



NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

**SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS**

**GRADUATE PROGRAM
COMPUTER SYSTEMS NETWORKING**

MSc THESIS

**An SDN QoE Monitoring Framework for VoIP and video
applications**

Maria-Evgenia I. Xezonaki

Supervisor: Lazaros Merakos, Professor

ATHENS

OCTOBER 2017



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΔΙΚΤΥΩΣΗ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

SDN Σύστημα Παρακολούθησης QoE για VoIP και video εφαρμογές

Μαρία-Ευγενία Ι. Ξεζωνάκη

Επιβλέπων: Λάζαρος Μεράκος, Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2017

MSc THESIS

An SDN QoE Monitoring Framework for VoIP and video applications

Maria-Evgenia I. Xezonaki

S.N.: M1473

SUPERVISOR: Lazaros Merakos, Professor

ADVISORY COMMITTEE **Lazaros Merakos, Professor**
Stathes Hadjiefthymiades, Associate Professor

October 2017

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

SDN Σύστημα Παρακολούθησης QoE για VoIP και video εφαρμογές

Μαρία-Ευγενία Ι. Ξεζωνάκη

A.M.: M1473

ΕΠΙΒΛΕΠΩΝ: Λάζαρος Μεράκος, Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ Λάζαρος Μεράκος, Καθηγητής
Στάθης Χατζηευθυμιάδης, Αναπληρωτής Καθηγητής

Οκτώβριος 2017

ABSTRACT

Lately, there has been a rapid rise of the mobile communications industry, since the use of mobile devices is spreading at a fast pace and is expected to continue its penetration into the daily routine of consumers. This fact, combined with the limitations of the current communications networks' structure, necessitates the development of new networks with increased capabilities, so that users can be served with the best possible quality of service and at the same time with the optimal network resources utilization. A new networking approach is Software Defined Networking (SDN) which decouples the control from the data plane, transforming the network elements to simple forwarding devices and making decisions centrally. The quality of service perceived by the user, or quality of experience (QoE), is considered to be a matter of great importance in software defined networks.

This diploma thesis aims at presenting SDN technology, reviewing existing research in the field of QoE on SDN networks and then developing an SDN application that monitors and preserves the QoE for VoIP and video applications. More specifically, the developed SDN QoE Monitoring Framework (SQMF) periodically monitors various network parameters on the VoIP/video packets transmission path, based on which it calculates the QoE. If it is found that the result is less than a predefined threshold, the framework changes the transmission path, and thus the QoE recovers.

The structure of this diploma thesis is the following: Chapter 1 presents the current state of communications networks and predictions for the future state, as well as the challenges that current networks will not be able to cope with. Chapter 2 then describes in detail the SDN technology in terms of architecture, main control-data plane communication protocol, use cases, standardization, advantages and disadvantages. Chapter 3 introduces the concept of QoE and lists well-known QoE estimation models for various applications types, some of which were used in this thesis. Relevant existing studies in the field of QoE on SDN networks as well as a comparative table can be found in chapter 4. The following chapters concern the framework implemented in the context of this diploma thesis: Chapter 5 describes in detail all the required tools and instructions for the development of SQMF, while Chapter 6 presents examples where the QoE in a network can face degradation. Finally, Chapter 7 analyzes in depth SQMF's design principles, logic and code files, provides a demonstration of its operation and evaluates it, whereas Chapter 8 briefly summarizes the conclusions and of this thesis and future work points.

SUBJECT AREA: Communications Networks

KEYWORDS: Software Defined Networks, Quality of Experience, Video, VoIP, Monitoring, SDN Controller, OpenDaylight, OpenFlow, Mininet

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια έχει σημειωθεί ραγδαία άνοδος του κλάδου των κινητών επικοινωνιών, αφού η χρήση των κινητών συσκευών εξαπλώνεται με ταχύτατους ρυθμούς και αναμένεται να συνεχίσει τη διεξόδου της στην καθημερινότητα των καταναλωτών. Το γεγονός αυτό, σε συνδυασμό με τους περιορισμούς που θέτει η τρέχουσα δομή των δικτύων επικοινωνιών, καθιστά αναγκαία την ανάπτυξη νέων δικτύων με αυξημένες δυνατότητες, ώστε να είναι δυνατή η εξυπηρέτηση των χρηστών με την καλύτερη δυνατή ποιότητα εμπειρίας και ταυτόχρονα τη βέλτιστη αξιοποίηση των πόρων του δικτύου. Μία νέα δικτυακή προσέγγιση αποτελεί η δικτύωση βασισμένη στο λογισμικό (Software Defined Networking - SDN), η οποία αφαιρεί τον έλεγχο από τις συσκευές προώθησης του δικτύου, και οι αποφάσεις λαμβάνονται σε κεντρικό σημείο. Η ποιότητα υπηρεσίας που αντιλαμβάνεται ο χρήστης, ή αλλιώς ποιότητα εμπειρίας, κρίνεται ζήτημα υψηλής σημασίας στα δίκτυα SDN.

Η παρούσα διπλωματική εργασία έχει ως στόχο την παρουσίαση της τεχνολογίας SDN, την επισκόπηση της υπάρχουσας έρευνας στο πεδίο της ποιότητας εμπειρίας σε SDN δίκτυα και στη συνέχεια την ανάπτυξη μίας SDN εφαρμογής η οποία παρακολουθεί και διατηρεί την ποιότητας εμπειρίας σε υψηλά επίπεδα για εφαρμογές VoIP και video. Πιο συγκεκριμένα, η εφαρμογή SQMF (SDN QoE Monitoring Framework) παρακολουθεί περιοδικά στο μονοπάτι μετάδοσης των πακέτων διάφορες παραμέτρους του δικτύου, με βάση τις οποίες υπολογίζει την ποιότητα εμπειρίας. Εάν διαπιστωθεί ότι το αποτέλεσμα είναι μικρότερο από ένα προσδιορισμένο κατώφλι, η εφαρμογή αλλάζει το μονοπάτι μετάδοσης, και έτσι η ποιότητα εμπειρίας ανακάμπτει.

Η δομή της παρούσας διπλωματικής εργασίας είναι η εξής: Στο κεφάλαιο 1 παρουσιάζεται η σημερινή εικόνα των δικτύων επικοινωνιών και οι προβλέψεις για τη μελλοντική εικόνα, καθώς και οι προκλήσεις στις οποίες τα σημερινά δίκτυα δε θα μπορούν να αντεπεξέλθουν. Στη συνέχεια στο κεφάλαιο 2 περιγράφεται αναλυτικά η τεχνολογία SDN ως προς την αρχιτεκτονική, το κύριο πρωτόκολλο που χρησιμοποιεί, τα σενάρια χρήσης της, την προτυποποίηση, τα πλεονεκτήματα και τα μειονεκτήματά της. Το κεφάλαιο 3 εισάγει την έννοια της ποιότητας εμπειρίας του χρήστη και παραθέτει ευρέως γνωστά μοντέλα υπολογισμού της για διάφορους τύπους εφαρμογών, που χρησιμοποιούνται στην παρούσα εργασία. Σχετικές υπάρχουσες μελέτες στο πεδίο της ποιότητας εμπειρίας σε δίκτυα SDN αλλά και συγκριτικός πίνακας μπορούν να βρεθούν στο κεφάλαιο 4. Τα επόμενα κεφάλαια αφορούν στην εφαρμογή SQMF που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας: το κεφάλαιο 5 περιγράφει αναλυτικά όλα τα προαπαιτούμενα εργαλεία και οδηγίες για την ανάπτυξη του SQMF, ενώ το κεφάλαιο 6 παρουσιάζει παραδείγματα όπου η ποιότητα εμπειρίας ενός δικτύου μπορεί να υποστεί μείωση. Τέλος, το κεφάλαιο 7 αναλύει σε βάθος τις σχεδιαστικές προδιαγραφές, τη λογική και τον κώδικα του SQMF και παρέχει επίδειξη της λειτουργίας του και αξιολόγησή του, ενώ το κεφάλαιο 8 συνοψίζει επιγραμματικά τα συμπεράσματα της παρούσας εργασίας και ανοιχτά θέματα για μελλοντική έρευνα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δίκτυα Επικοινωνιών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Δικτύωση Βασισμένη στο Λογισμικό, Ποιότητα Εμπειρίας, Υπηρεσίες βίντεο, Υπηρεσίες φωνής, Παρακολούθηση, SDN Controller, OpenDaylight, OpenFlow, Mininet

This MSc Thesis is dedicated to my family.

ACKNOWLEDGMENTS

This MSc Thesis constitutes the last step of my postgraduate studies in the Department of Informatics and Telecommunications of National and Kapodistrian University of Athens. Through this thesis I am given the opportunity to wholeheartedly thank my supervisor, Prof. Lazaros Merakos, for the confidence he showed me in undertaking the specific thesis subject, the chance he gave me to deal with such an interesting and challenging field and the aid he provided to me regarding technical issues. This is the second time he shows faith in my capabilities, after having also supervised me in my undergraduate thesis, therefore I can only be grateful towards him.

Subsequently, heartfelt thanks go to the PhD Candidate Eirini Liotou for her continuous guidance and ceaseless support during both the current thesis writing and the practical part's implementation. I cannot overlook the fact that she accepted to supervise me from the very first moment I asked her to, always being really reassuring about my progress and supportive when I was under pressure, thus making our cooperation very pleasant and fruitful.

Moreover, a person whose knowledge and aid have been of significant importance to the implementation of my thesis is George K. Petropoulos, whom I worked with for 7 months at Intracom Telecom. His help was crucial on my first contact with the technology used in this thesis as well as at a critical point before its delivery. To this end, I would like to mention his name and thank him for everything.

Finally, I cannot miss to thank my family and my friends for all the love, support and patience they showed throughout my undergraduate and postgraduate studies, as well as to all those who contributed in their own way to my effort's successful completion.

CONTENTS

PREFACE	18
1. INTRODUCTION	19
1.1 CURRENT NETWORKING STATE AND LIMITATIONS.....	19
1.2 DEVICES GROWTH	20
1.3 DATA AND VIDEO GROWTH.....	21
1.4 FIFTH GENERATION (5G) WIRELESS NETWORKS EMERGENCE	24
1.4.1 5G systems requirements	25
1.5 THE NEW NETWORKING PARADIGM	27
2. SOFTWARE-DEFINED NETWORKING (SDN)	29
2.1 DEFINITION(S) OF SDN.....	29
2.2 SDN ARCHITECTURE.....	30
2.3 OPENFLOW PROTOCOL.....	36
2.3.1 The Flow Table.....	36
2.3.2 The lookup process.....	38
2.4 OPENDAYLIGHT CONTROLLER.....	40
2.5 NETCONF, RESTCONF AND YANG.....	42
2.6 HISTORY AND STANDARDIZATION OF SDN.....	44
2.7 USE CASES OF SDN	47
2.8 ADVANTAGES OF SDN	48
2.9 CHALLENGES OF SDN.....	50
3. QUALITY OF EXPERIENCE (QOE)	52
3.1 INTRODUCTION TO QoE.....	52

3.2 QoE MANAGEMENT.....	54
3.3 QoE MODELS	58
3.3.1 Voice	58
3.3.2 Video	61
3.3.3 YouTube.....	63
4. STATE OF THE ART IN QOE FOR SOFTWARE - DEFINED NETWORKS	65
4.1 RESEARCH WORKS ON QoE FOR SDN	65
4.2 SUMMARIZING TABLE.....	78
5. ENVIRONMENT SETUP	81
5.1 SYSTEM REQUIREMENTS.....	81
5.2 SDN CONTROLLER DEPLOYMENT.....	82
5.2.1 Karaf features.....	82
5.3 MININET DEPLOYMENT	84
5.3.1 Mininet default topology	86
5.3.2 Changing topology size and type	86
5.3.3 Custom topologies.....	87
5.3.4 Using a remote controller in Mininet.....	87
5.4 FIRST ODL PROJECT CREATION	89
5.5 USE OF AN IDE	90
5.6 VoIP TRAFFIC CREATION.....	96
5.7 VIDEO TRAFFIC CREATION.....	102
5.7.1 Video adjustment to required specifications	102
6. QOE DEGRADATION CASES EXAMPLES.....	105
6.1 EXPERIMENTS ON VoIP APPLICATIONS.....	106
6.1.1 Experiment N° 1: Manual network limitations on VoIP	106
6.1.2 Experiment N° 2: Multiple sources of traffic on VoIP	107

6.2 EXPERIMENTS ON VIDEO APPLICATIONS	109
6.2.1 Experiment N° 1: Manual network limitations on video.....	109
6.2.2 Experiment N° 2: Multiple sources of traffic on video	110
7. IMPLEMENTATION ANALYSIS	112
7.1 DESIGN PRINCIPLES AND WORKFLOW	112
7.2 IMPLEMENTATION STRUCTURE	114
7.3 DETAILED IMPLEMENTATION ANALYSIS.....	117
7.4 SQMF DEMONSTRATION.....	129
7.5 SQMF EVALUATION	135
7.5.1 SQMF on VoIP traffic	135
7.5.2 SQMF on video traffic.....	138
8. CONCLUSION AND FUTURE WORK.....	142
ABBREVIATIONS - ACRONYMS	143
ANNEX.....	147
REFERENCES	148

LIST OF FIGURES

Figure 2.1: The lookup process in OF [32]	39
Figure 6.1: VoIP quality decrease in relation to total network packet loss	107
Figure 6.2: VoIP quality decrease in relation to number of VoIP packets, in parallel with iperf	108
Figure 6.3: Video quality decrease in relation to total network packet loss.....	110
Figure 6.4: Video quality instability for video streaming simultaneously with using iperf	111
Figure 7.1: QoE Monitoring workflow.....	112
Figure 7.2: SQMF classes correlation	129
Figure 7.4: Packet loss comparison between cases with and without SQMF in VoIP traffic generation, when a link failure occurs.....	138
Figure 7.5: MOS comparison between cases with and without SQMF in VoIP traffic generation, when a link failure occurs	138
Figure 7.6: Packet loss comparison between cases with and without SQMF in video streaming, when a link failure occurs.....	141
Figure 7.7: Vq comparison between cases with and without SQMF in video streaming, when a link failure occurs	141

LIST OF IMAGES

Image 1.1: Conventional networking [5]	19
Image 1.2: Cisco's prediction for 24.3 exabytes of monthly mobile traffic until 2019 [10]	21
Image 1.3: The impact of the smart devices' and connections' increase on data traffic, according to Cisco [10]	22
Image 1.4: Video use will constitute more than 69% of the data traffic until 2019 , according to Cisco [10]	22
Image 1.5: Cloud applications will account for 90% of mobile data traffic until 2019, according to Cisco [10]	23
Image 1.6: An IoE schematic representation, according to Cisco [10].....	24
Image 1.7: 5G systems use cases and the main corresponding challenges [18]	27
Image 1.8: The general shift in networking [5]	27
Image 1.9: Conventional networking VS Software-Defined networking [6]	28
Image 2.1: High-level SDN overview [25]	30
Image 2.2: Basic SDN components [27]	31
Image 2.3: Simplified view of SDN architecture [6]	31
Image 2.4: Simplified SDN architecture and interfaces [6]	32
Image 2.5: SDN architecture and its fundamental abstractions [6]	33
Image 2.6: SDN architecture with management function [27].....	34
Image 2.7: SDN overview [27]	35
Image 2.8: The ODL Boron version architecture [37]	40
Image 3.1: QoE influence factors belonging to context, human user and the technical system [61]	54
Image 3.2: Relationship between R factor and MOS [66].....	60
Image 3.3: The IQX hypothesis [68]	64
Image 4.1: The OpenCache architecture [31].....	65
Image 4.2: The QoS/QoE Mapping and Adjusting application overview [24].....	66

Image 4.3: SDN-Based architecture for QoE optimization in HTTP-based video streaming [70].....	67
Image 4.4: The SDN-based vehicular network architecture [71].....	68
Image 4.5: The DASH-SDN architecture [72]	69
Image 4.6: The SDN-based scheme for HTTP video quality optimization [21]	70
Image 4.7: Topology of the experimental testbed of [19].....	70
Image 4.8: SDN-enabled cloud video distribution system [73].....	71
Image 4.9: Level-2 QoS flows rerouting in ARVS [29]	72
Image 4.10: Level-1 QoS flows rerouting in ARVS [29]	73
Image 4.11: Traffic-aware SDN-based topology for video streaming flows identification [25]	74
Image 4.12: The architecture of joint routing and layer selecting system [74]	75
Image 4.13: The block diagram for the system proposed in [20]	76
Image 4.14: An overview of the TSDN model proposed in [75]	77
Image 4.15: OpenFlow-assisted QoE Fairness Framework [30]	78
Image 4.16: SDN architecture for QoE-driven service optimization and path assignment [22]	78
Image 5.1: The result of mvn -v command after Apache Maven successful installation	82
Image 5.2: The ODL DLUX Login Page	83
Image 5.3: VirtualBox import window	84
Image 5.4: Mininet login console	85
Image 5.5: Example Python script for a custom topology creation [88]	87
Image 5.6: Mininet messages after starting a topology	88
Image 5.7: Mininet pingall result	88
Image 5.8: The created Mininet topology shown in ODL DLUX UI	89
Image 5.9: Import a project into IntelliJ	91
Image 5.10: Select the project to import	92

Image 5.11: Select to import a project from an external model, choosing Maven.....	92
Image 5.12: Click on button <i>Environment settings</i> and configure Maven's home directory.....	93
Image 5.13: Select Maven project to import and click <i>Next</i>	93
Image 5.14: Configure a new JDK.....	94
Image 5.15: Choose the JDK installation folder and click <i>OK</i>	94
Image 5.16: The JDK resources	95
Image 5.17: Enter a name for the project and click <i>Finish</i>	95
Image 5.18: D-ITG Architecture [92].....	96
Image 5.19: First Mininet session for using D-ITG.....	99
Image 5.20: Second Mininet session for using D-ITG, with topology deployment	99
Image 5.21: Navigation in the bin folder of the D-ITG installation for both hosts	100
Image 5.22: The command starting the receiver host (h_2).....	100
Image 5.23: The commands for the creation of script describing the traffic characteristics, in the sender host (h_1).....	100
Image 5.24: The command starting the sender host (h_1).....	101
Image 5.25: The receiver host's log file, with information about the received packets	101
Image 5.26: The sender host's log file, with information about the sent packets	102
Image 5.27: The output of <i>ffmpeg</i> command	103
Image 5.28: The output of <i>ffprobe</i> command.....	104
Image 6.1: Topology with nine switches and two hosts, used for the current thesis	105
Image 7.1: Traffic forwarding according to QoE Monitoring.....	114
Image 7.2: SQMF implementation structure	114
Image 7.3: ODL modules, including <i>sqmf</i>	115
Image 7.4: The created RPCs	115
Image 7.5: Requested input for <i>startMonitoringLinks</i> RPC.....	116
Image 7.6: The controller message when the new topology is created	129
Image 7.7: The provided input for VoIP traffic	130

Image 7.8: The provided input for video traffic.....	130
Image 7.9: Prompt for video selection, after input for video application type	131
Image 7.10: OF rules established in ingress switch for QoE monitoring.....	131
Image 7.11: OF rules established in core switches for QoE monitoring	132
Image 7.12: OF rules established in egress switch for QoE monitoring	132
Image 7.13: Part of the controller output after QoE monitoring has started.....	132
Image 7.14: The traffic received by the destination (h ₂)	133
Image 7.15: Traffic stops being delivered at the destination when link failure occurs and no corrective actions are made.....	133
Image 7.16: The OF rules established in the ingress switch when low QoE is detected	134
Image 7.17: The OF rules established in the egress switch when low QoE is detected	134
Image 7.18: Instant pause and recovery of traffic flow in case of link failure, using SQMF	134
Image 7.19 : Controller output when low QoE is detected.....	135
Image 7.20: Calling <i>stopMonitoringLinks</i> RPC	135
Image 7.21: The controller message after calling <i>stopMonitoringLinks</i>	135

LIST OF TABLES

Table 2.1: Match fields of a flow table rule.....	37
Table 2.2: Summarized overview of the history of programmable networks [6].....	45
Table 3.1: <i>R</i> factors, quality ratings and the associated MOS [66]	61
Table 3.2: Conditions for deriving coefficient tables [67]	63
Table 3.3: Provisional coefficient table for the video quality estimation function [67].....	63
Table 4.1: Comparison between state-of-the-art works on QoE for SDN networks	79
Table 6.1: Parameters used for experiment N° 1 on VoIP traffic generation	106
Table 6.2: Delay, <i>R</i> and MOS according to the total network packet loss	106
Table 6.3: Delay, Packet Loss, <i>R</i> and MOS according to number of VoIP packets sent simultaneously with iperf	108
Table 6.4: Parameters used for experiments on video streaming.....	109
Table 6.5: <i>V_q</i> according to the total network packet loss.....	109
Table 6.6: Packet loss and <i>V_q</i> during video streaming simultaneously with iperf	110
Table 7.1: Parameters used for SQMF evaluation on VoIP traffic	135
Table 7.2: Delay, <i>R</i> , packet loss and MOS during VoIP traffic generation with link failure, without SQMF	136
Table 7.3: Delay, <i>R</i> , packet loss and MOS during VoIP traffic generation with link failure, with SQMF	137
Table 7.4: Packet loss and <i>V_q</i> during video streaming with link failure, without SQMF	139
Table 7.5: Packet loss and <i>V_q</i> during video streaming with link failure, with SQMF	140

PREFACE

The current MSc Thesis was conducted in the Department of Informatics and Telecommunications of National and Kapodistrian University of Athens, and more specifically under the support of the Communication Networks Laboratory of Telecommunications and Signal Processing sector. The thesis started being conducted in March 2017 and ended in October of the same year. Professor Lazaros Merakos and PhD Candidate Eirini Liotou – to whom I owe special thanks - were the supervisors of this MSc Thesis.

