NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

GRADUATE PROGRAM
COMPUTER, TELECOMMUNICATIONS AND NETWORK ENGINEERING

MSc THESIS

Implementation of live HTTP Adaptive Video Streaming over Mininet

Konstantina P. Chatzieleftheriou
Gerasimos D. Christodoulou

Supervisor: **Lazaros Merakos,** Professor

ATHENS

FEBRUARY 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΜΗΧΑΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση ζωντανής προσαρμοστικής ροής βίντεο στο περιβάλλον Mininet

Κωνσταντίνα Π. Χατζηελευθερίου
Γεράσιμος Δ. Χριστοδούλου

Επιβλέπων: **Λάζαρος Μεράκος,** Καθηγητής

ΑΘΗΝΑ

ΦΕΒΡΟΥΑΡΙΟΣ 2020

MSc THESIS

Implementation of live HTTP Adaptive Video Streaming over Mininet

Konstantina P. Chatzieleftheriou
S.N.:EN2180009

Gerasimos D. Christodoulou
S.N.:EN2180011

SUPERVISOR: **Lazaros Merakos,** Professor

ADVISORY COMMITTEE: **Lazaros Merakos**, Professor
Nikolaos Passas, LTS Department of Informatics and Telecommunications NKUA
Stathes Hadjiefthymiades, Associate Professor

February 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση ζωντανής προσαρμοστικής ροής βίντεο στο περιβάλλον Mininet

Κωνσταντίνα Π. Χατζηελευθερίου
Α.Μ.: EN2180009

Γεράσιμος Δ. Χριστοδούλου
Α.Μ.: EN2180011

ΕΠΙΒΛΕΠΩΝ: **Λάζαρος Μεράκος,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Λάζαρος Μεράκος,** Καθηγητής
Νικόλαος Πασσάς, ΕΔΙΠ Τμήματος Πληροφορικής και Τηλεπικοινωνιών ΕΚΠΑ
Στάθης Χατζηευθυμιάδης, Αναπληρωτής Καθηγητής

Φεβρουάριος 2020

ABSTRACT

Video streaming is growing into a well-known technology for media transmission over the Internet. Dynamic Adaptive Streaming over HTTP (DASH) permits transmitting data streams to a user with the largest feasible bit rate in different bandwidth situations which is especially important for wireless networks. The purpose of this thesis is to analyze DASH technology as far as live video streaming is concerned, as well as to examine the Quality of Experience (QoE) on SDN networks, namely to understand the user's perspective. Thus, in this thesis, a virtual SDN network was developed in the Mininet environment to simulate DASH technology. In each experiment, one server transmitted a video live to a client while the throughput was changing by adding extra traffic to the network. At the same time, the Wireshark application was monitoring the transmitted packets from the server to the client, storing various parameters related to the network, based on which the user's QoE was calculated. Specifically, the impact of the added traffic on the quality of the broadcast video was examined, and as a result the impact in terms of QoE was measured. We concluded that the transmitted data is entirely connected with the existing traffic in the network and in particular the higher the traffic, the poorer the selected resolution. However, we observed that the resolution of the broadcast video would not always change immediately, as a result of the existence of the buffer at the video client.

The structure of this thesis is the following: Chapter 1 provides a detailed overview of the mobile networks' evolution from the first generation to the fifth one. Chapter 2 describes in detail the HTTP Adaptive Streaming (HAS) technology in terms of its architecture, its advantages and disadvantages. It also describes the architecture of DASH technology, its implementation and the additional advantages it offers. Chapter 3 is about QoE, the way in which it is calculated through specific models, and its influence on HTTP video streaming technology. All the necessary programs in order to conduct the experiments presented in this thesis, as well as any necessary settings, can be found in Chapter 4. Chapter 5 provides an in-depth analysis of the design of the experiments, during which a decrease in the quality of the user's experience was observed. It also presents the obtained results. Finally, Chapter 6 summarizes the conclusions of this thesis and discusses future work topics.

SUBJECT AREA: Communications Networks

KEYWORDS: Software Defined Networks, Quality of Experience, Video, Monitoring, SDN Controller, Dynamic Adaptive Streaming over HTTP, Mininet

ΠΕΡΙΛΗΨΗ

Η μετάδοση βίντεο εξελίσσεται σε μια διαδεδομένη τεχνολογία για τη μετάδοση δεδομένων μέσω του Διαδικτύου. Η τεχνολογία Dynamic Adaptive Streaming over HTTP (DASH) επιτρέπει τη μετάδοση ροών δεδομένων σε ένα χρήστη με το μεγαλύτερο εφικτό ρυθμό δεδομένων κάτω από διαφορετικά εύρη ζώνης, κάτι ιδιαίτερα σημαντικό για τα ασύρματα δίκτυα. Σκοπός αυτής της διπλωματικής εργασίας είναι η ανάλυση της τεχνολογίας DASH όσον αφορά την ζωντανή μετάδοση βίντεο, καθώς και την εξέταση της ποιότητας εμπειρίας σε SDN δίκτυα από την πλευρά του χρήστη. Γι' αυτό τον σκοπό αναπτύχθηκε ένα εικονικό SDN δίκτυο στο περιβάλλον Mininet, ούτως ώστε να προσομοιωθεί η τεχνολογία DASH. Σε κάθε πείραμα, ένας server μετέδιδε ζωντανά ένα βίντεο σε έναν client, ενώ την ίδια στιγμή μεταβαλλόταν το throughput που απολάμβαναν και οι δύο, προσθέτοντας επιπλέον δικτυακή κίνηση στο δίκτυο. Ταυτόχρονα, η εφαρμογή Wireshark παρακολουθούσε τα πακέτα που μεταδίδονταν από τον server στον client, αποθηκεύοντας διάφορες παραμέτρους του δικτύου, βάση των οποίων υπολογίστηκε η ποιότητα εμπειρίας του χρήστη. Συγκεκριμένα, εξετάστηκε η επιρροή που είχε το μέγεθος της προστιθέμενης δικτυακής κίνησης στην ποιότητα του μεταδιδόμενου βίντεο κι ως αποτέλεσμα στην ποιότητα εμπειρίας του χρήστη. Συμπεράναμε ότι τα μεταδιδόμενα δεδομένα είναι εξολοκλήρου συνδεδεμένα με την δικτυακή κίνηση του δικτύου και μάλιστα πιο συγκεκριμένα, όσο μεγαλύτερη είναι η δικτυακή κίνηση, τόσο χειρότερη είναι η ποιότητα αναπαραγωγής του video. Παρόλα αυτά, παρατηρήσαμε ότι η ποιότητα του μεταδιδόμενου βίντεο δεν άλλαζε πάντα άμεσα, λόγω της ύπαρξης του buffer του βίντεο του τελικού χρήστη.

Η δομή της παρούσας διπλωματικής εργασίας είναι η εξής: Στο κεφάλαιο 1 παρουσιάζεται αναλυτικά η εξέλιξη των κινητών δικτύων από την πρώτη γενιά έως και την πέμπτη. Στη συνέχεια, στο κεφάλαιο 2 περιγράφεται αναλυτικά η τεχνολογία HTTP Adaptive Streaming (HAS) ως προς την αρχιτεκτονική, τα πλεονεκτήματα και τα μειονεκτήματά της. Επίσης περιγράφεται η αρχιτεκτονική της τεχνολογίας DASH, η εφαρμογή της και τα επιπλέον πλεονεκτήματα που προσφέρει. Το κεφάλαιο 3 αφορά την έννοια της ποιότητας εμπειρίας του χρήστη, τον τρόπο υπολογισμού της μέσω μοντέλων υπολογισμού καθώς και την επιρροή που έχει στην τεχνολογία HTTP video streaming. Όλα τα απαραίτητα προγράμματα για την διεξαγωγή των πειραμάτων που παρουσιάζονται στην παρούσα εργασία, όπως και οι ρυθμίσεις αυτών μπορούν να βρεθούν στο κεφάλαιο 4. Το κεφάλαιο 5 αναλύει σε βάθος τις σχεδιαστικές προδιαγραφές των πειραμάτων, κατά την διάρκεια των οποίων παρατηρήθηκε μείωση της ποιότητας εμπειρίας του χρήστη, ενώ παρουσιάζει και τα αποτελέσματα που προέκυψαν. Τέλος, το κεφάλαιο 6 συνοψίζει τα συμπεράσματα της παρούσας διπλωματικής εργασίας και αναφέρει επιγραμματικά ανοιχτά θέματα για μελλοντική έρευνα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δίκτυα Επικοινωνιών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Δικτύωση Βασισμένη στο Λογισμικό, Ποιότητα Εμπειρίας, Υπηρεσίες βίντεο, Dynamic Adaptive Streaming over HTTP, Mininet

Θέλουμε να ευχαριστήσουμε για την ανεκτίμητη βοήθεια, και στήριξή τους
τις οικογένειές μας καθώς και τους φίλους μας.

CONTENTS

PREFACE 13

1. INTRODUCTION: NETWORK DEVELOPMENT FROM 1G TO 5G 14

1.1 THE 1ST GENERATION OF MOBILE NETWORKS 1G 14

1.2 THE 2ND GENERATION OF MOBILE NETWORKS 2G 15

1.3 THE 3RD GENERATION OF MOBILE NETWORKS 3G 16

1.4 LONG-TERM EVOLUTION (LTE) 17

1.4.1 LTE development 18
1.4.2 More about LTE 18
1.4.3 LTE-A benefits 19
1.4.4 LTE architecture 22
1.4.5 OFDM - OFDMA - SC-FDMA technologies 22
1.4.6 LTE – QOS 24

1.5 THE 5TH GENERATION OF MOBILE NETWORKS 5G 24

2. HTTP ADAPTIVE STREAMING 26

2.1 INTRODUCTION 26

2.2 BENEFITS 26

2.3 DYNAMIC ADAPTIVE STREAMING OVER HTTP (DASH) 27

2.3.1 Additional advantages 28
2.3.2 Implementation 28

2.4 MEDIA PRESENTATION DESCRIPTION (MPD) 29

2.4.1 Segment referencing schemes 29

3. QUALITY OF EXPERIENCE (QOE) 31

3.1 INTRODUCTION 31

3.2 QoE FACTORS 31

3.3 QoE VS QoS – THE IQX HYPOTHESIS 32

3.4 QoE MEASURMENTS 32

3.5 QoE MANAGEMENT 33

3.6 HTTP VIDEO STREAMING INFLUENCED BY QoE FATORS 34
3.6.1 Initial delay 34
3.6.2 Stalling 34

4. ENVIRONMENT SETUP 35

4.1 SYSTEM REQUIREMETS 35
4.1.1 Operating system (OS) 35
4.1.2 Java 35
4.1.3 SDN controller deployment 35
4.1.4 Mininet 38
4.1.5 Mininet’s graphical user interface, MiniEdit 39
4.1.6 VLC media player 41
4.1.7 FFmpeg 42
4.1.8 Dynamic adaptive streaming over HTTP (DASH) 42
4.1.9 Project on advanced content (GPAC) 43
4.1.10 Wireshark 43

4.2 SERVER SETUP 47
4.2.1 Making requests in Node.js 48
4.2.2 Proposed server implementation 49

5. MININET EXPERIMENTS 56

5.1 BASIC EXPERIMENT 56
5.1.1 Preparation 56
5.1.2 Basic simulation 59

5.2 TRAFFIC GENERATOR 62

5.3 PSNR 67
5.3.1 PSNR calculation 68

6. CONCLUSION AND FUTURE WORK 71

ABBREVIATIONS - ACRONYMS 72

REFERENCES 77

LIST OF FIGURES

Figure 1: 1G Architecture 14
Figure 2: Timeline 16
Figure 3: 3G Architecture 17
Figure 4: LTE Development Timeline 18
Figure 5: Carrier Components 19
Figure 6: LTE-A base stations 20
Figure 7: Heterogeneous Networks 21
Figure 8: Coordinated Multi-Point transmission and reception 21
Figure 9: LTE Architecture 22
Figure 10: OFDM 23
Figure 11: FDM vs OFDM 23
Figure 12: OFDMA 23
Figure 13: 5G Architecture 25
Figure 14: Adaptive Streaming Overview 26
Figure 15: HTTP in a nutshell 26
Figure 16: HTTP Adaptive Streaming (HAS) 27
Figure 17: HTTP Dynamic Adaptive Streaming Timeline 28
Figure 18: Media Presentation Description (MPD) File 29
Figure 19: SegmentBase Arrangement 30
Figure 20: SegmentList Arrangement 30
Figure 21: SegmentTemplate Arrangement 30
Figure 22: QoE Factors 31
Figure 23: The IQX hypothesis diagram 32
Figure 24: QoE Modeling 33
Figure 25: QoE Management Entities 33
Figure 26: OpenDaylight Login Page 36
Figure 27: OpenDaylight Main Page 37
Figure 28: Topology Tab 37
Figure 29: Nodes Tab 38
Figure 30: Basic Topology 39
Figure 31: Pingall Output 39
Figure 32: MiniEdit main page 40
Figure 33: VLC Preferences (1) 41
Figure 34: VLC Preferences (2) 42
Figure 35: Main Page of Wireshark 44

Figure 36: Needed Columns..44
Figure 37: Column Preferences Option 45
Figure 38: Wireshark Preferences (1)..45
Figure 39: New Columns 45
Figure 40: Wireshark Preferences (2)..46
Figure 41: Wireshark Preferences (3)..46
Figure 42: Wireshark Main Page in the end...47
Figure 43: example.js output 48
Figure 44: The procedure described...49
Figure 45: Terminal window...56
Figure 46: bash.sh execution...58
Figure 47: Different versions of the video, mpd file, different segments 59
Figure 48: Custom topology...59
Figure 49: Mininet preferences...59
Figure 50: xterm h1 60
Figure 51: xterm h2 (1) 60
Figure 52: xterm h2 (2) 60
Figure 53: Video Playback...60
Figure 54: Output of the server terminal...61
Figure 55: Updated topology 62
Figure 56: Wireshark Conversation window 63
Figure 57: Download Rate (Mbps) Vs Resolution 1Mbps 63
Figure 58: Download Rate (Mbps) Vs Time 1Mbps 64
Figure 59: Download Rate (Mbps) Vs Resolution 3Mbps 64
Figure 60: Download Rate (Mbps) Vs Time 3Mbps 65
Figure 61: Download Rate (Mbps) Vs Resolution 5Mbps 65
Figure 62: Download Rate (Mbps) Vs Time 5Mbps 66
Figure 63: Download Rate (Mbps) Vs Resolution 10 Mbps 66
Figure 64: Download Rate (Mbps) Vs Time (sec) 10Mbps 67
Figure 65: PSNR (Average, Minimum, Maximum)...70

LIST OF TABLES

Table 1: Used tools..46
Table 2: Provided resolutions according to selected bandwidth 67
Table 3: PSNR results 69

PREFACE

The current MSc thesis was conducted in the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens, and more specifically under the support of the Communication Networks Laboratory of the Telecommunications and Signal Processing sector. The thesis started being conducted in June 2019 and ended in February 2020. Professor Lazaros Merakos and Dr. Eirini Liotou – to whom we owe special thanks - were the supervisors of this MSc Thesis.

1. INTRODUCTION: NETWORK DEVELOPMENT FROM 1G TO 5G

1.1 THE 1ST GENERATION OF MOBILE NETWORKS 1G

Mobile telecommunication networks have become increasingly more popular in the past few years due to the fast reformation from 1G to 5G mobile technology. Mobile telecommunication networks went through several stages of transformation until they reached their present form and will continue to evolve in the future. Each of the various evolution stages is called a generation. Hence, the first network is called 1st generation network – 1G.

The 80s is considered a revolutionary period due to the creation of 1G networks. The architecture of this model is based on Cellular Networks. Figure 1 illustrates this architecture. The initial theory divides the geographical area into the cells [1]. Cells have two characteristics: their size and shape. These two factors depend on general and geographical characteristics of the area. Examples of such characteristics are the existence of high barriers such as tall buildings as well as the possibility that the area is agricultural or urban. What is of high importance is the fact that each cell has a frequency range. Neighboring cells do not share the same frequency in order to avoid interference effects. [2]

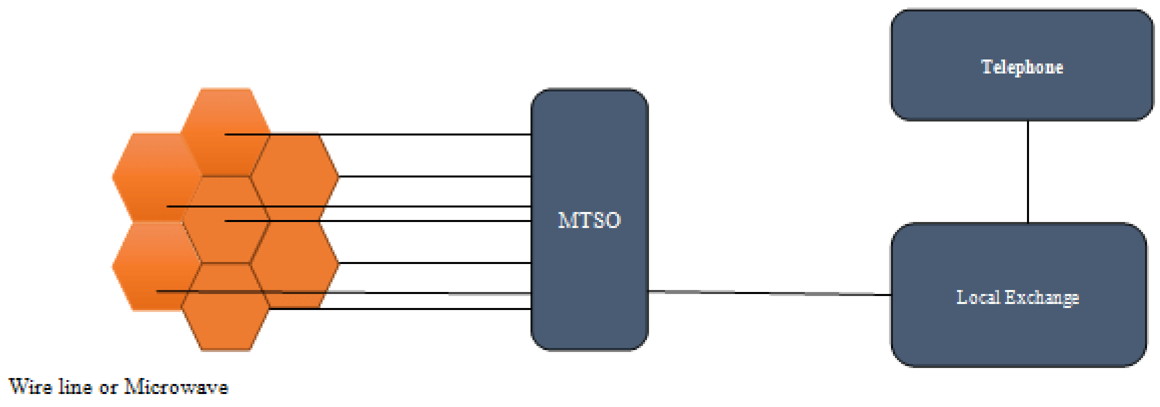


Figure 1: 1G Architecture

The first commercial cellular network came out in Japan in 1979 by Nippon Telegraph and Telephone (NTT). It was used for the first time in the metropolitan area of Tokyo. Finally, the whole Japanese population had outreach to cellular networks during a time period of 5 years. As for some technical aspects of 1G networks, the systems where purely analogue and functioned primarily in 150 MHz frequency.

Another point of interest in the history of cellular networks was the first mobile phone invented in 1972 by the American innovator Martin Cooper [3]. The device he invented was called "Motorola DynaTAC 8000X". It was really heavy and bulk for modern standards as it was 33 cm in height and weighted approximately a pound [4].

1G networks were undeniably revolutionary for the technology sector. However due to the complexity of human everyday needs, some problems came to the surface. Problems this first generation could not solve. For example, it was impossible to continue the call in a transition from one cell to another. This was restrictive for communication purposes as it would not allow for the user to move while using the phone. Another problem was the low system performance as the number of users who could communicate simultaneously was extremely limited. This was due to the fact that the available bandwidth was not enough. In general, the first systems did not leave any room for optimization and application of techniques such as compression and coding of information [5]. All the above, presuppose the use of digital signals [6].

1.2 THE 2ND GENERATION OF MOBILE NETWORKS 2G

The first generation networks undeniably belong to the past. The second generation networks did not follow the same course. Many of their features were taken into consideration in order to design the third generation.

The second generation networks used methods of digital signal modulation. The users were categorized according to Time Division Multiple Access (TDMA) [7] or Code Division Multiple Access (CDMA). That is in contrast to the first generation networks which transmitted analogue signals. Furthermore, in the second generation the separation of the users in simultaneous wireless media access was conducted by Frequency Division Multiple Access (FDMA).

There are four major standards for second generation networks. These are the Global System for Mobile (GSM) communications, the Digital AMPS (DAMPS), the Code Division Multiple Access (CDMA) IS-95 as well as the Personal Digital Cellular (PDC).

The GSM is by far the most successful and widespread system of this generation. The GSM was created by the need for a global system that would serve European citizens regardless of their country. The European Technical Standards Institute (ETSI) [8] was positioned as the responsible organization for the surveillance of this endeavor. This system became widely known. It had 400 mobile telecommunication networks in 140 countries, with over 350 million users. Finally, it may have started as a European system but it was adopted globally. Nevertheless, in 2001 the northamerican community for TDMA decided to adopt the system Wideband CDMA (WCDMA) which was defined by the Third Generation Partnership Project (3GPP).

The GSM started to function in 1990 in the bandwidth of 900MHz. The International Telecommunications Union (ITU) provided a pair of frequencies. The first one being from 890 to 915 MHz and the second from 935 to 960 MHz. The whole system was named GSM 900 or else known as Standard GSM.

During the year 1991, the system DCS 1800 was developed. In DCS 1800 the structure of GSM 900 is preserved. However, different pairs of frequencies are used: 1710 -1785 MHz (Uplink) and 1805 -1880 MHz (Downlink). The DCS 1800 was renamed to GSM 1800 during the 1990s in order to enforce the global use of GSM.

The GSM 1900 is used in various countries of the Americas. The structure of GSM 900 is again retained, with different pairs of frequencies: 1850 to 1910 MHz Up link and 1930 to 1990 MHz Down link. The E-GSM was defined by the European Radiocommunication Community during the end of the 90s in order to replace the classic GSM 900 preserving the its structure and increasing the bandwidth: 880 - 915 MHz Uplink and 925 - 960 MHz Downlink. Thus, mobile telephone networks were able to increase their storage space and cover the needs of the increasingly higher client traffic.

The technical features of the network architecture are divided into three parts:

1) The Mobile Station

2) The Base Station Subsystem

3) The NSS – Network Switching Subsystem

To conclude, the second generation networks made breakthroughs providing multiple new services. Firstly, it was the first time that a limited access to the Internet was allowed. Secondly, users could exchange written messages with each other. These messages are in fact the now well known, yet decreasing in popularity, Short Messaging Service (SMS) [9].

1.3 THE 3RD GENERATION OF MOBILE NETWORKS 3G

Third generation mobile networks showed the way for more and more services (Figure 2). It is a fact that their predecessors did not have the capability to offer any of those services. Some of those services are the VoIP (Voice over Internet Protocol) service as well as other services using the mobile phone.

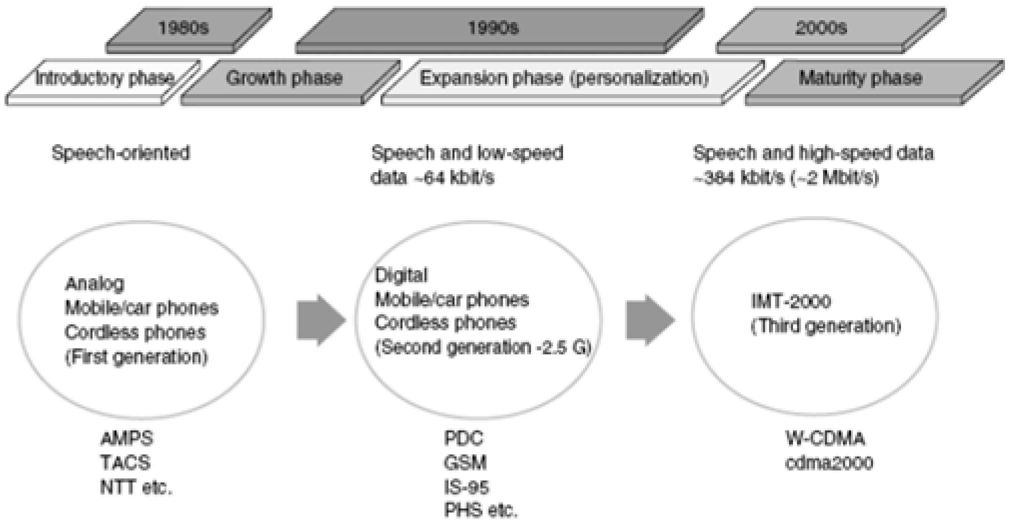

1980s
1990s
2000s
Introductory phase
Growth phase
Expansion phase (personalization)
Maturity phase
Speech-oriented
Speech and low-speed data ~64 kbit/s
Speech and high-speed data ~384 kbit/s (~2 Mbit/s)
Analog Mobile/car phones Cordless phones (First generation)
Digital Mobile/car phones Cordless phones (Second generation -2.5 G)
IMT-2000 (Third generation)
AMPS TACS NTT etc.
PDC GSM IS-95 PHS etc.
W-CDMA cdma2000

Figure 2: Timeline

As a result, second generation networks develop and the prototype cdma2000 was created and replaced CDMA, the Wideband-CDMA (W-CDMA) also known as Universal Mobile Telecommunication System (UMTS), a continuation of GSM, IS-136 and PDC. The main goal of the third generation networks is to offer services in any place any given moment in time. This means that a user who uses the network is capable of being serviced regardless of whether his/her geographical area has 3G signal. These services are Internet services that combine image with audio in high transmittance rates. It is important to note that the 3rd generation systems that prevailed in Europe was the UMTS, in North America the CDMA2000 and in China the TD-SCDMA. The UMTS (Universal Mobile Telecommunications Service) is an idea that started in 1996 as an upgrade of the GSM system.

It is backwards compatible with the 2G systems GSM, GSM, IS-136 and PDC as well as the 2.5G systems GPRS and EDGE. It uses the spread spectrum technique and requires a minimum band width of 5MHz. The transmit rates can reach and exceed 16 Mchip/sec/user. In the following, the architecture of UTMS is depicted. UTMS is an extension of the General Packed Radio Service (GPRS).

The CDMA2000, also known as IS-2000, is a technology for the development of CDMA One/IS – 95 3rd generation services (Figure 3). The evolutionary track of CDMA2000 was designed so that the investment and the effect on the user network is minimized without interruption of services for the final user. That is achieved through a backward and forward compatibility, reusing hardware, in-band migrating and setup of a hybrid network. This unique characteristic of CDMA2000 technologies offered to the user a significant time-to-market advantage against of the 3G technologies [9].

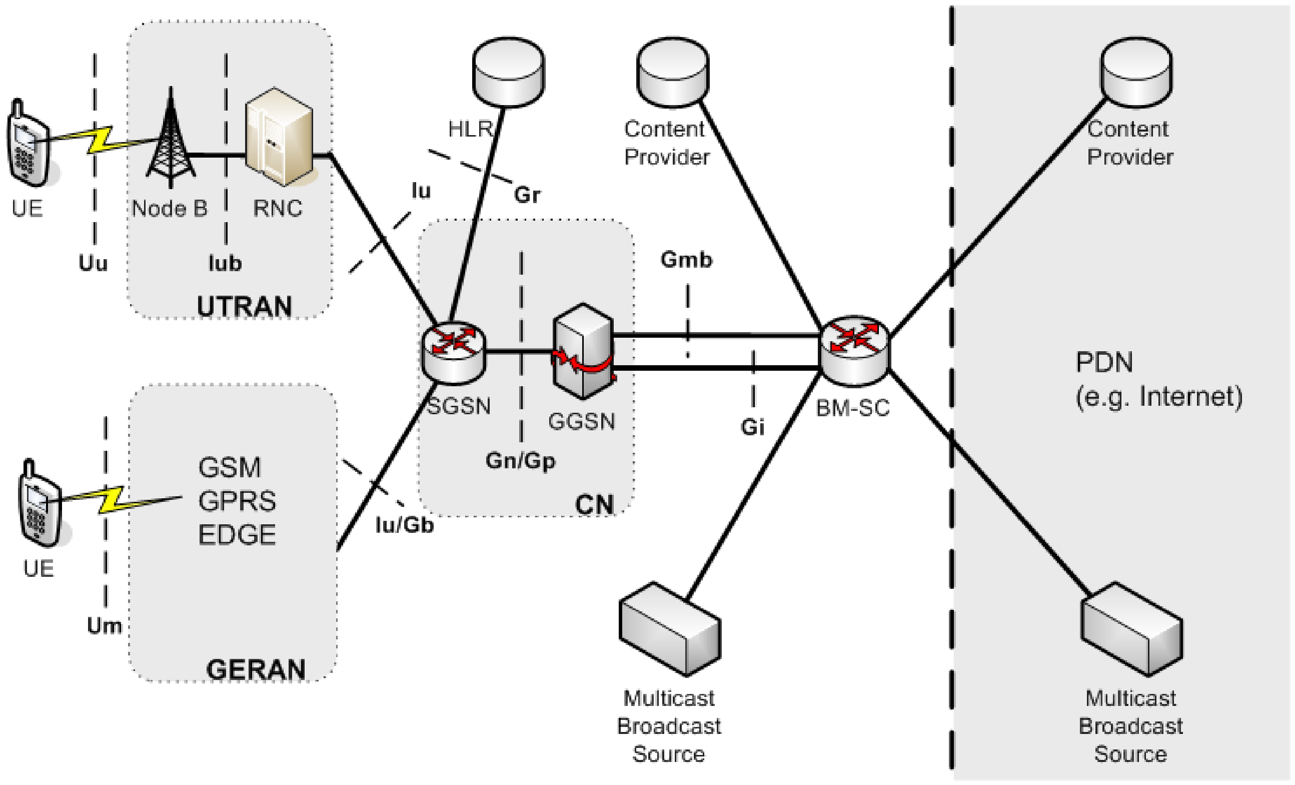


Figure 3: 3G Architecture

As it was mentioned above, the main goal was to use a large number of apps using the beginning of the coordinate system in a way possible. That cannot be achieved by 3G. The coverage everywhere on the planet is the desirable goal but unfortunately high data transmission speeds are not available everywhere. In addition, high speeds are not available in high movement subscriber speeds. Therefore, the rate is confined to about 380 Kbits/sec, while for slower speed it can reach 2Mbits/sec.

The TD-SCDMA was implemented by the China Academy of Communications Technology (CATT), together with the Siemens Information and Communication Mobile Group (ICMobile). The protocol is an alternative 3G technology and in 2003 the first device using it was presented.

Their view for 3G networks was extremely disappointing. Despite the fact that they include many upgrades in matters of security compared to 2G, their main goal for the domination of a global prototype was not achieved, with the exception of America where three incompatible systems were developed. The data transmission rates were not the expected, as voice cannot be transferred through the IP. Unfortunately, the 3G network did not respond to the promises of its manufacturers. However, this was the initial and trial version of the 3G experience. Lastly, the coming 4G will not only accomplish transmission of packets (and not circuits as in 3G) but also all its elements will be digital.

1.4 LONG-TERM EVOLUTION (LTE)

The Long-Term Evolution (LTE) is a standard for wireless broadband communication for mobile devices and data terminals. It is based on GSM / EDGE and UMTS / HSPA technologies and it was developed by 3GPP. Its first release was announced in Release 8, with minor improvements added in Release 9 later. The rapid increase in the use of data services along with the development of data applications, such as MMOG (Multimedia Online Gaming), mobile TV, Web 2.0 and streaming, has prompted 3GPP to work on LTE towards the 4th generation networks. However, while LTE is often referred as 4G LTE & Advance 4G, it does not meet the technical criteria of a 4G wireless network as they have been set by ITU-R. Yet, due to market demand and the significant progress that WiMAX, Evolved High Speed Packet Access and LTE standards have brought in 3G technologies, the ITU later decided that LTE and the other technologies mentioned before can be referred as 4G technologies. The LTE Advanced standard officially meets ITU-R specifications and can be considered IMT-Advanced.

