, Release 7
[44] 3GPP: Technical Specification Group Services and System Aspects (2006), IP Multimedia Subsystem (IMS), Stage 2, TS 23.228
[45] CISCO: Mobility Management Entity Overview, MME Administration Guide, StarOS Release 20
[46] https://sites.google.com/site/lteencyclopedia/lte-network-infrastructure-and-elements#TOC-3.2-HSS-Home-Subscriber-Server-
[47] 3GPP: Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3, Technical specification (TS) 24.301, Release 8
[48] 3GPP: Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 Application Protocol (X2AP), Technical specification (TS)36.423, Release 8
[49] http://www.masterltefaster.com/lte/userplane.php
[50] 3GPP: TS 29.060 V6.9.0 (2005-06), 3rd Generation Partnership Project, 650 Route des Lucioles - Sophia Antipolis, Valbonne - FRANCE, 2005-06.
[51] 3GPP: Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification, Technical specification (TS)36.321, Release 8
[52] 3GPP: Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification, Technical specification (TS)36.323, Release 8
[53] 3GPP: Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification, Technical specification (TS)36.331, Release 8
[54] S. Palat and P. Godin, “Network Architecture,” in LTE –The UMTS Long Term Evolution :From Theory to Practice, S. Sesia, I. Toufik, and M. Baker, Eds. West Sussex, UK: John Wiley and Sons, 2009, pp. 23 –50.
[55] H. Ekstrom, “QoS control in the 3GPP evolved packet system,”IEEE Communications Magazine,vol.47,no.2,pp.76–83,2009.
[56] “Qualinet White Paper on Definitions of Quality of Experience”, Dagstuhl Seminar, 2012.
[57] Seufert M, Egger S, Slanina M, Zinner T, Hoßfeld T, Tran-Gia P. A survey on quality of experience of http adaptive streaming.Communications Surveys Tutorials, IEEE 2014;PP(99), doi:10.1109/COMST.2014.2360940.
[58] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino, “Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC,” in Proc. IEEE CCNC, Las Vegas, NV, USA, 2012, pp. 127–131.
[59] T. Zinner, T. Hoßfeld, T. N. Minhas, and M. Fiedler, “Controlled vs. uncontrolled degradations of QoE—the provisioning-delivery hysteresis in case of video,” inProceedings of the EuroITV Workshop: Quality of Experience for Multimedia Content Sharing,Tampere,Finland,June 2010

[60] E. Yaacoub and Z. Dawy, Fair Optimization of Video Streaming Quality of Experience in LTE Networks using Distributed Antenna Systems and Radio Resource Management, Journal of Applied Mathematics, Special Issue on Fair Optimization and Networks: Models, Algorithms, and Applications.
[61] Hoßfeld T, Heegaard P, Varela M, Moller S. Formal Definition of QoE Metrics, Qual User Exp (2016) 1: 2. doi:10.1007/s41233-016-0002-1
[62] E. Liotou, D. Tsolkas, N. Passas, “A roadmap on QoE metrics and models” IEEE ICT, 2016. DOI: 10.1109/ICT.2016.7500363
[63] https://www.techopedia.com/definition/9049/quality-of-service
[64] G. Piro, N. Baldo, and M. Miozzo, “An LTE module for the ns-3 network simulator,” in Proc. of the 4th Int. ICST Conf. on Simulation Tools and Techniques, 2011, pp. 415–422.
[65] NS3 Evalvid module by GERCOM - www.gercom.ufpa.br
[66] J. Klaue, B. Rathke, and A. Wolisz, "EvalVid - A Framework for Video Transmission and Quality Evaluation", 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, pp. 255-272, Urbana, Illinois, USA, September 2003.
[67] Dimitrios J. Vergados, Angelos Michalas, Aggeliki Sgora, and Dimitrios D. Vergados. "A fuzzy controller for rate adaptation in MPEG-DASH clients." In 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC), pp. 2008-2012. IEEE, 2014.
[68] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner, “Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming,” in IEEE International Workshop on Quality of Multimedia Experience (QoMEX), 2014, pp. 111–116.
[69] L. Merakos, E. Liotou, D. Tsolkas, N. Passas, “A Survey on Parametric QoE Estimation for Popular Services” Journal of Network and Computer Applications · January 2017 . DOI: 10.1016/j.jnca.2016.10.016