1. INTRODUCTION

For many years, the Information and Communication Technologies (ICTs) have been representing highly profitable business areas with continuous developments of technologies, devices and services in order to serve all types of users [1]. The innovative and effective utilization of ICTs is becoming more and more important for the world economy improvement [2], as the industries' and services' capability to compete and evolve is increasingly depending on them [3]. Investments in ICTs can play a vital role in the pathway to economic recovery, given the strong external factors which positively affect the economy.

Undoubtedly, the communications and computer networking sector is one of the most crucial elements in the global ICT strategy, underpinning many other industries. It is one of the fastest growing and most dynamic sectors worldwide, allowing for the interconnection between either individual persons or institutions, companies, businesses, industries and in general every kind of functional departments worldwide [2]. Lately, a drive for changing the conventional networking architecture and moving towards new networking paradigms is beginning to show. This trend can be explained by a number of factors related to the current state in computer systems networking, as well as to the emerging needs of next generation networks.

1.1 CURRENT NETWORKING STATE AND LIMITATIONS

Despite the fact that computing has advanced rapidly over the past three decades, the way that networking is performed has remained virtually unchanged [4]. The current state of networking is characterized by the legacy technology which the majority of networks are built on. In conventional networking (Image 1.1), the networking protocols are distributed among the devices (i.e. routers, switches, firewalls and middle boxes) [5]. The distributed control and transport network protocols running inside the routers and switches are the key technologies that allow information, in the form of digital packets, to travel around the world [6], [7]. The most traditional networking approach is Internet Protocol (IP) networking and most of the public Internet still operates on hybrid IP version 4 / version 6 (IPv4/v6) services like Network Address Translation (NAT) [8].

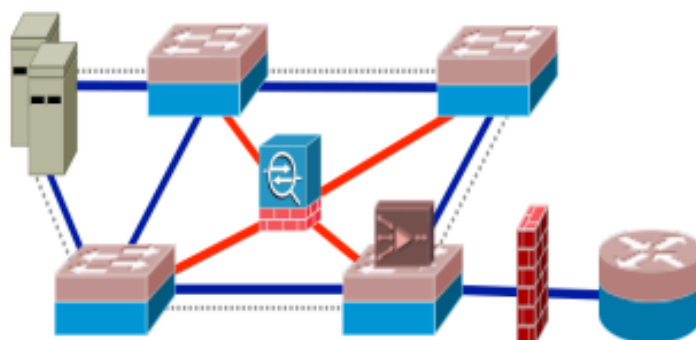


Image 1.1: Conventional networking [5]

Despite their widespread adoption, traditional IP networks have severe disadvantages and impose limitations on creating new, innovative services. The root cause of these limitations is the fact that the networks are built using devices (e.g. switches, routers) which have become exceedingly complex as they implement an ever-increasing number of distributed protocols and use closed and proprietary interfaces.

Historically, the “best” networks (those which are the most reliable, have the highest availability, and offer the fastest performance, etc.) are those built with custom silicon (Application-specific Integrated Circuits - ASICs) and purpose-built hardware. Due to

the fact that it takes a significant investment to build custom silicon and hardware, rigorous processes are required to ensure that vendors get the most out of each update or new iteration. This means that the ad hoc addition of features is virtually impossible, limiting customers who want new or different functionality to address their requirements to the vendor's timeline [9].

Consequently, the main limitations that emerge from the current networking state are the following:

- **Optimization difficulty:** It is hard for network operators to introduce new revenue generating services and optimize their expensive infrastructures, i.e. data centers, wide-area networks, and enterprise networks, as they continue having serious known problems with security, robustness, manageability, mobility and evolvability that have not been successfully addressed so far [4]. New networking features are commonly introduced via expensive, specialized and hard-to-configure equipment [5].
- **Capital costs:** Network capital costs have not been reducing fast enough and operational costs have been growing, putting excessive pressures on network operators. The transition from IPv4 to IPv6 started more than a decade ago and is still largely incomplete, while in fact IPv6 represents merely a protocol update. Due to the inertia of current IP networks, a new routing protocol can take 5 to 10 years to be fully designed, evaluated and deployed. Likewise, a clean-state approach to change the Internet architecture (e.g., replacing IP), is regarded as a daunting task – simply not feasible in practice. Ultimately, this situation has inflated the capital and operational expenses of running an IP network [4], [6].
- **Customization difficulty:** Even vendors and third parties are not able to provide customized cost effective solutions to address their customers' problems [4].
- **Configuration complexity:** To express the desired high-level network policies, network operators need to configure each individual network device separately and sometimes manually, using low-level and often vendor-specific commands, a procedure which is error-prone [5]. In addition to the configuration complexity, network environments have to endure the dynamics of faults and adapt to load changes. Automatic reconfiguration and response mechanisms are virtually non-existent in current IP networks. Enforcing the required policies in such a dynamic environment is therefore highly challenging [6].
- **Vertical integration:** The control plane (which decides how to handle network traffic) and the data plane (which forwards traffic according to the decisions made by the control plane) are tightly coupled, therefore there is no common view of the network [5]. They are bundled inside the networking devices, reducing flexibility and hindering innovation and evolution of the networking infrastructure [6]. It is hard to implement new features and protocols as this requires changing the control plane of all devices which are part of the topology [5].

1.2 DEVICES GROWTH

Over the past years, there is a spectacular increase in the number of users who subscribe to mobile broadband systems every year. The figures presented in Cisco's latest report [10] show an exponential increase in the number of devices, which will continue in the forthcoming years. Besides, more and more users are seeking faster internet access, more advanced mobile phones, and generally direct communication with other people and access to information [2]. Thus, they are increasingly employing mobile personal devices such as smartphones, tablets, and notebooks to access the internet. Every year, several new devices in different formats with sophisticated

capabilities and intelligence appear on the market. Stronger smartphones and laptops are becoming more popular nowadays, due to their additional multimedia capabilities. This fact has led to a significant increase in the production and distribution of wireless mobile devices and related services. In addition, as mobile network capacity is improving, the number of users with more than one device in their possession is increasing as well.

The Wireless World Research Forum (WWRF) has predicted that seven trillion wireless devices will serve seven billion users by the end of 2017. This means that the number of connections to wireless communications networks will reach a thousand times the world population [1], [2]. Almost half a billion (497 million) mobile devices and connections were added in 2014, whereas mobile devices and connections expected to grow globally to 11.5 billion by 2019 [10].

One of the most significant contributing factors behind the device revolution has been the democratization of workforce-technology in recent years. While it has provided a lot of value to people around the world, there is also a more pragmatic impact: the deluge of new vendors and equipment on the market today means that there is very little uniformity in terms of the way devices operate on a network. Since the majority of networks today cater to a variety of devices from different vendors simultaneously, it can be difficult for network management systems to keep up with each vendor's specific device requirements [8].

Therefore, the ICT sector is under pressure to accommodate these personal devices in a fine-grained manner while protecting corporate data and intellectual property and meeting compliance mandates [11].

1.3 DATA AND VIDEO GROWTH

Based on Cisco's annual reports, there is quantified evidence that there is a rapid growth of data traffic over wireless networks and it will continue [2]. Already in 2014, there was an increase in data traffic by 69% and it is expected to grow to 24.3 exabytes per month by 2019. Mobile data traffic will follow a compound annual growth rate (CAGR) of 57% from 2014 to 2019 [10], as shown in Image 1.2.

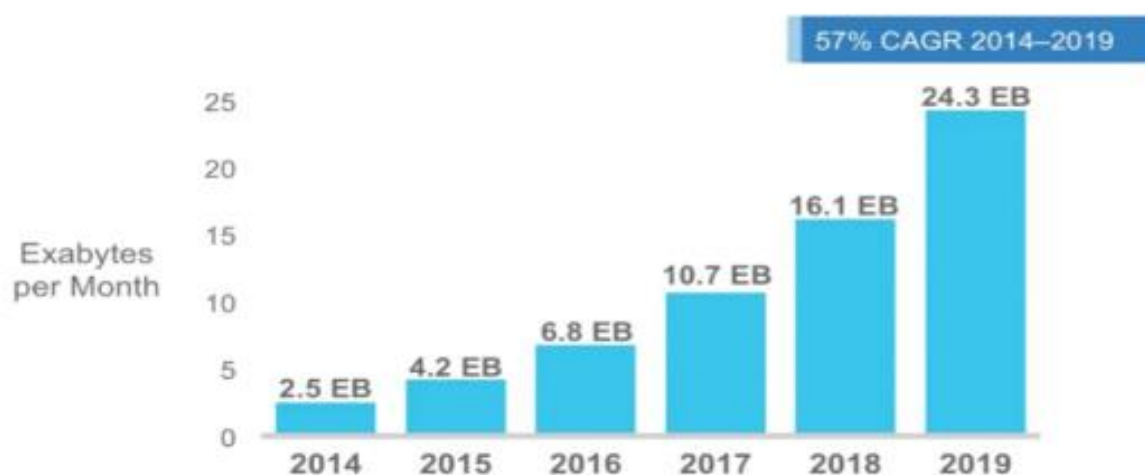


Image 1.2: Cisco's prediction for 24.3 exabytes of monthly mobile traffic until 2019 [10]

Image 1.3 depicts the impact of the aforementioned increase in mobile devices and connections in global data traffic. In particular, the global data traffic caused by new smart devices is projected to increase from 88% to 97% by 2019 [10].

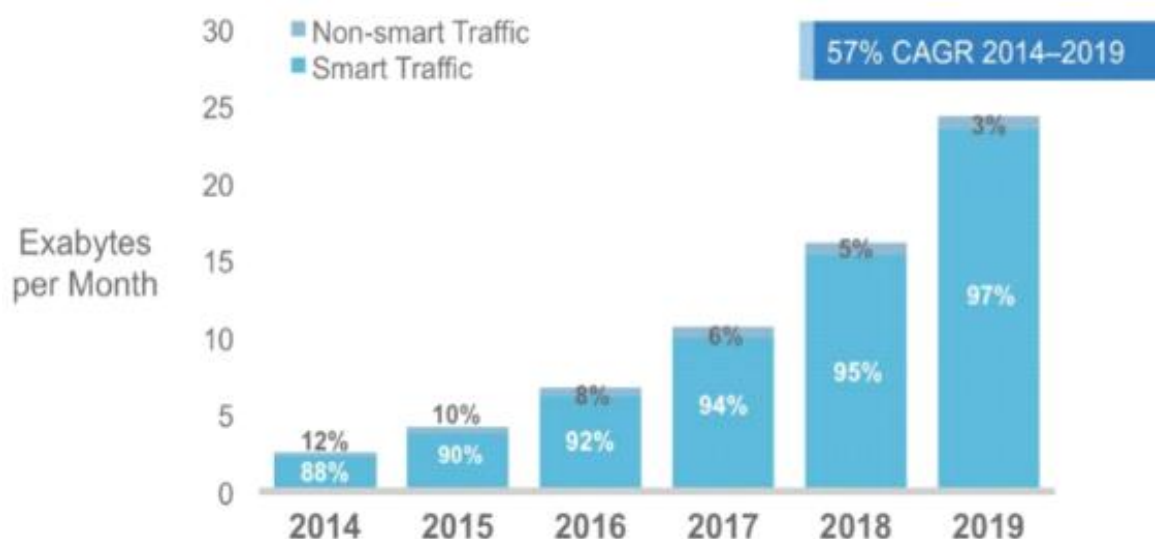


Image 1.3: The impact of the smart devices' and connections' increase on data traffic, according to Cisco [10]

A remarkable fact is the traffic that this new type of devices with increased potential, which are spreading at a rapid pace, can cause. Specifically, a single smartphone can generate as much traffic as 37 basic phones, a tablet as much traffic as 94 key phones, while a single laptop's traffic is equivalent to 119 phones'. Another factor of great importance in increasing data traffic in the forthcoming years is the rapid increase in video usage, given the higher bit rates compared to other types of applications. Video is already considered as a necessary asset by many mobile users as it occupies much of their everyday routine and habits, making it the most involved player in the global data traffic. High-resolution video is expected to spread, while the percentage of content watched via live streaming compared to downloaded content is expected to increase. Mobile video will follow a CAGR of 66% between 2014 and 2019. More specifically, out of a total of 24.3 exabytes per month in which data traffic is expected to arrive by 2019, 17.4 exabytes are expected to be due to video use, implying more than 69% of total traffic [10], as depicted in Image 1.4.

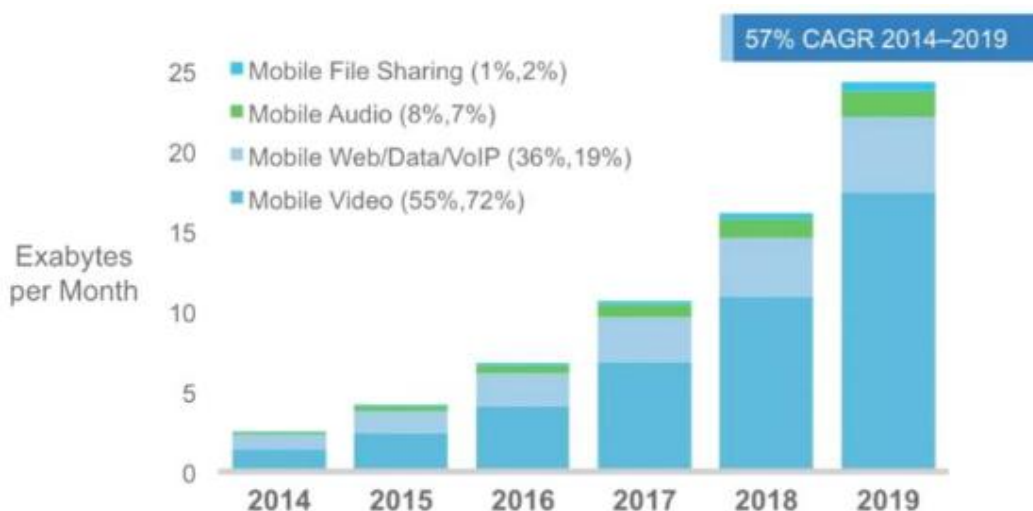


Image 1.4: Video use will constitute more than 69% of the data traffic until 2019 , according to Cisco [10]

Since many web video applications belong to the Cloud application class, Cloud traffic follows a video-like curve, and therefore a further increase is expected in Cloud-based mobile applications, which demonstrate unique latency and bandwidth features. Unlike

Cloud applications and services, mobile devices are subject to memory and speed limitations, which are a brake on providing multimedia applications at a satisfactory level. In fact, more complex applications (such as voice recognition, navigation) are often downloaded to a Cloud Server in order to relieve the mobile devices from processing and energy costs. Cloud applications and services (such as YouTube and Spotify) allow mobile users to overcome memory capacity limitations and mobile processing speeds. While this fact effectively ensures greater austerity in smartphone or tablet operations, it also stresses the need for a reliable, low-latency and high-bandwidth Internet connection. It should be noted that Cloud applications are estimated to generate up to 90% of the world's mobile data traffic by 2019 [10], as depicted in Image 1.5.



Image 1.5: Cloud applications will account for 90% of mobile data traffic until 2019, according to Cisco [10]

Enterprises have enthusiastically embraced both public and private cloud services, resulting in unprecedented growth of these services. Furthermore, the planning for cloud services must be performed in an environment of increased security, compliance, and auditing requirements, along with business reorganizations, consolidations, and mergers that can change assumptions overnight. Providing self-service provisioning, whether in a private or public cloud, requires elastic scaling of computing, storage, and network resources, ideally from a common viewpoint and with a common suite of tools.

Regarding the network traffic, another fact is that within the enterprise data centers, traffic patterns have changed significantly. In contrast to client-server applications where the bulk of the communication occurs between one client and one server, today's applications access different databases and servers. At the same time, users are changing network traffic patterns as they push for access to corporate content and applications from any type of device (including their own), connecting from anywhere, at any time. Finally, many enterprise data centers managers are contemplating a utility computing model, which might include a private cloud, public cloud, or some mix of both, resulting in additional traffic across the wide area network [8].

A last challenge occurs when handling today's big data or mega datasets, a procedure which requires massive parallel processing on thousands of servers, all of which need direct connections to each other. The rise of mega datasets is fueling a constant demand for additional network capacity in the data center. Operators of hyperscale data center networks face the daunting task of scaling the network to a previously unimaginable size and maintaining any-to-any connectivity, keeping their costs at a reasonable level at the same time [11].

Summarizing, as the amount of data that users consume on a daily basis continues to increase, the means of processing data will need to evolve to meet this new demand. When the public Internet was taking its first steps, today's ubiquity of broadband Internet could not have been predicted. Early iterations of today's networking protocols were well-suited to the archaic Internet of the 1990s, but in the context of the modern Internet – and even more so the Internet of the future – these protocols are obsolete [8].

1.4 FIFTH GENERATION (5G) WIRELESS NETWORKS EMERGENCE

The success of communication networks is reflected in fast growing technological advances. In only two decades the first generation mobile networks (1G) have been replaced from the wireless fourth generation communication networks 4G-Long Term Evolution (4G -LTE) [12], which can support data rates up to 1 Gbps for low mobility and up to 100 Mb/s for high mobility. The LTE systems and their evolution, LTE-Advanced (LTE-A) systems, have been developed and continue to evolve worldwide [2].

It is worth noting that according to the annual reports published by Cisco, it seems that around 2019, 4G connectivity will surpass the use of 2G connections. This is expected to happen given the rapid spread of mobile applications and the expected growth of Internet connections through mobile devices on the one hand, and on the other hand the need for optimal bandwidth and other network resources management. Thus, the development and adoption of 4G systems worldwide is facilitated [10].