1.4.1 LTE development

The development of the LTE standard did not take place within a short period of time. The creation of LTE was held in 2000 with the presentation of UMTS. The work over its specifications was done in 2004, then it was approved in 2008 and first presented in 2010 (Figure 4).

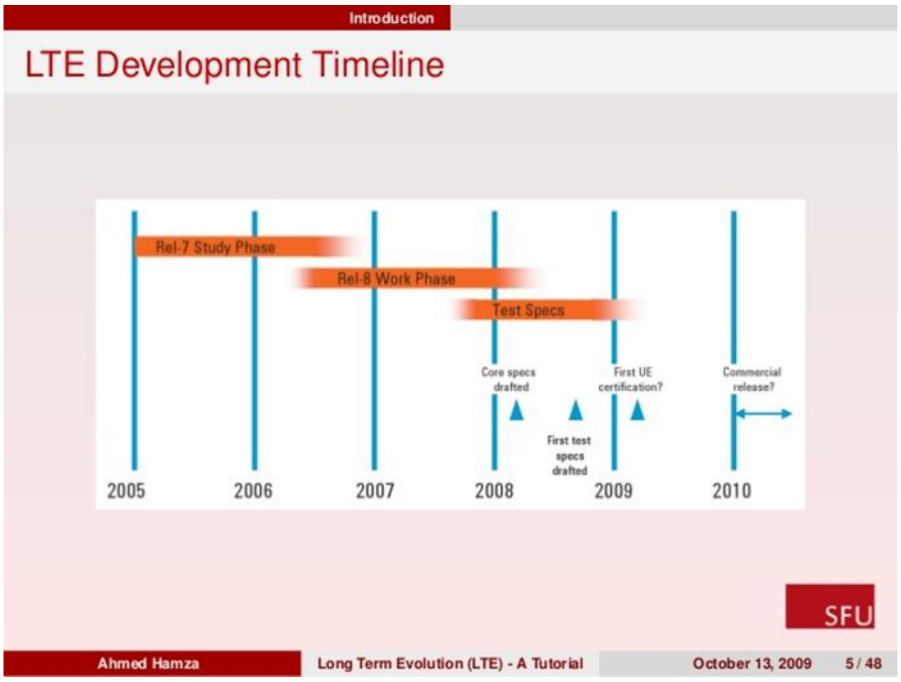


Figure 4: LTE Development Timeline

1.4.2 More about LTE

LTE was originally developed to achieve a higher transmission rate, specifically 300Mbps downlink and 75 Mbps uplink (maximum values). It is the ideal technology to support high data rates for services such as VoIP, multimedia streaming, videoconferencing.

It uses Time Division Duplex (TDD) and Frequency Division Duplex (FDD). In the FDD uplink and downlink, transmissions use different frequencies, while at TDD both transmissions use the same carrier and are separated by time. It supports flexible band carrier ranges together with both FDD and TDD. In an LTE system, designed with a gradient bandwidth of 1.4 MHz to 20 MHz, the bandwidth choice depends on the frequency band and the amount of the available spectrum. Moreover, all LTE devices must support Multiple Input Multiple Output (MIMO) transmissions, allowing the base station to transmit several data streams to the same player at the same time. All interfaces between network nodes in LTE are IP based, including the connection of base stations. Furthermore, Quality of Service (QoS) mechanisms have been standardized in all interfaces to ensure that voice calls, which require constant delay and bandwidth zone, can always be satisfied. It works collaboratively with existing GSM / EDGE / UMTS systems, 2G and 3G spectrum or new spectrum and it also supports handover and roaming on existing mobile networks. LTE has the ability to handle high speed moving devices such as the mobile of a user travelling by car. The network architecture is IP-based and is called Evolved Packet Core (EPC). It was designed to replace the GPRS Core Network and supports handovers for voice and data at base stations even with older network technology, such as GSM, UMTS and CDMA2000. One benefit of this architecture is its simplicity, which results in lower operating costs (for example, each the E-UTRA cell supports up to four times the voice or data capacity of HSPA).

1.4.3 LTE-A benefits

LTE-A is the latest and most advanced technology for LTE. From now on all subsequent improvements will be called LTE-A. LTE-A speeds reach 1 Gbps downlink and 500 Mbps uplink and have additional LTE-like mechanisms such as carrier aggregation and relaying.

According to 3GPP in the release for LTE-A specs we are able to have downlink speeds of 1 Gbps and uplink speeds of 500 Mbps. Additionally, less than 5 ms RTT is also reachable (Round Trip Time). The benefits that LTE-A provides are the following:

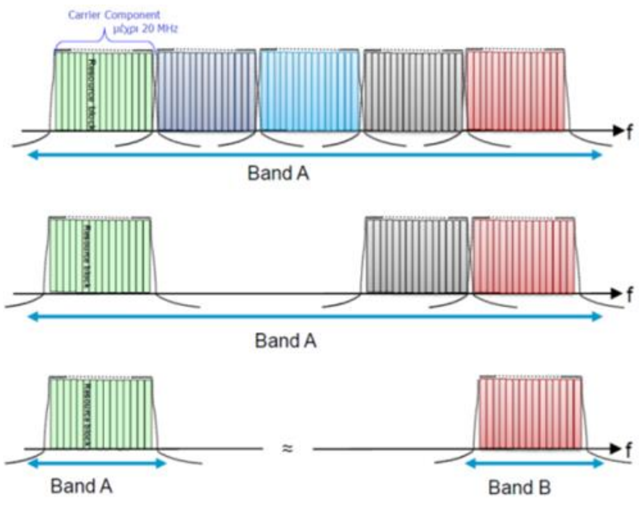


Figure 5: Carrier Components

1. Carrier Aggregation

In this component there is an increase of the frequency range from 20MHz that we used to have in LTE, to 100MHz with Carrier Aggregation technique, as it is very difficult to find 100MHz bands in a row that are not in use. LTE-A is looking for five (maximum) tracks that call them Carrier Components of different frequencies which uses them as if they were one (Figure 5). These five tracks can exist in 3 different ways:

• Intra Band Contiguous Allocation

In this way the tracks could be continuous, however LTE-A will still see them as separate tracks because it can use up to 20MHz.

• Intra Band Non Contiguous Allocation

In this way the pieces could be within the same spectrum length. That is to say they are in the immediate vicinity (in terms of frequency)

• Inter Band Non Contiguous Allocation

Finally, the tracks could be spaced too far apart, so as to belong to another band. However, LTE-A can find them and consider them as one.

2. Advanced MIMO techniques

MIMO stands for Multiple Input Multiple Output and it is the technique in which multiple antennas are established. In LTE-A base stations have up to 8 antennas and terminals, and up to 4 antennas.

As far as how multiple antennas can be used, there are three possibilities:

• Cooperative MIMO

In this case a terminal is on the border of 3 base stations. Each station uses 2 of its 8 antennas to transmit the signal. The terminal receives the same, yet weak signal 6 times because it is far from all the 3 base stations. In LTE-A the terminal is able to exploit these copies and compose a signal which is of very good quality (Figure 6).

• MU-MIMO (Multiple User MIMO)

In this case a base station can send 2 different signals to 2 users so as to provide more information.

• SU-MIMO (Single User MIMO)

In this case a base station can send the same signal 4 times from its 4 antennas in order to make it even more powerful with fewer errors. Therefore, it either achieves faster speeds or sends 4 different signals where the one will be a continuation of the other.

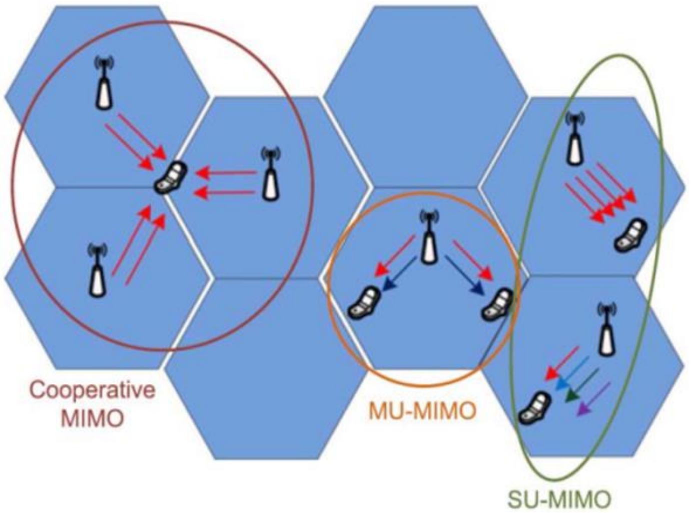


Figure 6: LTE-A base stations

3. Heterogeneous Networks and eICIC (enhanced Inter-Cell Interference Coordination)

Generally, there are many interference problems. In order to address this problem we use eICIC (enhanced Inter-Cell Interference Coordination) where base stations can communicate with each other and coordinate:

• In Time

It is the most common method used. In this method there are the Almost Blank Subframes (ABSFs), as they are called, which leave the macro cell deliberately empty and inform the femto cell to send to them exclusively so that there is no interference.

• In Frequency

In this case the stations understand the user who is having the problem and give him another frequency (in the same bandwidth) to communicate with no problem.

• In Power

In this case the base stations communicate after the problem and the femto cell emission power is reduced which either way has always the best signal and as a result its power loss does not cause any problems (Figure 7).

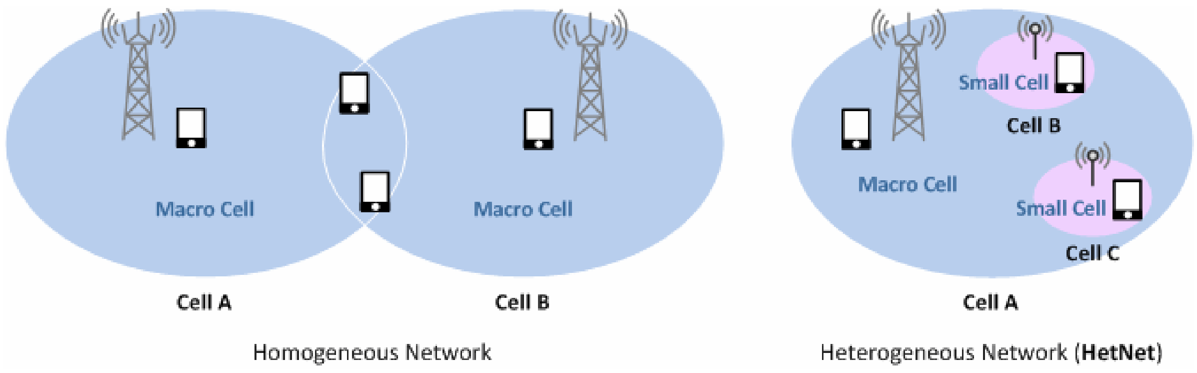


Figure 7: Heterogeneous Networks

4. Broadcasting

In many cases, terminals can be located between buildings either within the subway or generally at a point where they cannot be covered by a cell. This is where the Relay Nodes come in. They are meant to capture the signal from a cell and simply relay it to cover those areas.

5. Coordinated Multi-Point transmission and reception (CoMP)

Coordinated multi-transmission and reception is intended to cover users who are on the edge of the cell and are not satisfied with any signal from any cell in the area (Figure 8). This is similar to MIMO in a few words, but we could say it is a Distributed MIMO as the base stations talk to each other to decide what they will send.

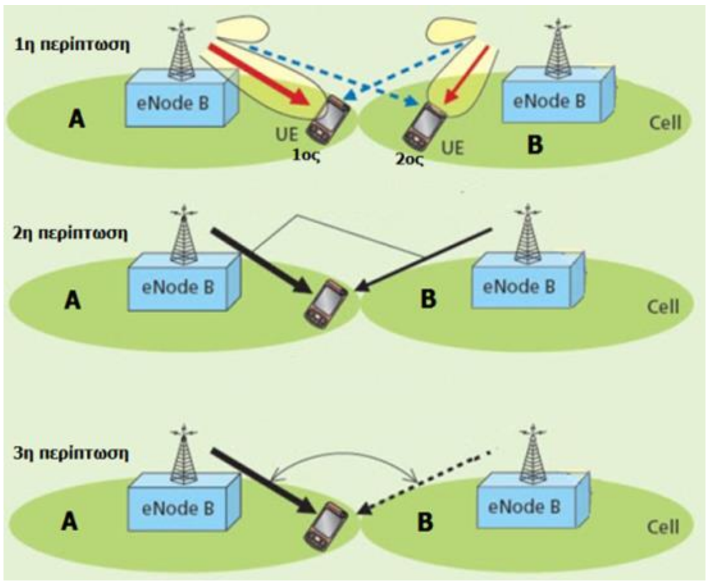


Figure 8: Coordinated Multi-Point transmission and reception

1.4.4 LTE architecture

As it can be observed from the image below, LTE holds data from UMTS. UMTS had UTRAN and LTE has Enhanced UTRAN (E-UTRAN). Additionally, UMTS had the B Nodes which were the base stations. LTE has the Enhanced Node B (eNodeB). The UE is the User Equipment, in other words the user terminal. The MME (Control Plane) / SAE (User Plane) Gateways are respectively MSC and SGSN. What is changing rapidly is the X2. X2 is an interface that allows eNodeBs to speak directly to each other, something that has not been the case until now. Finally, there is the S1 interface for the communication between the eNodeB and the Gateway (Figure 9).

Its main operating parameters are:

- Frequency Range: Uses UMTS FDD bands and UMTS TDD bands
- Channel Bandwidth: 1.4 MHz, 3 MHz, 5 MHz, 15 MHz, 20 MHz
- Modulation Schemes: QPSK, 16QAM, 64QAM for downlink and uplink
- Multiple Access: OFDMA for downlink and SC-FDMA for uplink
- MIMO: A technique in which we utilize multiple antennas for better signal quality
- Peak Data Rate: From 150-300 Mbps for downlink and 75 Mbps for uplink.

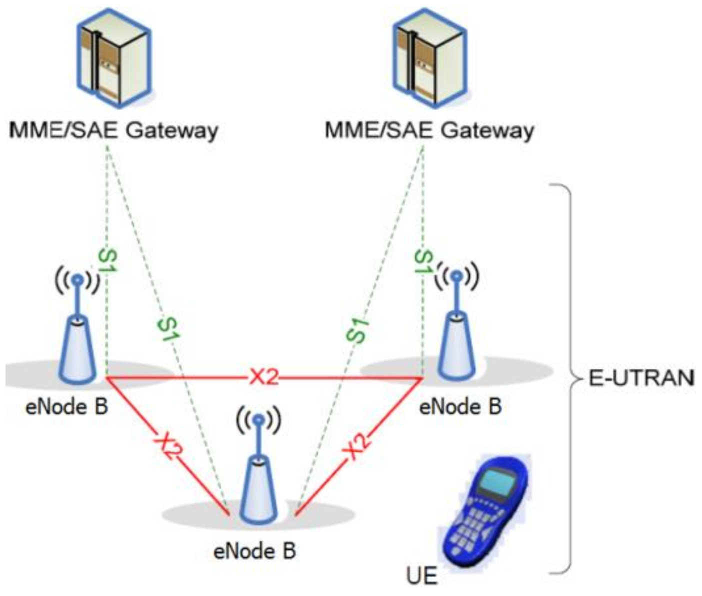


Figure 9: LTE Architecture

1.4.5 OFDM - OFDMA - SC-FDMA technologies

Orthogonal Frequency-Division Multiplexing (OFDM) technology is an encoding method of digital data to multiple frequency carriers (Figure 10). The basic natural LTE downlink resource can be interpreted as a time-frequency framework.

OFDM slots the time and splits the frequency. A user uses all the bits of frequency. Also, OFDM provides more frequencies in the same band than FDM, because the maximum of one pulse is at the minimum of another. At each different frequency track, a different track is transmitted. As a result, if one piece is affected only a small part of the transmission is affected rather than the whole transmission which can withstand some interference (Figure 11).

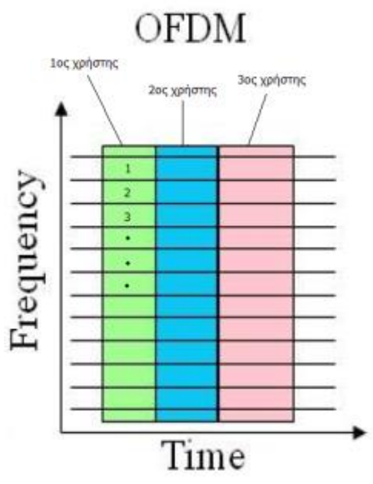


Figure 10: OFDM

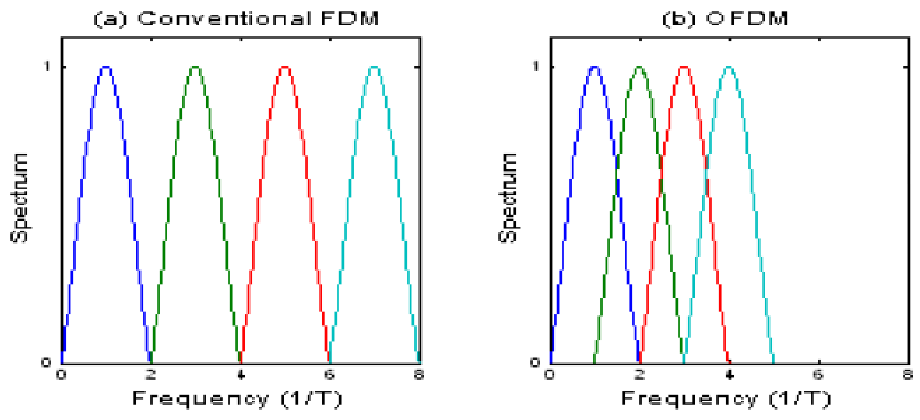


Figure 11: FDM vs OFDM

In OFDMA on the other hand, each slot does not have the same duration but fluctuates depending on the user. In addition, each user only gets a portion of the bits from the whole band. As a result, we can have even more users within a frequency band at the same time (Figure 12).

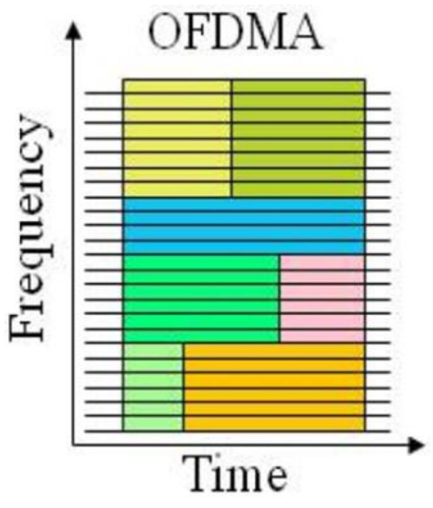


Figure 12: OFDMA

SC-FDMA is a new Single Carrier multiple access technique that is similar in structure and performance to OFDMA. Although it is a more sophisticated technique it consumes less power which is very important as a user primarily downloads and not uploads and as a result uploading consumes less power. In addition, since most users download, there will not be too many parallel users uploading, so there is no need of OFDMA for uplink.

1.4.6 LTE – QOS

QoS on LTE and LTE-A is provided through the bearers. A bearer is a virtual concept that defines how the user's data will be handled within the network. It is essentially a set of network settings designed to provide special treatment.

Each time a device enters a cell, a default bearer is registered, yet it is not unique. If the user starts a video call, another bearer can be created specifically for this call, which is called dedicated bearer.

Filtering which package goes to which bearer each time is determined by Traffic Flow Templates:

1. Uplink TFT: Sending from the user to the network, the packets are filtered from the terminals and put into a bearer which can be either default or dedicated.

2. Downlink TFT: When the data is coming from the network to the user the filtering will be done by the network to choose the correct bearer.

In QoS there are 9 different classes or 9 different bearers. Each class is characterized first by the QCI (QoS Class Identifier) which is essentially the identifier of each bearer and secondly by the ARP (Allocation and Retention Priority) which defines what will happen in high congestion situations. Furthermore, there are two types of bearers: GRB (Guaranteed Bit Rate) which provides specific bit rate requirement and Non-GBR.

The parameters that GBR bearers have are:

1. Guaranteed bit rate (GBR): This is the standard bit rate available.

2. Maximum bit rate (MBR): This is the maximum bit rate.

The non-GBR parameters are:

1. Access Point Name (APN) aggregate maximum bit rate (APN-AMBR): This is the maximum network transmission rate of all the non-GBR bearers

2. Per UE aggregate maximum bit rate (UE-AMBR): This is the maximum bit rate that a user from the network can reach.

1.5 THE 5TH GENERATION OF MOBILE NETWORKS 5G

The main purpose of this section is to understand the 5G logic. The network data flow constantly increases. As a result, their evolution is more than necessary. However, the constant development of mobile phone devices created the need for developed mobile phone networks. The main need is the large data volume received or transmitted as well as the transmission rate. The new technology will provide extra space which will allow the simultaneous connection of more than one devices. The data volume serviced must be increased up to 1000 times. Even the transmission rate reaches 100Mbps for 95% of the users using the network any given moment.
The 5G will dominate the so called "Internet of Things" where this technology is needed in order for it to function. Its own nature concerns an environment where many devices, often in a small space, are connected to each other and transmit data in real time. These devices include: smartphones, sensors, thermostats, cars, robots and other

technological artifacts connected to the 5G network. The older 4G network does not have the necessary bandwidth in order to manage the large volume of data that are transmitted by these devices. In addition, 5G will eliminate the time delay between the device and the host which communicates with the self-driving vehicles and the telemedicine applications. Such applications require a reliable connection with zero-time delay. That is because every loss or delay in the transport of data is literally a matter of life and death.

In order for 5G networks to achieve all the above-mentioned characteristics, it will be transmitted through electromagnetic waves of high frequency. Higher frequency equals to higher speed and wider bandwidth. The problem is that these waves cannot pass through walls, windows or roofs and the more they travel the more they are weakened. The companies, carriers of wireless mobile networks will have to install thousands or even millions of little transmission antennas in every traffic light, building or even every room. Therefore, even after the introduction of 5G, the new network will not replace its predecessor. At least for the first few years of its release it will only work as a supplement.

5G might be the newer technology for wireless networks that will use phones, smartwatches, cars and other devices in the next few years, but it is not available in every country simultaneously. Some estimation, such as of Ericsson, predict 1.5 billion users subscribing in a 5G network until 2024. The coverage will reach over the 40% of the global population.

A plethora of architecture scenarios have been proposed for the 5G. This is happening due to the resources and service speed of all the requests. It is important to note that 5G applications use mainly wide bandwidths, as for example in video streaming. However, the communication applications that deal with machine-to-machine and machine-to-human make life easier. For example, the introduction of many devices in the network as well as their connection will offer a plethora of new applications and services. In addition, a broad spectrum of automation in many new technology sectors is offered. A well-known model for that is the smart city.

As seen in Figure 13, this is considered the general architecture of 5G networks.

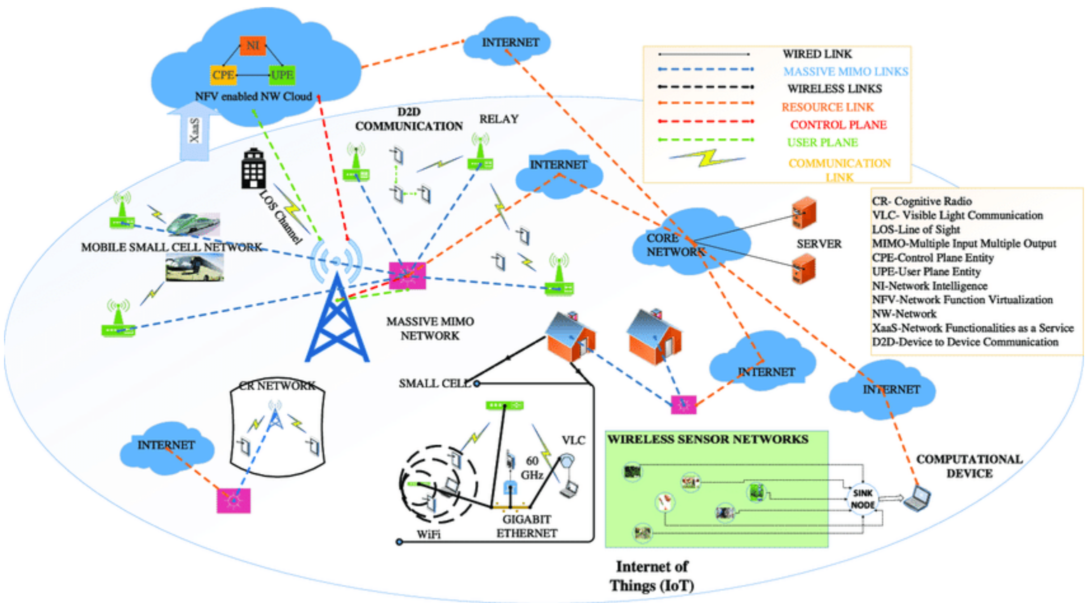


Figure 13: 5G Architecture

2. HTTP Adaptive Streaming

2.1 INTRODUCTION

HTTP Adaptive Streaming (HAS) is currently paramount as far as video technology for over-the-top video distribution is concerned, as video streaming applications account for more than 50% of the traffic on the Internet. In HTTP Adaptive Streaming the video is split into numerous segments and it is encoded in different quality versions. Each segment usually ranges from one to ten seconds and can be decoded separately of other segments. A HAS client first downloads a manifest file, which offers a full description of the available quality versions and segments of the video. Then he fetches the segment which is the most appropriate, based on its quality, in order to play the video smoothly. In this way, video streaming of best-effort is enabled. The selection of the segments to be downloaded, is determined according to the Rate Determination Algorithm (RDA), based on the network status and the filling level of the buffer (Figure 14, Figure 15). The RDA tries to improve the Quality of Experience (QoE) from the client's perspective. QoE depends on the times the video freezes, the average quality offered as well as the number of its alterations.

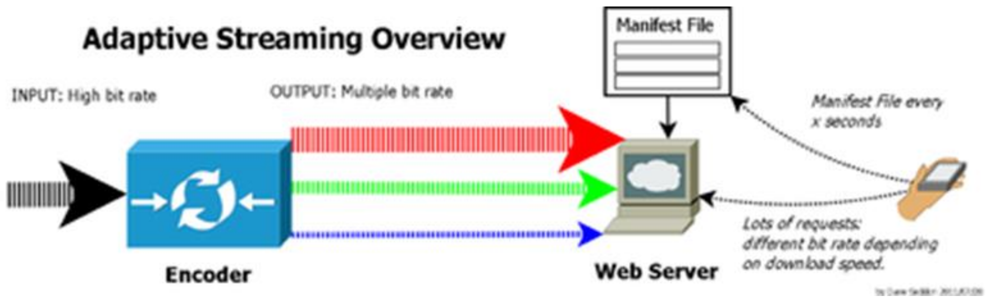

Adaptive Streaming Overview
INPUT: High bit rate
OUTPUT: Multiple bit rate
Manifest File
Manifest File every x seconds
Encoder
Web Server
Lots of requests: different bit rate depending on download speed.

Figure 14: Adaptive Streaming Overview

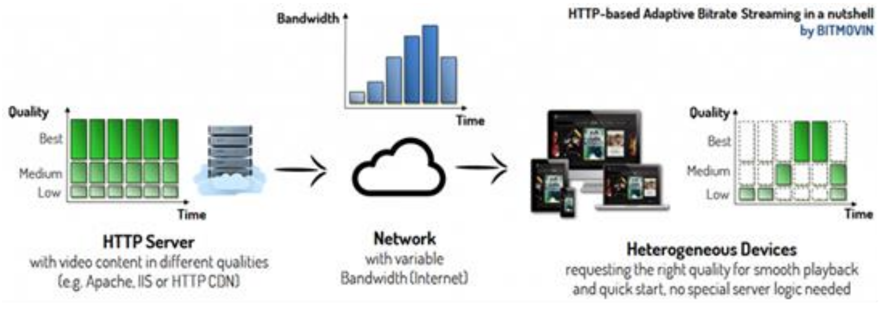

Bandwidth
Time
HTTP-based Adaptive Bitrate Streaming in a nutshell by BITMOVIN
Quality
Best
Medium
Low
Time
HTTP Server
with video content in different qualities (e.g. Apache, IIS or HTTP CDN)
Network
with variable Bandwidth (Internet)
Quality
Best
Medium
Low
Time
Heterogeneous Devices
requesting the right quality for smooth playback and quick start, no special server logic needed

Figure 15: HTTP in a nutshell

2.2 BENEFITS

HAS is superior to the traditional real-time streaming because it can adjust to different video qualities depending on the bandwidth of the network, so as to prevent the video from freezing (Figure 16). Furthermore, HTTP-based video streams are able to go over firewalls without difficulty and reuse already established HTTP servers, proxies and Content Delivery Network (CDN) nodes. As a result, companies like Microsoft, Apple, Adobe and Netflix have chosen to use this algorithm [10].

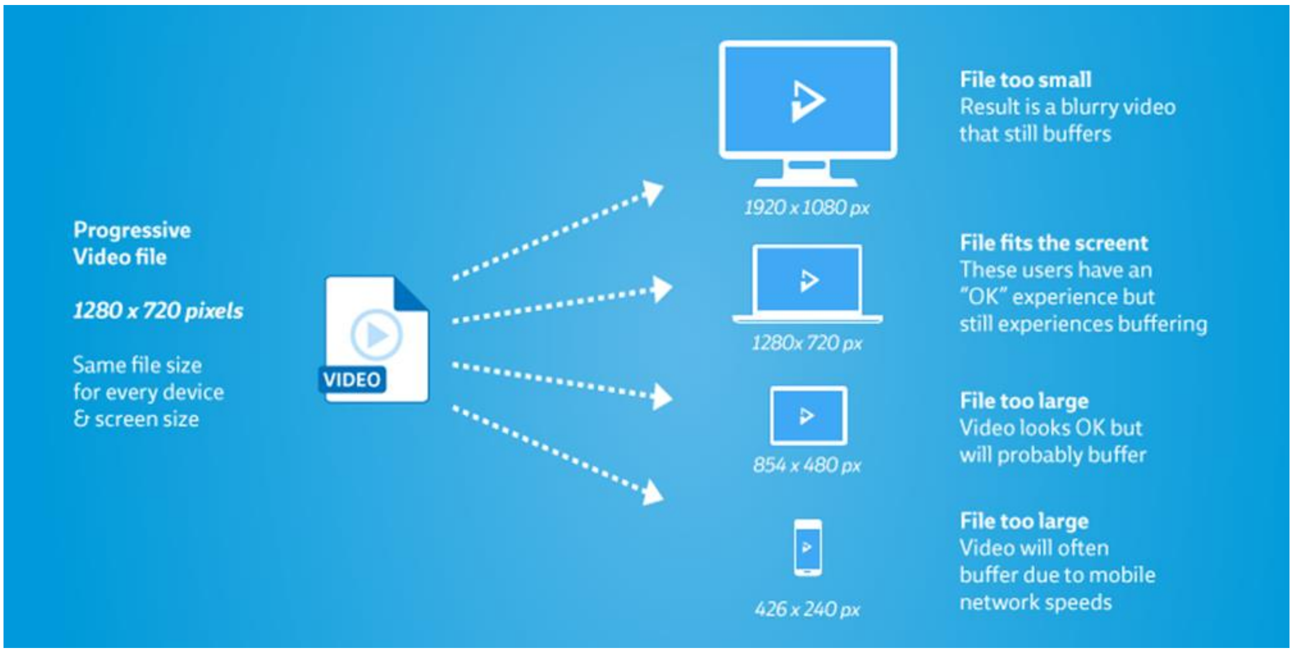

Progressive Video file
1280 x 720 pixels
Same file size for every device & screen size
VIDEO
1920 x 1080 px
File too small
Result is a blurry video that still buffers
1280x 720 px
File fits the screent
These users have an "OK" experience but still experiences buffering
854 x 480 px
File too large
Video looks OK but will probably buffer
426 x 240 px
File too large
Video will often buffer due to mobile network speeds

Figure 16: HTTP Adaptive Streaming (HAS)

HAS provides a solution to two main problems that traditional streaming was not able to do. The first one is quality. For example, a video with 1280x720 resolution will never play satisfactorily on a screen of 1920x1080 pixels. On the contrary, it will be stretched and as a result the quality shown will be low. The second, and equally important problem is buffering. Buffering appears when the video freezes because of poor network conditions. Most videos play at 24 frames every second, so the client requires to download at least 24 frames per second to prevent buffering from happening. In case of buffering, the video stream is not downloaded as rapidly as it should be, thus it needs to pause and fetch more data so as to start playing again. In other words, as a large video is downloaded more slowly than a smaller one, HAS adapts to the current internet connection of the client to keep the video playing [11]. As a result, content creators and video suppliers can support exceptional services. The speed of the internet connection can also be described as bitrate, which explains the reason why adaptive streaming is also referred as adaptive bitrate streaming. Bitrate is actually the rate of bits of data that are being transferred to the clients. Even though adaptive bitrate technology demands further encoding, it reduces the workflow and produces a better outcome. It is important to mention that avoiding buffering is more essential than retaining the quality of the video, since the client will be satisfied if the video continues playing even in a lower quality for a while.