However, given the explosion of data traffic in wireless networks during the next years, several challenges emerge which 4G systems are unable to face. This fact creates the need for switching to a technology which will satisfy the ever increasing demand for higher data rates, enhanced network capacity, better spectral and energy performance and higher mobility. Of course, the new technology needs to also meet the requirements of the wired network's parts, aiming for a fully integrated approach. The step which is expected to meet the above requirements is the fifth generation wireless communication networks technology, 5G.

Another factor that leads to the need to develop the next generation 5G systems is the obvious signs of moving towards the Internet of Everything (IoE) and Internet of Things (IoT). IoE and IoT are networks of physical objects or "things" that integrate electronic components, software, sensors and connectivity units in order to achieve higher quality of service by exchanging data with the manufacturer, administrator and / or other connected devices (Image 1.6). Each object is uniquely identifiable through a built-in computing system and is capable of operating within the existing web infrastructure. The IoE allows end users to access any content in any device from anywhere [10].

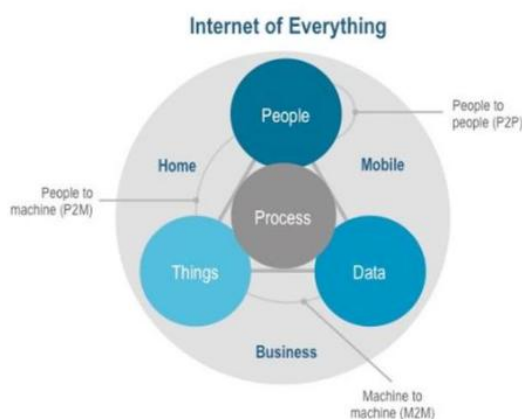


Image 1.6: An IoE schematic representation, according to Cisco [10]

1.4.1 5G systems requirements

The expected wireless data traffic explosion, as described earlier, combined with the transition to an “IoT world”, poses a set of requirements, which have to be met by the next generation broadband networks, in order to satisfy the increasingly emerging users’ needs. These requirements are mainly identified in the following fields [13], [14], [15]:

- Capacity and transmission rates enhancement
- Latency reduction
- Energy saving
- High reliability
- Support for various types and large volumes of terminal devices
- New spectrum management approaches
- Ultra-high service availability and reliability

The predicted increase in data traffic by 10.000 times raises the need for developing systems which will achieve higher transmission rates as well as higher spectral density levels. The below targets are set for the transmission rate metrics and constitute a challenge for the fifth generation networks [13], [14]:

- The **aggregate data rate**, or in other words the total data volume which can be served by the network (bits/s/area), needs to be **increased at least by 1000 times** compared to the fourth generation networks.
- The **edge rate**, or in other words the worst data rate that a user can reasonably expect to receive when in range of the network, is demanded to be **increased at least by 100 times** compared to the corresponding rate in the fourth generation networks, ranging from 100 Mbps to 1Gbps. This range depends on various factors, such as the network load and the cell size. However, in any case, goals are set for a high rate which will enable every user to at least receive services such as a high definition live-streaming video.
- As for the **peak rate**, which is the best-case data rate that a user can hope to achieve under any conceivable network configuration, it is required to be in range of **tens of Gbps**.

At the same time, a plethora of new and impressive applications is expected to be supported by the next generation networks, including unmanned vehicle control, remote medical monitoring and virtual reality applications. It can be easily understood that this applications class is particularly sensitive to delays, even compared to a high definition live-streaming video application. In order to make it possible to support the aforementioned applications, the aim is to reduce the roundtrip latency under 1ms and the jitter under 20 μ s. Such an aim requires that the control and signaling procedures, the time scheduling and allocation of resources, the establishment of new connections and several other functions must take place in the minimum possible time [13], [14].

The issue of energy saving constitutes a major issue of global concern, especially for the ICT sector. Reducing power consumption and increasing battery life have always been significant factors for mobile communications, which are becoming increasingly important as network infrastructure, devices and services are evolving. However, the expected increase in the transmission speed per connection as well as the exponential increase in the number of devices and base stations will inevitably lead to an increase

in energy consumption, which requires the development of technological solutions in order to maintain consumption at similar levels to those of the previous generation networks, or even better to achieve a reduction [16].

It is therefore expected that in the next years the energy requirements will skyrocket consumption numbers, raising the need for special care so that both providers (as the increase in energy consumption significantly increases the amount of operating costs) and users can benefit from lengthened battery life. More specifically, for next generation networks, a **10-year battery life** is set as a goal to be achieved, partly through the development of battery technology, but also through the efficient management of data traffic and signaling [2], [13].

On the one hand, the growth of network devices per user (one user will own multiple smartphones, tablets, laptops, etc.) and, on the other hand, the growth of Machine-to-Machine (M2M) communications and the expected transition to an “IoT world” imply the need for many different types of devices support and for efficient management of a large volume of simultaneous connections. As the number of connected devices is estimated to be billions, perhaps trillions in 2020 [17], network scalability becomes more and more important. An average macrocell is estimated to support up to 10,000 low bandwidth devices, alongside any high-end devices of mobile users. Therefore, there is a need for networks to be flexible in order to serve just as effectively a range of devices, from very simple ones that send sporadic data (such as sensor devices) , to devices sending large volumes of data (such as laptops) with a high frequency. Changes in the control plane and in the management of the network are considered necessary to support the above.

Finally, next-generation systems need to be cost-effective, in order to benefit both the provider and the end user. New solutions have to be found in order to achieve reduced Capital Expenditure (CAPEX) and Operational Expenses (OPEX) and therefore profitability for the providers as well as a lower average revenue per user (ARPU).

To sum up, as mobile communication networks are expected to become the focus of human-to-human, human-to-machine and machine-to-machine communication, they will have to compete with and overcome optical communications networks in terms of both Quality of Service (QoS), as well as reliability. Therefore, next generation networks should reach speeds of 10 Gbps and roundtrip delays not exceeding 1ms, while billions of simultaneous connections with devices of varying capabilities and requirements will be supported. Therefore, the network should be designed to be flexible, robust, reliable and energy-efficient [17].

Image 1.7 depicts the use cases and the corresponding challenges which the 5G systems are expected to deal with.

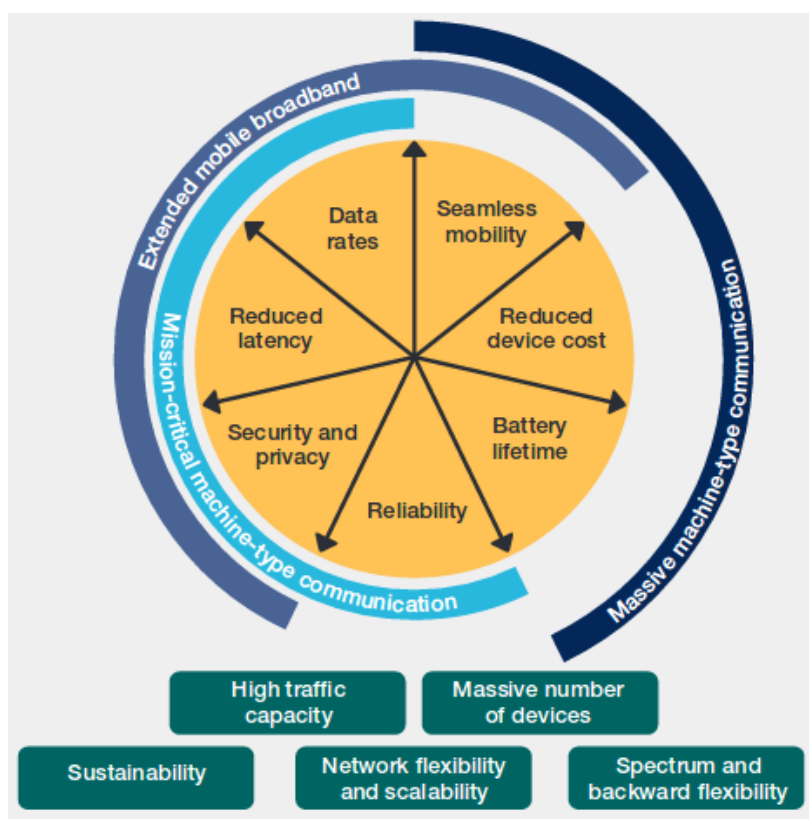


Image 1.7: 5G systems use cases and the main corresponding challenges [18]

1.5 THE NEW NETWORKING PARADIGM

Summarizing the points mentioned above, the explosion of mobile devices and content, server virtualization, and advent of cloud services are among the trends driving the networking industry to re-examine traditional network architectures. Many conventional networks are hierarchical, built with tiers of Ethernet switches arranged in a tree structure. This design made sense when client-server computing was dominant, but such a static architecture cannot deal with the dynamic computing and storage needs of today's computing environments, such as enterprise data centers, campuses, and carrier environments [11].

Taking into consideration the points mentioned above, it is evident that there is a need for a general shift in networking which has already started to take place and is depicted in Image 1.8.

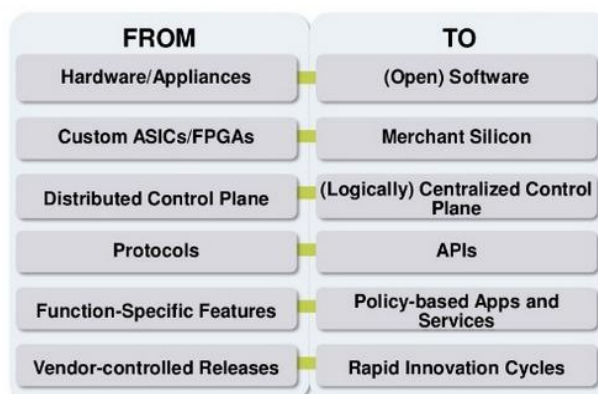


Image 1.8: The general shift in networking [5]

Software-defined networking (SDN) is an approach to computer networking meant to address the above challenges, deriving from the current network infrastructure's stability and avoiding the need for expensive uplifts or hardware investments [7], [11]. This is achieved by decoupling the system that makes decisions about where traffic is sent (known as control plane) from the underlying systems that forward traffic to the selected destination (known as the data plane). Image 1.9 shows the contrast between conventional networking and software-defined networking. SDN will be presented in detail in Chapter 2.

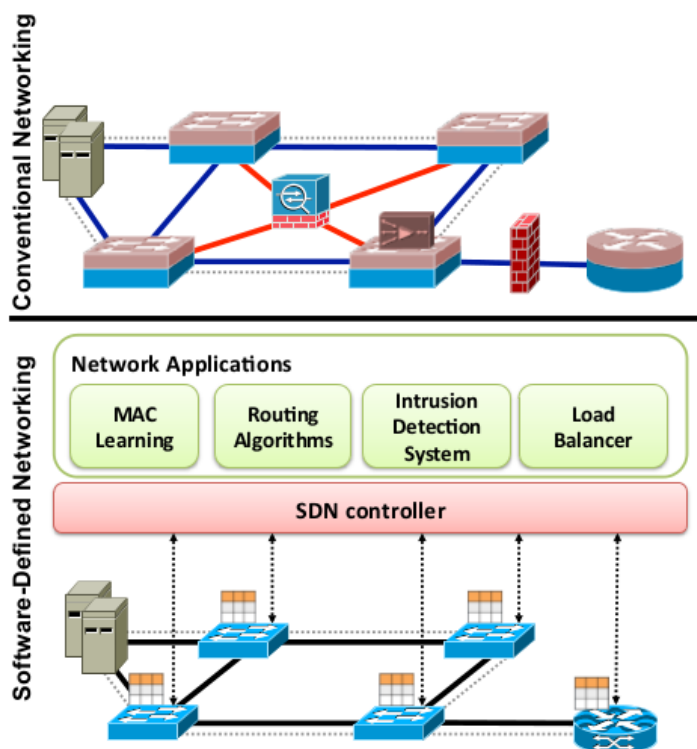


Image 1.9: Conventional networking VS Software-Defined networking [6]

2. SOFTWARE-DEFINED NETWORKING (SDN)

2.1 DEFINITION(S) OF SDN

Briefly summarizing the key aspects of the current and future networking state as explained in the previous chapter, the majority of traditional networks are built with Ethernet switches, which are arranged in a tree structure. This design was adequate in the time that client-server computing was at the forefront. But currently things have changed and the concept of virtualization has entered at various layers of the computing industry, spanning from the application/service layer down to networking and function virtualization [19]. The 5G mobile communication networks will be expected to support connections with data rates that are 10 to 100 times higher than current mobile networks with up to 1000 times more data by volume in any geographical area and up to 100 times as many connected devices. However, in order to meet the demanding Key Performance Indicators (KPIs) set for 5G networks, researchers will need to overcome a number of significant challenges. A significant challenge will be managing the anticipated exponential growth in multimedia traffic whilst meeting the quality expectations of end users. In addition to the predicted massive increase in the number of devices attached to 5G networks, the 5G KPIs expect that Ultra High-Definition (UHD) video will be supported across a range of applications such as Internet Protocol Television (IPTV) and Video on Demand services (VoD) with each UHD stream potentially requiring up to 16 times as much bandwidth as current High-Definition (HD) streams. According to Cisco's latest networking forecast, mobile video will account for about 75% of all mobile data traffic by 2019, representing a 13-fold increase since 2014 [20].

All the above mentioned factors prompt the ICT industry to revise their opinions about current traditional network architectures. The technology which will serve the emerging needs is called Software Defined Networking (SDN) [19].

As already mentioned, SDN is a relatively new approach to computer networking introduced to face the challenges deriving from the complex network protocols and functions and bloated network equipment. The idea of SDN originated at Stanford University and then became widely recognized by academia and industry areas [21] around 2009.

SDN is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures. First, it breaks the vertical integration by separating the network's control logic (the control plane, which used to be tied to a particular infrastructure element, and thus be vendor and device specific [22]) from the underlying routers and switches that forward the traffic (the data plane), using an open standard protocol for the communication between them [23]. Second, with the separation of the control and data planes, the function of control element no longer executes in the switches but rather in an external server [24]. The network switches become simple forwarding devices routing the traffic according to rules set to them by the control plane, and the control logic is implemented in a logically centralized controller (or network operating system) with a global view of the entire network as well as of all competing traffic flows traversing the network [20], thus simplifying policy enforcement and network (re)configuration and evolution [25]. Having simultaneous access to both views could provide tremendous potential benefit when managing the transmission of bandwidth-hungry, delay-intolerant multimedia flows over the 5G network [20]. The migration of control provides an abstraction of the underlying network for the applications residing on upper layers, enabling them to treat the network as a logical or virtual entity [21].

SDN's key attributes include:

- separation of data and control planes
- a uniform vendor-agnostic interface between control and data planes
- a logically centralized control plane that offers a consistent, system-wide programming interface to users and operators
- slicing and virtualization of the underlying network [4].

The Open Networking Foundation (ONF) is the group that is most associated with the development and standardization of SDN. According to the ONF, “*Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow™ protocol is a foundational element for building SDN solutions*” [26].

The networking industry has on many occasions shifted from the original view of SDN, by referring to anything that involves software as being SDN. A much less ambiguous definition of SDN can be given based on four pillars:

- The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
- Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN context, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and middle boxes. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables.
- Control logic is moved to an external entity, the so-called SDN controller or Network Operating System (NOS). The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.
- The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition [6].

2.2 SDN ARCHITECTURE

In implementation, a high-level overview of the SDN architecture is illustrated in Image 2.1, and can be regarded as consisting of three layers: infrastructure layer, control layer and application layer.

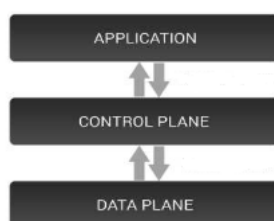


Image 2.1: High-level SDN overview [25]

Image 2.2 introduces the basic SDN components. The initial view of the three layers (red text) is translated to data plane, control plane and application plane respectively (black text) [27].

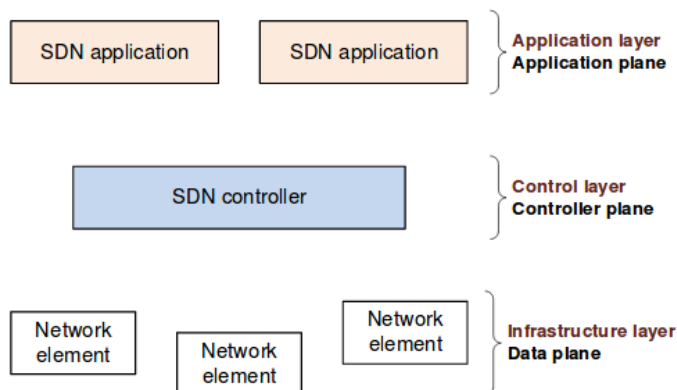


Image 2.2: Basic SDN components [27]

A simplified view of this architecture is shown in Image 2.3. It is important to emphasize that a logically centralized programmatic model does not postulate a physically centralized system. In fact, the need to guarantee adequate levels of performance, scalability and reliability would preclude such a solution. Instead, production-level SDN network designs resort to physically distributed control planes [6].

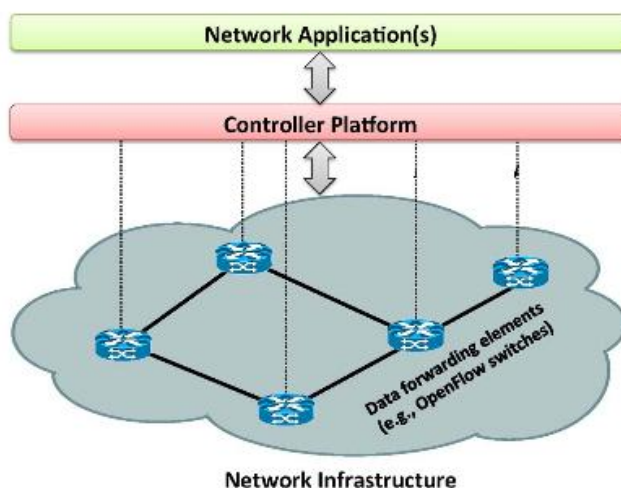


Image 2.3: Simplified view of SDN architecture [6]

The **data plane** comprises a set of one or more network elements, each of which contains a set of traffic forwarding or traffic processing resources. Resources are always abstractions of underlying physical capabilities or entities.

The **control plane** comprises a set of one or more SDN controllers, each of which has exclusive control over a set of resources exposed by one or more network elements in the data plane (its span of control). The minimum functionality of the SDN controller is to faithfully execute the requests of the applications it supports, while isolating each application from all others. To perform this function, an SDN controller may communicate with peer SDN controllers, subordinate SDN controllers, or non-SDN environments, as necessary. A common but non-essential function of an SDN controller is to act as the control element in a feedback loop, responding to network events to recover from failure, re-optimize resource allocations, or otherwise.

The implementation of the SDN control plane can follow a centralized, hierarchical, or decentralized design. Initial SDN control plane proposals focused on a centralized solution, where a single control entity has a global view of the network. While this simplifies the implementation of the control logic, it has scalability limitations as the size and dynamics of the network increase. To overcome these limitations, several approaches have been proposed in the literature that fall into two categories, hierarchical and fully distributed approaches. In hierarchical solutions, distributed controllers operate on a partitioned network view, while decisions that require network-wide knowledge are taken by a logically centralized root controller. In distributed approaches, controllers operate on their local view or they may exchange synchronization messages to enhance their knowledge. Distributed solutions are more suitable for supporting adaptive SDN applications [11].

A key issue when designing a distributed SDN control plane is to decide on the number and placement of control entities. An important parameter to consider while doing so is the propagation delay between the controllers and the network devices, especially in the context of large networks. Other objectives that have been considered involve control path reliability, fault tolerance, and application requirements [11].

The **application plane** comprises one or more applications, each of which has exclusive control of a set of resources exposed by one or more SDN controllers. An application may invoke or collaborate with other applications. An application may act as an SDN controller in its own right [27].

In an SDN-enabled world, new open interfaces exist between the application plane, the data plane and the control plane. These are illustrated in Image 2.4.

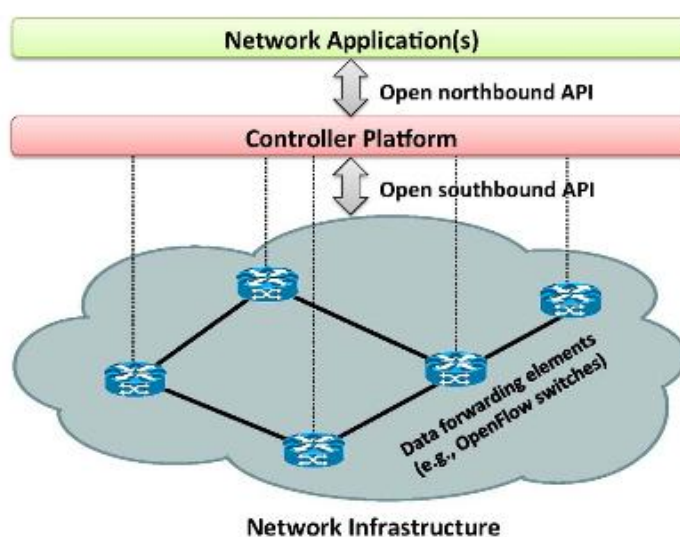


Image 2.4: Simplified SDN architecture and interfaces [6]

The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller [6]. The controller exercises direct control over the state in the data-plane elements via this well-defined application programming interface (API). The interface that bridges the data plane and the control plane is called the **Southbound-API (SBI)**. It enables the externalization of the control plane from the forwarding device to the logically-centralized network control plane (controller). As a software entity, the controller can be freely programmed and adapted to the network according to the operator's requirements.

While the SBI is an important component of SDN, a significant additional value of SDN lies within the **Northbound-API (NBI)** interface between the control plane and the application plane, i.e. applications running on top of or interacting with the network itself. This enables applications to be executed on top of the network as well as the exchange of information about the application and network state. It makes the controller programmable and responding to changes in the network [28]. SDN architecture segregates control planes from forwarding devices. The segregated control planes are combined in design and therefore regarded as a centralized controller which has a global view of the entire network. Therefore, the controller presents a level of abstraction of the underlying network. In other words the controller acts as middleware that provides a higher level of abstraction to network developers [23]. Hence, flexible reconfiguration of the network becomes possible by the introduction of a centralized controller [25].

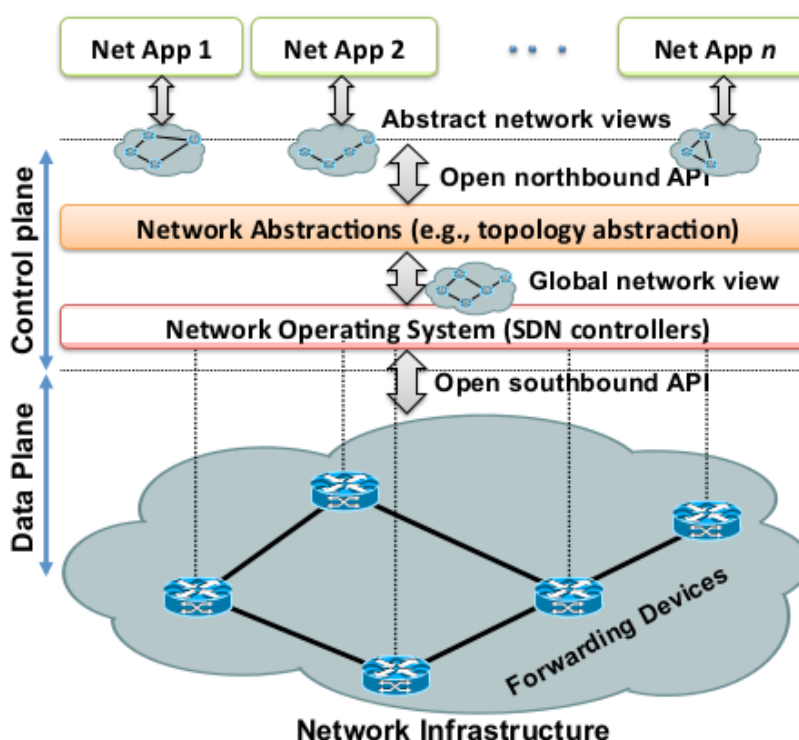


Image 2.5: SDN architecture and its fundamental abstractions [6]

Image 2.5 summarizes thoroughly the SDN architecture and its fundamental abstractions. Keeping in mind the components described, Image 2.6: SDN architecture with management function [27] adds the management function, which is often omitted from simplified SDN representations. Although many traditional management functions may be bypassed by the direct application-control plane interface (NBI), certain management functions are still essential. In the data plane, management is at least required for initially setting up the network elements, assigning the SDN-controlled parts and configuring their SDN controller. In the control plane, management needs to configure the policies defining the scope of control given to the SDN application and to monitor the performance of the system. In the application plane, management typically configures the contracts and service license agreements (SLAs). In all planes, management configures the security associations that allow distributed functions to safely intercommunicate [25].

Each application, SDN controller and network element has a functional interface to a manager. The minimum functionality of the manager is to allocate resources from a

resource pool in the lower plane to a particular client entity in the higher plane, and to establish reachability information that permits the lower and higher plane entities to mutually communicate. Additional management functionality is not precluded, subject to the constraint that the application, SDN controller, or network element (NE) have exclusive control over any given resource [25].

Image 2.6 summarizes the SDN architecture including the management function. It shows distinct application, controller and data planes, with controller plane interfaces (CPIs) designated as reference points between the SDN controller and the application plane (A-CPI) and between the SDN controller and the data plane (D-CPI). The information exchanged across these interfaces should be modeled as an instance of a protocol-neutral information model [27].

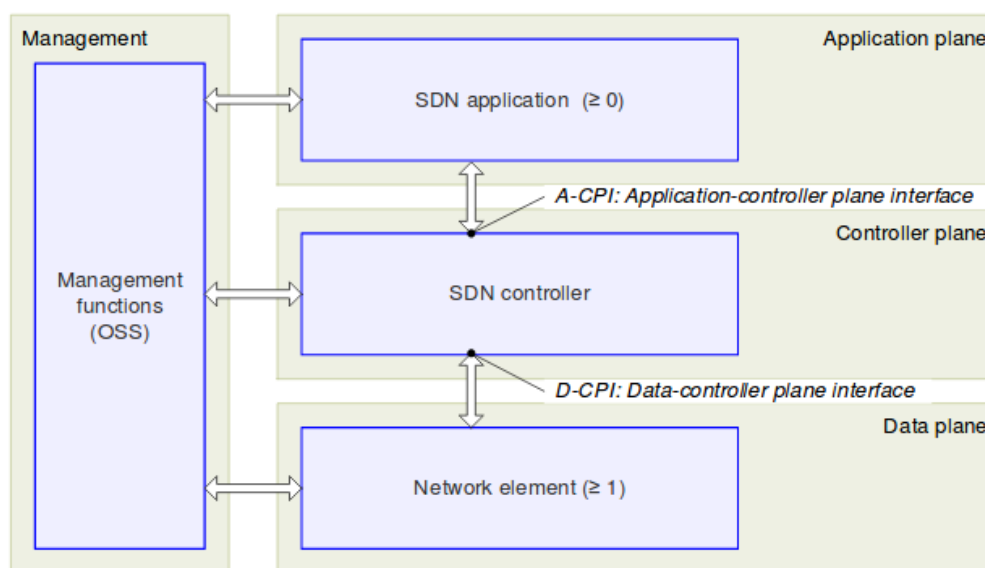


Image 2.6: SDN architecture with management function [27]

In other words, the infrastructure layer (data plane) comprises network elements, which expose their capabilities toward the control layer (control plane) via interfaces southbound from the controller. The SDN applications exist in the application layer (application plane), and communicate their network requirements toward the control plane via northbound interfaces (NBIs). In the middle, the SDN controller translates the applications' requirements and exerts low-level control over the network elements, while providing relevant information up to the SDN applications. An SDN controller may orchestrate competing application demands for limited network resources according to policies. The concept of a data plane in the context of the SDN architecture includes traffic forwarding and processing functions. A data plane may include the necessary minimum subset of control and management functions [27].

In order to sum up the current subchapter, the following list which summarizes the major SDN architectural components and interfaces is presented. Image 2.7 also provides an illustrative summary.

- **SDN Applications:** SDN Applications are programs that explicitly, directly, and programmatically communicate their network requirements and desired network behavior to the SDN Controller via a NBI. In addition they may consume an abstracted view of the network for their internal decision-making purposes. An SDN Application consists of one SDN Application Logic and one or more NBI Drivers. SDN Applications may themselves expose another layer of abstracted network control, thus offering one or more higher-level NBIs through respective NBI agents.

- **SDN Controller:** The SDN Controller is a logically centralized entity in charge of translating the requirements from the SDN Application layer down to the SDN Datapaths and providing the SDN Applications with an abstract view of the network (which may include statistics and events). An SDN Controller consists of one or more NBIs, the SDN Control Logic, and the SBI. Its definition as a logically centralized entity neither prescribes nor precludes implementation details such as the federation of multiple controllers, the hierarchical connection of controllers, communication interfaces between controllers, nor virtualization or slicing of network resources.
- **SDN Datapath:** The SDN Datapath is a logical network device that exposes visibility and uncontested control over its advertised forwarding and data processing capabilities. The logical representation may encompass all or a subset of the physical substrate resources. An SDN Datapath comprises an SBI agent and a set of one or more traffic forwarding engines and zero or more traffic processing functions. These engines and functions may include simple forwarding between the Datapath's external interfaces or internal traffic processing or termination functions. One or more SDN Datapaths may be contained in a single (physical) network element—an integrated physical combination of communications resources, managed as a unit. An SDN Datapath may also be defined across multiple physical network elements.
- **SDN SBI:** The interface defined between an SDN Controller and an SDN Datapath, which provides at least programmatic control of all forwarding operations, capabilities advertisement, statistics reporting and event notification. One value of SDN lies in the expectation that the SBI is implemented in an open, vendor-neutral and interoperable way.
- **SDN NBIs:** Interfaces between SDN Applications and SDN Controllers which typically provide abstract network views and enable direct expression of network behavior and requirements. This may occur at any level of abstraction (latitude) and across different sets of functionality (longitude). One value of SDN lies in the expectation that these interfaces are implemented in an open, vendor-neutral and interoperable way [11].

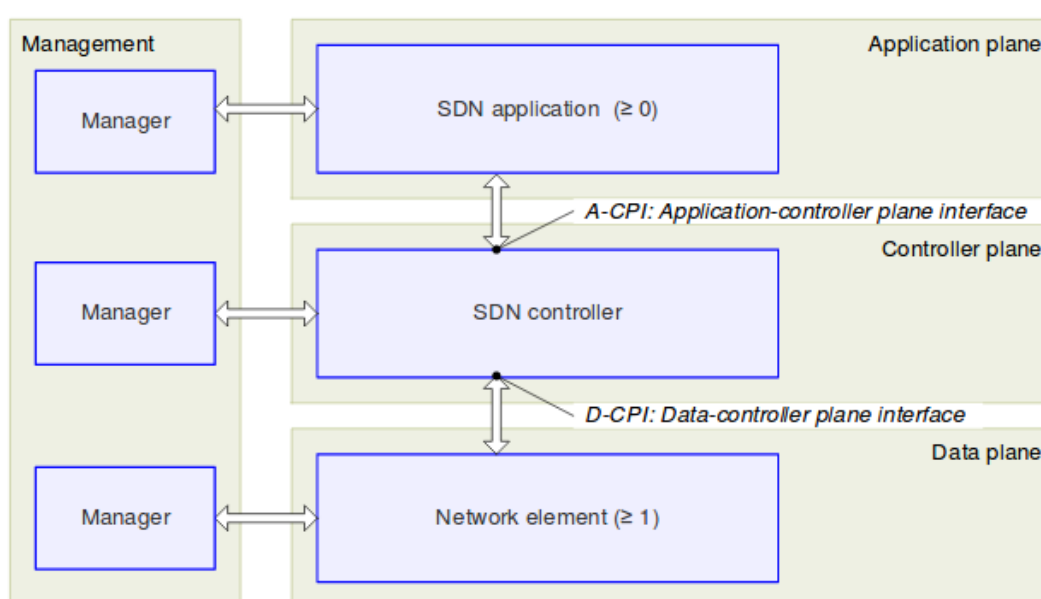


Image 2.7: SDN overview [27]

2.3 OPENFLOW PROTOCOL

SDN is commonly associated with the OpenFlow (OF) protocol since its emergence in 2011, which is the first standardized communication protocol defined between the control and the forwarding layers of the SDN architecture [11], [29]. Currently OF is the prominent SDN protocol for the communication between the Layer 2 networking devices (i.e. switches and routers, both physical and virtual) and the controller of the network [30] in an open and vendor-agnostic manner [31]. OF enables the network control plane to define cross-layer forwarding rules, which can be established and handled by OpenFlow-enabled devices. Based on the SDN architecture, together with the OF protocol, network devices are transformed to fully programmable forwarding elements.

In 2008, the OF protocol was proposed and studied in many studies. Aided by ONF's promotion, OF has become the standard for realizing the Southbound API. OF switches report statistics of flows to the controller with help from OF. Therefore, researchers and developers can easily access the statistics in applications through the Northbound API. These statistics are invaluable, since they reflect the traffic situation within the network [25].

OF allows experimenters, researchers, protocol developers and network administrators to exploit the true capabilities of a network in a quick, easily deployable and flexible manner. With the centralized network perspective that OF provides through its controller, an experimenter has an overarching view of the current status in the network. In addition, they have the ability to introduce, at run-time, new functionality without having to specifically modify any of the networking devices. Network administrators are able to make decisions about how data flows should be routed between network devices and switches along the optimized paths in networks.

OF's recent popularity is in part due to its open and vendor-agnostic nature. It provides powerful tools and enables the implementation of a diverse range of functionality and network behavior. With the characteristics of central management and flexibility property in SDN architecture, OF can help service providers to achieve better QoS performance by offering traffic differentiation [29], [30].

It is important to note that ONF is also working on open APIs between the SDN control and applications layers. What these will provide are the means for the applications to use network services and capabilities as needed, without knowing the network specifics, such as network topology. Applications on the application layer can thus issue requests, which are translated by the control layer to device specific configurations [22].

The controller collects information transmitted by the OF switch through the OF protocol and instructs the OF switch how to forward packets. To work in an OF environment, any device (i.e. switch/router) that wants to communicate with an SDN Controller must support the OF protocol [9].

2.3.1 The Flow Table

Each OF switch has its own flow table for administrators to define the paths of the packets. A flow table consists of flow entries or **rules**. Each rule contains the following fields:

- **Match fields:** Fields to match against OpenFlow packets. These fields consist of the ingress port and the packet headers and, optionally, some metadata specified by a previous flow table.

- **Priority:** Field to match the precedence of the flow entry. Higher values are higher priorities.
- **Counters:** Fields which are increased by one when a packet is matched.
- **Instructions:** Fields for the modification of the action set or pipeline processing.
- **Timeouts:** The maximum timespan or idle time before a flow is expired by the switch.
- **Cookie:** Opaque data value handled and selected by the controller. A cookie may be used by the controller to filter flow statistics, flow modification and deletion.

The match fields and priority taken together, uniquely identify each flow table entry in a flow table [32].

In Table 2.1, the match fields are presented. When an OF packet is received by an OF switch, it is buffered and these fields are matched against the corresponding fields of the packet. Each flow entry may contain one or more wildcarded fields. In this case, a wildcarded field matches against all the possible values of that field.

Table 2.1: Match fields of a flow table rule

Ingress port	MAC src	MAC dest	Ether type	VLAN id	VLAN priority	IP src	IP dest	IP proto	IP ToS bits	TCP/UDP src port	TCP/UDP dst port
--------------	---------	----------	------------	---------	---------------	--------	---------	----------	-------------	------------------	------------------

Each OF switch is permitted to contain multiple flow tables. The main purpose is to allow the administrators to decide whether to compare the packet to another flow table or to deliver the packet to the controller and let the controller handle it when the packet cannot be matched to any of the rules in one flow table [24].

Each flow entry contains a set of instructions that are executed when a packet matches the entry. Such instructions result in an **action** set, which performs changes to the incoming packet and/or pipeline processing. It should be noted that a switch must reject a flow entry, if it is unable to execute the instructions associated with this flow entry. The most important instruction types are:

- **Apply-Actions action(s):** This instruction may be used for the modification of the packet between two tables or for the execution of multiple actions of the same type. It applies the specific action(s) immediately to the packet, without changing the Action Set. Such actions are described as an action list.
- **Write-Actions action(s):** Merges the specified action(s) into the current action set.
- **Goto-Table [TABLE_ID]:** Indicates the next table in the processing pipeline. The next table-id must be greater than the current table-id.

The Apply-Actions instruction includes an action list. The actions of an action list are executed in the order specified by the list and are applied immediately to the packet. Each action is executed on the packet in sequence and that execution starts with the first action in the list. Some possible actions are:

- **Output:** According to this action, a packet is forwarded to a specified OpenFlow port. OpenFlow switches must support forwarding to physical ports, switch-defined logical ports and the required reserved ports.

- **Set-Queue:** It sets the queue id for an incoming packet. When the packet is forwarded to a port using the output action, the queue id specifies which queue, attached to this port, is used for scheduling and forwarding the packet. More specifically, the forwarding behavior is determined by the configuration of the queue and is used for the basic QoS support.
- **Drop:** This result can come from empty instruction sets or empty action buckets in the processing pipeline, or after the execution of a Clear-Actions instruction. In other words, there is no explicit action to represent drop, but packets whose action sets have no output actions should be dropped.
- **Group:** Process the packet through the specified group.
- **Push-Tag/Pop-Tag:** Switches may support the ability to push and pop tags from the packet. For instance, the ability to push/pop VLAN tags is suggested to be supported.
- **Set-Field:** The Set-Field actions modify the values of respective header fields in the packet. Such actions are identified by their field type.
- **Change-TTL:** Such actions result in the modification of the values of the IPv4 TTL, IPv6 Hop Limit or MPLS TTL in the packet.

The output action in the action set is executed last. An output action is ignored only in the case that both an output action and a group action are specified in an action set because the group action takes precedence. The packet is dropped unless an output or a group action (or both) was specified in an action set [32].

An example flow rule is the following:

```
cookie=0x2b0000000000000f6, duration=145.503s, table=0, n_packets=0,
n_bytes=0, priority=2, in_port=3, actions=output:2
```

where:

- **cookie** is a unique identifier of the flow
- **n_packets** (or **n_bytes**) is the number of packets (or number of bytes) matched by this flow
- **priority** shows the order in which the specific flow rule will be examined
- **in_port** is the match field of the flow. In this case, packets coming from port 3 are matched
- **actions** is the field containing the actions that will be performed. In this case, the packet will be forwarded to output 2.