There are two more benefits over traditional streaming methods which is important to mention. At first, as HAS is guided by the client, he is the one responsible for any adaptation. This is really beneficial as repeated connections between the server and the client can be decreased. In addition, session state information does not have to be maintained on every client which boosts scalability, too.

2.3 DYNAMIC ADAPTIVE STREAMING OVER HTTP (DASH)

The Motion Picture Expert Group (MPEG) suggested the Dynamic Adaptive Streaming over HTTP (DASH) standard in 2010, which became a Draft International Standard in January 2011, and an International Standard in November 2011. In 2015 MPEG LA announced an MPEG-DASH patent portfolio license which includes patents that are necessary to the standard (Figure 17). A lot of organizations contributed in order to develop this technology with the most important ones being Maxell, TNO and NTT. The DASH technology is connected to Adobe Systems HTTP Dynamic Streaming, Apple

Inc. HTTP Live Streaming (HLS) and Microsoft Smooth Streaming. It is also based on HAS in 3GPP and on HAS in Open IPTV Forum [12].

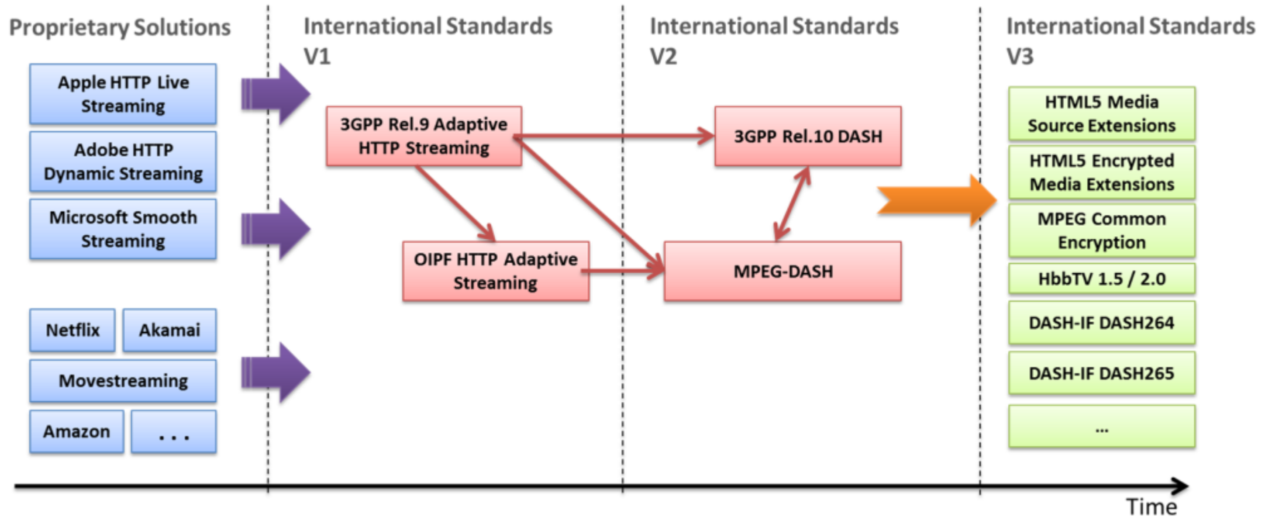

Proprietary Solutions
International Standards V1
International Standards V2
International Standards V3
Apple HTTP Live Streaming
Adobe HTTP Dynamic Streaming
Microsoft Smooth Streaming
Netflix
Akamai
Movestreaming
Amazon
...
3GPP Rel.9 Adaptive HTTP Streaming
OIPF HTTP Adaptive Streaming
3GPP Rel.10 DASH
MPEG-DASH
HTML5 Media Source Extensions
HTML5 Encrypted Media Extensions
MPEG Common Encryption
HbbTV 1.5 / 2.0
DASH-IF DASH264
DASH-IF DASH265
...
Time

Figure 17: HTTP Dynamic Adaptive Streaming Timeline

2.3.1 Additional advantages

DASH has many advantages over other adaptive streaming techniques. In the last few years, DASH has been transformed into a pioneered standard. HTML5 Media Source Extensions (MSE) presented the playback feature by adding the HTML5 video and audio tag. More importantly, the HTML5 Encrypted Media Extensions (EME) support streaming between different clients protected by interoperable Digital Rights Management (DRM) tools[1] MPEG Common Encryption (MPEG-CENC) in combination with the Hybrid broadcast broadband TV (HbbTV 1.5 and HbbTV 2.0) allows streaming even on SmartTV platforms. In addition, DASH is in fact an audio/video codec agnostic standard which means that it does not specify the adaptive bitrate streaming (ABR) logic as well as the underlying application layer protocol. Hence, it is able to be used together with any protocol [13].

2.3.2 Implementation

DASH is an adaptive bitrate streaming standard in which a multimedia file is divided into one or more segments and transferred to a client using HTTP GET requests. The adaptation is performed by the client for every segment. The client will change to a higher quality when the network bandwidth allows it. This is really beneficial as the client is aware of its network abilities and throughput. The client then requests the Media Presentation Description (MPD) file.

[1] Digital rights management (DRM) tools or technological protection measures (TPM) are a group of access control methods in order to limit the use and distribution of hardware and copyrighted works, for example of software and multimedia content

2.4 MEDIA PRESENTATION DESCRIPTION (MPD)

A Media Presentation Description (MPD) is an XML file (Figure 18). It is a hierarchical data model that includes at least one Period. Every Period describes segment information such as timing, URL, video resolution, bit rates, different view angles or with different codecs, audio components for different languages and even subtitle or caption components. Those components are characterized by specific features which cannot alter during the Period. However, the client is able to adapt during a Period to different resolutions or bitrates that are available in it. Media components are usually organized in AdaptationSets and every Period can include numerous of them. It also contains Subsets which limit associations between AdaptationSets and gives valuable designing information. An AdaptationSet includes a group of Representations that consist of information about different resolutions, bitrates etc. A variety of Representations enables the client to adapt and play the video smoothly regardless of the network conditions and bandwidth. Representations are divided into Segments which are specified by a URL or occasionally by an additional byte range. Both Segments and Representations are equal in terms of their length and are organized according to the media presentation timeline. Opposing to other techniques, DASH allows different segment lengths determined by different scenarios. For example, bigger Segments in length permit better compression or reduced overhead, while shorter Segments are ideal for live streaming and reliable network conditions as far as the bandwidth is concerned. It is also possible to divide each Segment into smaller Subsegments which will demonstrate smaller parts of the Segment.

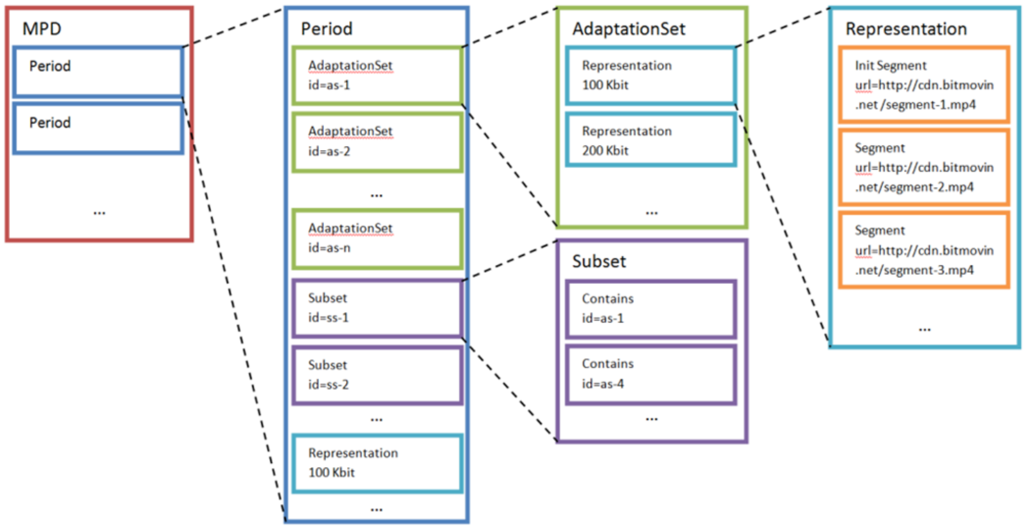


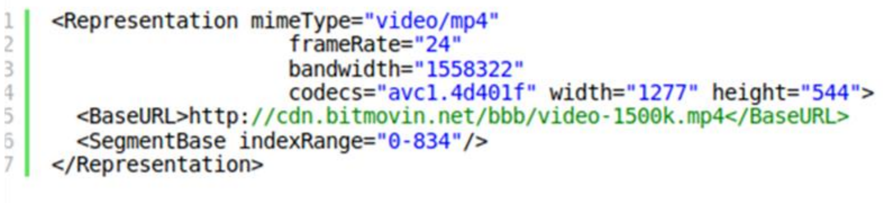
Figure 18: Media Presentation Description (MPD) File

While the video is being broadcast, it is not feasible to alter Representations. As a result, Segments cannot overlap and also dependencies between segments are not permitted. In order to overcome this restriction, MPEG - DASH presented Stream Access Points (SAP) which presents time and position in segments at which random access and switching can occur.

2.4.1 Segment referencing schemes

Segments can be arranged in many ways, for instance SegmentBase, SegmentList and SegmentTemplate, which can provide additional information.

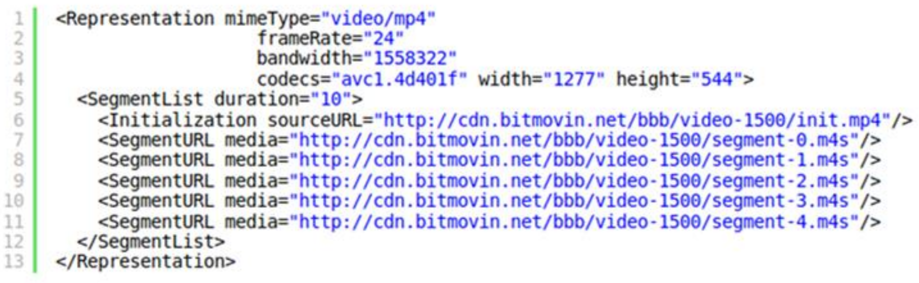
SegmentBase (Figure 19) is the least effective way to represent segments because it will be used when only one media segment is present in every Representation. Otherwise, either SegmentList or SegmentTemplate must be used.



```
<Representation mimeType="video/mp4"
                    frameRate="24"
                    bandwidth="1558322"
                    codecs="avc1.4d401f" width="1277" height="544">
  <BaseURL>http://cdn.bitmovin.net/bbb/video-1500k.mp4</BaseURL>
  <SegmentBase indexRange="0-834"/>
</Representation>
```

Figure 19: SegmentBase Arrangement

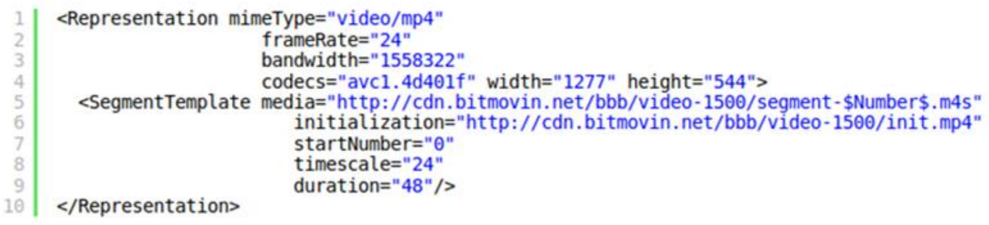
SegmentList (Figure 20) includes a list of SegmentURL elements which will be played at the order they are presented. A SegmentURL element contains a URL to a segment and possibly a byte range.



```
<Representation mimeType="video/mp4"
                    frameRate="24"
                    bandwidth="1558322"
                    codecs="avc1.4d401f" width="1277" height="544">
  <SegmentList duration="10">
    <Initialization sourceURL="http://cdn.bitmovin.net/bbb/video-1500/init.mp4"/>
    <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment-0.m4s"/>
    <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment-1.m4s"/>
    <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment-2.m4s"/>
    <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment-3.m4s"/>
    <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment-4.m4s"/>
  </SegmentList>
</Representation>
```

Figure 20: SegmentList Arrangement

Last but not least, SegmentTemplate (Figure 21) element creates a list of segments from each template. Its list consists of a few lines that show the way to create a large list of segments, opposing to SegmentList whose lists are way bigger.



```
<Representation mimeType="video/mp4"
                    frameRate="24"
                    bandwidth="1558322"
                    codecs="avc1.4d401f" width="1277" height="544">
  <SegmentTemplate media="http://cdn.bitmovin.net/bbb/video-1500/segment-$Number$.m4s"
                      initialization="http://cdn.bitmovin.net/bbb/video-1500/init.mp4"
                      startNumber="0"
                      timescale="24"
                      duration="48"/>
</Representation>
```

Figure 21: SegmentTemplate Arrangement

3. Quality of Experience (QoE)

3.1 INTRODUCTION

Quality of Experience (QoE), originated from Quality of Service (QoS), is a method that measures the level of satisfaction or dissatisfaction of a client's experience with a particular service such as a phone call, TV broadcasting or streaming to name just a few. QoE is actually connected with the network, not only the client himself and concentrates on the whole service experience. Yet, when talking from the client's point of view, for his complete service experience, QoE is a subjective measure based on his aesthetic and needs. In other words, QoE offers an evaluation as for the individual's expectations, emotions, apprehension, understanding and fulfillment. In 2013, The European Cooperation in Science and Technology (COST Association) has defined QoE 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*" [14].

3.2 QoE FACTORS

QoE takes into account three main factors, known as influence factors that are responsible for the experience that a customer perceives. Firstly, the Human Influence Factors that refer to the visual and auditory comprehension of the individual, the gender, the age and the emotional condition that he is in. These are the low-level processing factors [15]. On the other hand, the high-level processing factors are his mental processes, his social and cultural background as well as personal wishes and ambitions. Secondly, the System Influence Factors connected with the Content, the Media (encoding, resolution, sample rate etc.), the Network (bandwidth, delay, buffering, etc.) and the Device (screen resolution, display size, etc.). Finally, the Context Influence Factors related to Physical (location and space), Temporal (time of day, frequency of use, etc.), Social (personal relations), Economic, Task (multitasking, interruptions, task type), Technical and Information (relationship between systems) context (Figure 22).

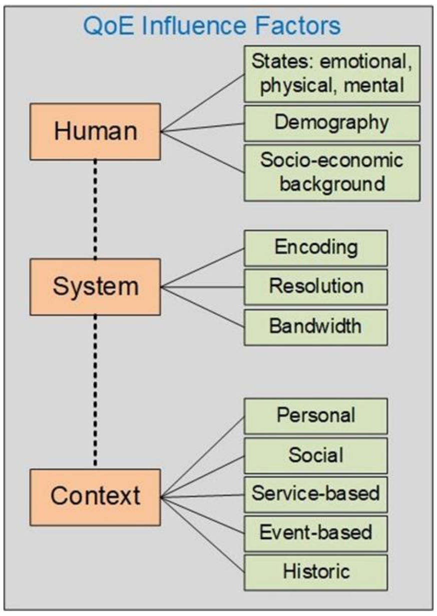


Figure 22: QoE Factors

3.3 QoE VS QoS – THE IQX HYPOTHESIS

QoS is not the same as QoE. The first one concerns purely technical aspects while the latter also reflects end-user satisfaction [16]. Same QoS in two users does not necessarily mean the same QoE in each one. However, there is an exponential relationship between QoS and QoE which is called "the IQX hypothesis" and is defined as $QoE = \alpha * e^{-bQoS} + \gamma$ (Figure 23):

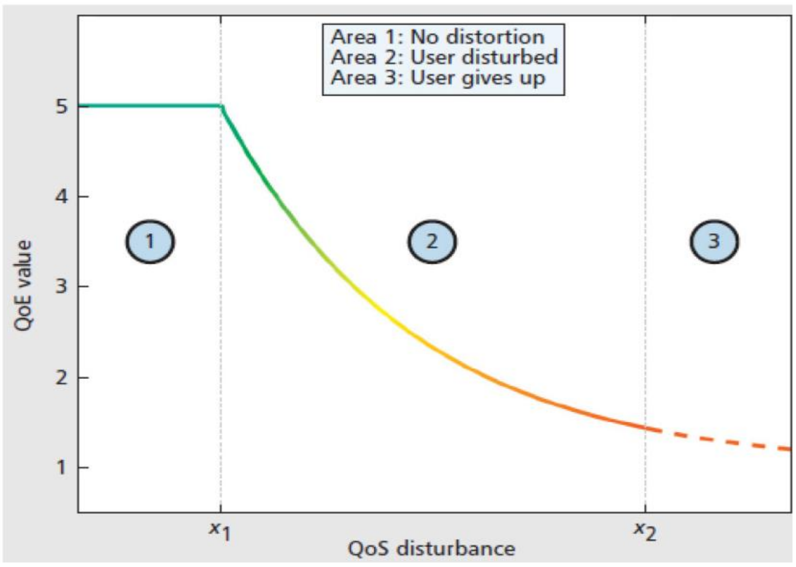


Figure 23: The IQX hypothesis diagram

The logic behind this relationship is that the more QoS we actually pay for, the higher QoE we expect to have. However, this relationship is negatively exponential which means that if we have very good QoS, any slight drop will cost a lot to QoE. Whereas, if we have very low QoS, a further decline will not matter much. In addition, it is important to highlight two areas at the beginning and end of the graph (Constant Optimal (Region 1) and Unacceptable (Region 3)). In the first one we see that even if QoS falls slightly, QoE will remain the same. This is very important to the providers, as they can reduce their development costs and resources by reducing QoS a little. On the other hand, in the Unacceptable area both QoS and QoE are very low, so the user will probably give up using the service.

3.4 QoE MEASURMENTS

It is very important to find a way to measure QoE. QoE measurement is possible by using the QoE Modeling which includes subjective and objective ways of measuring QoE (Figure 24). The difference between subjective and objective assessment is that the first is based on experiments on the subjective view of the client that is using a service. Subsequently, all subjective estimates are collected and are then used to form the objective assessment of QoE.

The subjective way, on condition that it is properly designed, is the most reliable measurement for QoE as it is very accurate and valid and ensures consistency between subjective measurements. However, it is not real-time and the results cannot be reproduced at any time. It is also a time consuming and costly process that requires very good planning so it is complicated, too. The subjective way may also be biased by users' opinions due to unconscious psychological factors. For example, users may be greedy for their QoE demands, so their ratings can be poor.

As for the objective way of measurement, it is an automated QoE forecast, which means that same inputs always have the same outputs. Also, it can be either real-time or proactive. Nevertheless, it is really complex and it may not always represent the reality. Finally, there is not a universal model to use, so depending on the service we want to evaluate, it must be different.

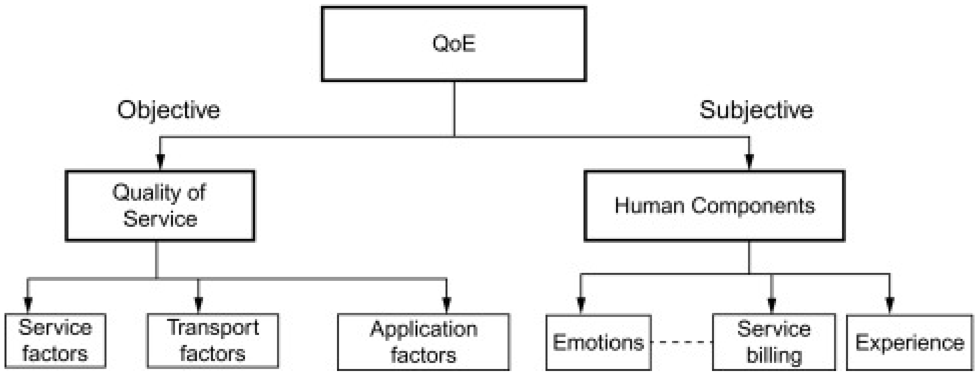


Figure 24: QoE Modeling

3.5 QoE MANAGEMENT

Figure 25 illustrates the three basic QoE Management Entities. The QoE Controller is the entity that collects the needed data from the network and sends it to the QoE Monitor entity. QoE Monitor then implements the QoE evaluation model and calculates a rating for the network. Next, this rating is sent to the QoE Manager entity which will take some decisions concerning the network and will also update it through the QoE Controller. Decisions can vary depending on the QoE Manager. For example, a possible decision could be to give priority to a certain user.

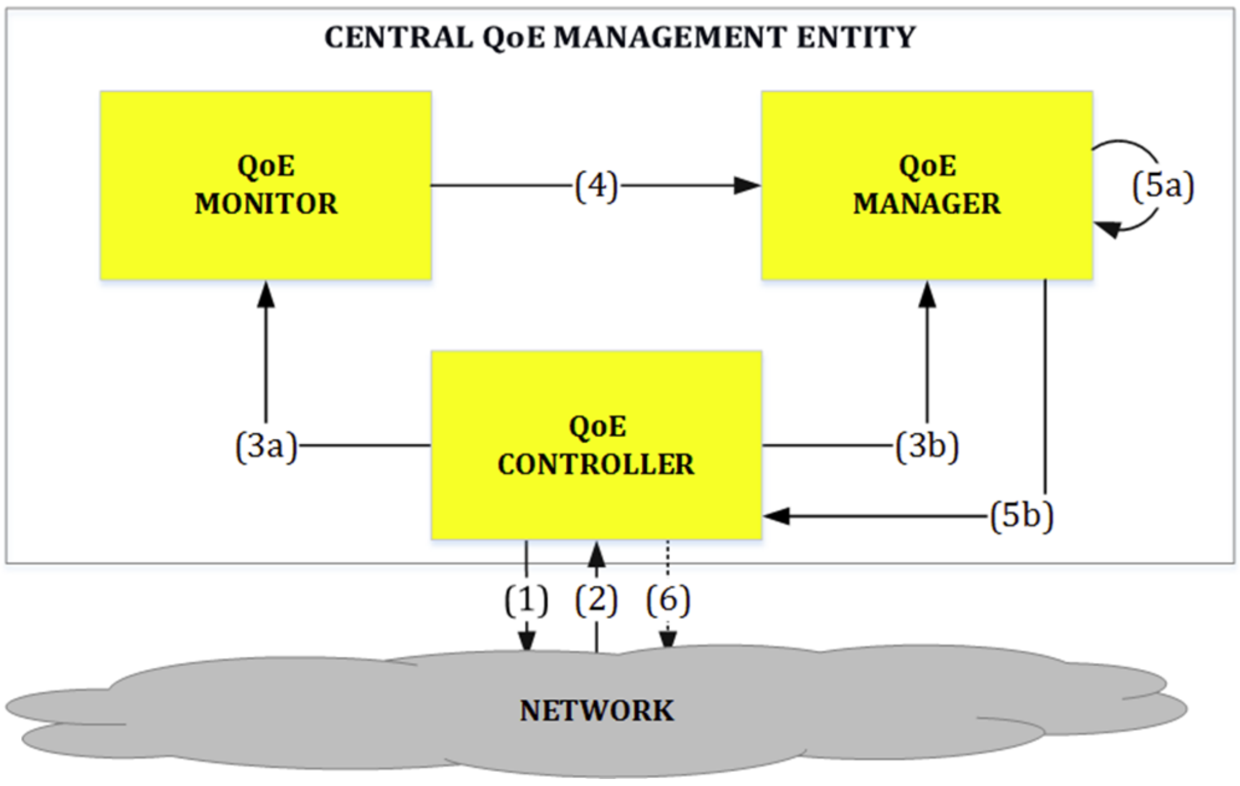


Figure 25: QoE Management Entities

3.6 HTTP VIDEO STREAMING INFLUENCED BY QoE FATORS

HTTP video streaming or video on demand streaming combines download and playback at the same time. The video is transmitted to the client via HTTP and is saved to a buffer. Even if the video has not been downloaded completely, yet only an adequate number of data is available, the video can start playing from the buffer. However, problems in the network like packet loss, insufficient bandwidth, delay and jitter can reduce the throughput and create delays. As a result, the buffer loads more slowly or in some cases depletes. Consequently, the playback has to be paused till sufficient data has been delivered. These interruptions are called stalling or rebuffering. Stalling together with the initial delay of the video playback are the two most important factors that influence QoE.

3.6.1 Initial delay

Initial delay is defined as the time since the user chose to watch the video, or in other words the time difference between the first HTTP request sent to the server and the moment the playback begins. In every service, video streaming has a small initial delay, which depends on the available transmission data rate. Usually, even though the received amount of segments is sufficient to start the video playback, there is a small delay in order to fill the buffer with more data. This takes place in case of throughput reduction, that below a certain threshold will cause stalling. Therefore, saved data in the buffer allows throughput to return to previous levels and possibly avoid stalling. However, another problem arises. The more buffered data to prevent stalling, the greater the initial delay. On the other hand, providing less buffered playtime reduces initial delay, but increases the likelihood of stalling occurrence. In order to determine which of these two scenarios is more preferable, the impact on the end-user QoE was taken into account. The results of a survey [17] showed that 90% of the users preferred initial delays to stalling. As the users are accustomed to delays before launching a service, they are willing to tolerate them to some extent. Stalling, however, is an unexpected event during the service, without acknowledgment of its duration in advance and creates more inconvenience.

3.6.2 Stalling

Stalling is defined as the interruption of the video playback, since it has already started because of playout buffer depletion. When the buffer is empty, playback stops until it gains a certain amount of playtime data. This process is called rebuffering. As a result, increased stalling duration reduces QoE. It seems that this is probably the parameter with the greater impact on QoE. Therefore, it is also the parameter that mostly concerns video streaming providers. In this regard, a service that uses HTTP Adaptive Streaming should try to avoid stalling by all means.

4. Environment Setup

This chapter demonstrates all the requirements so as to operate the SDN environment. In this thesis, the following steps were followed to develop the SDN framework.

4.1 SYSTEM REQUIREMETS

4.1.1 Operating system (OS)

The simulation in the current thesis uses Ubuntu 18.04 LTS OS.

4.1.2 Java

OpenDaylight Controller, used for this thesis, runs in Java Virtual Machine (JVM). Due to the fact that this is a Java application, it can potentially run in any operating system and hardware as long as it supports Java. However, a Java 8-compliant JVM is necessary. For this reason, we executed the following commands from the terminal:

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

Then, the JAVA_HOME variable was set, by adding the following at the end of the /etc/profile file:

```
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin
export JAVA_HOME
export PATH
```

The path of the variable can be checked by running this command:

```
echo $JAVA_HOME
```

4.1.3 SDN controller deployment

ODL Controller must be installed and configured independently, as a separate application. ODL Controller can be used as a remote controller; however only the default features can be installed and used. For its installation the following steps were followed:

First, we visited the OpenDaylight downloads page[2] and we downloaded the OpenDaylight Controller Boron SR4. When the file had successfully been downloaded, we unzipped it, opened a terminal in this directory and executed **./bin/karaf** so as to start the ODL controller. For its termination “shutdown –f” or “logout” must be typed.

The OpenDaylight controller is deployed on the concept of Apache Karaf. Apache Karaf is a modular open source runtime environment. It is a lightweight, powerful, and enterprise ready container powered by OSGi and can host any kind of applications. Apache Karaf supports the provisioning of applications as well as modules using the concept of Karaf Features, which means that it provides a simple and flexible way to arrange applications, using different features.

[2]https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation

A feature describes an application as:

- a name
- a version
- an optional description (eventually with a long description)
- a set of bundles
- optionally a set configurations or configuration files
- optionally a set of dependency features

Once a feature is installed, Apache Karaf installs all resources at the same time. In other words, it will unconsciously resolve and install all bundles, configurations, and dependency features described in the feature [18].

In the current thesis several features were installed using the following command in the karaf console:

```
feature:install[FEATURE_NAME]
```

One necessary feature is odl-dlux-all, which provides a Web based user interface for OpenDaylight. The UI can be found in this link:

```
http://localhost:8181/index.html
```

and it is only available when karaf is executed which means that the controller is running. In any other case, the web page will not load. We can enter, using admin for both username and password (Figure 26).

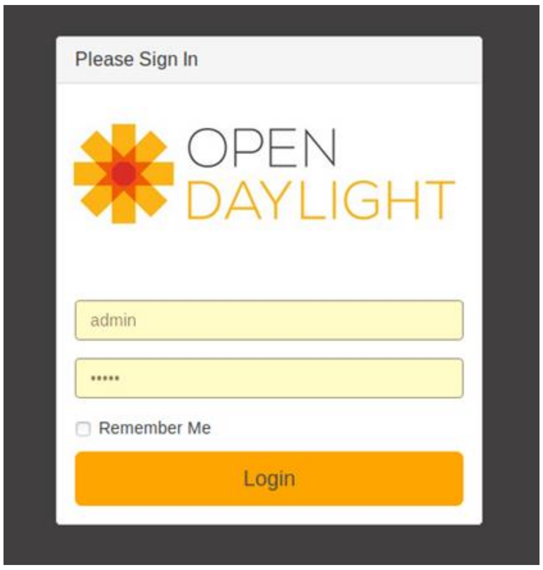


Figure 26: OpenDaylight Login Page

Along with the odl-dlux-all feature, the features odl-dlux-core, odl-dlux-node and odl-dlux-yangui have also been installed and can be found in the panel on the left of ODL DLUX's page (Figure 27).