2.3.2 The lookup process

The counters associated with this particular flow entry must be increased and the instruction set included in the selected flow entry must be applied. In case of multiple matching flow entries with the same highest priority, the chosen flow entry is undefined. This process is illustrated in Figure 2.1:

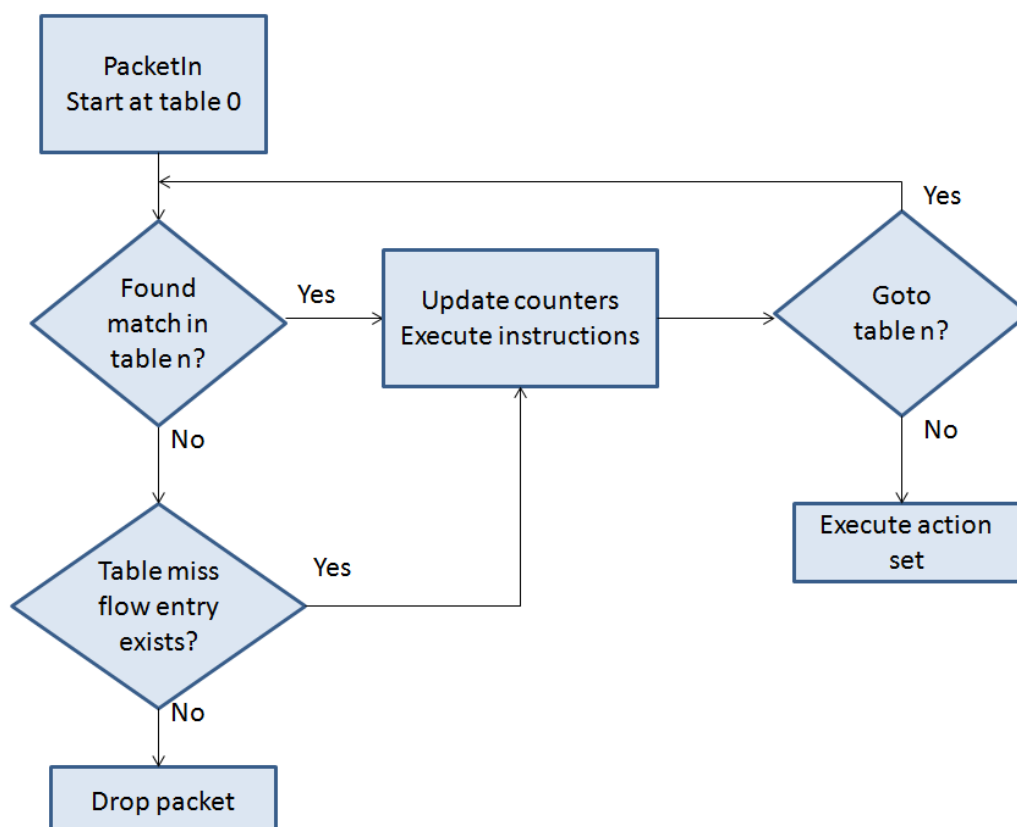


Figure 2.1: The lookup process in OF [32]

The lookup process starts in the first table and ends either with a match in one of the tables of the pipeline or with a miss (when no rule is found for that packet). A packet matches a flow table entry if the values in the packet match fields, used for the lookup, match those specified in the flow entry. Each packet is matched against the table and only the highest priority entry that matches the packet must be selected. In case of a successful match, the action(s) specified in the rule are executed. If there is no matching rule in the flow tables, the packet is either dropped or an OpenFlow message containing the packet header is sent to the controller for processing. The controller calculates the action the network element should take with regard to the packet and communicates it. Furthermore, the controller can specify a flow rule and send it to the network element(s). This way, all following packets of the flow are treated the same way by the network, and the controller does not need to be involved any longer. The controller can also introduce new flow rules or modify existing ones without being triggered by an incoming packet. For example, the controller may adhere to a pre-programmed schedule or implement a network policy. This is where the flexibility of SDN comes into play. Where traditional network devices would have to be reconfigured by an administrator, SDN enables the automatic and seamless implementation of changes in the forwarding behavior of the network. These changes can be triggered by external entities via the northbound API [6], [33].

Moreover, every flow table must support a table-miss flow entry to process table misses. This flow entry defines how to handle packets that are not matched against other flow entries in the flow table. As a result, such packets may be sent to the controller, be dropped or be directed to a subsequent table. If such a table-miss entry does not exist, by default, packets unmatched by flow entries are discarded [32].

2.4 OPENDAYLIGHT CONTROLLER

As already mentioned, SDN is an industry movement for building programmable networks that are flexible and responsive to organizations' and users' needs. OpenDaylight (ODL) is the largest open source SDN controller which is helping lead this transition. By uniting the industry around a common SDN platform, the ODL community is delivering interoperable, programmable networks to service providers, enterprises, universities and a variety of organizations around the globe [34].

Announced in April 2013 and hosted by the Linux Foundation, ODL is open to anyone, including end users and customers, and it provides a shared platform for those with SDN goals to work together to find new solutions. Under the Linux Foundation, ODL includes support for the OpenFlow protocol, but it is not limited to this and can also support other open SDN standards. ODL Controller exposes open northbound APIs, which are used by applications. These applications use the controller to collect information about the network, run algorithms to conduct analytics, and then use the ODL Controller to create new rules throughout the network.

The ODL Controller is implemented solely in software, and is kept within its own Java Virtual Machine (JVM). This means that it can be deployed on hardware and operating system platforms that support Java [35]. The first code from the ODL project was released in 2014, including an open controller, a virtual overlay network, protocol plugins and switch device enhancements [36].

The latest stable version of ODL, released in May 2016, is Boron. This version is the one chosen to be used in the scope of the current diploma thesis. The Boron Controller, similarly to the previous releases, consists of three key blocks:

- The controller platform
- Northbound applications and services
- Southbound plugins and protocols

Image 2.8 depicts the overall architecture setup and components of ODL Boron.

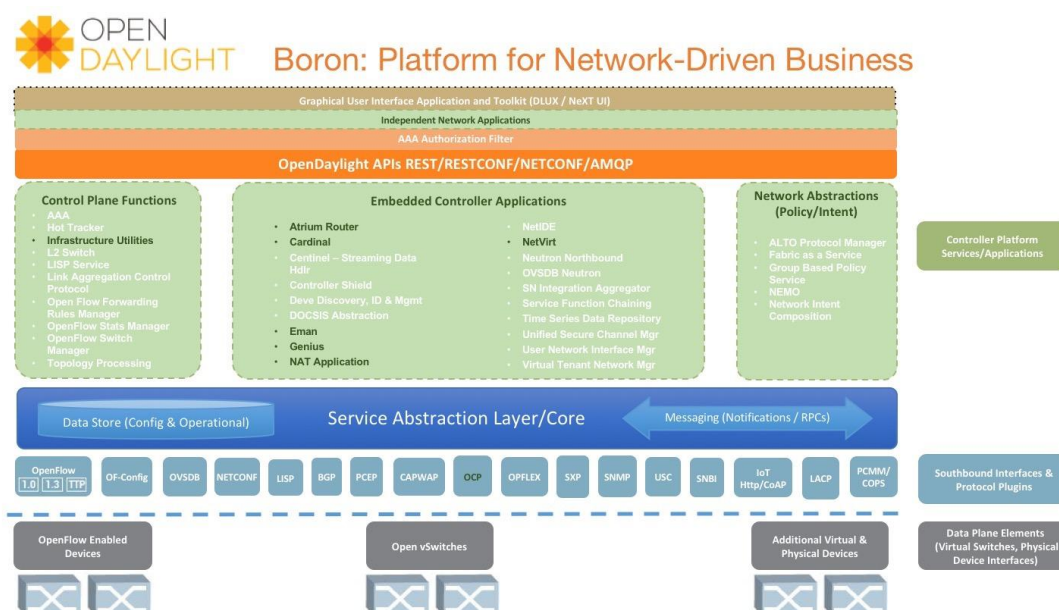


Image 2.8: The ODL Boron version architecture [37]

The controller platform is a modular layer and has a northbound and southbound interface. The northbound interface provides controller services and a set of common

REST APIs that applications can leverage to manage networking infrastructure configuration.

The southbound interface implements protocols to manage and control the underlying networking infrastructure. The southbound level has multiple plugins that either implement various networking protocols or directly communicate with hardware. OF and NETCONF are the best known and most widely used configuration and management protocols.

The controller platform communicates with the underlying network infrastructure using southbound plugins and provides basic networking services via a set of managers, such as Topology Manager and Switch Manager. Any custom application can use these network services.

The Base Network Service Functions are provided by the following platform managers and components and are the following:

- **Topology Manager:** Stores and handles information about the managed networking devices. When the controller starts, the Topology Manager creates the root node in the topology operational subtree. Then it listens for notifications and updates this subtree with topology details, including all discovered switches and their interconnections. Notifications from other components, such as the Switch Manager or Device Manager, may also provide relevant topology information.
- **Statistics Manager:** Implements statistics collection, sending statistics requests to all enabled nodes (managed switches) and storing responses in the statistics operational subtree. The Statistics Manager also exposes northbound APIs to return information on the following:
 - Node Connectors (switch ports)
 - Flows
 - Meters
 - Tables
 - Group statistics
- **Switch Manager:** Provides network nodes (switches) and node connectors (switch ports) details. As soon as the controller discovers network components, their parameters are saved to the Switch Manager data tree. Northbound APIs can be used to get information on the discovered nodes and port devices.
- **Forwarding Rules Manager:** Manages basic forwarding rules (such as OpenFlow rules), resolves their conflicts, and validates them. The Forwarding Rules Manager communicates with southbound plugins and loads OpenFlow rules into the managed switches.
- **Inventory Manager:** Queries and updates information about switches and ports managed by ODL, guaranteeing that the inventory database is accurate and up-to-date.
- **Host Tracker:** Stores information about the end hosts (data layer address, switch type, port type, network address), and provides APIs that retrieve end node information. Host Tracker may work in a static or dynamic way. In case of dynamic operation, the Host Tracker uses the Address Resolution Protocol (ARP) to track the status of the database. In static mode, the Host Tracker database is populated manually via northbound APIs.

The central concept of the ODL controller is the Service Abstraction Layer (SAL), which connects the protocol plugins and Service Network Function Modules. Because the original API-Driven SAL (AD-SAL) approach proved ineffective, ODL and all dependent projects are migrating to Model-Driven SAL (MD-SAL). The Model-Driven SAL provides a common approach to plugin development, enabling unification between both northbound and southbound APIs and the data structures used in various components of the controller. In MD-SAL, all status-related data are stored in the form of a document object model (DOM), known as a data stores.

The following two types of data stores are used in the ODL Controller:

- The **operational** data store, which controller modules use to store certain temporary runtime information
- The **configuration** data store, used to store the current status of the controller. An application or external end-user can post data, either through MD-SAL transaction or RESTCONF, to this data store. The individual objects are stored in a parent-child hierarchy and accessible through YANG instance identifiers. Once an instance identifier is created, a **read** or **write** transaction can be performed to that location in the data store [38].

MD-SAL uses YANG as the modeling language for describing all network services. After one defines the necessary YANG models, a compiler outputs appropriate Java interfaces, and the next step is to implement those auto-generated Java interfaces [37].

MD-SAL currently provides infrastructure services for:

- Data Store
- Remote Procedure Call (RPC) / Service routing
- Notification subscription and publish services [39].

2.5 NETCONF, RESTCONF AND YANG

The model-driven approach is being increasingly used in the networking domain to describe the functionality of network devices, services, policies and network APIs. The protocols of choice are Network Configuration Protocol (NETCONF) and Representational State Transfer Configuration (RESTCONF) Protocol; the modeling language of choice is Yet Another Next Generation (YANG) Language [39]. All three are explained below in detail.

Since the beginning of the SDN discussion a few years back, proponents of OF have been behind the movement to control all devices in the network. This has unquestionably been the case within the confines of the data center. OF appears to solve data center issues well, even its earliest versions. However, there are many cases where, in order to provide an end-to-end Wide-area Network (WAN) service or provide inter-data center connectivity, the use of OF is up to now insufficient. Even in SDN, there still is persistent state on network devices and OF doesn't automatically configure itself.

Extensions contained in newer OF versions address many early limitations, such as transactional actions and dealing with multiple entities that can monitor and/or control the same device. In all fairness, OF has made astonishing progress since its inception in 2009, but the capabilities needed for WAN device configuration and control are recent additions.

On the other hand, a protocol which has existed for some time is now appearing more often, especially in discussions pertaining to network automation. The protocol is called

NETCONF and was established by the Internet Engineering Task Force (IETF) in 2006. NETCONF was introduced to help automate configuration management and monitoring for a large population of WAN devices (routers, switches and transport gear). Vendors such as Cisco, Juniper, Ericsson and others have been early proponents of NETCONF, thereby establishing momentum for the technology [40].

NETCONF provides the fundamental programming features for comfortable and robust automation of network services [41]. It defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated [42]. It also defines configuration and operational conceptual data stores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these data stores. It uses an XML- based data encoding for the configuration and operational data, as well as for its protocol messages [39].

Mechanisms are provided to install, update, and delete the configuration of network devices, such as routers, switches, and firewalls [42]. The NETCONF protocol operations are realized as RPCs [43].

In order to develop applications in the SDN controller, the following are required:

- A Domain-Specific Language (DSL) to describe internal and external system behavior.
- Modeling tools for the controller aligned with the modeling tools for the devices.
- Code generation from models:
 - Enforce standard API contracts
 - Generate boilerplate code performing repetitive and error-prone tasks
 - Produce functionality equivalent APIs for different language bindings
 - Model-to-model adaptations for services and devices
 - Consumption of aligned device models

The above requirements are satisfied with YANG [44].

YANG is a data modeling language for the definition of data sent over the NETCONF protocol. YANG can be used to model both configuration data and state data of network elements in a tree format, as well as define the format of event notifications emitted by network elements and allow data modelers to define the signature of remote procedure calls that can be invoked on network elements via the NETCONF protocol [45]. These characteristics make it suitable for use as an Interface Description Language (IDL) in a model-driven system [39]. It is a human readable and easy to learn representation, with reusable types and groupings and extensibility through augmentation mechanisms. It is a full, formal contract language with rich syntax and semantics to build applications on [41]. The language, being protocol independent, can then be converted into any encoding format, e.g. XML or JSON, that the network configuration protocol supports [45]. In the SDN controller, YANG is being used as a general purpose modeling language [46].

Finally, RESTCONF is a Representational State Transfer (REST)-like protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the data stores defined in NETCONF. Configuration data and state data are exposed as resources that can be retrieved with a GET request. Data is encoded in either XML or JSON [39].

2.6 HISTORY AND STANDARDIZATION OF SDN

Albeit a fairly recent concept, SDN leverages on networking ideas with a longer history. In particular, it builds on work made on programmable networks, such as active networks, programmable Asynchronous Transfer Mode (ATM) networks as well as on proposals for control and data plane separation.

In order to present a historical perspective, Table 2.2 summarizes different instances of SDN-related work prior to SDN, splitting it into five categories. Along with the categories defined, the second and third columns of the table mention past initiatives (pre-SDN, i.e., before the OF-based initiatives that sprung into the SDN concept), and recent developments that led to the definition of SDN. Data plane programmability has a long history. Active networks represent one of the early attempts on building new network architectures based on this concept. The main idea behind active networks is for each node to have the capability to perform computations on, or modify the content of, packets. To this end, active networks propose two distinct approaches: programmable switches and capsules. The former does not imply changes in the existing packet or cell format. It assumes that switching devices support the downloading of programs with specific instructions on how to process packets. The second approach, on the other hand, suggests that packets should be replaced by tiny programs, which are encapsulated in transmission frames and executed at each node along their path.

Forwarding and Control Element Separation (ForCES) and OF represent recent approaches for designing and deploying programmable data plane devices. In a manner different from active networks, these new proposals rely essentially on modifying forwarding devices to support flow tables, which can be dynamically configured by remote entities through simple operations such as adding, removing or updating flow rules, i.e., entries on the flow tables.

The earliest initiatives on separating data and control signaling date back to the decades of 1980 and 1990. Initiatives such as ForCES and Path Computation Element (PCE) proposed the separation of the control and data planes for improved management in Ethernet and Multiprotocol Label Switching (MPLS) networks, respectively.

More recently, initiatives such as OF and NOX proposed the decoupling of the control and data planes for Ethernet networks. Interestingly, these recent solutions do not require significant modifications on the forwarding devices, making them attractive not only for the networking research community, but even more to the networking industry.

The concept of a network operating system was reborn with the introduction of OF-based network operating systems, such as NOX, Onix and ONOS. Indeed, network operating systems have been in existence for decades. One of the most widely known and deployed is the Cisco IOS, which was originally conceived back in the early years of the decade of 1990. Despite being more specialized network operating systems, targeting network devices such as high-performance core routers, these NOSs abstract the underlying hardware to the network operator, making it easier to control the network infrastructure as well as simplifying the development and deployment of new protocols and management applications.

The open signaling movement worked towards separating the control and data signaling, by proposing open and programmable interfaces. Curiously, a rather similar movement can be observed with the recent advent of OF and SDN, with the lead of the ONF. This type of movement is crucial to promote open technologies into the market, hopefully leading equipment manufacturers to support open standards and thus fostering interoperability, competition, and innovation [6].

Table 2.2: Summarized overview of the history of programmable networks [6]

Standardization Organization	Working Group	Focus	Outcomes
ONF	Architecture & Framework	SDN architecture, defining architectural components and interfaces	SDN Architecture
	Northbound Interfaces	Definition of standard NBIs for SDN controllers	
	Testing and Interoperability	Specification of OpenFlow conformance test suites	Conformance tests
	Extensibility	Development of extensions to OpenFlow protocol, producing specifications of the OpenFlow switch (OF-WIRE) protocol	OF-WIRE 1.4.0
	Configuration & Management	OAM (operation, administration, and management) capabilities for OF protocol, producing specifications of the OF Configuration and Management (OF-CONFIG) protocol	OF-CONFIG 1.2, OpenFlow Notifications Framework
	Forwarding Abstractions	Development of hardware abstractions and simplification of behavioral descriptions mapping	OpenFlow Table Type Patterns
	Optical Transport	Specification of SDN and control capabilities for optical transport networks by means of OpenFlow	Use cases Requirements
	Wireless & Mobile	Specification of SDN and control capabilities for wireless and mobile networks by means of OpenFlow	
	Migration	Methods to migrate from conventional networks to SDN-based networks based on OpenFlow	Use cases
	Market Education	Dissemination of ONF initiatives in SDN and OpenFlow by releasing White Papers and Solution Briefs	SDN White Paper
IETF	Application-Layer Traffic Optimization (ALTO)	Provides applications with network state information	Architectures for the coexistence of

			SDN and ALTO
	Forwarding and Control Element Separation (ForCES)	Protocol specifications for the communication between control and forwarding elements	Protocol specification
	Interface to the Routing System (I2RS)	Real-time or event driven interaction with the routing system in an IP routed network	Architecture
	Network Configuration (NETCONF)	Protocol specification for transferring configuration data to and from a device	NETCONF protocol
	Network Virtualization Overlays (NVO3)	Overlay networks for supporting multi-tenancy in the context of data center communications (i.e., VM communication)	Control plane requirements
	Path Computation Element (PCE)	Path computation for traffic engineering and path selection based on constrains	ABNO framework, Cross stratum path computation
	Source Packet Routing in Networking (SPRING)	Specification of a forwarding path at the source of traffic	OpenFlow interworking SDN controlled use cases
	Abstraction and Control of Transport Networks (ACTN) BoF	Facilitate a centralized virtual network operation	Virtual network framework
IRTF	Software-Defined Networking Research Group (SDNRG)	Prospection of SDN for the evolution of Internet	SDN operator perspective, SDN Architecture Service / Transport separation
ITU-T	SG 11	Signaling requirements using SDN technologies in broadband access networks	Q. Supplement, SDN Q.SBAN
	SG 13	Functional requirements and architecture for SDN and networks of the future	Recommendation Y.3300
	SG 15	Specification of a transport network control plane architecture to support SDN control of transport networks	
	SG 17	Architectural aspects of security in SDN and security services	

		using SDN	
BBF	BBF Service Innovation and Market Requirements	Requirements and impacts of deploying SDN in broadband networks	SD – 313
MEF	The Third Network	Service orchestration in network as a Service and NFV environments	
IEEE	802	Applicability of SDN to IEEE 802 infrastructure	
OIF	Carrier WG	Transport SDN networks	Requirements for SDN enabled transport networks
ODCA	SDN/Infrastructure	Requirements for SDN in cloud environments	Usage model
ETSI	NFV ISG	Orchestration of network functions, including the combined control of computing, storage and networking resources	NFV Architecture
ATIS	SDN Focus Group	Operational aspects of SDN and NFV	Operation of SDN

2.7 USE CASES OF SDN

As already mentioned, SDN has been paid a lot of attention to by researchers and scientists over the past few years. It proves to be an adaptable, comparatively cost-efficient and dynamic solution. The sophistication of SDN allows it to cater to the high bandwidth needs of applications. Although it has been getting a lot of hype, any real world use cases for it can rarely be figured out, even though there are plenty of them [47]. SDN theory has dominated conversation in the networking industry, but also SDN uses cases are beginning to emerge, showing how the technology can result in cost efficiency and network flexibility in both enterprise and service provider environments [48]. Below a few selected SDN use cases which depict how and why it can provide flexibility and deal with some key network problems are presented [47].