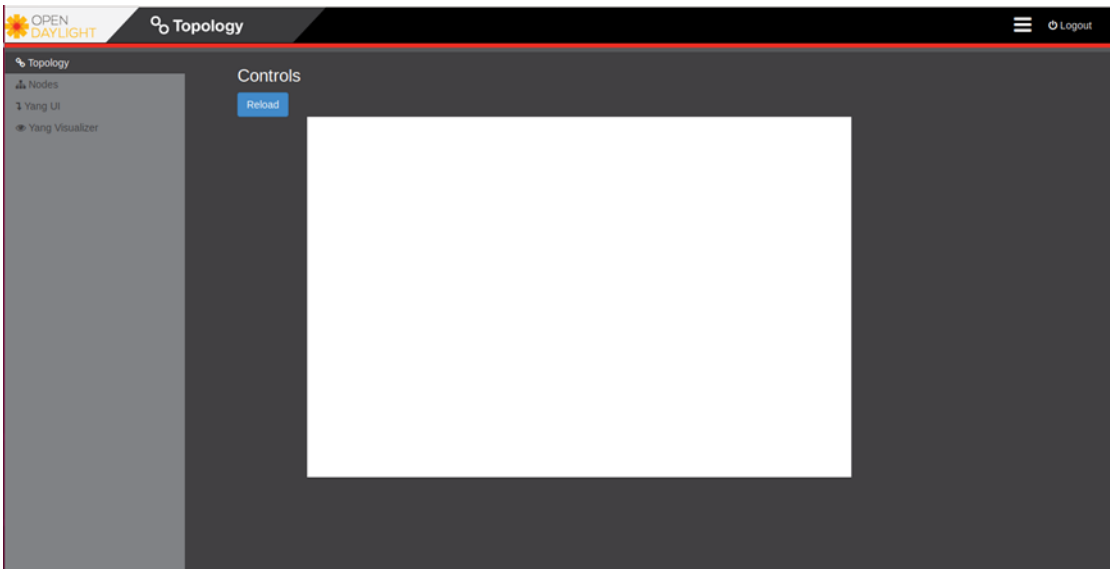


Figure 27: OpenDaylight Main Page

odl-dlux-core feature: provides the topology tab and shows the graphical representation of the network topology (Figure 28). Switches are represented as blue boxes, and available hosts as black boxes. Switches and hosts are connected by lines. We can view source and destination ports if we click on hosts, links, or switches.

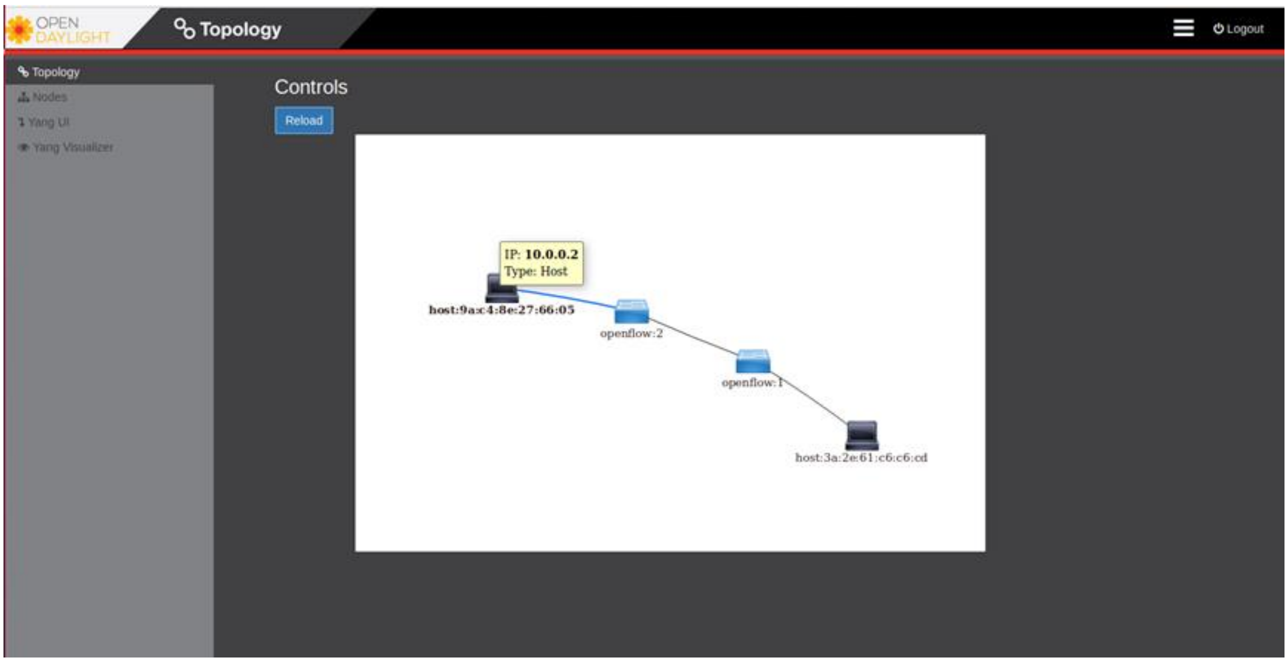
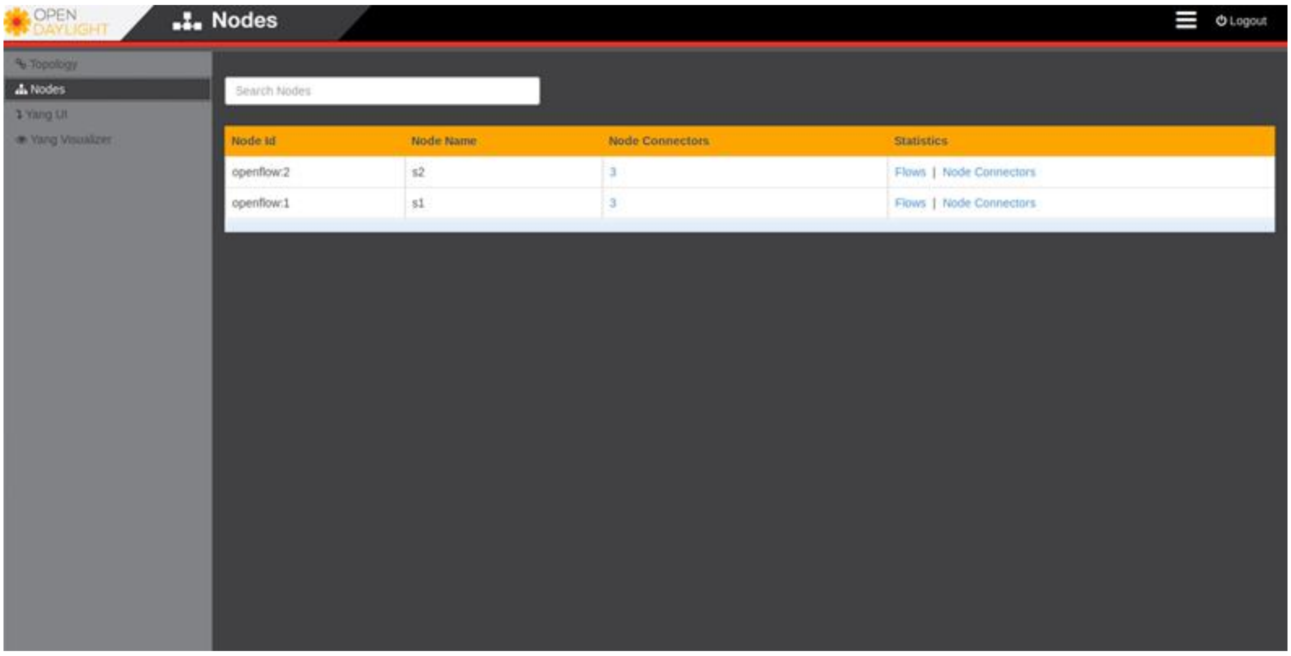


Figure 28: Topology Tab

odl-dlux-node feature: Provides the Nodes tab. It demonstrates a table of all the nodes, node connectors and the statistics (Figure 29). In addition, we can enter a node ID in the Search Nodes tab and look for it, click on the Node Connector number to view more details about the port ID or name, the number of ports per switch or even the MAC Address. Also, we can click on Flows in the Statistics column to view the Flow Table Statistics for any node like the table ID. Finally, by clicking on Node Connectors we can view the Node Connector Statistics for a specific node ID.



Node Id	Node Name	Node Connectors	Statistics
openflow:2	s2	3	Flows | Node Connectors
openflow:1	s1	3	Flows | Node Connectors

Figure 29: Nodes Tab

odl-dlux-yangui feature: It provides the Yang UI tab which is an ODL DLUX-based application designed to simplify application development and testing [19].

4.1.4 Mininet

In order to deploy SDN applications with different network topologies, a network emulation tool is required, which in our case is Mininet [20]. For its installation, the steps below were followed:

First, we got the source code with the following command which will get the latest and greatest version of Mininet:

```
git clone git://github.com/mininet/mininet
```

The command to install Mininet is:

```
Mininet/util/install.sh -a
```

This installs everything that is included in the Mininet VM, including dependencies like Open vSwitch. By default, these tools will be built in directories created in the home directory.

After the installation is completed, the basic Mininet functionality is tested with the command:

```
sudo mn --test pingall
```

The basic mininet topology (Figure 30) as well as the output of the pingall command (Figure 31) are shown at the respective figures.

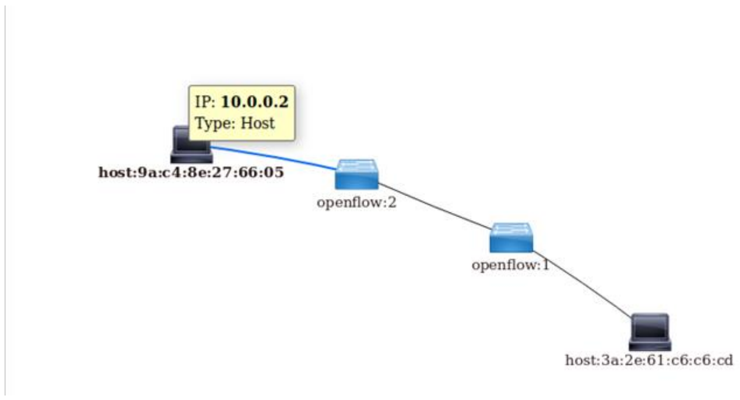
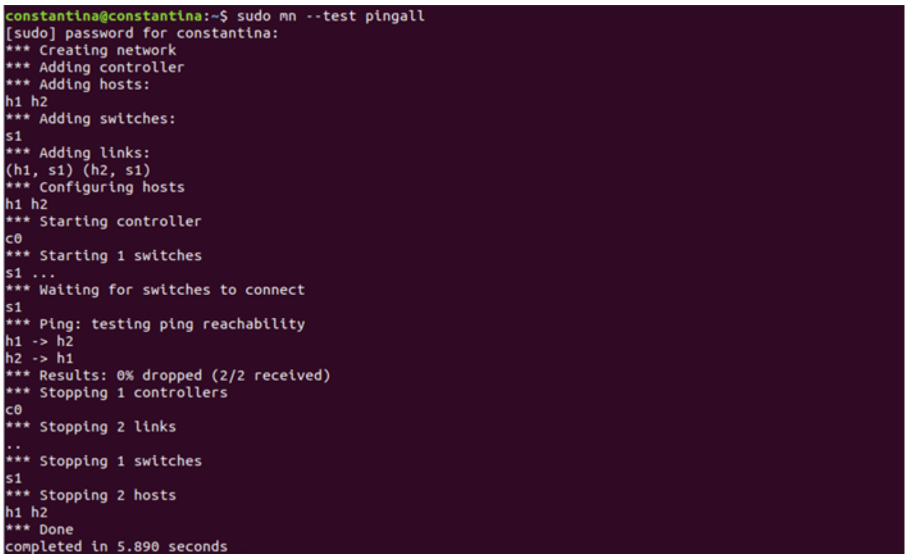


Figure 30: Basic Topology



```
constantina@constantina:~$ sudo mn --test pingall
[sudo] password for constantina:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.890 seconds
```

Figure 31: Pingall Output

4.1.5 Mininet’s graphical user interface, MiniEdit

In the terminal, we execute the following command in order to open Mininet’s graphical user interface, MiniEdit:

```
sudo ~/mininet/examples/miniedit.py
```

The Mininet network simulator includes MiniEdit. It is a simple GUI editor for Mininet. MiniEdit is an experimental tool in order to show how Mininet can be extended. The MiniEdit script is located in Mininet’s examples folder. Mininet demands root privileges, so we use the sudo command. MiniEdit has a simple user interface. Its main page shows a canvas with a variety of tools on the left and a menu bar on the top of the window (Figure 32).

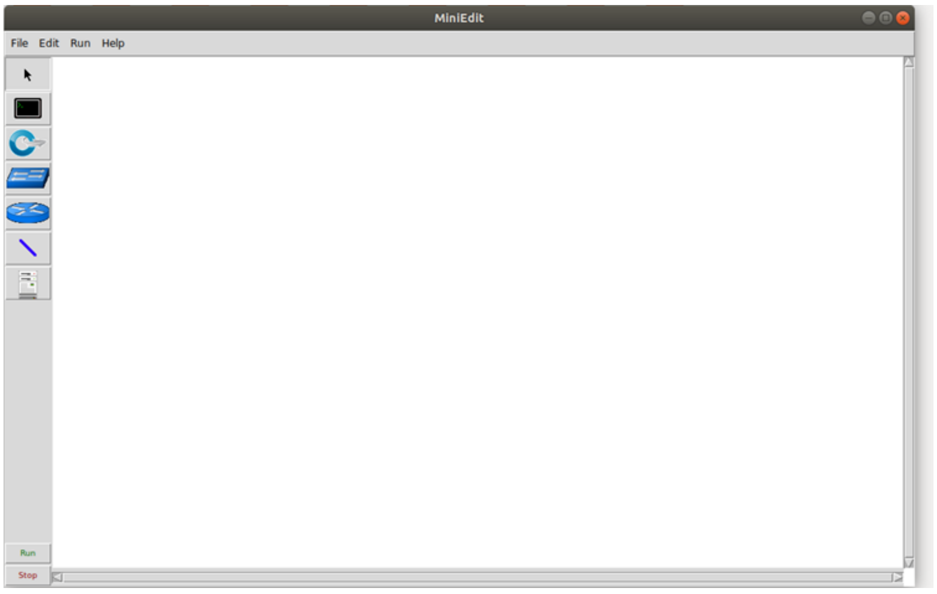


Figure 32: MiniEdit main page

The icons, from top to bottom, represent specific tools:

The Select tool enables us to drag nodes around on the canvas. Surprisingly, we do not have to use the Select tool to select a node or a link. By just either right-clicking or pressing the Delete key, we can select the node or the link that we desire and display the configuration menu or even remove each element.

The Host tool creates nodes which represent host computers. Clicking on the tool and then clicking in any place on the canvas we can place a node. On condition that we do not select another tool, we can continue adding hosts. By right-clicking on the hosts and selecting Properties we are able to configure them.

The Switch tool creates OpenFlow-enabled switches. All switches must be connected to a controller. Again, we can configure each switch by right-clicking on it and selecting Properties.

The Legacy Switch tool creates a learning Ethernet switch with default settings. In contrast to the switches above, these are capable of operating without a controller and also cannot be configured.

The Legacy Router tool creates a basic router that will operate separately and cannot be configured, too. It is basically a host with IP Forwarding enabled.

The NetLink tool creates links between nodes. We can connect elements by clicking any node and dragging the link to the other one. We may configure each link by right-clicking on it and selecting Properties.

The Controller tool creates a controller. We can add as many controllers as we want. By default, the MiniEdit creates a mininet openFlow reference controller, which affects the behaviour of a learning switch. We can configure other controller types, too. We may configure each controller by right-clicking on it and selecting Properties.

Run
Stop

The Run starts Mininet simulation experiment. On the other hand, the Stop button stops it. When MininEdit simulation is running, other functions are available by right-clicking on any element.

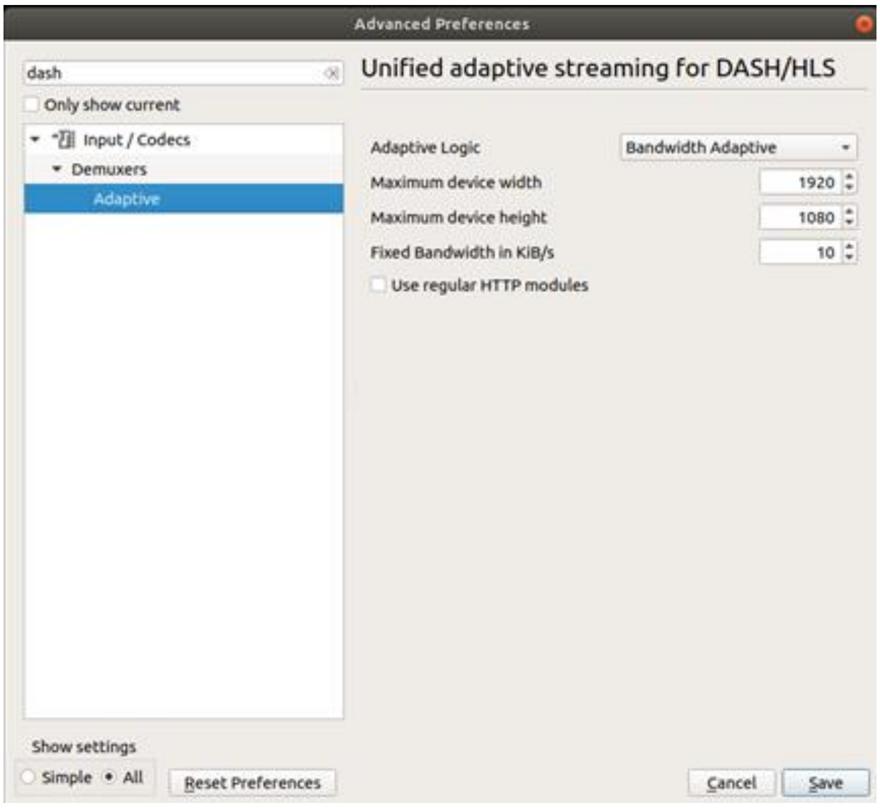
4.1.6 VLC media player

VLC media player is a free and open-source portable cross-platform media player software and streaming media server designed by the VideoLAN project. VLC is used for different desktop OS and mobile systems, like Android, iOS, iPadOS, Tizen, Windows 10 Mobile and Windows Phone. It provides many audio and video compression techniques and file formats, such as DVD-Video, video CD and streaming protocols. It is also capable of broadcasting over computer networks and transcoding multimedia data which is exactly the reason why we used it for our thesis. The default package of VLC consists of a wide variety of free decoding and encoding libraries. The libavcodec library from the FFmpeg program offers many of VLC's codecs, however the VLC player primarily works with its own muxers and demuxers. It also has its own protocols. In addition, it uses the libdvdcss DVD decryption library so as to support Demuxers playback of encrypted DVDs on Linux and macOS for the first time [21].

The command to install VLC is:

```
sudo apt install vlc
```

For this thesis and for the sake of Adaptive streaming, certain settings are necessary so that the VLC would be able to support Adaptive Streaming for DASH/HLS [22]. In the tab Tools we select Preferences. Then, we select the tab Input/Codecs → Demuxer → Adaptive and we type dash (Figure 33).


Advanced Preferences

dash

Only show current

Input / Codecs
Demuxers
Adaptive

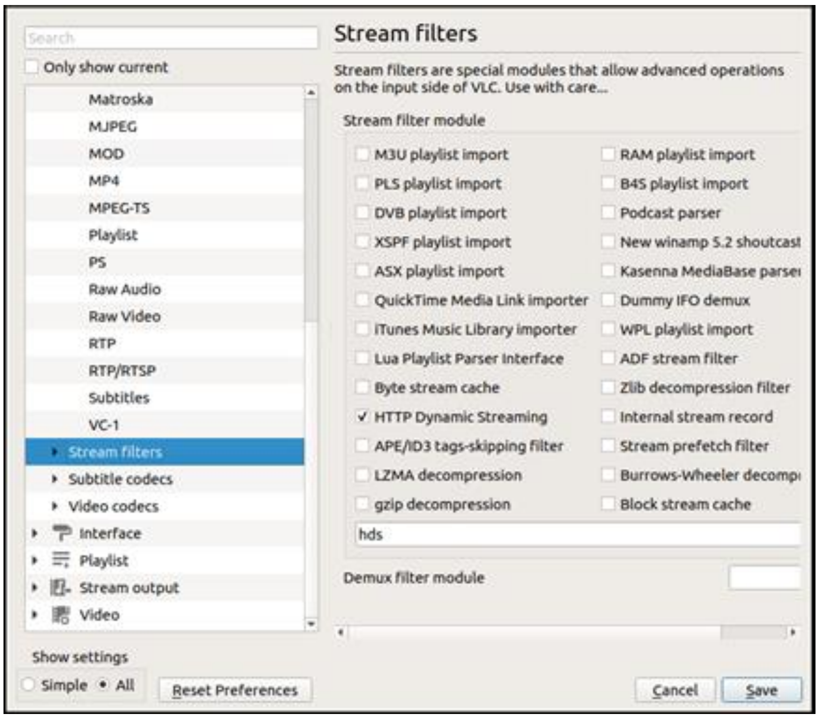
Unified adaptive streaming for DASH/HLS

Adaptive Logic: Bandwidth Adaptive
Maximum device width: 1920
Maximum device height: 1080
Fixed Bandwidth in KiB/s: 10
Use regular HTTP modules

Show settings: Simple [✓ All]

Figure 33: VLC Preferences (1)

Furthermore, on the same tab Input/Codecs we select Stream filters and we tick the box HTTP Dynamic Streaming (Figure 34).



Stream filters

Stream filters are special modules that allow advanced operations on the input side of VLC. Use with care...

Stream filter module

- [ ] M3U playlist import
- [ ] RAM playlist import
- [ ] PLS playlist import
- [ ] B4S playlist import
- [ ] DVB playlist import
- [ ] Podcast parser
- [ ] XSPF playlist import
- [ ] New winamp 5.2 shoutcast
- [ ] ASX playlist import
- [ ] Kasenna MediaBase parser
- [ ] QuickTime Media Link importer
- [ ] Dummy IFO demux
- [ ] iTunes Music Library importer
- [ ] WPL playlist import
- [ ] Lua Playlist Parser Interface
- [ ] ADF stream filter
- [ ] Byte stream cache
- [ ] Zlib decompression filter
- [✓] HTTP Dynamic Streaming
- [ ] Internal stream record
- [ ] APE/ID3 tags-skipping filter
- [ ] Stream prefetch filter
- [ ] LZMA decompression
- [ ] Burrows-Wheeler decompr
- [ ] gzip decompression
- [ ] Block stream cache

hds

Demux filter module

Figure 34: VLC Preferences (2)

Finally, we save our preferences.

4.1.7 FFmpeg

FFmpeg is a free and open-source project which contains a wide software set of libraries and programs which was extensively used in our thesis for handling video, audio, and other multimedia files and streams. The name of the project comes from the MPEG video model, along with "FF" which stands for "fast forward" [23]. The basic body is the FFmpeg program itself which is created for operations though the command line window for video and audio files. It is extensively used for format transcoding, basic editing (trimming and concatenation), video scaling, video post-production effects, and standards compliance (SMPTE, ITU).

FFmpeg includes libavcodec which is an audio/video codec library, libavformat (Lavf), an audio/video container mux and demux library and finally the basic FFmpeg command line program which is a transcoder of multimedia files.

FFmpeg is similar to other software projects, and its libraries are the basis of other software media players like VLC as mentioned before, YouTube and iTunes. It is a greatly effective tool for converting both common and uncommon media files into a single common format.

For the installation process in Ubuntu we must type the following commands in the command window [24]:

```
sudo apt-get update

sudo apt-get install ffmpeg
```

4.1.8 Dynamic adaptive streaming over HTTP (DASH)

Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is an adaptive bitrate streaming procedure that implements high quality streaming of media files over traditional HTTP web servers [25]. It is the first adaptive bit-rate HTTP-based streaming solution that is an international standard. MPEG-DASH splits the content into a string of small HTTP-based file segments, which consist of a small recap of the video, regardless of its duration. Then the segments are available at a range of different bit rates. While the MPEG-DASH client plays the video, the client runs an Adaptive Bit Rate (ABR) algorithm to mechanically choose the segment to be displayed, with the highest bit rate possible. The current MPEG-DASH client dash.js provides not only buffer-based bit rate adaptation algorithms (BOLA) but also hybrid (DYNAMIC). For these reasons, an MPEG-DASH client can smoothly fit to unsteady network conditions and support high quality playback with less stalls or re-buffering events.

We execute the following command to install Node on Ubuntu, which will also install NPM and other dependent packages [26].

```
sudo apt-get install nodejs
```

Node.js is a Javascript-based software development platform mainly used for servers. In our thesis we used this software in order to create a server. Its development will be explained in another chapter.

4.1.9 Project on advanced content (GPAC)

It is an application of the MPEG-4 Systems standard and it is written in ANSI C. GPAC supports means for media playback, vector graphics and 3D rendering, MPEG-4 authoring and distribution. GPAC provides three sets of tools based on a core library called libgpac. The first is MP4Client which is a multimedia player, MP4Box which is a multimedia packager as well as some server tools for multiplexing and streaming which are still under development [27].

For this thesis, we used MP4Box for the preparation of the HTTP Adaptive Streaming content, specifically to create the DASH manifest and associated files [28]. To install MP4Box we run the command below:

```
sudo apt-get install gpac
```

4.1.10 Wireshark

Wireshark is a free, open-source packet analyzer, complying with the GNU General Public License. The main reason for its use in our thesis is the fact that it provides network troubleshooting, analysis, software and communications protocol development, which were extremely helpful in order to collect certain data. At first the project was called Ethereal but was later renamed Wireshark because of brand controversies. Wireshark is a cross-platform [29]. It uses two main tool-kits, the Qt widget which performs the user interface along with the pcap in order to monitor packets. It is available on Linux, macOS, BSD, Solaris and Microsoft Windows. In addition, there is a non-GUI version of Wireshark, called Tshark. Wireshark lets us monitor traffic by using different controllers in the network interface.

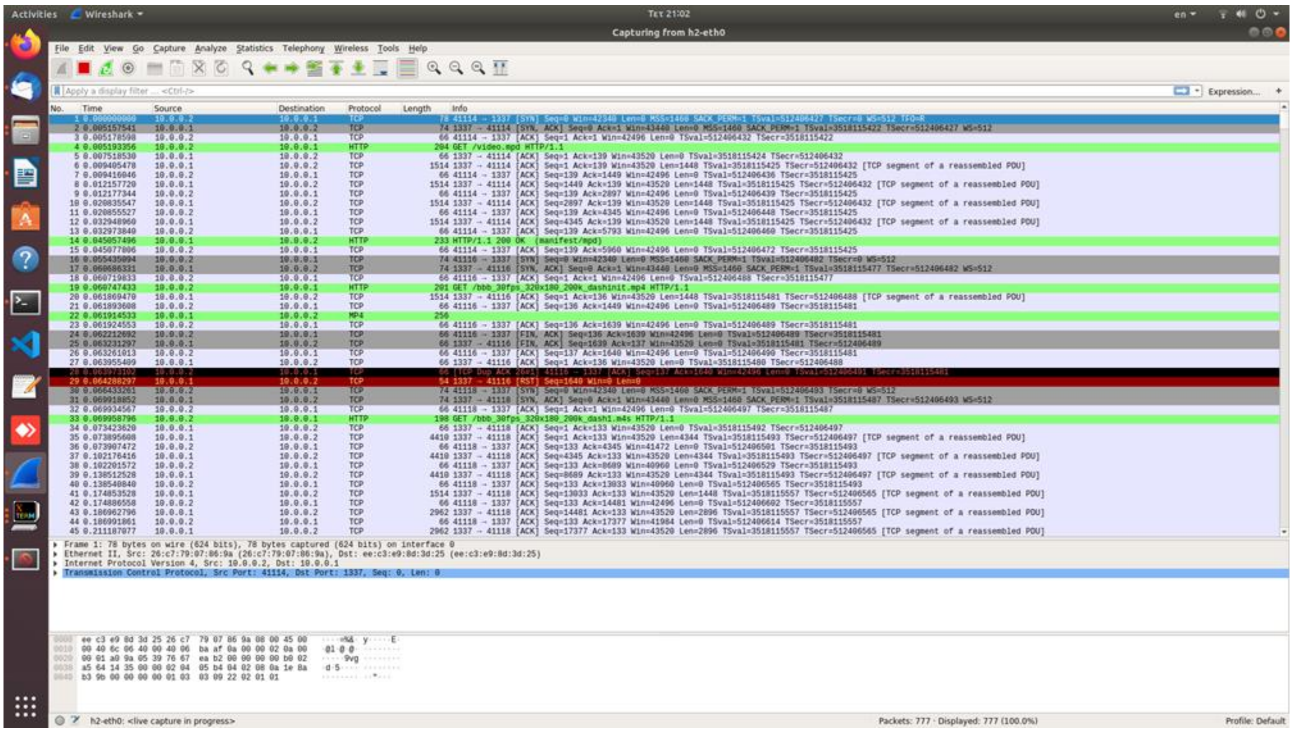
Before we install Wireshark, we need to install all the dependencies needed using the following command [30]:

```
sudo apt-get install build-essential checkinstall libcurl4-openssl-dev bison flex qt5-efault
qttools5-dev libssl-dev libgtk-3-dev libpcap-d
```

Then, we install Wireshark by typing:

```
sudo apt-get install wireshark
```

Figure 35 shows the main page of Wireshark and the data displayed.


Activities
Wireshark
Τετ 21:02
Capturing from h2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
h2-eth0: <live capture in progress>
Packets: 777 · Displayed: 777 (100.0%)
Profile: Default

```
No.  Time         Source     Destination  Protocol  Length  Info
  1 0.000000000   10.0.0.2   10.0.0.1     TCP         78 41114 → 1337 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=512406427 TSecr=0 WS=512 TFO=R
  2 0.005157541   10.0.0.1   10.0.0.2     TCP         74 1337 → 41114 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=3518115422 TSecr=512406427 WS=512
  3 0.005178598   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=512406432 TSecr=3518115422
  4 0.005193356   10.0.0.2   10.0.0.1     HTTP       204 GET /video.mpd HTTP/1.1
  5 0.007518530   10.0.0.1   10.0.0.2     TCP         66 1337 → 41114 [ACK] Seq=1 Ack=139 Win=43520 Len=0 TSval=3518115424 TSecr=512406432
  6 0.009405478   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41114 [ACK] Seq=1 Ack=139 Win=43520 Len=1448 TSval=3518115425 TSecr=512406432 [TCP segment of a reassembled PDU]
  7 0.009416046   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=139 Ack=1449 Win=42496 Len=0 TSval=512406436 TSecr=3518115425
  8 0.012157720   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41114 [ACK] Seq=1449 Ack=139 Win=43520 Len=1448 TSval=3518115425 TSecr=512406432 [TCP segment of a reassembled PDU]
  9 0.012177344   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=139 Ack=2897 Win=42496 Len=0 TSval=512406439 TSecr=3518115425
 10 0.020835547   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41114 [ACK] Seq=2897 Ack=139 Win=43520 Len=1448 TSval=3518115425 TSecr=512406432 [TCP segment of a reassembled PDU]
 11 0.020855527   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=139 Ack=4345 Win=42496 Len=0 TSval=512406448 TSecr=3518115425
 12 0.032948960   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41114 [ACK] Seq=4345 Ack=139 Win=43520 Len=1448 TSval=3518115425 TSecr=512406432 [TCP segment of a reassembled PDU]
 13 0.032973840   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=139 Ack=5793 Win=42496 Len=0 TSval=512406460 TSecr=3518115425
 14 0.045057496   10.0.0.1   10.0.0.2     HTTP       233 HTTP/1.1 200 OK  (manifest/mpd)
 15 0.045077806   10.0.0.2   10.0.0.1     TCP         66 41114 → 1337 [ACK] Seq=139 Ack=5960 Win=42496 Len=0 TSval=512406472 TSecr=3518115425
 16 0.055435094   10.0.0.2   10.0.0.1     TCP         74 41116 → 1337 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=512406482 TSecr=0 WS=512
 17 0.060686331   10.0.0.1   10.0.0.2     TCP         74 1337 → 41116 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=3518115477 TSecr=512406482 WS=512
 18 0.060719833   10.0.0.2   10.0.0.1     TCP         66 41116 → 1337 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=512406488 TSecr=3518115477
 19 0.060747433   10.0.0.2   10.0.0.1     HTTP       201 GET /bbb_30fps_320x180_200k_dashinit.mp4 HTTP/1.1
 20 0.061869470   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41116 [ACK] Seq=1 Ack=136 Win=43520 Len=1448 TSval=3518115481 TSecr=512406488 [TCP segment of a reassembled PDU]
 21 0.061893608   10.0.0.2   10.0.0.1     TCP         66 41116 → 1337 [ACK] Seq=136 Ack=1449 Win=42496 Len=0 TSval=512406489 TSecr=3518115481
 22 0.061914533   10.0.0.1   10.0.0.2     MP4        256
 23 0.061924553   10.0.0.2   10.0.0.1     TCP         66 41116 → 1337 [ACK] Seq=136 Ack=1639 Win=42496 Len=0 TSval=512406489 TSecr=3518115481
 24 0.062212692   10.0.0.2   10.0.0.1     TCP         66 41116 → 1337 [FIN, ACK] Seq=136 Ack=1639 Win=42496 Len=0 TSval=512406489 TSecr=3518115481
 25 0.063231297   10.0.0.1   10.0.0.2     TCP         66 1337 → 41116 [FIN, ACK] Seq=1639 Ack=137 Win=43520 Len=0 TSval=3518115481 TSecr=512406489
 26 0.063261013   10.0.0.2   10.0.0.1     TCP         66 41116 → 1337 [ACK] Seq=137 Ack=1640 Win=42496 Len=0 TSval=512406490 TSecr=3518115481
 27 0.063955409   10.0.0.1   10.0.0.2     TCP         66 1337 → 41116 [ACK] Seq=1 Ack=136 Win=43520 Len=0 TSval=3518115480 TSecr=512406488
 28 0.063973102   10.0.0.2   10.0.0.1     TCP         66 [TCP Dup ACK 26#1] 41116 → 1337 [ACK] Seq=137 Ack=1640 Win=42496 Len=0 TSval=512406491 TSecr=3518115481
 29 0.064288297   10.0.0.1   10.0.0.2     TCP         54 1337 → 41116 [RST] Seq=1640 Win=0 Len=0
 30 0.066433201   10.0.0.2   10.0.0.1     TCP         74 41118 → 1337 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=512406493 TSecr=0 WS=512
 31 0.069918852   10.0.0.1   10.0.0.2     TCP         74 1337 → 41118 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=3518115487 TSecr=512406493 WS=512
 32 0.069934567   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=512406497 TSecr=3518115487
 33 0.069958796   10.0.0.2   10.0.0.1     HTTP       198 GET /bbb_30fps_320x180_200k_dash1.m4s HTTP/1.1
 34 0.073423620   10.0.0.1   10.0.0.2     TCP         66 1337 → 41118 [ACK] Seq=1 Ack=133 Win=43520 Len=0 TSval=3518115492 TSecr=512406497
 35 0.073895608   10.0.0.1   10.0.0.2     TCP       4410 1337 → 41118 [ACK] Seq=1 Ack=133 Win=43520 Len=4344 TSval=3518115493 TSecr=512406497 [TCP segment of a reassembled PDU]
 36 0.073907472   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=133 Ack=4345 Win=41472 Len=0 TSval=512406501 TSecr=3518115493
 37 0.102176416   10.0.0.1   10.0.0.2     TCP       4410 1337 → 41118 [ACK] Seq=4345 Ack=133 Win=43520 Len=4344 TSval=3518115493 TSecr=512406497 [TCP segment of a reassembled PDU]
 38 0.102201572   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=133 Ack=8689 Win=40960 Len=0 TSval=512406529 TSecr=3518115493
 39 0.138512528   10.0.0.1   10.0.0.2     TCP       4410 1337 → 41118 [ACK] Seq=8689 Ack=133 Win=43520 Len=4344 TSval=3518115493 TSecr=512406497 [TCP segment of a reassembled PDU]
 40 0.138540840   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=133 Ack=13033 Win=40960 Len=0 TSval=512406565 TSecr=3518115493
 41 0.174853528   10.0.0.1   10.0.0.2     TCP       1514 1337 → 41118 [ACK] Seq=13033 Ack=133 Win=43520 Len=1448 TSval=3518115557 TSecr=512406565 [TCP segment of a reassembled PDU]
 42 0.174886558   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=133 Ack=14481 Win=42496 Len=0 TSval=512406602 TSecr=3518115557
 43 0.186962796   10.0.0.1   10.0.0.2     TCP       2962 1337 → 41118 [ACK] Seq=14481 Ack=133 Win=43520 Len=2896 TSval=3518115557 TSecr=512406565 [TCP segment of a reassembled PDU]
 44 0.186991861   10.0.0.2   10.0.0.1     TCP         66 41118 → 1337 [ACK] Seq=133 Ack=17377 Win=41984 Len=0 TSval=512406614 TSecr=3518115557
 45 0.211187077   10.0.0.1   10.0.0.2     TCP       2962 1337 → 41118 [ACK] Seq=17377 Ack=133 Win=43520 Len=2896 TSval=3518115557 TSecr=512406565 [TCP segment of a reassembled PDU]

▸ Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▸ Ethernet II, Src: 26:c7:79:07:86:9a (26:c7:79:07:86:9a), Dst: ee:c3:e9:8d:3d:25 (ee:c3:e9:8d:3d:25)
▸ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
▸ Transmission Control Protocol, Src Port: 41114, Dst Port: 1337, Seq: 0, Len: 0

0000  ee c3 e9 8d 3d 25 26 c7  79 07 86 9a 08 00 45 00   ····=%&· y·····E·
0010  00 40 6c 06 40 00 40 06  ba af 0a 00 00 02 0a 00   ·@l·@·@· ········
0020  00 01 a0 9a 05 39 76 67  ea b2 00 00 00 00 b0 02   ·····9vg ········
0030  a5 64 14 35 00 00 02 04  05 b4 04 02 08 0a 1e 8a   ·d·5···· ········
0040  b3 9b 00 00 00 00 01 03  03 09 22 02 01 01         ········ ··"···
```

Figure 35: Main Page of Wireshark

There are six default columns in Wireshark. At first, “No.” column. Each frame has a unique number from the beginning of the pcap. The first one is always number 1. Next, the “Time” column which breaks down time to nanoseconds from the beginning of the pcap. The first frame is always 0.000000, too. The “Source” column represents the address of the source which can be an IPv4, IPv6, or Ethernet address. Moreover, the “Destination” column represents the address of the destination which can also be an IPv4, IPv6, or Ethernet address. The ”Protocol” column saves the used protocol in the Ethernet frame, IP packet or TCP segment. The protocol can be anything among the following: ARP, DNS, TCP, HTTP, etc. Finally, the “Length” column stands for the length of the frame in bytes.

Next, we thoroughly present all the adaptations made, so as to display exactly the data we needed. We right-click on any column header and check the columns that we want to display or uncheck the columns that we desire to hide. For the experiments conducted in our thesis we hide the columns No. and Length (Figure 36) as the information does not come in handy [31].

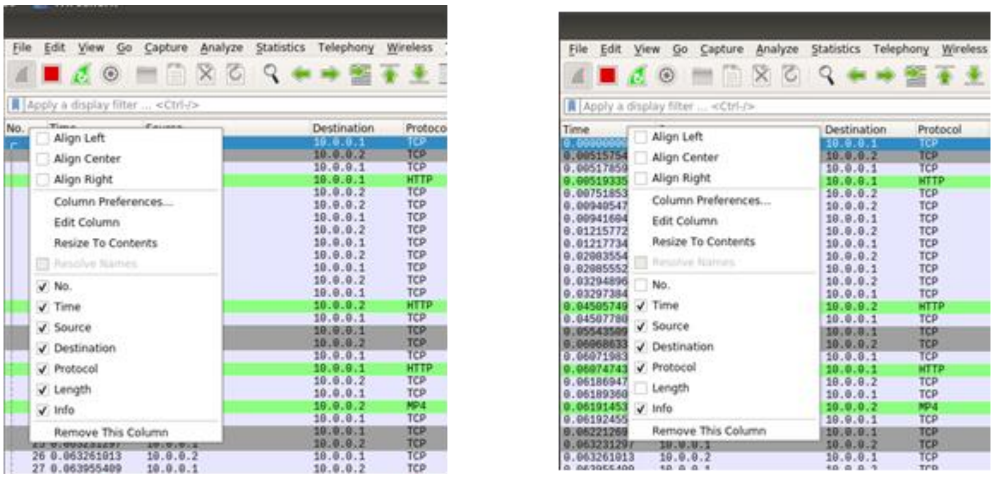
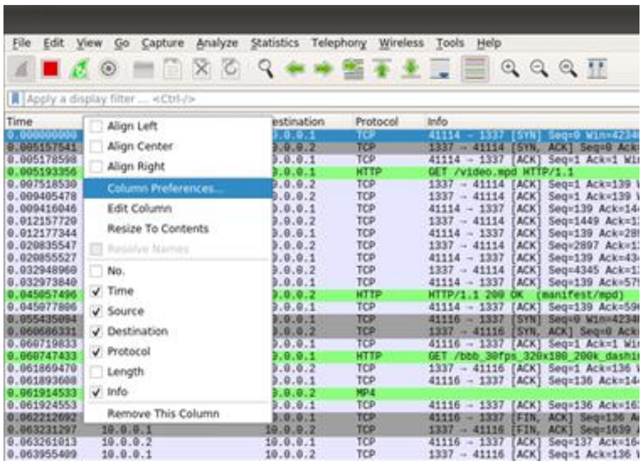


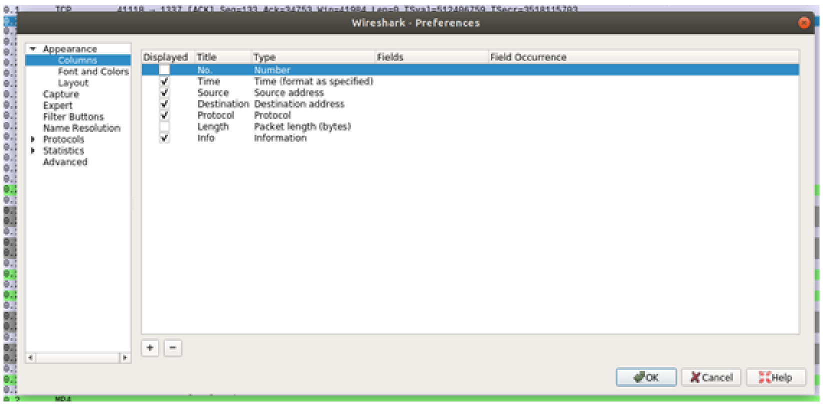
Figure 36: Needed Columns

In order to add a new column, we right-click on any column header and select Column Preferences as shown in Figure 37 and Figure 38.



```
Time                    estination  Protocol  Info
0.000000000             .0.0.1      TCP       41114 → 1337 [SYN] Seq=0 Win=4234
0.005157541             .0.0.2      TCP       1337 → 41114 [SYN, ACK] Seq=0 Ack
0.005178598             .0.0.1      TCP       41114 → 1337 [ACK] Seq=1 Ack=1 Wi
0.005193356             .0.0.1      HTTP      GET /video.mpd HTTP/1.1
0.007518530             .0.0.2      TCP       1337 → 41114 [ACK] Seq=1 Ack=139
0.009405478             .0.0.2      TCP       1337 → 41114 [ACK] Seq=1 Ack=139
0.009416046             .0.0.1      TCP       41114 → 1337 [ACK] Seq=139 Ack=14
0.012157720             .0.0.2      TCP       1337 → 41114 [ACK] Seq=1449 Ack=1
0.012177344             .0.0.1      TCP       41114 → 1337 [ACK] Seq=139 Ack=28
0.020835547             .0.0.2      TCP       1337 → 41114 [ACK] Seq=2897 Ack=1
0.020855527             .0.0.1      TCP       41114 → 1337 [ACK] Seq=139 Ack=43
0.032948960             .0.0.2      TCP       1337 → 41114 [ACK] Seq=4345 Ack=1
0.032973840             .0.0.1      TCP       41114 → 1337 [ACK] Seq=139 Ack=57
0.045057496             .0.0.2      HTTP      HTTP/1.1 200 OK  (manifest/mpd)
0.045077806             .0.0.1      TCP       41114 → 1337 [ACK] Seq=139 Ack=59
0.055435094             .0.0.1      TCP       41116 → 1337 [SYN] Seq=0 Win=4234
0.060686331             .0.0.2      TCP       1337 → 41116 [SYN, ACK] Seq=0 Ack
0.060719833             .0.0.1      TCP       41116 → 1337 [ACK] Seq=1 Ack=1 Wi
0.060747433             .0.0.1      HTTP      GET /bbb_30fps_320x180_200k_dashi
0.061869470             .0.0.2      TCP       1337 → 41116 [ACK] Seq=1 Ack=136
0.061893608             .0.0.1      TCP       41116 → 1337 [ACK] Seq=136 Ack=14
0.061914533             .0.0.2      MP4
0.061924553             .0.0.1      TCP       41116 → 1337 [ACK] Seq=136 Ack=16
0.062212692             .0.0.1      TCP       41116 → 1337 [FIN, ACK] Seq=136 A
0.063231297  10.0.0.1   10.0.0.2    TCP       1337 → 41116 [FIN, ACK] Seq=1639
0.063261013  10.0.0.2   10.0.0.1    TCP       41116 → 1337 [ACK] Seq=137 Ack=16
0.063955409  10.0.0.1   10.0.0.2    TCP       1337 → 41116 [ACK] Seq=1 Ack=136
```

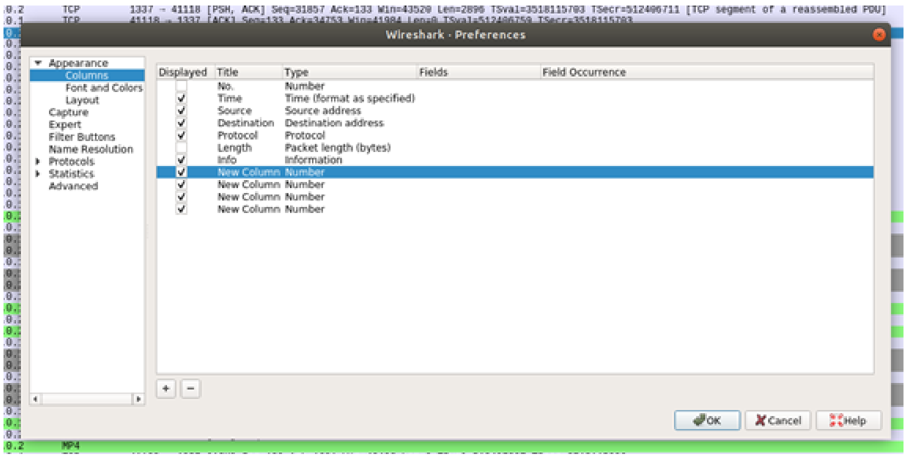
Figure 37: Column Preferences Option



Displayed	Title	Type	Fields	Field Occurrence
	No.	Number		
✓	Time	Time (format as specified)		
✓	Source	Source address		
✓	Destination	Destination address		
✓	Protocol	Protocol		
	Length	Packet length (bytes)		
✓	Info	Information		

Figure 38: Wireshark Preferences (1)

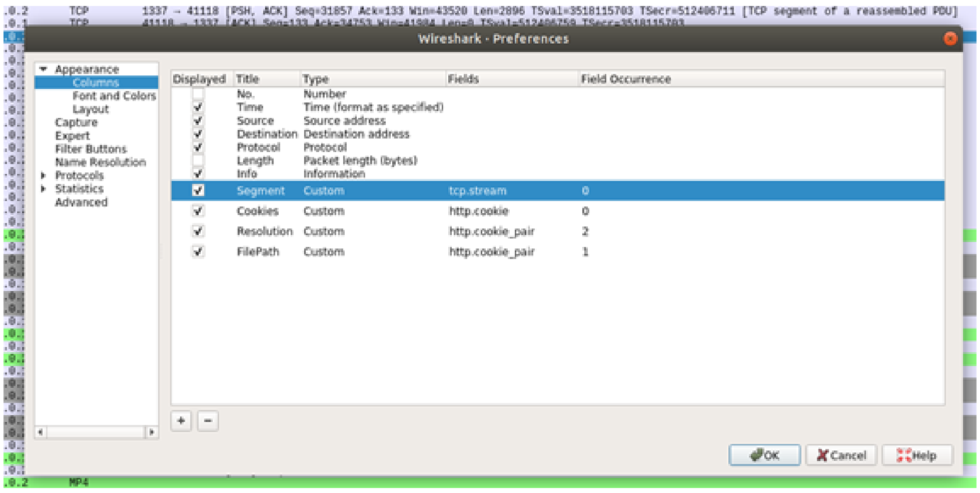
Next, we select the '+' button four times, to add 4 new columns as shown in Figure 39.



Displayed	Title	Type	Fields	Field Occurrence
	No.	Number		
✓	Time	Time (format as specified)		
✓	Source	Source address		
✓	Destination	Destination address		
✓	Protocol	Protocol		
	Length	Packet length (bytes)		
✓	Info	Information		
✓	New Column	Number		
✓	New Column	Number		
✓	New Column	Number		
✓	New Column	Number		

Figure 39: New Columns

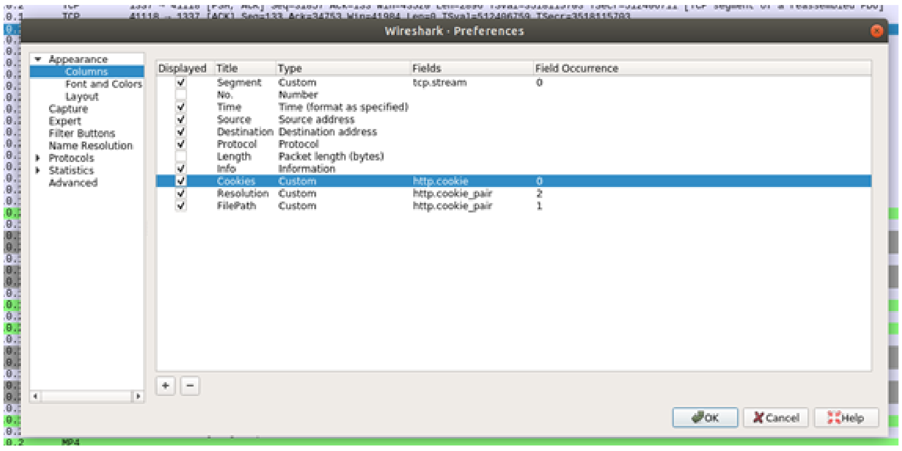
We name these new columns Segment, Cookies, Resolution and FilePath. The column Segment shows a full picture of the request made not only from the client to the server but also from the server to the client. The Cookie column demonstrates extra information that the server sends to the client apart from the object itself. This information is actually the Resolution of the video as well as the FilePath of the file which can also be seen in two different columns separately, the Resolution and the FilePath column. We also edit the Type, Fields and Fields Occurrence fields as shown in Figure 40:



Displayed	Title	Type	Fields	Field Occurrence
	No.	Number		
✓	Time	Time (format as specified)		
✓	Source	Source address		
✓	Destination	Destination address		
✓	Protocol	Protocol		
	Length	Packet length (bytes)		
✓	Info	Information		
✓	Segment	Custom	tcp.stream	0
✓	Cookies	Custom	http.cookie	0
✓	Resolution	Custom	http.cookie_pair	2
✓	FilePath	Custom	http.cookie_pair	1

Figure 40: Wireshark Preferences (2)

It is also possible to change the order of the columns by clicking and dragging them wherever we want. For example, we can transfer the Segment column to the top as can be seen in Figure 41.



Displayed	Title	Type	Fields	Field Occurrence
✓	Segment	Custom	tcp.stream	0
	No.	Number		
✓	Time	Time (format as specified)		
✓	Source	Source address		
✓	Destination	Destination address		
✓	Protocol	Protocol		
	Length	Packet length (bytes)		
✓	Info	Information		
✓	Cookies	Custom	http.cookie	0
✓	Resolution	Custom	http.cookie_pair	2
✓	FilePath	Custom	http.cookie_pair	1

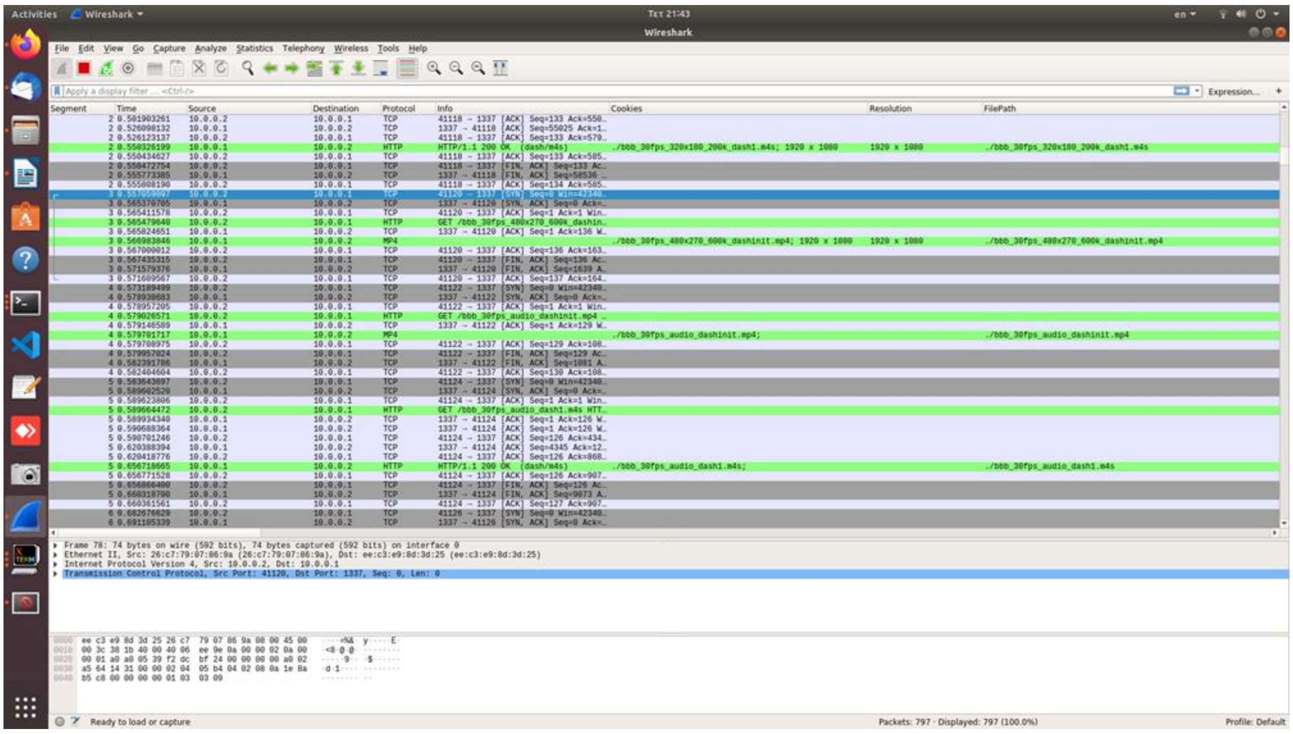
Figure 41: Wireshark Preferences (3)

We click OK and save the changes made. The Wireshark main page will look this way (Figure 42) in the end.

Finally, the table shows all the tools used in our thesis, together with their versions:

Table 1: Used tools

TOOL	VERSION
Java Virtual Machine (JVM)	1.8.0_222
SDN Controller	Boron SR4
Mininet	2.3.0d6
VLC	3.0.8 Vetinari
FFmpeg	3.4.6-0ubuntu0.18.04.1
Nodejs	v8.10.0
MP4Box – GPAC	0.5.2-DEV-revVersion
Wireshark	2.6.10



```
Segment Time         Source    Destination Protocol Info                                      Cookies                                                Resolution   FilePath
      2 0.501903261  10.0.0.2  10.0.0.1    TCP      41118 → 1337 [ACK] Seq=133 Ack=550…
      2 0.526098132  10.0.0.1  10.0.0.2    TCP      1337 → 41118 [ACK] Seq=55025 Ack=1…
      2 0.526123137  10.0.0.2  10.0.0.1    TCP      41118 → 1337 [ACK] Seq=133 Ack=579…
      2 0.550326199  10.0.0.1  10.0.0.2    HTTP     HTTP/1.1 200 OK  (dash/m4s)               ./bbb_30fps_320x180_200k_dash1.m4s; 1920 x 1080        1920 x 1080  ./bbb_30fps_320x180_200k_dash1.m4s
      2 0.550434627  10.0.0.2  10.0.0.1    TCP      41118 → 1337 [ACK] Seq=133 Ack=585…
      2 0.550472754  10.0.0.2  10.0.0.1    TCP      41118 → 1337 [FIN, ACK] Seq=133 Ac…
      2 0.555773385  10.0.0.1  10.0.0.2    TCP      1337 → 41118 [FIN, ACK] Seq=58536 …
      2 0.555808190  10.0.0.2  10.0.0.1    TCP      41118 → 1337 [ACK] Seq=134 Ack=585…
      3 0.557659097  10.0.0.2  10.0.0.1    TCP      41120 → 1337 [SYN] Seq=0 Win=42340…
      3 0.565370705  10.0.0.1  10.0.0.2    TCP      1337 → 41120 [SYN, ACK] Seq=0 Ack=…
      3 0.565411578  10.0.0.2  10.0.0.1    TCP      41120 → 1337 [ACK] Seq=1 Ack=1 Win…
      3 0.565479640  10.0.0.2  10.0.0.1    HTTP     GET /bbb_30fps_480x270_600k_dashin…
      3 0.565824651  10.0.0.1  10.0.0.2    TCP      1337 → 41120 [ACK] Seq=1 Ack=136 W…
      3 0.566983846  10.0.0.1  10.0.0.2    MP4                                                ./bbb_30fps_480x270_600k_dashinit.mp4; 1920 x 1080     1920 x 1080  ./bbb_30fps_480x270_600k_dashinit.mp4
      3 0.567000012  10.0.0.2  10.0.0.1    TCP      41120 → 1337 [ACK] Seq=136 Ack=163…
      3 0.567435315  10.0.0.2  10.0.0.1    TCP      41120 → 1337 [FIN, ACK] Seq=136 Ac…
      3 0.571579376  10.0.0.1  10.0.0.2    TCP      1337 → 41120 [FIN, ACK] Seq=1639 A…
      3 0.571609567  10.0.0.2  10.0.0.1    TCP      41120 → 1337 [ACK] Seq=137 Ack=164…
      4 0.573189499  10.0.0.2  10.0.0.1    TCP      41122 → 1337 [SYN] Seq=0 Win=42340…
      4 0.578930683  10.0.0.1  10.0.0.2    TCP      1337 → 41122 [SYN, ACK] Seq=0 Ack=…
      4 0.578957205  10.0.0.2  10.0.0.1    TCP      41122 → 1337 [ACK] Seq=1 Ack=1 Win…
      4 0.579026571  10.0.0.2  10.0.0.1    HTTP     GET /bbb_30fps_audio_dashinit.mp4 …
      4 0.579146589  10.0.0.1  10.0.0.2    TCP      1337 → 41122 [ACK] Seq=1 Ack=129 W…
      4 0.579701717  10.0.0.1  10.0.0.2    MP4                                                ./bbb_30fps_audio_dashinit.mp4;                                     ./bbb_30fps_audio_dashinit.mp4
      4 0.579708975  10.0.0.2  10.0.0.1    TCP      41122 → 1337 [ACK] Seq=129 Ack=108…
      4 0.579957024  10.0.0.2  10.0.0.1    TCP      41122 → 1337 [FIN, ACK] Seq=129 Ac…
      4 0.582391786  10.0.0.1  10.0.0.2    TCP      1337 → 41122 [FIN, ACK] Seq=1081 A…
      4 0.582404604  10.0.0.2  10.0.0.1    TCP      41122 → 1337 [ACK] Seq=130 Ack=108…
      5 0.583643697  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [SYN] Seq=0 Win=42340…
      5 0.589602520  10.0.0.1  10.0.0.2    TCP      1337 → 41124 [SYN, ACK] Seq=0 Ack=…
      5 0.589623806  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [ACK] Seq=1 Ack=1 Win…
      5 0.589664472  10.0.0.2  10.0.0.1    HTTP     GET /bbb_30fps_audio_dash1.m4s HTT…
      5 0.589934340  10.0.0.1  10.0.0.2    TCP      1337 → 41124 [ACK] Seq=1 Ack=126 W…
      5 0.590688364  10.0.0.1  10.0.0.2    TCP      1337 → 41124 [ACK] Seq=1 Ack=126 W…
      5 0.590701246  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [ACK] Seq=126 Ack=434…
      5 0.620388394  10.0.0.1  10.0.0.2    TCP      1337 → 41124 [ACK] Seq=4345 Ack=12…
      5 0.620418776  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [ACK] Seq=126 Ack=868…
      5 0.656718665  10.0.0.1  10.0.0.2    HTTP     HTTP/1.1 200 OK  (dash/m4s)               ./bbb_30fps_audio_dash1.m4s;                                        ./bbb_30fps_audio_dash1.m4s
      5 0.656771528  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [ACK] Seq=126 Ack=907…
      5 0.656866400  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [FIN, ACK] Seq=126 Ac…
      5 0.660318700  10.0.0.1  10.0.0.2    TCP      1337 → 41124 [FIN, ACK] Seq=9073 A…
      5 0.660361561  10.0.0.2  10.0.0.1    TCP      41124 → 1337 [ACK] Seq=127 Ack=907…
      6 0.682676629  10.0.0.2  10.0.0.1    TCP      41126 → 1337 [SYN] Seq=0 Win=42340…
      6 0.691105339  10.0.0.1  10.0.0.2    TCP      1337 → 41126 [SYN, ACK] Seq=0 Ack=…

▸ Frame 78: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: 26:c7:79:07:86:9a (26:c7:79:07:86:9a), Dst: ee:c3:e9:8d:3d:25 (ee:c3:e9:8d:3d:25)
▸ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
▸ Transmission Control Protocol, Src Port: 41120, Dst Port: 1337, Seq: 0, Len: 0

0000  ee c3 e9 8d 3d 25 26 c7  79 07 86 9a 08 00 45 00   ····=%&· y·····E·
0010  00 3c 38 1b 40 00 40 06  ee 9e 0a 00 00 02 0a 00   ·<8·@·@· ········
0020  00 01 a0 a0 05 39 f2 dc  bf 24 00 00 00 00 a0 02   ·····9·· ·$······
0030  a5 64 14 31 00 00 02 04  05 b4 04 02 08 0a 1e 8a   ·d·1···· ········
0040  b5 c8 00 00 00 00 01 03  03 09                     ········ ··

Ready to load or capture          Packets: 797 · Displayed: 797 (100.0%)          Profile: Default
```

Figure 42: Wireshark Main Page in the end

4.2 SERVER SETUP

Below we present a simple web server, created in node.js which responds "Hello World" to any request.

```
var http = require('http');
http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Hello World\n');
        }).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

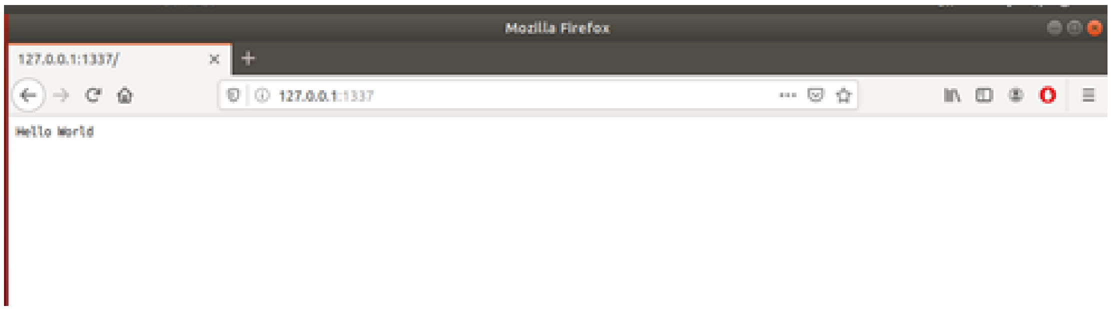
In order to include the Hyper Text Transfer Protocol (HTTP) module we need to add the following method:

```
var http = require('http');
```

The HTTP module enables us to create an HTTP server that listens to a specific port. The server then is able to respond whenever the client makes a request. The creation of such a server is made by using the method createServer(). According to the previous code, we create a server that listens to http://127.0.0.1 or localhost to the port 1337. The server responds to each request with its status code which in our case is 200 that means success. It also sends back the response headers which is the Content-Type argument. We can save the server that we have just created to a file named "example.js" and test it with the following command:

```
node example.js
```

Figure 43 shows the output. For our demonstration we have chosen to use the Firefox Web Browser.



Hello World

Figure 43: example.js output

4.2.1 Making requests in Node.js

Requests can also be made in the node.js environment. For instance, we can rename the previous web server file to “server.js” and in addition we can create a new file called “client.js” which contains the following.

At first, util is necessary for many different operations in the node.js environment, so it should be placed at the beginning of the code.