- **Video and collaboration applications:** Video and collaboration applications have become critical for the success of an organization. Most of these applications typically are more efficient when they use multicast technology. IP multicast technology is mature and available, but it is still difficult to deploy and troubleshoot. It is not as widely deployed as IP unicast technology, and it has forced many organizations to deploy video applications using other mechanisms based on IP unicast forwarding. SDN is an ideal solution for these types of applications, as the SDN controller knows the topology, sources and listeners, and can build an efficient multicast topology and program the network on an on-demand basis [49].

The use of SDN in video applications allows better control over the network and gives providers the ability to ensure that the QoS levels are maintained, without having to expand capacity, since they are able to both provision and de-provision the network space based solely on need [47], [48].

- **Application Aware Routing:** With personal applications such as Facebook and YouTube competing with corporate applications, networks need to prioritize and forward the traffic based on an application. SDN can provide a simple and consistent way of identifying applications, and program the network to prioritize and forward the traffic appropriately. In fact, any application can communicate with the SDN controller to provide application specific-needs, and request the SDN controller to program the flow appropriately [49].
- **Data Center Optimization:** Using SDN, networks can be optimized in order to improve application performance by detecting and taking into account affinities and orchestrating workloads with networking configuration (mice/elephant flows) [50]. SDN allows scaling of bandwidth between servers, without significant hardware expenses [48].
- **Converged storage:** Converged storage is an architecture that consists of an amalgam of computing resources and storage in a single unit. They can be used for platform development for storage centric, server centric or even hybrid (storage-server) workloads [47]. SDN can be used with the goal of virtualizing the network and making it agile enough to keep up server and storage virtualization. Many famous data services providing companies are beginning to use SDN while creating programmable fabrics across storage and data center technologies. With this approach, companies can offer software as a Service and data services [48].
- **Routing and Service Convergence:** The value of SDN for service providers with dense and highly distributed networks lies specifically in the ability to provide them more options on locating resources thereby conferring a competitive advantage when delivering certain kinds of services. SDN may reduce the policy complexity by enabling scalability in inter-domain and providing validation for the claims of seamless evolution [51].
- **Cloud-based Networks:** The introduction and deployment of cloud-based services have emerged as an important solution that offers enterprises a cost-effective business model. However, many network functions have extreme characteristics and performance requirements, which have created new challenges such as servers and network virtualization, mobile clouds, and security. These need to be addressed with intelligent network virtualization, high-speed packet processing, and load-balancing. SDN can be seen as a new and complementary technology to virtualization, which is poised to tackle the challenges of network-enabled cloud and web-scale deployments [51].
- **Wireless and Mobility Settings:** Software Defined Wireless Networking (SDWN) is an SDN technology for wireless that provides radio resource and mobility management, routing, and multi homing. SDWN can provide a programmable wireless data plane to allow modular and declarative programming interfaces across the wireless stack. This allows programming the enterprise-specific requirements (e.g., an airport, a restaurant, public library) in a wireless access point (WAP) [51].

These were only some of the many possible applications of SDN, which is extended beyond the aforementioned cases. It's inevitable that even more will appear once the technology receives more recognition and public approval [47], [52].

2.8 ADVANTAGES OF SDN

SDN has captured the industry's attention because it brings significant benefits to the entire network and cloud ecosystems [4]. It has potential use cases and benefits for

almost every part of the network in both Layer 2 and Layer 3 segments [53] and offers considerable technological and financial advantages [7].

Adopting an SDN methodology has a myriad of benefits including flexibility, scalability, redundancy, and performance. In a traditional network, there might be certain limited hardware and software pieces. When a network requires additional resources, there will be considerable cost in buying new hardware and licensing. With SDN, the network is abstracted onto software, leaving more choice and flexibility in purchasing hardware. In addition, a growing network can be more easily supported by SDN because a network administrator or engineer can simply add more virtual switches or routers rather than purchase costly equipment and licensing. A software-defined network is also portable, which allows the flexibility in choosing and moving to cloud storage, public or private. Abstracting the network onto a cloud could present many benefits as well: less hardware to manage onsite, lower energy bills, and greater uptime [53].

Moreover, SDN benefits the network operators and owners across various domains of use. For example, the data center operators use SDN for network virtualization to support multi-tenancy across computing, storage, and networking in a unified way and to integrate soft-appliances to reduce capital and operational expenses and be more agile in meeting customer requirements. Service providers use SDN to create highly cost-efficient wide-area networks with virtualization to interconnect their geographically distributed data centers for a cloud infrastructure. SDN within and across data centers enables network virtualization, customization, and optimization for their customers that hasn't been possible before. Service providers and their vendors see the benefits of SDN in traffic engineering, service chaining, and other use cases for simplifying management and control of the edge and core networks to help reduce operational expenses and future capital expenses. Cellular operators view SDN as a way to build CAPEX- and OPEX-efficient backhaul and packet core networks that can be more agile in supporting mobile services.

SDN also benefits the large group of network equipment and third party vendors. It allows them to innovate faster by creating software-based solutions to meet their customers' requirements in various domains of use. Equipment vendors have an opportunity to sell a new class of products and solutions and network operators and owners can grow their infrastructure rapidly and roll out new innovative capabilities and services, giving vendors more opportunities for revenue growth.

One of SDN's most important advantages is the potential for automation. By using programmatic controls to automate functions within a network, SDN can significantly increase speed and efficiency while reducing the risk of human error. The business can focus on innovation, rather than operational tasks. Moreover, reducing the time needed to manage the network and deploy new resources or applications can also greatly increase an organization's agility and the speed with which new services can be deployed [7].

To sum up the current chapter, the most important benefits of SDN, as stated by the ONF, are:

- **Direct programmability:** The control plane is directly programmable because it is decoupled from the data plane.
- **Agility:** Abstracting the control plane from the data plane lets administrators dynamically adjust traffic flows to meet changing needs.
- **Central (logically) management:** The SDN controller maintains a global view of the infrastructure network which appears to applications and policy engines as a single, logical switch.

- **Programmable configuration:** SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves, because the programs do not depend on proprietary software.
- **Open standards implementation and vendor neutrality:** When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols [26].

2.9 CHALLENGES OF SDN

Although SDN is a favorable solution for IT and cloud providers and enterprises, SDN faces some challenges which hinder its performance and implementation. The list of SDN challenges contains scalability, performance, security, interoperability and reliability issues. Enterprises and networking organizations must overcome several obstacles to fully realize SDN's benefits [54], [55].

Following are presented some common challenges in software defined networks, created by the paradigm shift of software-defined services from traditional hardware-based networking:

- **SDN Reliability/Fault Tolerance:** In a traditional network when one network or many network devices fail, network data flow is routed through another or nearby nodes or devices to continue data flow continuity. Therefore, the existing networks survive failures or bugs for any of the devices. However in the centralized controller architecture of SDN, the controller is a single point of control and therefore a single point of failure, as a single controller is in charge of all the networks. Thus, in case of a bug or a failure in the central controller, the whole network collapses since there is no alternate controller.

To address this issue the cloud organization needs to focus on how to efficiently utilize main controller functions that can increase network reliability. The SDN controller should have the ability to support multiple-path solutions or fast traffic rerouting to active links if there is a path/link failure. If the main controller fails, the newer architectures support an alternate controller which can handle traffic flow [56].

- **Scalability:** In SDN, as already mentioned, the data and control planes are decoupled. The decoupling process has its own drawbacks, such as the fact that the SDN controller becomes the bottle neck in a situation where the network scales the number of switches and number of nodes up [54]. In particular, large networks with volumes of networking requests can overwhelm controllers. As networks grow, the bottleneck tightens and network performance degrades.

Scalability may be improved with a decentralized control architecture or similar solution, such as split or fully distributed control planes. But such solutions can introduce new obstacles such as convergence and countless control instances to configure and manage [55].

- **Performance:** The network's performance is another important area to look into. Performance is the greatest issue for all networks. Regardless of how robust, secure, scalable, or interoperable a network is, it's unusable if it lacks performance.

The separate control and data plan architecture can introduce latency into SDN. In large networks this can build to an unacceptable level of delay, degrading network performance. Related, controller response time and throughput can contribute to poor performance, with the combined effect causing scalability issues.

The solution for many performance issues in large and growing networks is to push more intelligence to the data plane or move to a distributed control plane architecture of some type. While this can improve SDN performance, it is not very close to the intent of SDN, as it replicates traditional networks built on fully distributed intelligent devices. A balance has to be sought where virtualization is maintained without degrading network performance or introducing potential single points of failure [54], [55].

- **Security:** Because the control plane plays such a central function in an SDN architecture, security strategies must focus on protecting the controller and authenticating an application's access to the control plane. New services can introduce security threats as programmers and network administrators may unwittingly introduce at-risk code and extend the threat network wide through a centralized or partially distributed controller. Related, SDN's virtual nature can result in the creation of countless network segments, each with its own risk and security requirements [55].

Security needs to be everywhere within a software-defined network. SDN security needs to be built into the architecture, as well as delivered as a service to protect the availability, integrity, and privacy of all connected resources and information [57].

- **Rapid on-demand growth accommodation:** Unlike legacy infrastructure in the SDN world we can have multiple overlay topologies running on top of the physical network. Whenever a new service starts, it deploys the necessary virtual infrastructure, and thus the number of monitored elements can grow rapidly with increased demand – outstripping traditional capacity management.

The solution is to deploy performance monitoring within both physical and virtual appliances. When extra performance management capacity is needed, spinning up additional virtual appliances on demand enables performance monitoring to flex with the demands of an SDN environment and still provide answers in seconds [58].

- **Interoperability:** For new networks, implementing SDN is fairly straightforward - all network devices are SDN-ready. Transitioning a legacy network to SDN is another story as the legacy network is likely supporting active business and networking systems. Enterprises and most networking environments have to transition to SDN, requiring a period of interoperability with a hybrid legacy-SDN infrastructure. They need guarantees that services running on existing networks will not be disrupted.

SDN and legacy network nodes can operate together, with help from an appropriate protocol that supports SDN communications while providing backward compatibility with existing IP and MPLS control plane technologies - reducing the cost, risk, and disruption of services while transitioning to SDN [55].

3. QUALITY OF EXPERIENCE (QoE)

3.1 INTRODUCTION TO QoE

As already explained in Section 1.3, network applications such as online video streaming have seen a huge growth in popularity during recent years and at the same time, HD video traffic has already surpassed that of Standard Definition (SD). It is expected that this trend will continue further in the forthcoming years. Undoubtedly, high quality video streaming has already become an essential part of many consumers' lives and with the introduction of UHD content, providers will continue to push user expectations in the availability of higher video quality and bitrates. Moreover, the rapid evolution of mobile networks is driven by the growth of packet data applications such as mobile video applications and mobile streaming services. Also, heterogeneous network structures, severe channel impairments, and complex traffic patterns make mobile scenarios much more unpredictable than their wired counterparts [59], [28], [60].

Therefore, the importance of taking care of user satisfaction with service provisioning has been realized [60]. One can easily understand that the requirements for today's network applications are diverse and network and content providers are thus immensely interested in ensuring a high degree of satisfaction for their end-users. Understanding and measuring quality of communication services and underlying networks from an end-user perspective has attracted increased attention over the course of the last decade [61].

Networks try to support the requirements based on Quality of Service (QoS) parameters, such as throughput, latency and jitter. However, the performance of a specific application cannot be determined by simply relying on QoS metrics. A growing awareness of the scientific community that technology-centric QoS concept is not powerful enough to cover every relevant performance aspect of a given application or service has been witnessed [61]. Network level metrics traditionally used by network administrators are not adequate to indicate how satisfied a user is with his video streaming experience. In addition, research shows that there is not always a direct or deterministic correlation of the impact of the network-level metrics to the users' satisfaction. Thus, the evaluation of network applications should be based on user-centric metrics that provide a better indication of the satisfaction of the end-users and define the Quality of Experience (QoE) [59], [28].

The notion of QoE appeared around the beginning of the current century, mainly promoted by industry [60]. QoE is a measure of the delight or annoyance of a customer's experience towards a specific service, or in other words determines how well that network is satisfying the end user's requirements. 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. It focuses on the entire service experience; it is a holistic concept with its roots in telecommunications. QoE is an emerging multidisciplinary field based on social psychology, cognitive science, economics, and engineering science, focused on understanding overall human quality requirements. In short, QoE provides an assessment of human expectations, feelings, perceptions, cognition and satisfaction with respect to a particular product, service or application [61].

QoE has historically emerged from QoS, which attempts to objectively measure service parameters. QoS measurement is most of the time not related to a customer, but to the media or network itself. On the contrary, QoE takes into consideration the end-to-end connection and applications that are currently running over that network connection and how multimedia elements are satisfying or meeting the end user's requirements. It is a

purely subjective measure from the user's perspective of the overall quality of the service provided, by capturing people's aesthetic and hedonic needs. The relationship between a performance-based QoS parameter has a resulting effect on the end user's QoE, since a high network performance is required to meet QoE objectives [61], [62].

The QoE of a service is considered as being good if the user is content with the service as a whole. The degree of the latter is reflected in KPIs, addressing:

- Reliability aspects such as service availability, service accessibility, service access time, and continuity of service
- Comfort aspects such as session quality, ease of use, and level of support [60].

The IT industry applies the QoE model to businesses and services. QoE aims at taking into consideration every factor that contributes to a user's perceived quality of a system or service. This includes system, human and contextual factors. The following so-called "influence factors" have been identified and classified as described below and are depicted in Image 3.1:

- **Human Influence Factors**, such as:
 - Low-level processing (visual and auditory acuity, gender, age, mood, etc.)
 - Higher-level processing (cognitive processes, socio-cultural and economic background, expectations, needs and goals, other personality traits, etc.)
- **System Influence Factors**, which can be:
 - Content-related
 - Media-related (encoding, resolution, sample rate, etc.)
 - Network-related (bandwidth, delay, jitter, etc.)
 - Device-related (screen resolution, display size, etc.)
- **Context Influence Factors**, which can be related to:
 - Physical context (location and space)
 - Temporal context (time of day, frequency of use, etc.)
 - Social context (inter-personal relations during experience)
 - Economic context
 - Task context (multitasking, interruptions, task type)
 - Technical and information context (relationship between systems)

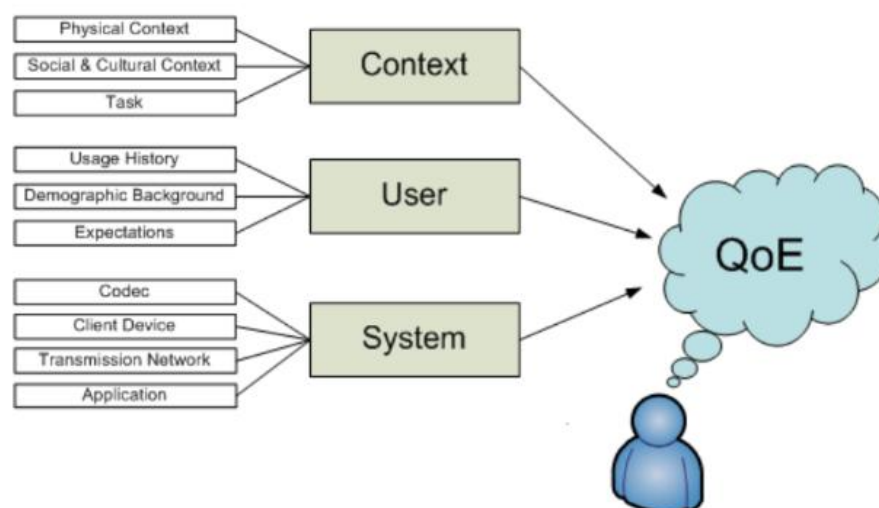


Image 3.1: QoE influence factors belonging to context, human user and the technical system [61]

Studies in the field of QoE have typically focused on system factors, primarily due to its origin in the QoS and network engineering domains. Through the use of dedicated test laboratories, the context is often sought to be kept constant. However, studies investigating context and human factors have become more popular. Research has shown that human factors account for observed variations in multimedia quality ratings, including socio-cultural and economic background as well as user expectations [61].

Key environmental factors impact QoE assessment. These include:

- Hardware, such as wired or cordless devices
- Application criticality, for example, texting versus audio/video
- Working environment, for example, fixed or mobile [63].

Therefore, a major challenge for future networks is to dynamically adapt to QoE demands of the applications in the network. This is especially true for networks with limited resources, like today's access networks [28].

3.2 QoE MANAGEMENT

As an important measure of the end-to-end performance at the service level from the user's perspective, QoE is an important metric for the design of systems and engineering processes. So, when designing systems, the expected output, i.e. the expected QoE, is often taken into account also as a system output metric and optimization goal [64]. The central question for QoE research and engineering is how to operationalize the concept in terms of performing reliable, valid, and objective measurements. This challenge is framed by the overarching question of quantifying and measuring quality. Since inclusion of the human end-user's perspective is the defining aspect of QoE, conducting measurements merely on a technical level (e.g. by just assessing conventional end-to-end QoS integrity parameters) is not sufficient. In particular, QoE also accounts for user requirements, expectations and contextual factors like task and location.

Thus, quality assessment schemes are needed that act as translator between a set of technical (QoS) and non-technical (subjective and contextual) key influence factors and user perception, or ultimately, user experience. These can be categorized into subjective and objective quality assessment methods, depending on whether human subjects are involved in the assessment process or not.

Subjective Quality Assessment Methods are based on gathering information from human assessors who are exposed to different test conditions or stimuli during the process. In general, a panel of assessors is subjected to various quality levels, something which leads to some form of explicit or implicit response. In most cases, quantitative methods derived from neighboring disciplines such as psychophysics and psychometrics are used to obtain information regarding assessors' judgment in the form of ratings that describe their perception of the respective quality experienced. In addition, qualitative methods such as focus groups or interviews are used, particularly in order to find out which influence factors or features contribute to QoE and how. Subjective tests are typically conducted in a controlled laboratory setting and require careful planning in terms of which variables and influence factors need to be controlled, measured and monitored [61].

The typical result of a subjective test campaign is the individual assessor's ratings which are typically aggregated into so-called mean opinion scores (MOS). The MOS expresses the average quality judgment of a panel given a certain test condition regarding the overall quality experienced or along a certain quality dimension (e.g. picture quality). It is typically based on an ordinal five-point scale:

- Bad
- Poor
- Fair
- Good
- Excellent

MOS is a widely used measure for assessing the quality of media signals; it is a limited form of QoE measurement, relating to a specific media type, in a controlled environment and without explicitly taking into account user expectations. The MOS as an indicator of experienced quality has been used for audio and speech communication, as well as for the assessment of quality of Internet video, television and other multimedia signals, and web browsing. Due to inherent limitations in measuring QoE in a single scalar value, the usefulness of the MOS is often debated [64].

Objective Quality Assessment Methods are another approach being investigated with the purpose to automatically predict QoE at high accuracy on behalf of algorithmic processing of input parameters.

Objective quality assessment approaches can be categorized on behalf of the following criteria:

- **Targeted service:** Service type, e.g. IPTV, Voice-over-IP (VoIP) telephony, video conferencing, mobile TV, web browsing.
- **Model type:** Utilization of a reference signal, i.e. Full Reference (FR), Reduced Reference (RR), No Reference (NR).
- **Application:** Codec testing, network planning, verification of QoS classes, monitoring, etc.
- **Model input:** Parametric description of the processing path (i.e. protocol information or planning values), additional payload information from bitstream, reconstructed signal, combinations of parameters and signal, etc.
- **Model output:** Overall quality or specific quality aspects in terms of MOS or another index.

- **Modeling approach:** Psychophysical (i.e. explicit modeling of the human perceptual system) vs. empirical approaches (based on extracting characteristic system features by conducting experiments).

However, objective metrics are only useful if their measurements closely correlate with subjective quality. Therefore, an integral part of the design process is the derivation of quality models that map quantifiable influence factors to predicted MOS values. To this end, the data obtained from subjective quality experiments are required to find model functions that provide an optimum fit with human quality perception [61].

Briefly comparing the two afore mentioned approaches, subjective quality evaluation requires a lot of human resources, establishing it as a time-consuming process. Objective evaluation methods, on the other hand, can provide such results faster, but require large amount of machine resources and sophisticated apparatus configurations. Despite obvious speed and economy advantages, the caveat with objective metrics is that they are only an approximation of a limited number of aspects of human quality perception. Therefore, they can provide inaccurate or inconclusive results, particularly when applied to new conditions they were not initially trained or designed for. Objective metrics therefore need to be developed carefully, their application scope needs to be clearly defined and continuously validated against data from subjective experiments. For these reasons, much further research is required until QoE can be, if at all, accurately measured using objective metrics only [61]. Thus, objective evaluation methods are based and make use of multiple metrics [64].

The QoE monitoring and measurement process encompasses the acquisition of data related to the network environment and conditions, terminal capabilities, user, context, and application/service specific information and its quantification. The parameters can be gathered via probes at different points in the communication system, at different moments, as well as by various methods. A diversity of QoE monitoring and measurement points, moments, and methods together with the selection of the key QoE influence factors (IFs) for a given service additionally increases the complexity of this process.

In order to be able to manage and optimize QoE, knowledge regarding the root cause of unsatisfactory QoE levels or QoE degradations is necessary. A layered approach relates network-level KPIs with user-level application specific Key Quality Indicators (KQIs), for example, service availability, usability, reliability, etc., which then provide input for a QoE estimation model. Additional input to a QoE estimation model may then be provided by user-, context-, and device-related IFs. Knowledge regarding this mapping between KPIs and KQIs will provide valuable input regarding the analysis of the root causes of QoE degradation. Hence, monitoring probes inserted at different points along the service delivery chain to collect data regarding relevant KPIs are necessary.

While monitoring at the client side provides the best insight into the service quality that users actually perceive, a challenge lies in providing QoE information feedback to the network, service/application, content, or cloud provider to adapt, control, and optimize the QoE. This client side monitoring point poses the issues of users' privacy, trust, and integrity, since users may cheat in order to receive better performance. Consequently, collecting data from within the network without conducting client side monitoring (in an either objective or subjective manner), and vice versa, will not generally provide sufficient insight into QoE. Hence, accurate monitoring of QoE needs to employ both monitoring from within the network and at the client side [65].

The QoE monitoring and measurement process is complex due to the diversity of factors affecting QoE, data acquisition points, and timings, as well as methods of

collecting data, and the lack of consensus regarding these issues. The main challenge in this process is to determine what, where, when and how to collect data.

Firstly, one needs to determine which data to acquire, which is specified by the QoE metrics selection which depends on the service type and context. The decision regarding data that should be acquired considering the wide spectrum of QoE IFs is challenging, but it is the prerequisite for any QoE monitoring and measurement approach. Secondly, choosing a location where to collect data is another critical issue in the QoE assessment process, that is, determine the location of monitoring probes. As previously mentioned, data can be collected within the network, at the client side, or both (depending also on whether measurements are conducted for QoE modeling purposes or for QoE control purposes). The QoE monitoring and measurement within the network may include data collection at different points such as the base stations within the various access networks, the gateways or routers within the core network, or the servers in the service/application, content, or cloud domains. Additionally, the acquired parameters may be derived from application level, network level, or a combination thereof. Each acquisition location addresses the specific challenges discussed previously. Furthermore, if performing in-service QoE management (e.g., QoE-driven dynamic (re)allocation of network resources), collected data generally needs to be communicated to an entity performing QoE optimization decisions. Hence, the passing of data to a control entity needs to be addressed. Thirdly, one should determine when to collect data (before / after the service is developed or after the service is delivered). Additionally, how often data should be monitored and measured needs to be considered. Finally, how to perform the data acquisition is determined by the where and when clauses. The QoE monitoring process implies computational operations, hence computational complexity and battery life of mobile devices need to be considered [65].

It may be concluded that different actors involved in the service provisioning chain will monitor and measure QoE in different ways, focusing on those parameters over which a given actor has control (e.g. a network provider will monitor how QoS-related performance parameters will impact QoE, a device manufacturer will monitor device-related performance issues, while application developers will be interested in how the service design or usability will affect QoE).

Having chosen the proper QoE metrics, monitoring and measurements approach, it is important to provide mechanisms utilizing this information for improving service performance, network planning, optimization of network resources, specification of SLAs among operators, and so forth.

Following QoE modeling, monitoring, and measurements, the ultimate goal of QoE management is to control QoE via QoE optimization and control mechanisms. Such mechanisms yield optimized service delivery with (potentially) continuous and dynamic delivery control in order to maximize the end-user's satisfaction and optimally utilize limited system resources. From an operator point of view, the goal would be to maintain satisfied end users (in terms of their achieved QoE) in order to limit customer churn, while efficiently allocating available wireless network resources. QoE optimization as such may be considered a very challenging task due to a number of issues characteristic for converged all-IP wireless environments, including limited bandwidth and its variability, the growth of mobile data, the heterogeneity of mobile devices and services, the diversity of usage contexts, and challenging users' requirements and expectations, as well as the strive to achieve cost efficiency.

In light of the above, several QoE-centric network management solutions have been proposed, which aim to improve the QoE delivered to the end-users. In this perspective,

network resources and multimedia services are managed in order to guarantee specific QoE levels instead of classical QoS parameters, which are unable to reflect the actual delivered QoE. A pure QoE-centric management is challenged by the nature of the Internet itself, as the Internet was not originally designed to support today's complex and high demanding multimedia services. As an example, network nodes can become QoE-aware by estimating the status of the multimedia service as perceived by the end-users. This information can then be used to improve the delivery of the multimedia service over the network and proactively improve the users' QoE. This gives the service provider and network operator the capability to minimize the storage and network resources by allocating only the resources that are sufficient to maintain a specific level of user satisfaction [64].

3.3 QoE MODELS

As already mentioned, a prerequisite to successful QoE management is QoE modeling, which aims to model the relationship between different measurable QoE IFs and quantifiable QoE dimensions (or features) for a given service scenario. Such models serve the purpose of making QoE estimations, given a set of conditions, corresponding as closely as possible to the QoE as perceived by end users. Based on a given QoE model specifying a weighted combination of QoE dimensions and a further mapping to IFs, a QoE management approach will then aim to specify KQIs and their relation with measurable parameters, along with quality thresholds, for the purpose of fulfilling a set optimization goal (e.g., maximizing QoE to maximize profit, maximizing number of "satisfied" customers). An important issue to note is that different actors involved in the service provisioning chain will use a QoE model in different ways, focusing on those parameters over which a given actor has control (e.g., a network provider will consider how QoS-related performance parameters will impact QoE, while a content or service provider will be interested in how the service design or usability will impact QoE) [65]. In this subsection, some well-known QoE models for voice, video and YouTube are presented.

3.3.1 Voice

Lately, there has been shown an increasing interest in supporting voice applications over both the public Internet and private intra-nets, i.e., VoIP. An important aspect of VoIP is the development of a performance monitoring model to track the quality of the voice transmission [66]. The QoE model used in the scope of the current diploma thesis is the International Telecommunications Unit – Telecommunications Standardization Sector (ITU-T) **G.107** model for VoIP (or E-model).

The ITU-T's E-Model is a network planning tool used in the design of hybrid circuit-switched and packet-switched networks for carrying high quality voice applications. The tool estimates the relative impairments to voice quality when comparing different network equipment and network designs. The tool provides the means to estimate the MOS rating of voice quality over these planned network environments [66].

The G.107 E-model constitutes a formula that can be used for the computation of voice transmission quality. A basic result of the E-Model is the calculation of the Transmission Rating factor (*R* factor), which is a simple measure of voice quality ranging from a best case of 100 to a worst case of 0. The *R*-factor is defined in terms of several parameters associated with a voice channel across a mixed Switched Circuit Network (SCN) and a Packet Switched Network (PSN). The parameters included in the computation of the *R*-factor are fairly extensive covering such factors as echo, background noise, signal loss, codec impairments, and others [66]. *R* is given from the following relationship:

$$R = R_0 - I_s - I_d - I_{e\text{-eff}} + A \quad (1)$$

where:

- R_0 represents the basic signal-to-noise ratio
- I_s expresses the signal-to-noise impairments associated with typical SCN paths
- I_d represents the impairments caused by the mouth-to-ear delay of the path
- $I_{e\text{-eff}}$ represents equipment impairments caused by the losses within the low bit-rate gateway codecs
- The Advantage or Expectation Factor A allows for compensation of impairment factors when the user benefits from other types of access to the user.

An interesting aspect of the E-Model is that these terms, i.e., I_s , I_d , and $I_{e\text{-eff}}$ are additive and further, that the delay and packet loss contributions are isolated into I_d and $I_{e\text{-eff}}$, respectively. This does not imply that delay and packet loss are un-correlated in the underlying transport media, but only that their contributions to the estimated impairments are separable.

The Expectation Factor covers those intangible quantities that are difficult (or impossible) to quantify. This term accounts for lowered customer expectations of quality because of, e.g., a cell phone user's tendency to tolerate lower quality in exchange for the convenience afforded by mobility, or in exchange for a lower price. For the most part it is difficult to estimate the Expectation Factor. For this reason, the Expectation Factor is dropped from the computation of R .

Also, the signal-to-noise impairment factor I_s is a function of several parameters, none of which are a function of the underlying packet transport. However, the ITU-T Rec. G.107 recommends a set of default values for these parameters for planning purposes. Because this is not the focus of our discussion, and is dependent upon the method to access the VoIP network, the default recommendations are used for all but a few parameters, i.e. all except for the delay and packet loss parameters. For example, it is sufficient to assume that echo cancellers are present and working properly (no echo) [66].

Overall, the E-model in its standard format can be used for network planning purposes only. The extended version may be simplified under specific assumptions to enable its use for quality monitoring purposes. The most significant assumptions are:

- The codec used is G.729a
- Packet loss is up to 16% and random
- The Advantage factor A is neglected.

Therefore, the R factor is simplified to:

$$R = 94.2 - I_d - I_e - \text{eff}$$

In the case of the baseline scenario where no network or equipment impairments exist, the R factor is given by:

$$R = R_0 = 94.2$$

I_d is expressed as:

$$I_d = 0.024d + 0.11(d - 177.3)H(d - 177.3)$$

where:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

and d is the average packet transmission delay. Also, the packet loss rate p relates to the parameter I_{e-eff} as follows:

$$I_{e-eff} = 11 + 40 \ln(1 + 10p)$$

By substituting these values to expression (I) above, a simplified expression for R occurs:

$$R = 94.2 - 0.024d - 0.11(d - 177.3)H(d - 177.3) - 11 - 40\ln(1 + 10p)$$

where delay d is expressed in milliseconds and packet loss p as a decimal number.

Although the R factor can be used as an assessment value, it is recommended that it is used to derive the corresponding MOS values, which are comparable with results provided by subjective methods. The R -factor relates to the MOS through the following expression:

$$MOS = \begin{cases} 1, & \text{if } R < 0, \\ 1 + 0.035R + R(R - 60)(100 - R) \cdot 7 \cdot 10^{-6}, & \text{if } 0 \leq R \leq 100, \\ 4.5, & \text{if } R > 100. \end{cases}$$

The relationship between the R factor and MOS is graphically presented in Image 3.2 [66].

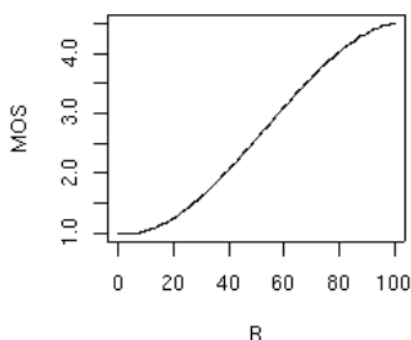


Image 3.2: Relationship between R factor and MOS [66]

Typically, the values of the R factor are categorized as shown in Table 3.1. Connections with R factors of less than 60 are expected to provide a poor quality of service to users [66].

Table 3.1: R factors, quality ratings and the associated MOS [66]

R Factor	Quality of Voice Rating	MOS
$90 < R < 100$	Best	4.34 – 4.5
$80 < R < 90$	High	4.03 – 4.34
$70 < R < 80$	Medium	3.60 – 4.03
$60 < R < 70$	Low	3.10 – 3.60
$50 < R < 60$	Poor	2.58 – 3.10

3.3.2 Video

The E-model, and specifically version ITU-T **G.1070**, can also be applied to video applications. This version constitutes a computational model for video-telephony applications over IP networks, which is useful as a QoE/QoS planning tool for assessing the combined effects of variations in several video and speech parameters that affect QoE [67].

Hence, this model is a framework for estimating the QoE of video-telephony applications in order to will guarantee the end-users' satisfaction. The model contains on three basic pillars: the video-alone quality estimation, the speech-alone quality estimation and the multimedia quality integration, and formulae are provided for each one of them. The video quality estimation function, ranging from **5 (best) to 1 (worst)**, is the following:

$$V_q = 1 + I_{coding} * I_{transmission}$$

where:

- I_{coding} represents the video quality affected by the coding distortion and is described by the formula:

$$I_{coding} = I_{ofr} \exp \left\{ - \frac{(\ln(Fr_v) - \ln(O_{fr}))^2}{2D_{FrV}^2} \right\}$$

- $I_{transmission}$ represents the video quality affected by the transmission process and is described by the formula:

$$I_{transmission} = \exp \left\{ - \frac{P_{plv}}{D_{Pplv}} \right\}$$

and:

- Fr_v is the **video frame rate** (frames/sec). It can be extracted from the video specifications
- Br_v is the **video bit rate** (bits/sec). It can be computed using the formula:

$$Br_v = Fr_v \frac{\text{bitsReceived}}{N * (1 - Ppl_v)}$$

where N is a frame sliding window, so that each output value depends on the N preceding frames.

- Ppl_v is the **video packet loss rate**. It can be computed using the formula:

$$Ppl_v = \frac{\text{lost packets}}{\text{lost packets} + \text{received packets}}$$

These three parameters are necessary to be known, and all the rest can be estimated as in relation to them as follows:

$$\checkmark O_{fr} = v_1 + v_2 Br_v$$

$$\checkmark I_{ofr} = v_3 - \frac{v_3}{1 + \left(\frac{Br_v}{v_4}\right)^{v_5}}$$

$$\checkmark D_{Fr_v} = v_6 + v_7 Br_v$$

$$\checkmark D_{Ppl_v} = v_{10} + v_{11} \exp\left(-\frac{Fr_v}{v_8}\right) + v_{12} \exp\left(-\frac{Br_v}{v_9}\right)$$

The coefficients $v_1 - v_{12}$ can be derived using a standard methodology provided in [67], as long as codec type, video format, key frame interval and video display size are known. Their default values for specific sets of configurations (shown in Table 3.2) may be found in Table 3.3 [67]. The required configurations include:

- The codec type, which is the way that a video was encoded.
- The video format, which is the video resolution or the number of distinct pixels in each dimension that can be displayed.
- The key frame interval, which is one of the frames in a video that provide the best summary of the video content and can be thought of as a reference point of the video.
- The video display size, which is an assumption regarding the size of the screen where the video is being watched.

The current thesis implementation covers these five default cases.

Table 3.2: Conditions for deriving coefficient tables [67]

Factors	#1	#2	#3	#4	#5
Codec type	MPEG-4	MPEG-4	MPEG-2	MPEG-4	ITU-T H.264
Video format	QVGA	QQVGA	VGA	VGA	VGA
Key frame interval (s)	1	1	1	1	1
Video display size (inch)	4.2	2.1	9.2	9.2	9.2

Table 3.3: Provisional coefficient table for the video quality estimation function [67]

Coefficients	#1	#2	#3	#4	#5
v_1	1.431	7.160	4.78	1.182	5.517
v_2	0.02228	0.02215	0.0122	0.0111	0.0129
v_3	3.759	3.461	2.614	4.286	3.459
v_4	184.1	111.9	51.68	607.86	178.53
v_5	1.161	2.091	1.063	1.184	1.02
v_6	1.446	1.382	0.898	2.738	1.15
v_7	0.0003881	0.0005881	0.0006923	-0.000998	0.000355
v_8	2.116	0.8401	0.7846	0.896	0.114
v_9	467.4	113.9	85.15	187.24	513.77
v_{10}	2.736	6.047	1.32	5.212	0.736
v_{11}	15.28	46.87	539.48	254.11	-6.451
v_{12}	4.170	10.87	356.6	268.24	13.684

3.3.3 YouTube

The E-model described in Section 3.3.2 handles real-time, hence lossy, video delivery. A different type of video content delivery that of streaming pre-encoded video, such as YouTube. The difference lies in the fact that YouTube uses Transmission Control Protocol (TCP)-based connections, and therefore does not suffer from packet losses. The key factors that affect the YouTube video delivery quality are:

- Number of stalling events, N
- Duration of stalling events, L
- Total video duration, T (so that it is compared to the total stalling events duration)
- Initial delay (video start-up delay)

The QoE model for YouTube follows the so-called IQX hypothesis, which describes a QoS-to-QoE mapping. The use of such a QoE-QoS relationship is straightforward; by inserting measured QoS values into the corresponding exponential formula, their impact on QoE can be assessed immediately. Typically, the QoE parameter and user

perception decrease when the QoS parameter increases [67]. The IQX hypothesis curve between the QoE and the QoS disturbance consists of three clearly distinguishable regions, as shown in Image 3.3.