```
const http = require('http');

const util = require('util');
```

The request() method needs the following mandatory settings. Hostname is actually the localhost or in other words 127.0.0.1, the port requested is 1337 and finally the method can be among the following: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS or TRACE. For this example, we use the method GET. This method is used to request data from a specified resource and it is one of the most common methods. The data can be an htlm page, an audio file etc.

```
const options = {
        hostname: 'localhost',
        port: 1337,
        path: '/',
        method: 'GET'
};
```

The command console.log is used to check the outcome of the request.

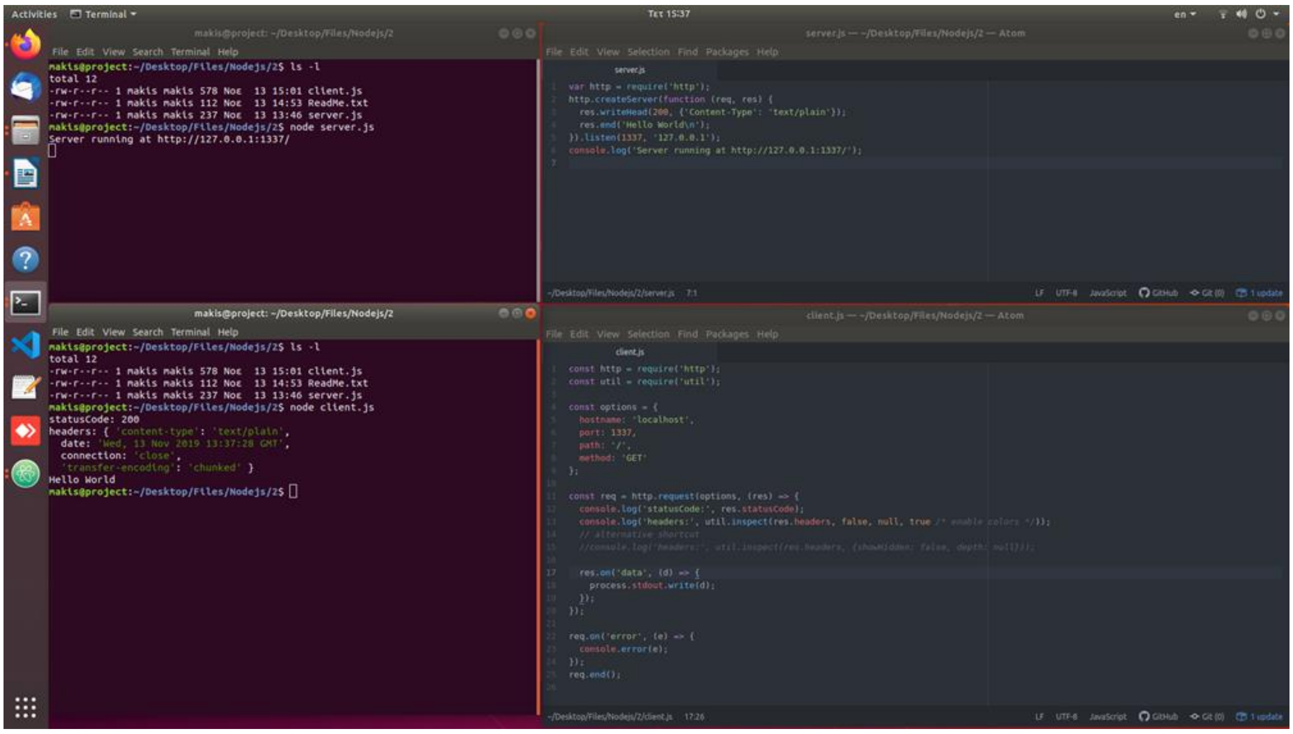
```
const req = http.request(options, (res) => {
  console.log('statusCode:', res.statusCode);
  console.log('headers:', util.inspect(res.headers, false, null, true /* enable colors */));
  // alternative shortcut
  //console.log('headers:', util.inspect(res.headers, {showHidden: false, depth: null}));


  res.on('data', (d) => {
    process.stdout.write(d);
  });
});
```

```
req.on('error', (e) => {
  console.error(e);
});
req.end();
```

The client can make the request by typing the following to the terminal:

```
 node client.js
```

Figure 44 shows the procedure in every detail as well as the results given in the command line. The text editor used is Atom.


Activities
Terminal
Τετ 15:37
makis@project: ~/Desktop/Files/Nodejs/2
File Edit View Search Terminal Help
server.js — ~/Desktop/Files/Nodejs/2 — Atom
File Edit View Selection Find Packages Help
~/Desktop/Files/Nodejs/2/server.js 7:1
client.js — ~/Desktop/Files/Nodejs/2 — Atom
~/Desktop/Files/Nodejs/2/client.js 17:26
LF UTF-8 JavaScript GitHub Git (0) 1 update

```
makis@project:~/Desktop/Files/Nodejs/2$ ls -l
total 12
-rw-r--r-- 1 makis makis 578 Νοε  13 15:01 client.js
-rw-r--r-- 1 makis makis 112 Νοε  13 14:53 ReadMe.txt
-rw-r--r-- 1 makis makis 237 Νοε  13 13:46 server.js
makis@project:~/Desktop/Files/Nodejs/2$ node server.js
Server running at http://127.0.0.1:1337/
```

server.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

```
makis@project:~/Desktop/Files/Nodejs/2$ ls -l
total 12
-rw-r--r-- 1 makis makis 578 Νοε  13 15:01 client.js
-rw-r--r-- 1 makis makis 112 Νοε  13 14:53 ReadMe.txt
-rw-r--r-- 1 makis makis 237 Νοε  13 13:46 server.js
makis@project:~/Desktop/Files/Nodejs/2$ node client.js
statusCode: 200
headers: { 'content-type': 'text/plain',
  date: 'Wed, 13 Nov 2019 13:37:28 GMT',
  connection: 'close',
  'transfer-encoding': 'chunked' }
Hello World
makis@project:~/Desktop/Files/Nodejs/2$
```

client.js

```
const http = require('http');
const util = require('util');

const options = {
  hostname: 'localhost',
  port: 1337,
  path: '/',
  method: 'GET'
};

const req = http.request(options, (res) => {
  console.log('statusCode:', res.statusCode);
  console.log('headers:', util.inspect(res.headers, false, null, true /* enable colors */));
  // alternative shortcut
  //console.log('headers:', util.inspect(res.headers, {showHidden: false, depth: null}));

  res.on('data', (d) => {
    process.stdout.write(d);
  });
});

req.on('error', (e) => {
  console.error(e);
});
req.end();
```

Figure 44: The procedure described

4.2.2 Proposed server implementation

Based on the knowledge gained in the previous sections, we are now able to present the server used in our thesis.

```
var http = require('http');
var fs = require('fs');
var path = require('path');
var fs2 = require('fs');
```

Firstly, the server creates a folder named statistics. In this folder a .csv file will be created after the completion of the procedure, which contains several statistic information about it. The created file is called ‘filePath + ','+ myCurrentTime().csv ‘ as shown below.

```
//Create directoty for statistics of our server
var dir = './statistics';
```

```
if (!fs2.existsSync(dir)){
   fs2.mkdirSync(dir);
}

//Start write to new File
//let writeStream = fs2.createWriteStream('statistics/results_'+ myCurrentDate() +'.csv');
//writeStream.write('FilePath,DateTime\n');

var filePath = '';
var server = http.createServer(function (request, response) {
```

The following part of the code is executed when the client makes a request. In case a segment file is requested, the data is inserted in the file described above.

```
   filePath = '.' + request.url;
   var datetimeRequestStart = myCurrentDate();

   if(filePath.includes(".mpd") === true){
      console.log('(manifest) : request starting... : ' + filePath + '\n');
   }
   else if ( filePath.includes("bbb_30fps_audio") === true){
      console.log('(audio) : request starting... : ' + filePath + '\n');
   }
   else if ( filePath.includes("bbb_30fps") === true){
      console.log('(segment) : request starting... : ' + filePath + '\n');
      // write to a new line in file named secret.txt
      //writeStream.write(filePath + ','+ myCurrentTime()  +  '\n');
   }

   if (filePath == './')
      filePath = './index.html';
```

The following code is executed in order to identify the content type of the requested object. The general type of its object is specified as text/html.

```
   var extname = path.extname(filePath);
   var contentType = 'text/html';
   switch (extname) {
          case '.m4s':
         contentType = 'dash/m4s';
```

```
            break;
       case '_dashinit.mp4':
            contentType = 'dashinit/mp4';
            break;
        case '.mpd':
            contentType = 'manifest/mpd';
            break;
        case '.mp4':
            contentType = 'video/mp4';
            break;
        case '.m4a':
            contentType = 'audio/m4a';
            break;
    }
```

The data is sent to the client accompanied with an appropriate message. In case of an ENOENT error, the server responds ‘404 Not Found’. ENOENT errors occur when the server does not possess the requested data.

```
    fs.readFile(filePath, function(error, content) {
        if (error) {
            if(error.code == 'ENOENT'){
                fs.readFile('./404.html', function(error, content) {
                    response.writeHead(404, {"Content-Type": "text/html"});
                    response.write("404 Not Found\n");
                    response.end(content, 'utf-8');
                });
            }
```

The server responds ‘Sorry, check with the site admin for error: '+error.code+' ..’ where error.code stands for 500.

```
            else {
                response.writeHead(500);
                response.end('Sorry, check with the site admin for error: '+error.code+' ..\n');
                response.end();
            }
        }
```

The server responds to a successfully made request with the status code 200. It also sends the requested object as well as some relevant information about it in the format of a Cookie. This information contains the name of the object and the resolution when its type is relevant.

```
        else {
          //response.setHeader('Cookie', ['filePath=' + filePath, 'resolution=' + myResolution(filePath)]);
          response.setHeader('Cookie', [ filePath, myResolution(filePath)]);
          response.writeHead(200, { 'Content-Type': contentType  });
          response.end(content, 'utf-8');
        }
    });
  }).listen(1337, '10.0.0.1');
  console.log("Running node.js %s on %s-%s", process.version, process.platform, process.arch);
  console.log('Server running at http://10.0.0.1:1337/');


//}).listen(1337, '127.0.0.1');
//console.log("Running node.js %s on %s-%s", process.version, process.platform, process.arch);
//console.log('Server running at http://127.0.0.1:1337/');
```

The server terminates and also ends the procedure of the creation of the file which was described previously. In addition, an appropriate message appears, informing the user that the server is no longer online.

```
server.on('close', function() {
  // the finish event is emitted when all data has been flushed from the stream
  writeStream.on('finish', () => {
     console.log('Create a new file with statistics of the experiment.\n');
  });


  // close the stream
  writeStream.end();


  console.log('Server stopped.\n');
});
```

In case of an abrupt termination (kill), which means that the user has typed Ctrl + C or Ctrl + X, the process ends smoothly as shown below:

```
process.on('SIGINT', function() {
  console.log('User typed : ^C');
  server.close();
});
```

The following auxiliary functions were created so as to make our server more understandable.

The function myCurrentDate() returns the current date in this form YYYYMMDD_hh:mm:ss.

```
function myCurrentDate() {
  let date_ob = new Date();
  // current date
  // adjust 0 before single digit date
  let date = ("0" + date_ob.getDate()).slice(-2);
  // current month
  let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);
  // current year
  let year = date_ob.getFullYear();
  // current hours
  let hours = date_ob.getHours();
  // current minutes
  let minutes = date_ob.getMinutes();
  // current seconds
  let seconds = date_ob.getSeconds();
  // current date of server in YYYYMMDD_HH:MM:SS format 3
  let myDate = year  + month  + date + "_" + hours + ":" + minutes + ":" + seconds

  return myDate;             // Function returns myDate
}
```

The function myCurrentTime() returns the current time in this form hh:mm:ss.sss[4].

```
function myCurrentTime() {
  let date_ob = new Date();
  // current time
  // current hours
  let hours = date_ob.getHours();
  // current minutes
  let minutes = date_ob.getMinutes();
  // current seconds
```

[3] The abbreviation YYYYMMDD_hh:mm:ss stands for: YYY: year ,MM: month ,DD:day ,hh: hour ,mm: month ,ss: second.

[4] The abbreviation hh:mm:ss.sss stands for: hh: hour ,mm: month , ss.sss: second.

```
  let seconds = date_ob.getSeconds();
  // current milliseconds
  let milliseconds = date_ob.getMilliseconds();
  // current date of server in HH:MM:SS format
  let myTime = hours + ":" + minutes + ":" + seconds + '.' + milliseconds

  return myTime;            // Function returns myTime
}
```

According to the name of the file (filePath) the function myResolution(filePath) finds and returns the resolution of the video. The possible resolutions are '430 x 242', '640 x 360', '849 x 480', '1280 x 720' and '1920 x 1080'.

```
function myResolution(filePath){
  //Find data from: https://stackoverflow.com/questions/10003683/extract-get-a-number-from-a-string

  if(filePath.includes(".mpd") === true){
    return ''
  }
  else if ( filePath.includes("video_audio") === true){
    return ''
  }
  else if ( filePath.includes("video") === true){
    var myNum = filePath.replace(/^\D+|\D.*$/g, "");
    var myResolution = '1920 x 1080';
    if( myNum === '242')
      myResolution = '430 x 242';
    else if( myNum === '360')
      myResolution = '640 x 360';
    else if( myNum === '480')
      myResolution = '849 x 480';
    else if( myNum === '720')
      myResolution = '1280 x 720';
    else if( myNum === '1080')
      myResolution = '1920 x 1080';

      return myResolution
  }
}
```

The function simplifiedUnits(input), converts the number input to Bits[5].

```
var units = ["B", "kB", "MB", "TB"];
function simplifiedUnits(input) {
   var unit = units[0];
   var i = 0;
   while (input > 1024 && ++i) {
      unit = units[i];
      input /= 1024;
   }
   return Math.round(input) + " " + unit;
}
```

As mentioned before, the server can be executed with the following command [32][33][34]:

```
 node server.js
```

[5] Bits can be presented as "B", "kB", "MB", "TB".

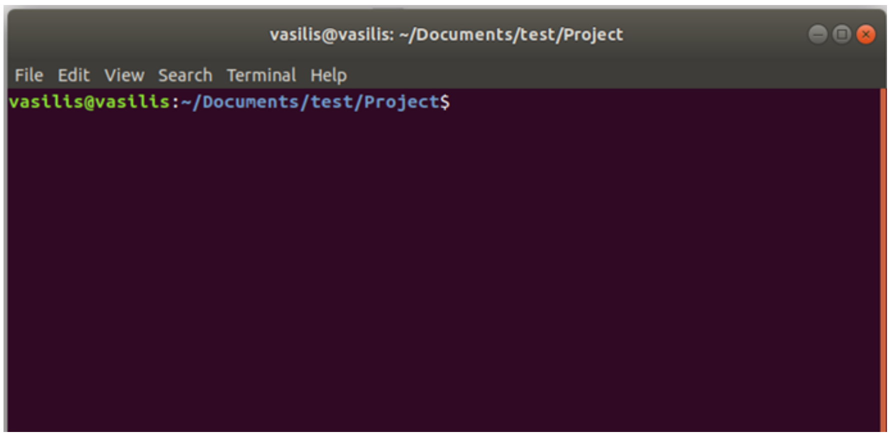
5. Mininet Experiments

In this chapter we thoroughly explain the experiments conducted in our thesis. These experiments were carried out in order to examine the Quality of Experience (QoE) on SDN networks, namely to understand the user's perspective.

5.1 BASIC EXPERIMENT

5.1.1 Preparation

In order to carry out an experiment we need a video with resolution 1920x1080 [35]. In the file where the video is located, we open a new terminal window (Figure 45).


vasilis@vasilis: ~/Documents/test/Project
File Edit View Search Terminal Help

```
vasilis@vasilis:~/Documents/test/Project$
```

Figure 45: Terminal window

Then, we need to create five different quality versions of the video as well as an audio file. Thus, we execute the "bash.sh" script, which we have created. This script converts every .mp4 file in the current folder and subfolder to a multi-bitrate video in MP4-DASH [36]. It creates five different quality versions, an audio file and an .mpd manifest file. Below, we explain in every detail the code written for this script.

1. In the beginning, we find the current directory name and save it to a variable.

```
MYDIR=$(dirname $(readlink -f ${BASH_SOURCE[0]}))
SAVEDIR=$(pwd)
```

2. Then, we check whether the needed programs, ffmpeg and MP4Box, are installed. If not, a message appears in the terminal informing the user.

```
# Check programs
if [ -z "$(which ffmpeg)" ]; then
    echo "Error: ffmpeg is not installed"
    exit 1
fi

if [ -z "$(which MP4Box)" ]; then
    echo "Error: MP4Box is not installed"
    exit 1
fi
```

3. Next, we change directory.

```
cd "$MYDIR"
```

4. Then, there is the main part of the script. We find all the .mp4 or .mov files in the current folder and subfolder. We save the name of each file without its extension (.mp4 or .mov) and we start converting each file to a multi-bitrate video in MPEG-DASH.

```
TARGET_FILES=$(find ./ -maxdepth 1 -type f \( -name "*.mov" -or -name "*.mp4" \))
for f in $TARGET_FILES
do
  fe=$(basename "$f") # fullname of the file
  f="${fe%.*}" # name without extension

  if [ ! -d "${f}" ]; then #if directory does not exist, convert
    echo "Converting \"$f\" to multi-bitrate video in MPEG-DASH"
```

5. At first, there is the conversion of the audio file.

```
ffmpeg -i "${fe}" -c:a copy -vn "${f}_audio.mp4"
```

- -i - the input file
- -c:a - the audio codec
- -vn - do not encode video

6. Next, there are the conversions of the video files to five different quality versions. The video is encoded using H.264 codec, so that it has a key frame every 24 frames. In other words, this lets us have the video segmented by chunks of 1 second in length. The bitrate is calculated according to the buffer size. It should be smaller than the rate since the segments are 1 second long, so as to ensure that the encoding is close to the requested rate. The frames in an H.264 video are organized in units called GOPs (Group Of Pictures) [37]. The frames are then divided into three different types, the I-frame that saves the entire picture, the P-frame which saves only the differences between the present and the earlier pictures and the B-frame that stores changes with previous or future pictures.

```
ffmpeg -i "${fe}" -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 5300k -maxrate 5300k -bufsize 2650k -vf 'scale=-1:1080' "${f}_1080.mp4"

  ffmpeg -i "${fe}" -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 2400k -maxrate 2400k -bufsize 1200k -vf 'scale=-1:720' "${f}_720.mp4"

  ffmpeg -i "${fe}" -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 1060k -maxrate 1060k -bufsize 530k -vf 'scale=-1:478' "${f}_480.mp4"

  ffmpeg -i "${fe}" -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 600k -maxrate 600k -bufsize 300k -vf 'scale=-1:360' "${f}_360.mp4"

  ffmpeg -i "${fe}" -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 260k -maxrate 260k -bufsize 130k -vf 'scale=-1:242' "${f}_242.mp4"
```

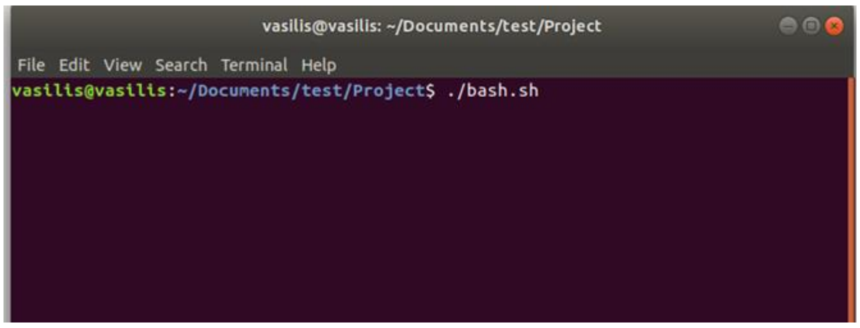
- -i - the input file

- -an - do not encode audio
- -c:v libx264 specifies the audio codec to use, in this case we want h264
- Keyint option defines the maximum length of the GOP or in other words the maximum interval between each keyframe
- min-keyint option defines the minimum length of the GOP. This is necessary as the encoder might insert a keyframe before the keyint value and in this way we can limit that
- no-scenecut option eliminates the case when the encoder adds additional I-frames, which also happen to use a lot of resources
- -b:v is the bitrate we want to encode the video to
- -maxrate the maximum bit rate
- -bufsize is the "rate control buffer"
- -vf 'scale -1:X' is used to resize the video, with X being the height of it
- Each new file is given a different name in combination with its original one

Now, there are one audio and five video files. Then, the Media Presentation Description (MPD) file is created. The MPD file stores all the data for the current project in a database format [38]. This file is like an index which mentions the different video and audio tracks available, along with their bitrate, size and order. The MPD file will then be loaded by the VLC player. The dash option defines the duration of each segment, which in our case is 1000 ms so as to match the 1 second segments of the videos. We add the flag -profile dashavc264:live to make it compatible with dash.js player.

```
        if [ -e "${f}_audio.mp4" ]; then
         MP4Box -dash 1000 -rap -frag-rap -profile onDemand  "${f}_1080.mp4" "${f}_720.mp4" "${f}_480.mp4" "${f}_360.mp4" "${f}_242.mp4" "${f}_audio.mp4" -out "${f}.mpd"
      fi
```

To execute the script we type the following (Figure 46):



```
vasilis@vasilis:~/Documents/test/Project$ ./bash.sh
```

vasilis@vasilis: ~/Documents/test/Project
File Edit View Search Terminal Help

Figure 46: bash.sh execution

Now, we have created the needed versions, the audio file, the mpd file and moreover many different segments of the video (.m4s) (Figure 47) [39].

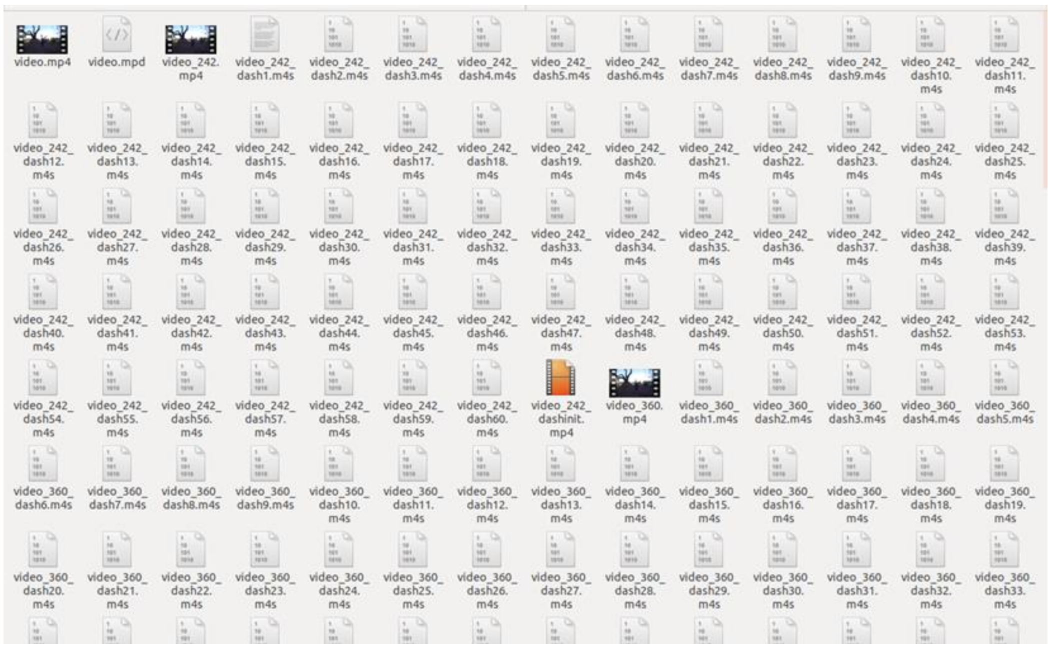


Figure 47: Different versions of the video, mpd file, different segments

5.1.2 Basic simulation

A custom topology in MiniEdit may look like this in Figure 48:

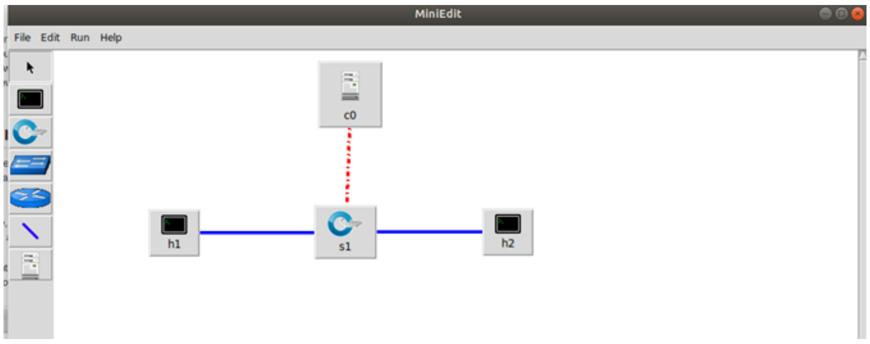
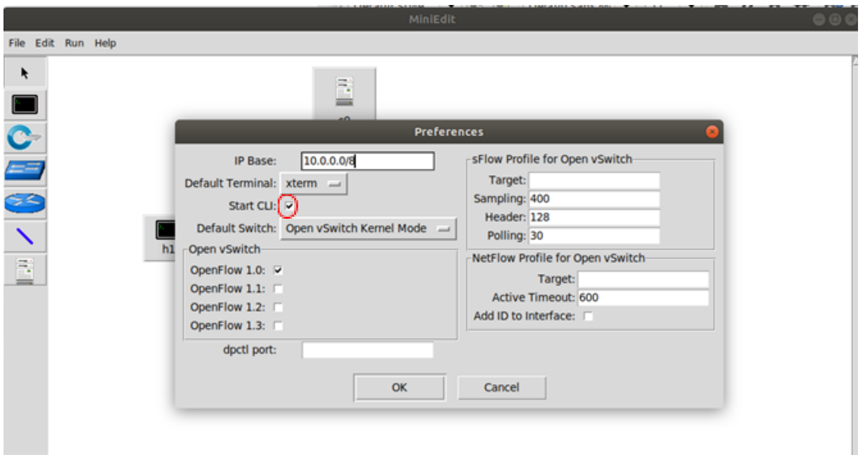


Figure 48: Custom topology

To set MiniEdit preferences, we must click on Edit → Preferences and check the Start CLI box (Figure 49):



IP Base: 10.0.0.0/8
Default Terminal: xterm
Start CLI: ✓
Default Switch: Open vSwitch Kernel Mode
Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1:
OpenFlow 1.2:
OpenFlow 1.3:
dpctl port:
sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30
NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface:

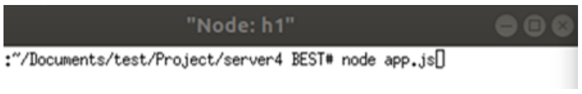
Figure 49: Mininet preferences

This is important as the MiniEdit console by default does not give access to the Mininet command line interface. However, if we check the Start CLI box, we are able to use the Mininet CLI during a simulation.

Now, we can run the experiment. In Mininet command line interface we execute the following command and the xterms of each host will appear:

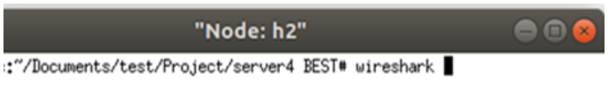
```
mininet> xterm h1 h2 h2
```

The h1 host represents the server and the h2 the client. The client not only does he need to execute the video player but also the wireshark program, so as to monitor the packets transmitted. For this reason, we execute the following commands in the different terminals as demonstrated below (Figure 50 - Figure 52), in the following order:



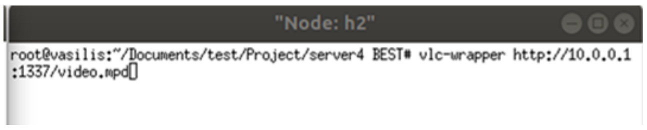
```
:~/Documents/test/Project/server4 BEST# node app.js
```

Figure 50: xterm h1