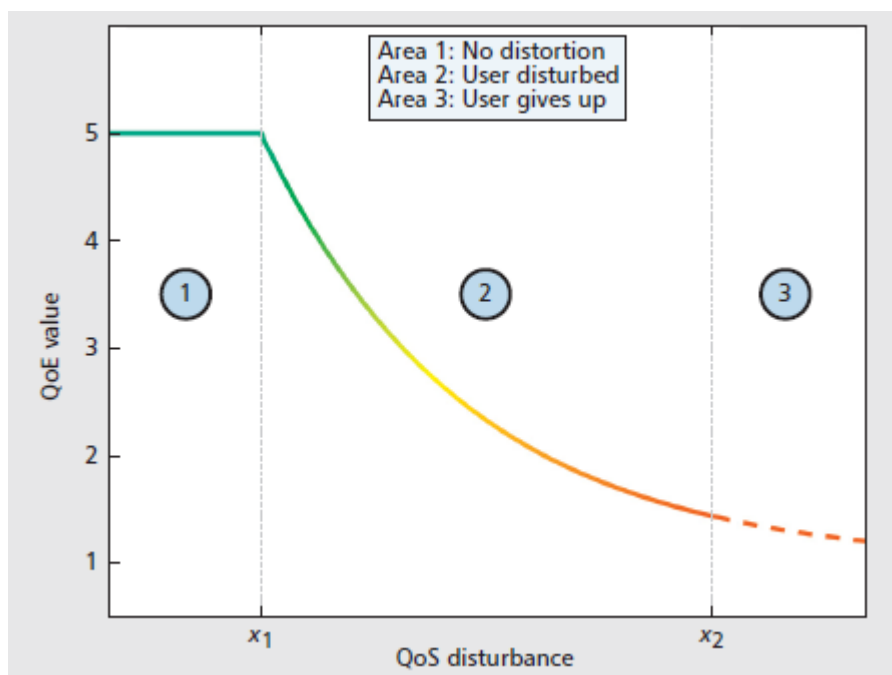


Image 3.3: The IQX hypothesis [68]

- **Area 1: constant optimal QoE.** For a vanishing QoS disturbance (i.e., in case of a transparent network), the user considers the QoE equivalent to that of the reference. A slight growth of the QoS disturbance may not affect the QoE at all. For instance, small delays and delay variations may be eliminated by a jitter buffer, without the user noticing the additional delay.
- **Area 2: sinking QoE.** When the QoS disturbance exceeds a certain threshold, the former quasi-optimal QoE level cannot be maintained anymore. As the QoS disturbance grows, the QoE and thus the user satisfaction sinks. In case of a high QoE, a certain additional QoS disturbance might have a considerable impact on the QoE, while for low QoE, the particular additional QoS disturbance might not be that critical anymore.
- **Area 3: unacceptable QoE.** As soon as the QoS disturbance reaches another threshold, the outcome of the transmission might become unacceptably bad in quality, or the service might stop working because of technical constraints such as timeouts. A user might give up using the service at that point; this is illustrated by the dashed line.

The mapping function for YouTube QoE has the following form [68]:

$$QoE(L,N) = \alpha * e^{-\beta(L)*N} + \gamma$$

4. STATE OF THE ART IN QoE FOR SOFTWARE - DEFINED NETWORKS

As already explained, the SDN paradigm is an emerging and very promising architecture, considered to be suitable for the high-bandwidth, dynamic nature of today's applications [1]. Therefore, many industrial and academic parties have focused their research activities on the concept of SDN. In particular, the QoE aspect of SDN has gained more and more attention over the past few years. This has resulted in the publication of several research papers, studies and solution proposals trying to provide approaches for user QoE enhancement aided by SDN, the most important of which are presented below [31]. Next follows a table summarizing the contributions of each research work and comparing the approaches against several parameters.

4.1 RESEARCH WORKS ON QoE FOR SDN

A very innovative paper which comes up with a solution proposal for VoD efficient distribution is [31]. The paper is motivated by the fact that the users' requests for video content are currently handled individually and each request is served by an independent unicast flow, which leads to multiple duplicate flows transferring the same video content and thus increasing the overhead for the network. The solution introduced is OpenCache, which is an in-network caching service whose architecture follows the principles of SDN and is shown in Image 4.1. The OpenCache Controller is the main entity of this architecture, as it orchestrates the content caching and distribution functionalities. The network also contains OF switches, which are connected to the users in order to transfer the requested content to them, as well as to OpenCache nodes, where the content is stored. The OF controller is dynamically instructed to set the necessary flows in the switches, the key-value store holds the requests for content to be cached and finally the VoD server is the primary source for the video assets. The proposed solution has been deployed in the GOFF European testbed and the results were extracted using the Scootplayer evaluation tool, showing that OpenCache improves network utilization, reduces the distribution load in the network and minimizes the distance between the VoD server and the users, thus leading to enhancements in the QoE in terms of throughput, bit rate, as well as start-up and buffering times.

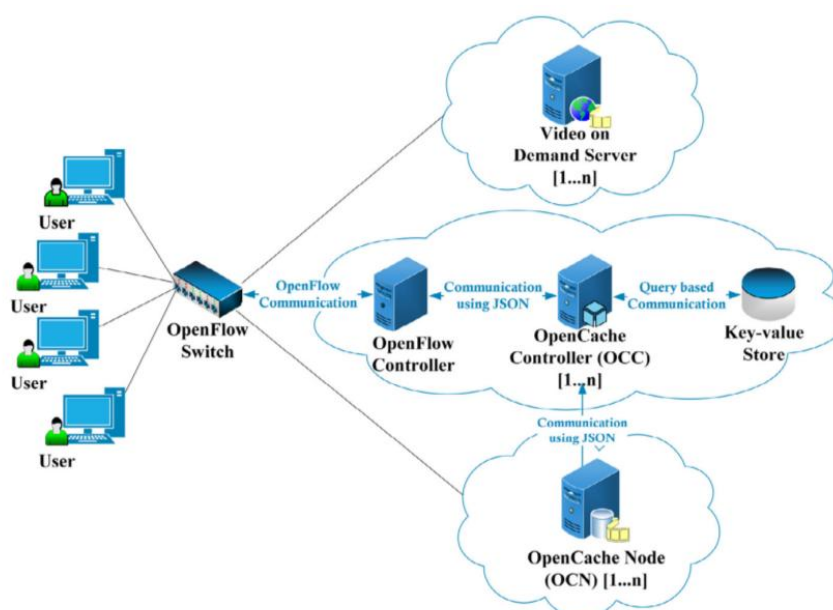


Image 4.1: The OpenCache architecture [31]

The authors of [24] also focus their attention on QoS/QoE and implement a QoS/QoE mapping and adjusting application, which can be used by an ISP to monitor users' QoE after having watched some video content and adjusting it, if necessary. The system's general overview is shown in Image 4.2. A Network Architecture Application (NAApp) is designed in order to obtain information from the OpenFlow switches (e.g. network architecture, QoE threshold etc.) and store it to a database as well as monitoring the switches topology, in case a change happens. Subsequently a QoE Measure Interface (QoEMI) collects users' parameters generated during the video watching process as well as the score given by users after having watched the video, and delivers them to a QoE Application (QoEApp). The QoEApp is the application's core component, as it is responsible for performing the algorithm which will show if an adjustment is needed, and also for applying it, in such case. The adjustments are performed by calculating all possible paths to the users and transmitting the traffic via multiple paths. The application was implemented using OpenvSwitch and RYU controller and the results showed that when used by the ISP in a given case of multiple users simultaneously watching the same video, it improved the QoE of those users who were receiving a low score.

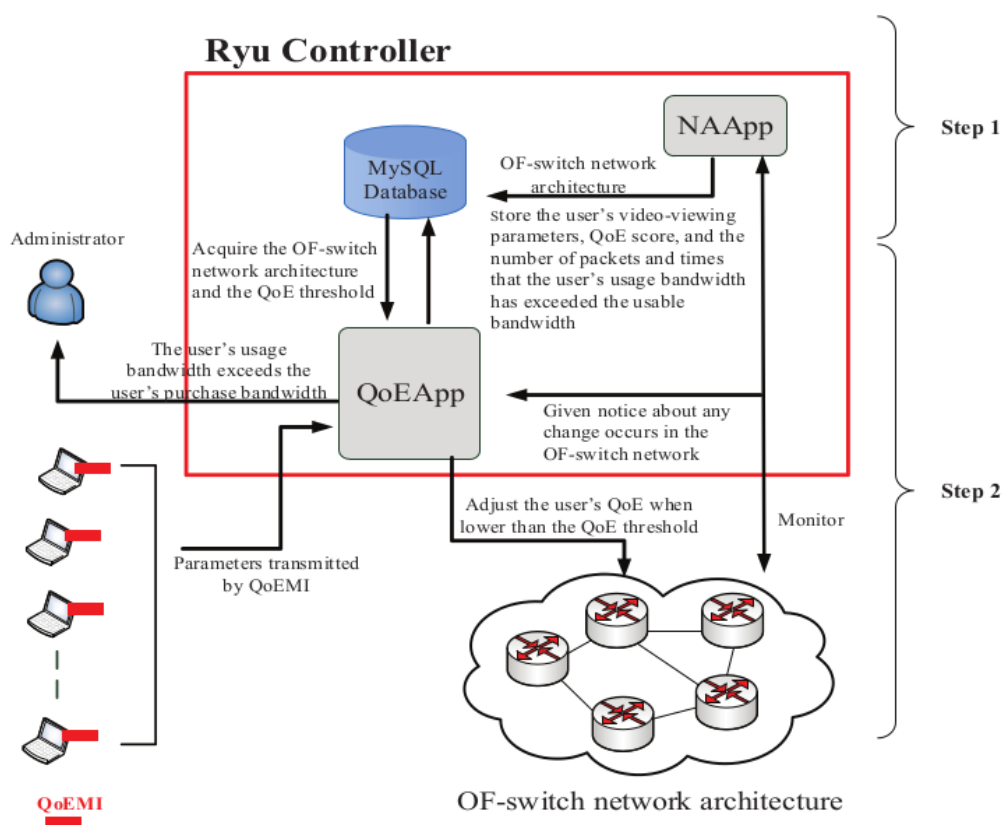


Image 4.2: The QoS/QoE Mapping and Adjusting application overview [24]

Subsequently, [69] examines the benefits that SDN and Network Functions Virtualization (NFV) can bring both to network operators and to users, studying a use case of a video application request from a user and its provision to his home network. It is explained that for each application there exists a service chain, which is a number of services necessary for its operation in compliance with the user's request. For instance, in the case of home networks, the service chain contains cache storage, bandwidth optimizer, type of service and traffic prioritizer, which need to be supported. The use case of the video application is divided into three sub use cases, and more specifically into video streaming request, HD video request and video conference. For the first sub case some packet losses can be accepted and the video is thus offered with best effort

quality. The second case requires high QoS and bandwidth, therefore some more network functions, such as cache storage and bandwidth accelerator, need to be implemented. In the sub case of video conference, priority must be given to voice in comparison to video and congestion problems when several attendees are participating should be overcome. As a result, this case also requires some additional network functions, such as traffic prioritizer and load balancer. In all cases, many network function elements can be virtualized, becoming Virtual Network Functions (VNFs), and be placed centrally instead of each individual user terminal, something which can lead to reduced OPEX for the operators as well as reduced cost for the end users, as the authors conclude.

Another very interesting proposal of a bandwidth management solution aiming to optimize the QoE of multiple video streaming sessions is presented in [70]. The motivation for this paper is the rapid increase of video traffic over the past year which consequently introduces an increased need for bandwidth on the existing network infrastructures. The authors focus on jointly optimizing bandwidth allocation and video rate selection and to this end they propose an SDN-based architecture shown in Image 4.3. The architecture contains a Video QoE Optimization Application (VOQA), which obtains information through interacting with the SDN Controller, and subsequently extracts and publishes statistics useful for the network operators. It also has the ability to adjust the network bandwidth allocation and coordinate the video rate selections among the different HyperText Transfer Protocol (HTTP) Adaptive Streaming (HAS) streams. The multi-client bandwidth allocation problem and its restrictions are modeled mathematically and offer flexibility in supporting variations of quality-based bandwidth allocation. The system's evaluation using Cisco routers showed that the proposed solution can provide same levels of QoE to 75% more users compared to the conventional bandwidth allocation and video rate selection.

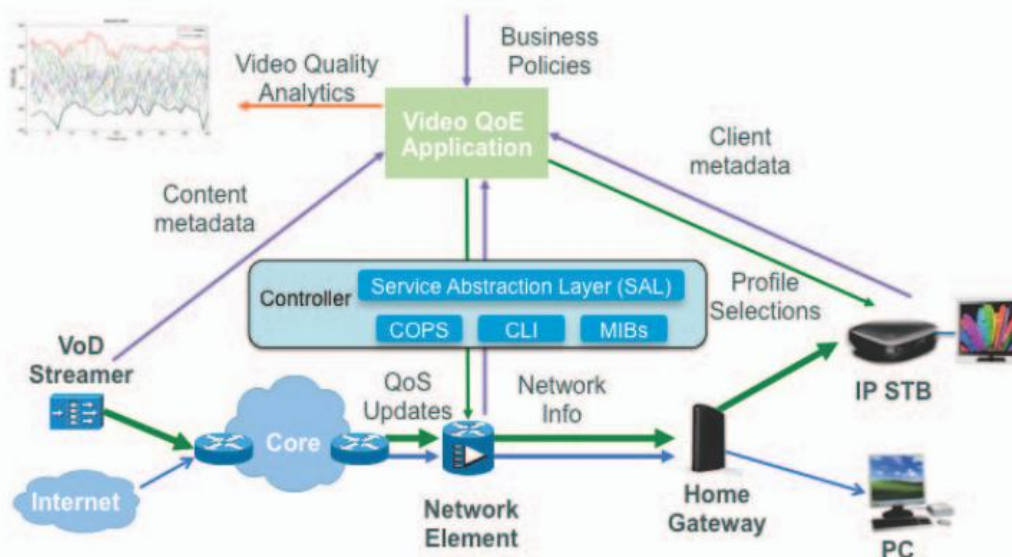


Image 4.3: SDN-Based architecture for QoE optimization in HTTP-based video streaming [70]

Next presented is paper [71], focusing on the very interesting case of vehicular networks, motivated by the fact that the high mobility and the limited transmission range of Road-Side Units (RSUs) lead to interference due to dynamic topological changes. It is necessary to divide the network resources fairly among the vehicles in the network, in order to minimize interference and satisfy as many as possible vehicles' requirements in QoE. In this direction the authors propose a software-defined flow and power management model to be implemented in the controller. The RSUs are modeled using a queuing theoretical approach and an SDN- and IEEE 802.11p-based architecture for

vehicular networks is introduced (Image 4.4), where the data plane contains the RSUs and the vehicles connected to them and the control plane manages the topology and consists of the Flow Management and the Power Management components. The Flow Management model classifies the vehicles to satisfied and unsatisfied, depending on whether each vehicle's QoE is kept above a threshold or QoE degradations are caused by new coming vehicles. Once the unsatisfied vehicles have been detected, the Power Management model adjusts their transmission power so that they connect to a new RSU with optimal signal level. The coordination of the vehicles' signal levels is performed by the controller through a reorganization of the OF flow tables so that the match fields contain a redefined version of Flow Label. The proposed model and architecture are evaluated in MATLAB environment using Exponential, Gaussian and Linear model for the estimation of unsatisfied vehicles' optimal signal levels by the controller. The evaluation shows that the Exponential model is the most suitable option, serving an average of 8% more unsatisfied vehicles.

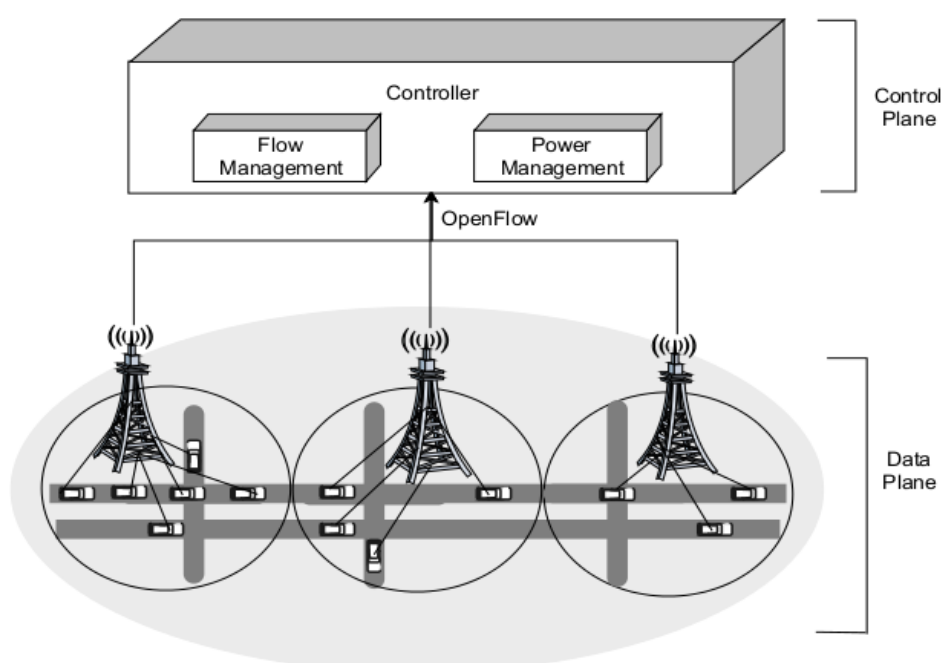


Image 4.4: The SDN-based vehicular network architecture [71]

Research work [72] constitutes an effort towards the direction of HTTP adaptive video streaming. In particular the authors introduce a dynamic SDN-based traffic shaping technique, namely DASH-SDN, which utilizes the long idle periods of an HTTP video player in order to temporarily allocate the unused bandwidth to other active players, and therefore increase the throughput and enhance the video service and the QoE. The SDN-based DASH-SDN architecture proposed is shown in Image 4.5 and is divided into two parts, the wireless infrastructure and the mobile devices.

- The first part is responsible for several key functions, such as network monitoring, flow inspection and bandwidth management.
- The second part consists of the flow manager, which handles the measurements about each received chunk and actually implements the traffic shaping, as well as of the mobile controller, which communicates with the SDN controller and controls the flow manager.

- The SDN controller is responsible for computing fair shaping rates and sending them to the mobile controllers, which in turn impose these limits through the TCP flow control. In other words, the actual traffic shaping is implemented on the mobile devices through TCP flow control, and more specifically through the modification of two fields in the TCP ACK packet according to the value received by the mobile controller.

The authors conducted relative experiments using android devices and considering different scenarios, and the results extracted showed for all cases that the proposed technique achieved up to 40% reduction in the quality fluctuation, up to 13% increase in the bandwidth utilization and up to 15% decrease in the Wi-Fi power consumption, compared to the static technique.

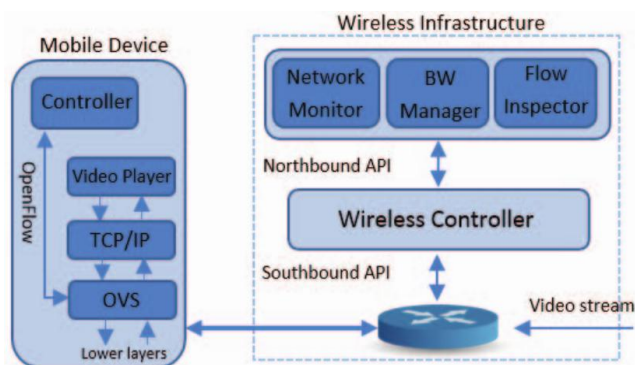


Image 4.5: The DASH-SDN architecture [72]

Due to the several defects such as scalability, intelligence and underlying abstract problems that current Content Delivery Network (CDN) architectures face with large-scale video services, the authors of [21] propose an HTTP video content delivery scheme deployed on an SDN network for improving the quality of HTTP video and the user QoE. The scheme's architecture (Image 4.6) consists of a number of user terminals, which can connect to the Internet through programmable storage routers, which are routers with storage capabilities, controlled by an SDN controller. The role of the programmable storage routers is to periodically request and receive video content from the video source server and forward users' requests to the SDN controller. The controller in turn specifies the closest programmable storage router to the user and instructs it to detect whether the requested content exists in its storage. If it does, it is directed to the user, otherwise the user request will be forwarded to a nearby programmable storage router. This approach appears to be advantageous over requesting content from the video source server or a CDN edge server, as the distance and process of video transmission are significantly shortened. In order to prove their claims, the authors conducted five sets of experiments for HTTP video service simulation in both SDN environment and current network environment, including CDN. The results show that the round-trip delay is significantly reduced with the proposed SDN scheme and the video quality and user QoE are improved, even in poor network conditions.