```
:~/Documents/test/Project/server4 BEST# wireshark
```

Figure 51: xterm h2 (1)



```
root@vasilis:~/Documents/test/Project/server4 BEST# vlc-wrapper http://10.0.0.1
:1337/video.mpd
```

Figure 52: xterm h2 (2)

First, we start the server in host 1. Secondly, we start monitoring the packets, open the VLC player and load the MPD file in the client in host 2. Finally, we are able to watch the video playing (Figure 53) as well as the output of the server terminal (Figure 54).

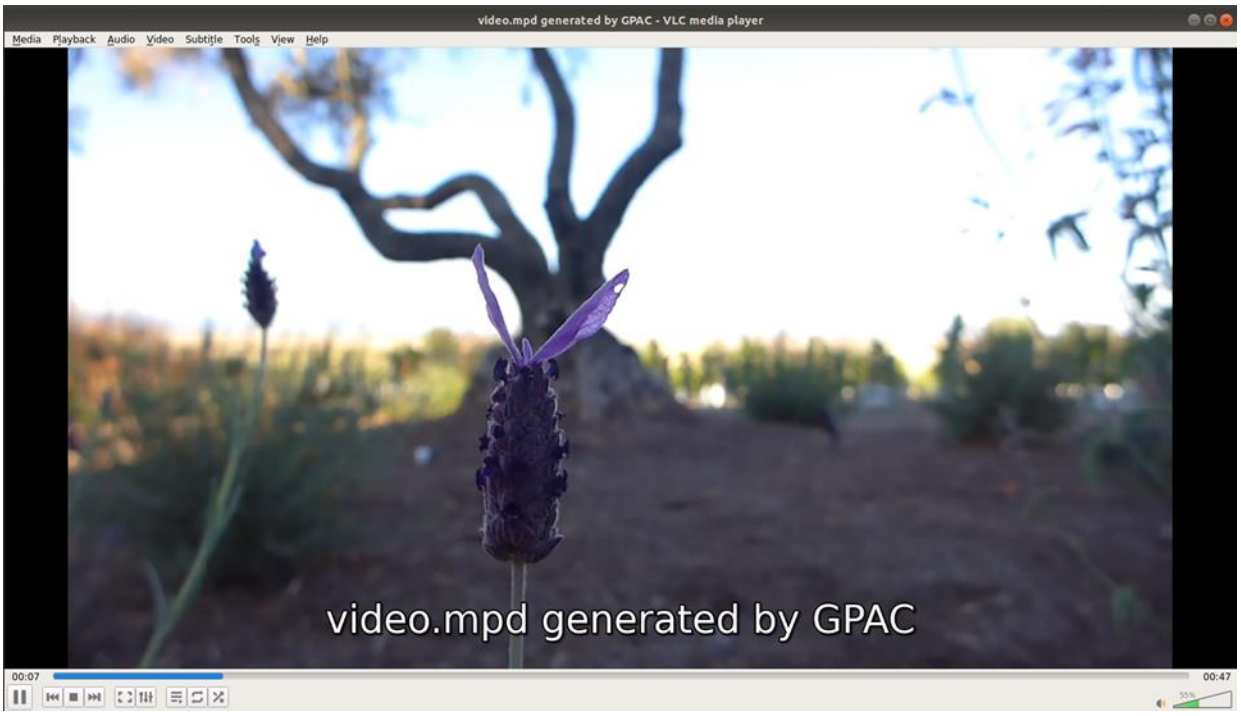
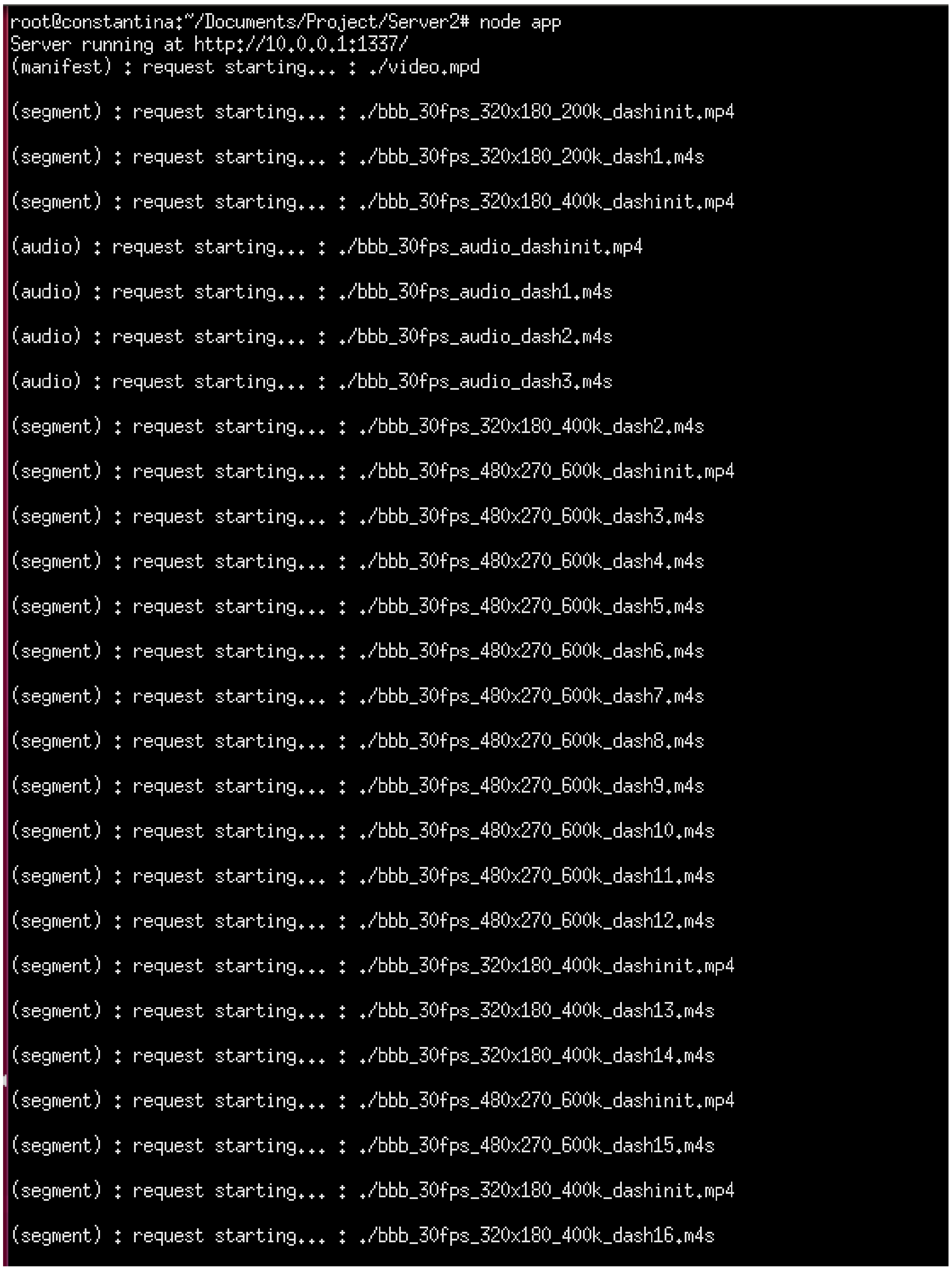

video.mpd generated by GPAC - VLC media player
Media Playback Audio Video Subtitle Tools View Help
video.mpd generated by GPAC
00:07
00:47

Figure 53: Video Playback



```
root@constantina:~/Documents/Project/Server2# node app
Server running at http://10.0.0.1:1337/
(manifest) : request starting... : ./video.mpd

(segment) : request starting... : ./bbb_30fps_320x180_200k_dashinit.mp4

(segment) : request starting... : ./bbb_30fps_320x180_200k_dash1.m4s

(segment) : request starting... : ./bbb_30fps_320x180_400k_dashinit.mp4

(audio) : request starting... : ./bbb_30fps_audio_dashinit.mp4

(audio) : request starting... : ./bbb_30fps_audio_dash1.m4s

(audio) : request starting... : ./bbb_30fps_audio_dash2.m4s

(audio) : request starting... : ./bbb_30fps_audio_dash3.m4s

(segment) : request starting... : ./bbb_30fps_320x180_400k_dash2.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dashinit.mp4

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash3.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash4.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash5.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash6.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash7.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash8.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash9.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash10.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash11.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash12.m4s

(segment) : request starting... : ./bbb_30fps_320x180_400k_dashinit.mp4

(segment) : request starting... : ./bbb_30fps_320x180_400k_dash13.m4s

(segment) : request starting... : ./bbb_30fps_320x180_400k_dash14.m4s

(segment) : request starting... : ./bbb_30fps_480x270_600k_dashinit.mp4

(segment) : request starting... : ./bbb_30fps_480x270_600k_dash15.m4s

(segment) : request starting... : ./bbb_30fps_320x180_400k_dashinit.mp4

(segment) : request starting... : ./bbb_30fps_320x180_400k_dash16.m4s
```

Figure 54: Output of the server terminal

The output of the server terminal demonstrates the order of the requested segments, both video and audio, which are being transmitted from the server to the client during the live broadcast, as well as the initial request of the manifest file. Even before the video starts playing, the server starts transmitting segments so that the buffer will be filled with data and stalling will be prevented. Also, when the available bandwidth is high, the initial delay is minor and the opposite.

Before stopping the simulation with the Stop button in MiniEdit console window, we have to quit from the CLI by typing exit at the Mininet prompt in the MiniEdit console window.

5.2 TRAFFIC GENERATOR

In this chapter we demonstrate the way to add extra traffic in order to conduct several experiments. In the beginning, we need to update the topology introduced before as shown in Figure 55.

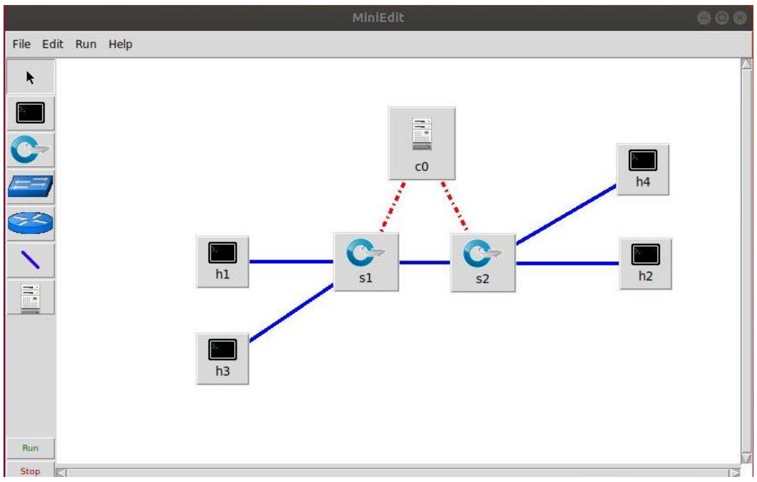


Figure 55: Updated topology

First of all, we add a new switch between switch s1 and host h2. Then, we connect the new switch (s2) to the controller. Next, we add two new hosts h3 and h4, which we connect to switch s1 and s2 respectively. Our aim is to create a new UDP server in host h3 and a new client that receives data in host h4 [40]. In order to achieve that, the first step is to open the terminals of h3 and h4. Then, in the terminal of host h3, we run the iperf command, as a UDP server [(-s) run as server, (-u) UDP]:

```
iperf -s -u
```

Right after that, in the h4 terminal we run the iperf command, however now as a UDP client [(-c h3) run as client with server on h3, (-u) UDP (-b 490m) 490 Mbps bandwidth, (-t 80) for 80 seconds]:

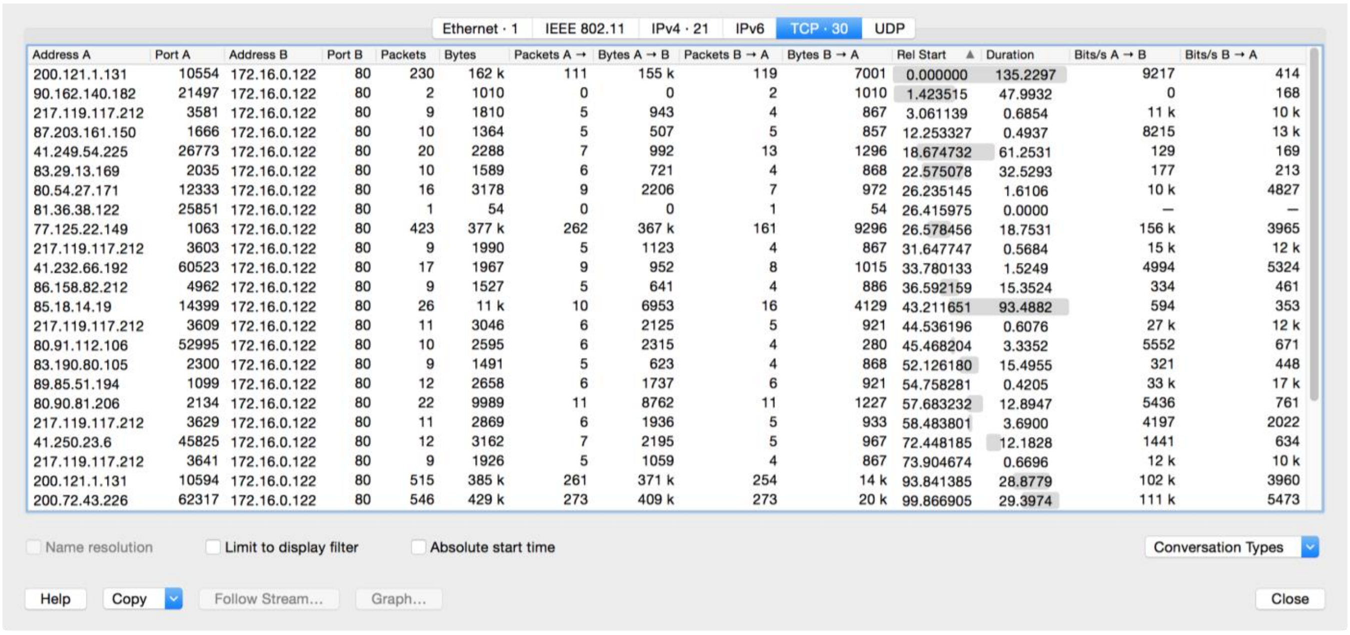
```
iperf -c h3 -u -b 490m -t 80
```

As a result, traffic has been generated between switches s1 and s2.

We conducted the experiment described using the following bandwidths between h1-s1 and s2-h2:

a. 1 Mbps
b. 3 Mbps
c. 5 Mbps
d. 10 Mbps

We also created traffic after the first 30 seconds of the playback. The results of each experiment are shown below (Figure 57-Figure 64). We compared the download rate of the video (Megabits per second) to the time during which the experiment was conducted (second), as well as to the resolution that the video was broadcast. All the data was derived using the Wireshark application. Specifically, we exported data as CSV (Comma Separated Values) files from the Wireshark GUI using: File -> Export Packet Dissections -> As CSV. Also, other valuable data was exported using the conversation window (Figure 56). The available columns in the conversation window are the following: addresses, packet counters, byte counters and the next four additional columns: the start time of the conversation (“Rel Start”) or (“Abs Start”), the duration of the conversation in seconds, and the average bits (not bytes) per second in each direction. A timeline graph is also drawn across the “Rel Start” / “Abs Start” and “Duration” columns. Each row in the list shows the statistical values for exactly one conversation. Using the Copy button the list values were copied to the clipboard in CSV (Comma Separated Values) [41].



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A →	Bytes A → B	Packets B → A	Bytes B → A	Rel Start ▲	Duration	Bits/s A → B	Bits/s B → A
200.121.1.131	10554	172.16.0.122	80	230	162 k	111	155 k	119	7001	0.000000	135.2297	9217	414
90.162.140.182	21497	172.16.0.122	80	2	1010	0	0	2	1010	1.423515	47.9932	0	168
217.119.117.212	3581	172.16.0.122	80	9	1810	5	943	4	867	3.061139	0.6854	11 k	10 k
87.203.161.150	1666	172.16.0.122	80	10	1364	5	507	5	857	12.253327	0.4937	8215	13 k
41.249.54.225	26773	172.16.0.122	80	20	2288	7	992	13	1296	18.674732	61.2531	129	169
83.29.13.169	2035	172.16.0.122	80	10	1589	6	721	4	868	22.575078	32.5293	177	213
80.54.27.171	12333	172.16.0.122	80	16	3178	9	2206	7	972	26.235145	1.6106	10 k	4827
81.36.38.122	25851	172.16.0.122	80	1	54	0	0	1	54	26.415975	0.0000	—	—
77.125.22.149	1063	172.16.0.122	80	423	377 k	262	367 k	161	9296	26.578456	18.7531	156 k	3965
217.119.117.212	3603	172.16.0.122	80	9	1990	5	1123	4	867	31.647747	0.5684	15 k	12 k
41.232.66.192	60523	172.16.0.122	80	17	1967	9	952	8	1015	33.780133	1.5249	4994	5324
86.158.82.212	4962	172.16.0.122	80	9	1527	5	641	4	886	36.592159	15.3524	334	461
85.18.14.19	14399	172.16.0.122	80	26	11 k	10	6953	16	4129	43.211651	93.4882	594	353
217.119.117.212	3609	172.16.0.122	80	11	3046	6	2125	5	921	44.536196	0.6076	27 k	12 k
80.91.112.106	52995	172.16.0.122	80	10	2595	6	2315	4	280	45.468204	3.3352	5552	671
83.190.80.105	2300	172.16.0.122	80	9	1491	5	623	4	868	52.126180	15.4955	321	448
89.85.51.194	1099	172.16.0.122	80	12	2658	6	1737	6	921	54.758281	0.4205	33 k	17 k
80.90.81.206	2134	172.16.0.122	80	22	9989	11	8762	11	1227	57.683232	12.8947	5436	761
217.119.117.212	3629	172.16.0.122	80	11	2869	6	1936	5	933	58.483801	3.6900	4197	2022
41.250.23.6	45825	172.16.0.122	80	12	3162	7	2195	5	967	72.448185	12.1828	1441	634
217.119.117.212	3641	172.16.0.122	80	9	1926	5	1059	4	867	73.904674	0.6696	12 k	10 k
200.121.1.131	10594	172.16.0.122	80	515	385 k	261	371 k	254	14 k	93.841385	28.8779	102 k	3960
200.72.43.226	62317	172.16.0.122	80	546	429 k	273	409 k	273	20 k	99.866905	29.3974	111 k	5473

Figure 56: Wireshark Conversation window

a. 1 Mbps

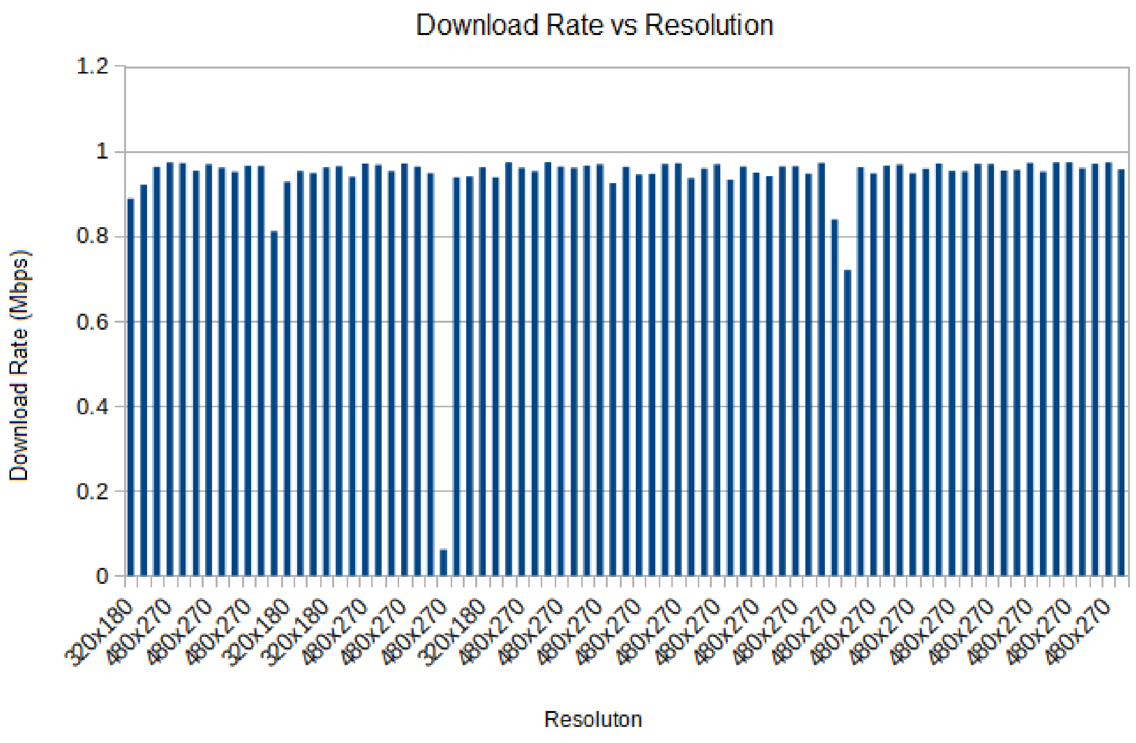


Figure 57: Download Rate (Mbps) Vs Resolution 1Mbps

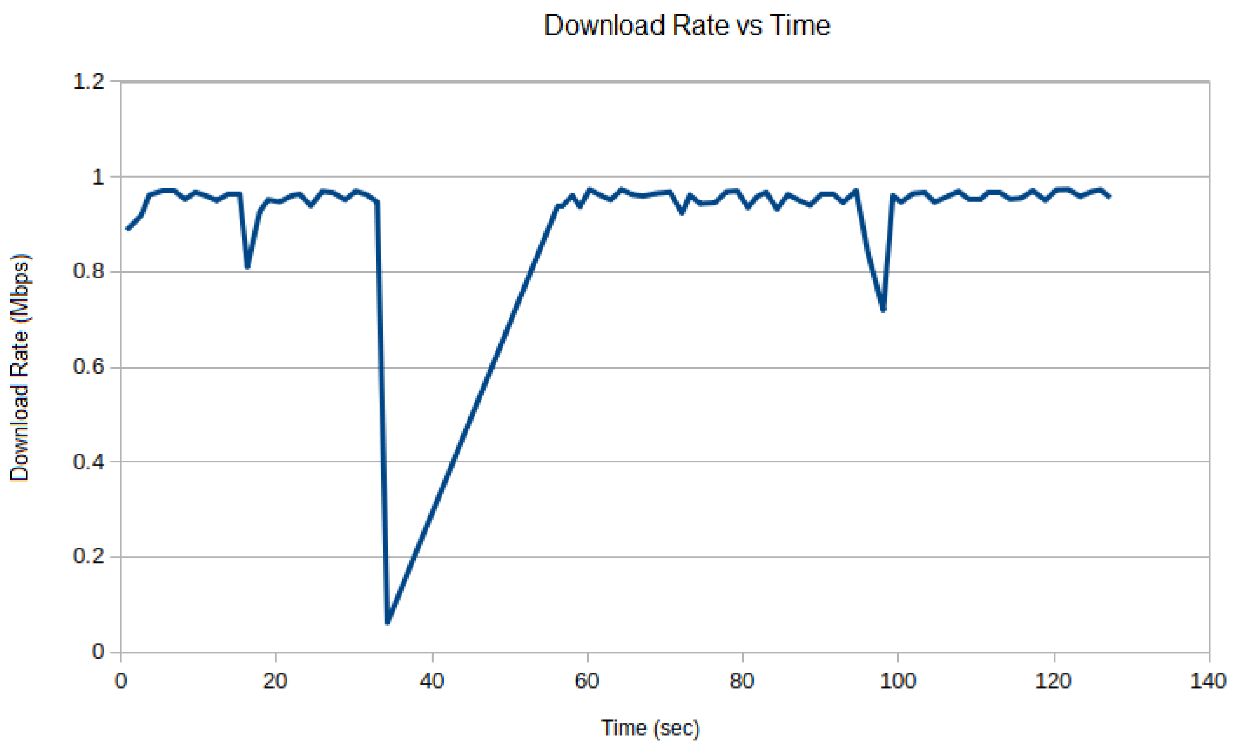


Figure 58: Download Rate (Mbps) Vs Time 1Mbps

b. 3 Mbps

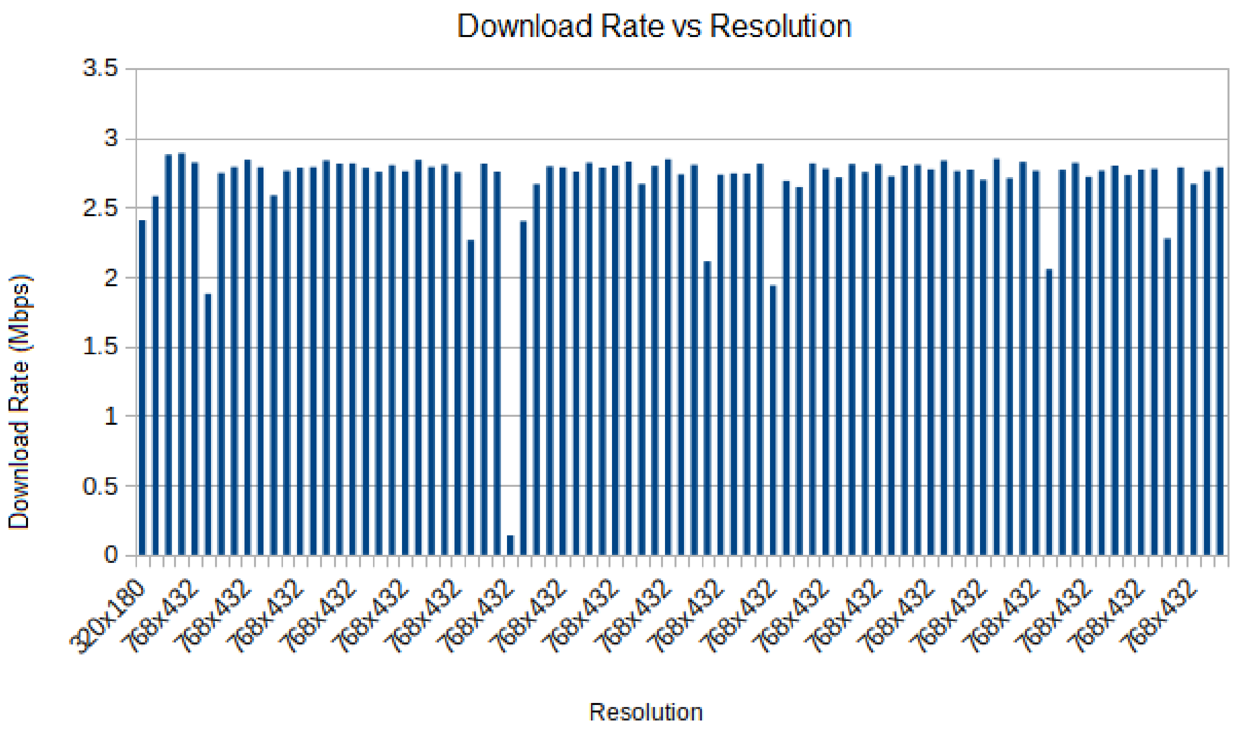


Figure 59: Download Rate (Mbps) Vs Resolution 3Mbps

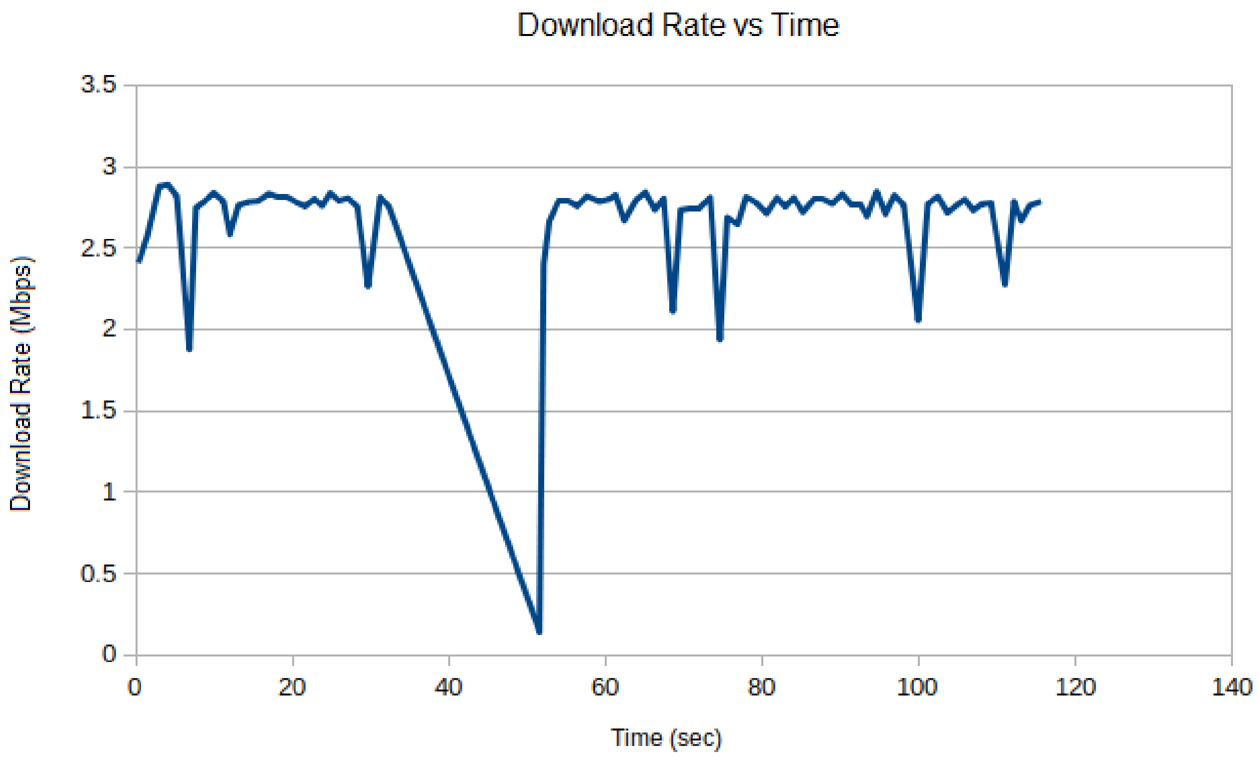


Figure 60: Download Rate (Mbps) Vs Time 3Mbps

c. 5 Mbps

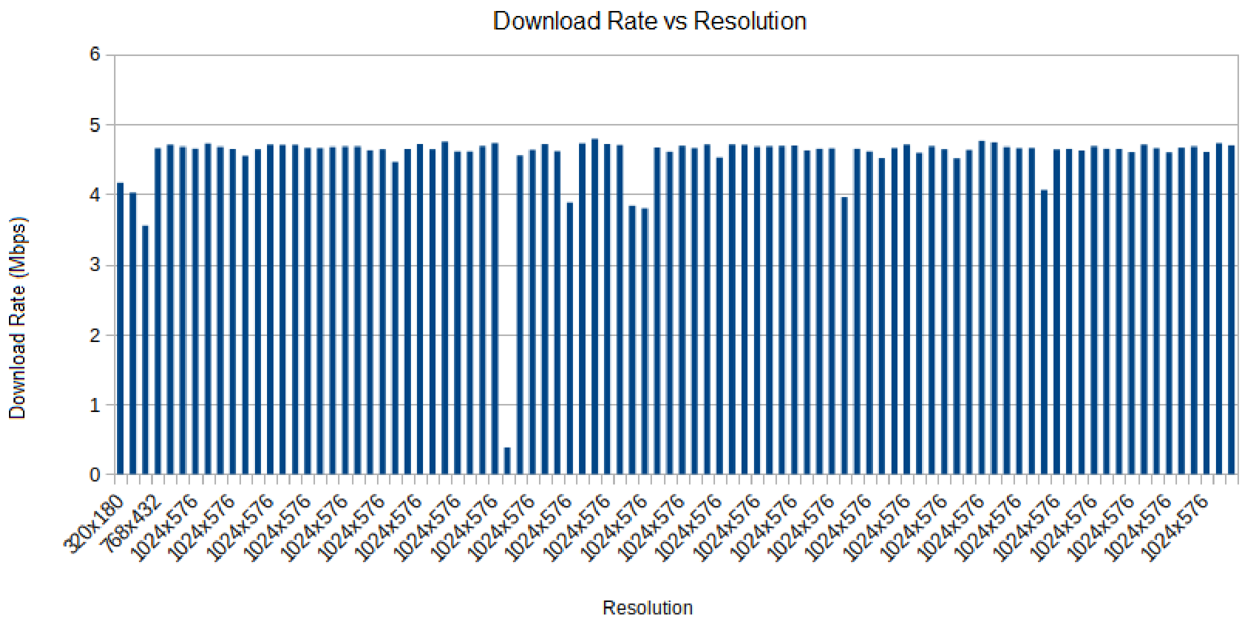


Figure 61: Download Rate (Mbps) Vs Resolution 5Mbps

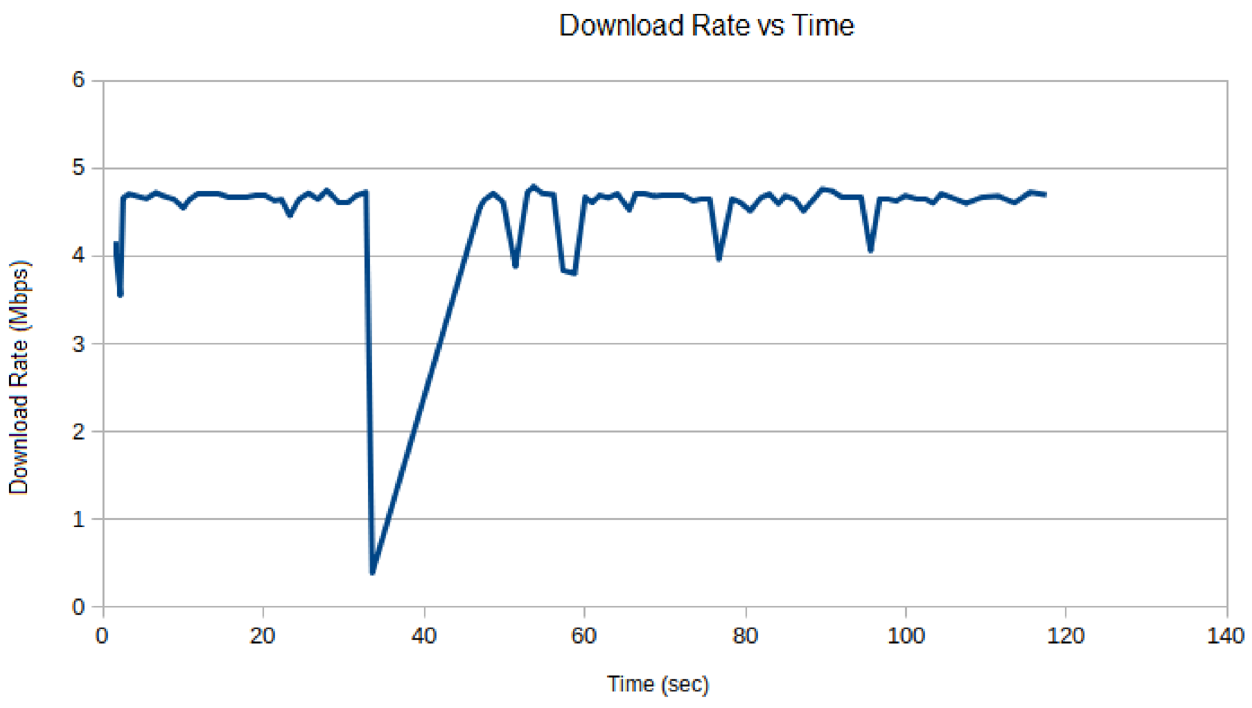


Figure 62: Download Rate (Mbps) Vs Time 5Mbps

d. 10 Mbps

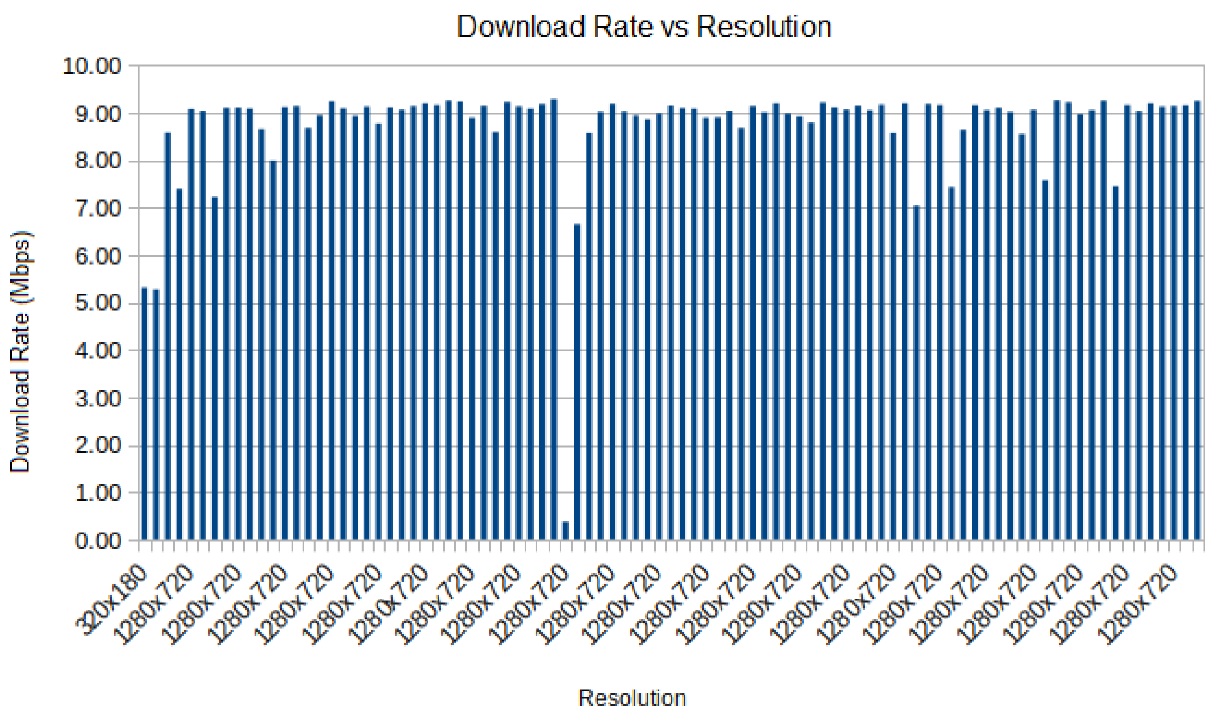


Figure 63: Download Rate (Mbps) Vs Resolution 10 Mbps

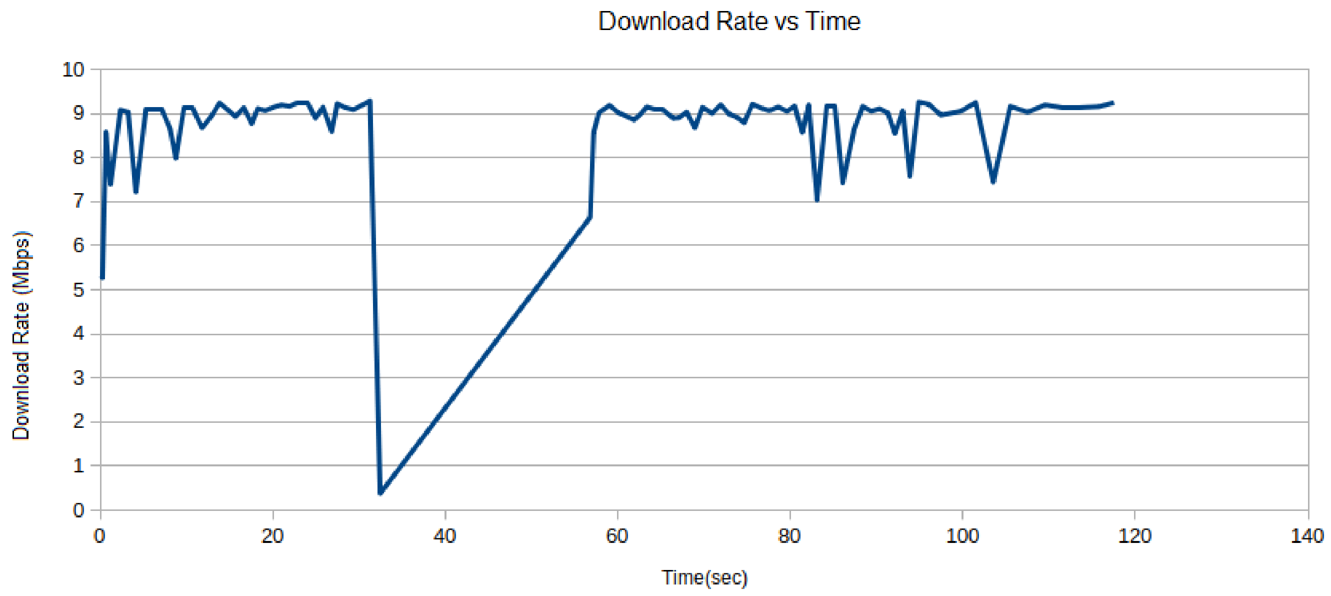


Figure 64: Download Rate (Mbps) Vs Time (sec) 10Mbps

It is clear that in the thirtieth second, or in other words when we add more traffic in the network, the download rate at the client reduces. However, this does not mean that the resolution will always change instantly. As described in Chapter 3.6, the video which is transmitted to the client via HTTP, is saved to a buffer. As long as a sufficient number of data of the video is available, the video can continue playing from the buffer. Furthermore, the lower the bandwidth is, the smaller resolution is offered. Yet, the resolution that the video starts playing is always the lowest (320x180), regardless the bandwidth that we have chosen. It is important to mention though that there is a big difference between 1 Mbps and 10 Mbps bandwidths. The first allows only the two lowest resolutions to be played, whereas the last the two highest. Furthermore, stalling was observed when the bandwidth was the lowest. Specifically, the selected bandwidths provide the following resolutions:

Table 2: Provided resolutions according to selected bandwidth

Bandwidth	**Resolutions**			
1 Mbps	320x180	480x270	-	-
3 Mbps	320x180	-	768x432	-
5 Mbps	320x180	-	768x432	1024x576
10 Mbps	320x180	-	768x432	1024x576

5.3 PSNR

Peak Signal-To-Noise Ratio, also known as PSNR, describes the ratio between the maximum possible power of a signal and the power of noise which influences the accuracy of its representation. PSNR is generally used to measure QoE. The signal is actually the original data and the noise is the error imported by compression or traffic in the network. The PSNR (in dB) is defined as [42]:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

Common values for the PSNR in lossy image and video compression are between the range of 30 and 50 dB, where higher values reflect the best QoE. A number between 20 dB and 25 dB, as far as wireless communication quality loss is considered, is acceptable.

5.3.1 PSNR calculation

After carrying out the experiments described in Chapter 5.1, using 1Mbps, 3Mbps, 5Mbps and 10Mbps as selected bandwidths, we measured the PSNR for each of them from the client's perspective. In order to achieve that, we followed the following steps as the video that was finally broadcast is not available and we needed to recreate it.

At first, using the cat command we created small parts of the broadcast video through its .m4s files and their corresponding init files.

```
cat bbb_30fps_320x180_200k_dashinit.mp4
bbb_30fps_320x180_200k_dash1.m4s    > part1.mp4
...
cat    bbb_30fps_768x432_1500k_dashinit.mp4
bbb_30fps_768x432_1500k_dash51.m4s bbb_30fps_768x432_1500k_dash52.m4s
bbb_30fps_768x432_1500k_dash53.m4s bbb_30fps_768x432_1500k_dash54.m4s
bbb_30fps_768x432_1500k_dash55.m4s bbb_30fps_768x432_1500k_dash56.m4s
bbb_30fps_768x432_1500k_dash57.m4s bbb_30fps_768x432_1500k_dash58.m4s
bbb_30fps_768x432_1500k_dash59.m4s bbb_30fps_768x432_1500k_dash60.m4s
bbb_30fps_768x432_1500k_dash61.m4s bbb_30fps_768x432_1500k_dash62.m4s
bbb_30fps_768x432_1500k_dash63.m4s bbb_30fps_768x432_1500k_dash64.m4s
bbb_30fps_768x432_1500k_dash65.m4s bbb_30fps_768x432_1500k_dash66.m4s
bbb_30fps_768x432_1500k_dash67.m4s bbb_30fps_768x432_1500k_dash68.m4s
bbb_30fps_768x432_1500k_dash69.m4s bbb_30fps_768x432_1500k_dash70.m4s
bbb_30fps_768x432_1500k_dash71.m4s bbb_30fps_768x432_1500k_dash72.m4s
bbb_30fps_768x432_1500k_dash73.m4s bbb_30fps_768x432_1500k_dash74.m4s
bbb_30fps_768x432_1500k_dash75.m4s bbb_30fps_768x432_1500k_dash76.m4s
bbb_30fps_768x432_1500k_dash77.m4s bbb_30fps_768x432_1500k_dash78.m4s
bbb_30fps_768x432_1500k_dash79.m4s bbb_30fps_768x432_1500k_dash80.m4s
```

```
bbb_30fps_768x432_1500k_dash81.m4s bbb_30fps_768x432_1500k_dash82.m4s
bbb_30fps_768x432_1500k_dash83.m4s    > part10.mp4
```

Secondly, all the parts (part1.mp4 ... part10.mp4) must be put together so as to recreate the final broadcast video. It is really important to mention that all these videos must have the same dimensions in the end (1920 x 1080) in order to be compared to the original one. We merge the parts using the command below:

```
ffmpeg -i part1.mp4 -i part2.mp4 -i part3.mp4 -i part4.mp4 -i part5.mp4 -i part6.mp4 -i part7.mp4 -i part8.mp4 -i part9.mp4 -i part10.mp4 -filter_complex"[0:v]scale=1920:1080:force_original_aspect_ratio=1[v0];[1:v]scale=1920:1080:force_original_aspect_ratio=1[v1];[2:v]scale=1920:1080:force_original_aspect_ratio=1[v2];[3:v]scale=1920:1080:force_original_aspect_ratio=1[v3];[4:v]scale=1920:1080:force_original_aspect_ratio=1[v4];[5:v]scale=1920:1080:force_original_aspect_ratio=1[v5];[6:v]scale=1920:1080:force_original_aspect_ratio=1[v6];[7:v]scale=1920:1080:force_original_aspect_ratio=1[v7];[8:v]scale=1920:1080:force_original_aspect_ratio=1[v8];[9:v]scale=1920:1080:force_original_aspect_ratio=1[v9];[v0][0:a][v1][1:a][v2][2:a][v3][3:a][v4][4:a][v5][5:a][v6][6:a][v7][7:a][v8][8:a][v9][9:a]concat=n=10:v=1:a=1[v][a]"-map [v] -map [a] finalxMbps.mp4
```

The name of the final broadcast video is finalxMbps.mp4 with x representing the selected bandwidth, in other words 1 Mbps, 3 Mbps, 5 Mbps or 10 Mbps.

Finally, we are able to compare the original video to the transmitted ones using this command:

```
ffmpeg -i output.mp4 -i finalxMbps.mp4 -lavfi '[0]scale=1920:1080[a];[a][1]psnr' -f null -
```

The results of the four comparisons are demonstrated in the Table below:

Table 3: PSNR results

Bandwidth	Y	U	V	Average	Minimum	Maximum
1 Mbps	20.389619	27.810548	34.136401	21.914616	9.987742	27.492456
3 Mbps	21.587332	29.178792	35.549621	23.121611	10.004065	28.895605
5 Mbps	22.379611	29.371822	37.341615	23.895887	9.988636	28.673364
10 Mbps	22.223866	29.201896	37.258051	23.740011	9.984147	29.090003

As we had expected, while the bandwidth increases, the PSNR values are larger. Moreover, the results for 5 Mbps and 10 Mbps are really close since the resolutions provided and the throughput during the experiment were similar. Figure 64 demonstrates the variations of the PSNR Average, Minimum and Maximum values:

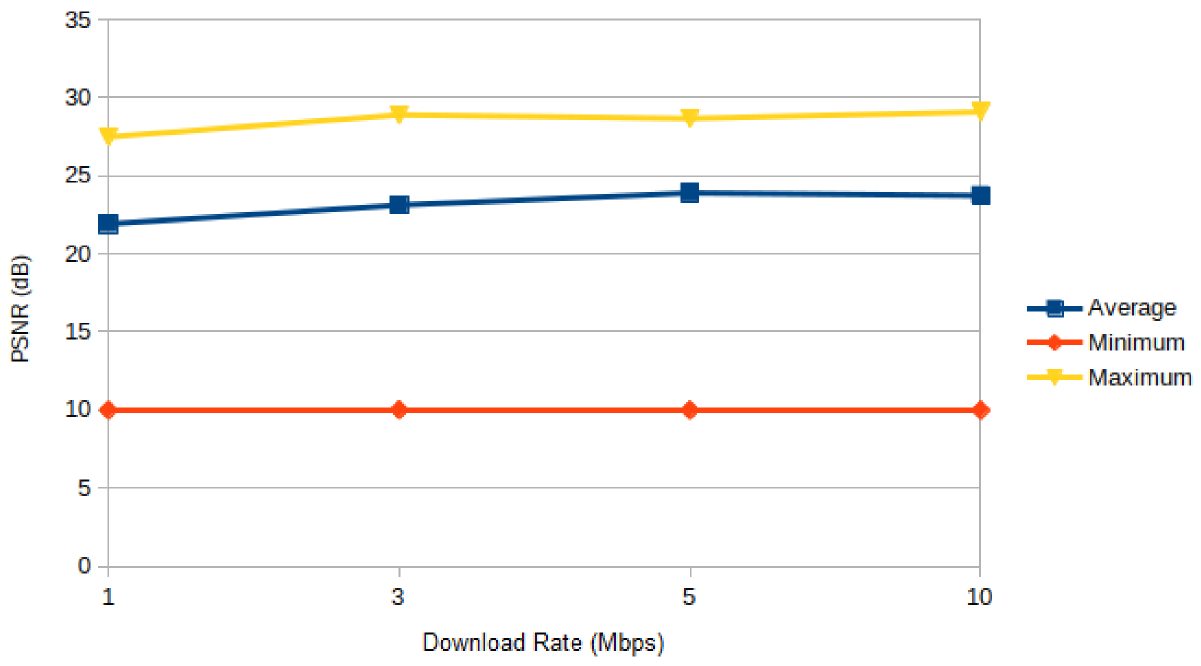


Figure 65: PSNR (Average, Minimum, Maximum)

6. CONCLUSION AND FUTURE WORK

In this thesis we investigated how to deliver DASH-based content from a server through Mininet environment. For this purpose, we created a custom Mininet topology, proposed a server-client implementation for video streaming, and implemented the HTTP Adaptive Video Streaming logic on top of this environment. Furthermore, we presented results for a set of experiments aimed at studying the delivery of streaming video under varying bandwidth conditions. Our findings indicate that the transmitted data is utterly connected with the existing traffic in the network. By establishing a bandwidth varying channel, the effects with regard to video resolutions that would be selected by the media player were as expected, namely the higher the bandwidth the higher the selected resolution. Yet, we noticed that the resolution of the broadcast video would not always change instantly. On condition that a sufficient number of data of the video are available, the video continued playing from the buffer. Also, smaller resolution was offered when the bandwidth was low.

One big challenge that we faced and overcame in the end, was the recreation of the transmitted video in order to calculate the QoE of the client. That was quite difficult as the broadcast was live and we were not able to save the video on the client’s device. However, as the segments of the video were known in the order of which they were transmitted, we were able to put them together and recreate the video.

In our thesis all the video segments are uploaded to a single server. An interesting future direction would be splitting video segments to two different servers and as a result creating two different MPD files for the client. That would be challenging as then the client should not only decide which segment he needs to request according to the available bandwidth, but also determine from which server he should request it from.

Finally, another research approach would be as far as the controller on the SDN environment is concerned. An SDN controller manages flow control to the switches and routers to deploy intelligent networks. A differently designed network controller would determine different paths for the packets across the network of switches and this would have an impact on the user’s QoE.

ABBREVIATIONS - ACRONYMS

1G	First-Generation Cellular Network
2G	Second-Generation Cellular Network
3G	Third-Generation Cellular Network
3GPP	3rd Generation Partnership Project
4G	Fourth-Generation Cellular Network
5G	Fifth-Generation Cellular Network
ABR	Adaptive Bit Rate
ABS	Adaptive Bitrate Streaming
ABSFs	Almost Blank Subframes
ADONIS	Article Delivery Over Network Information Systems
ALISE	Association for Library Collections and Technical Services
AMBR	Aggregate Maximum Bit Rate
ANSI	American National Standards Institute
APN	Access Point Name
ARP	Address Resolution Protocol
ARP	Allocation and Retention Priority
ASO	Adaptive Streaming Overview
AVC	Advanced Video Coding
BOLA	Buffer Occupancy based Lyapunov Algorithm
CDMA	Code Division Multiple Access
CATT	China Academy of Communications Technology
CDN	Content Delivery Network
CLI	Command Line Interface
CoMP	Coordinated Multi-Point
COST	Cooperation in Science and Technology
CSV	Comma Separated Values
DAMPS	Digital AMPS
DASH	Dynamic Adaptive Streaming over HTTP

DL-TFT	DownLink - Traffic Flow Template
DNS	Domain Name System
DRM	Digital Rights Management
EDGE	Enhanced Data rates for GSM Evolution
EGSM	European GSM
eICIC	enhanced Inter-Cell Interference Coordination
EME	Encrypted Media Extensions
eNodeB	Enhanced Node B
EPC	Evolved Packet Core
ETSI	European Technical Standards Institute
E-UTRA	Evolved Universal Terrestrial Radio Access
E-UTRAN	Enhanced - UTRAN
FDD	Frequency Division Duplex
FDM	Frequency Division Multiplexing
FDMA	Frequency Division Multiple Access
FFmpeg	Fast Forward MPEG
GGSN	Gateway GPRS Support Node
GPAC	Project on Advanced Content
GPRS	General Packet Radio Service
GRB	Guaranteed Bit Rate
GSM	Global System for Mobile
GUI	Graphical User Interface
HAS	HTTP Adaptive Streaming
HbbTV	Hybrid broadcast broadband TV
HLR	Home Location Register
HLS	HTTP Live Streaming
HSPA	High Speed Packet Access
HTTP	HyperText Transfer Protocol

ICMobile	Information and Communication Mobile Group
IMT- Advanced	International Mobile Telecommunications - Advanced
IoT	Internet of Things
IP address	Internet Protocol address
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IS-95	Interim Standard - 95
ITU	International Telecommunication Union
ITU-R	International Telecommunication Union - Radiocommunication Sector
JVM	Java Virtual Machine
LTE	Long-Term Evolution
LTE-A	LTE Advanced
MAC	Media Access Control
MBR	Maximum Bit Rate
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MMOG	Multimedia Online Gaming
mobile TV	Mobile TeleVision
MPD	Media Presentation Description
MPEG	Motion Picture Expert Group
MPEG-CENC	Motion Picture Expert Group - Common Encryption
MSC	Mobile Switching Centre
MSE	Media Source Extensions
MU-MIMO	Multiple User MIMO
NKUA	National and Kapodistrian University of Athens
Non-GBR	Non-Guaranteed Bit Rate
NPM	Node Package Manager

NSS	Network Switching Subsystem
NTT	Nippon Telegraph and Telephone
ODL controller	OpenDaylight controller
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OS	Operating System
PDC	Personal Digital Cellular
PSNR	Peak Signal-To-Noise Ratio
QAM	Quadrature Amplitude Modulation
QCI	QoS Class Identifier
QoE	Quality of Experience
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RDA	Rate Determination Algorithm
RTT	Round-Trip Time
SAE	System Architecture Evolution
SAP	Stream Access Points
SC-FDMA	Single Carrier FDMA
SDN	Software Defined Networking
SGSN	Serving GPRS Support Node
SMS	Short Messaging Service
SU-MIMO	Single User MIMO
TCP/IP	Transmission Control Protocol / Internet Protocol
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TD-SCDMA	Time Division Synchronous Code Division Multiple Access
TEI	Text Encoding Initiative
UDP	User Datagram Protocol

UE	User Equipment
UE-AMBR	User Equipment - Aggregate Maximum Bit Rate
UL-TFT	Uplink Traffic Flow Template
UMTS	Universal Mobile Telecommunication System
UNISIST	Universal System for information in Science and technology
URL	Uniform Resource Locator
UTMS	Universal Mobile Telecommunications System
VLC	VideoLAN Client
VoIP	Voice over Internet Protocol
W3C	World Wide Web Consortium
WCDMA	Wideband Code Division Multiple Access
WiMAX	Worldwide Interoperability for Microwave Access
XML	Extensible Markup Language
ΕΕΧΙ	Ένωση Ελλήνων Χρηστών Internet
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

REFERENCES

[1] https://en.wikipedia.org/wiki/Cellular_network

[2] Chapter 2.1 http://www.ijcta.com/documents/volumes/vol5issue5/ijcta2014050534.pdf

[3] https://en.wikipedia.org/wiki/Martin_Cooper_(inventor)

[4] https://en.wikipedia.org/wiki/Motorola_DynaTAC.

[5] http://www.cslab.ece.ntua.gr/~sgouros/MM-Old/notes4.htm

[6] https://el.wikipedia.org/wiki/%CE%A8%CE%B7%CF%86%CE%B9%CE%B1%CE%BA%CF%8C_%CF%83%CE%AE%CE%BC%CE%B1

[7] https://www.sciencedirect.com/topics/computer-science/time-division-multiple-access.

[8] https://www.etsi.org/

[9] https://www.researchgate.net/publication/216645569_Short_message_service_SMS_language_and_written_language_skills_Educators'_perspectives

[10] https://bitmovin.com/adaptive-streaming/

[11] https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming

[12] https://www.synopi.com/mpeg-dash/

[13] https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/

[14] https://en.wikipedia.org/wiki/European_Cooperation_in_Science_and_Technology

[15] https://en.wikipedia.org/wiki/Quality_of_experience

[16] T. Hoßfeld et al., “Quantification of YouTube QoE via crowdsourcing,” in Proc. IEEE Int. Symp. Multimedia, Dana Point, CA, USA, 2011, pp. 494–499.

[17] A Survey on Quality of Experience of HTTP Adaptive Streaming Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia

[18] https://karaf.apache.org/manual/latest/provisioning

[19] https://opensourceeducation.net/install-ffmpeg-flvtool2-and-mp4box-on-ubuntu-14-04

[20] http://mininet.org/download/

[21] https://en.wikipedia.org/wiki/VLC_media_player

[22] https://www.videolan.org/vlc/download-ubuntu.html

[23] https://en.m.wikipedia.org/wiki/Ffmpeg

[24] https://tecadmin.net/install-ffmpeg-on-linux/

[25] https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP#cite_note-DASH-ABRLOGIC-1

[26] https://tecadmin.net/install-latest-nodejs-npm-on-ubuntu/

[27] https://en.wikipedia.org/wiki/GPAC_Project_on_Advanced_Content

[28] https://opensourceeducation.net/install-ffmpeg-flvtool2-and-mp4box-on-ubuntu-14-04

[29] https://en.wikipedia.org/wiki/Wireshark

[30] https://linuxtechlab.com/install-wireshark-linux-centosubuntu/

[31] https://unit42.paloaltonetworks.com/unit42-customizing-wireshark-changing-column-display/

[32] https://nodejs.org/api/errors.html

[33] https://nodejs.org/api/process.html

[34] https://www.w3schools.com/nodejs/nodejs_http.asp

[35] //http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/

[36] https://video.stackexchange.com/questions/24680/what-is-keyint-and-min-keyint-and-no-scenecut

[37] https://blog.desgrange.net/post/2017/04/17/encode-videos-dynamic-adaptive-streaming-http.html

[38] https://rybakov.com/blog/mpeg-dash/

[39] https://gist.github.com/dvlden/b9d923cb31775f92fa54eb8c39ccd5a9

[40] https://onl.wustl.edu/NPR_Tutorial/Filters,_Queues_and_Bandwidth/Generating_Traffic_With_Iperf.html

[41] https://www.wireshark.org/docs/wsug_html_chunked/ChStatConversations.html

[42] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio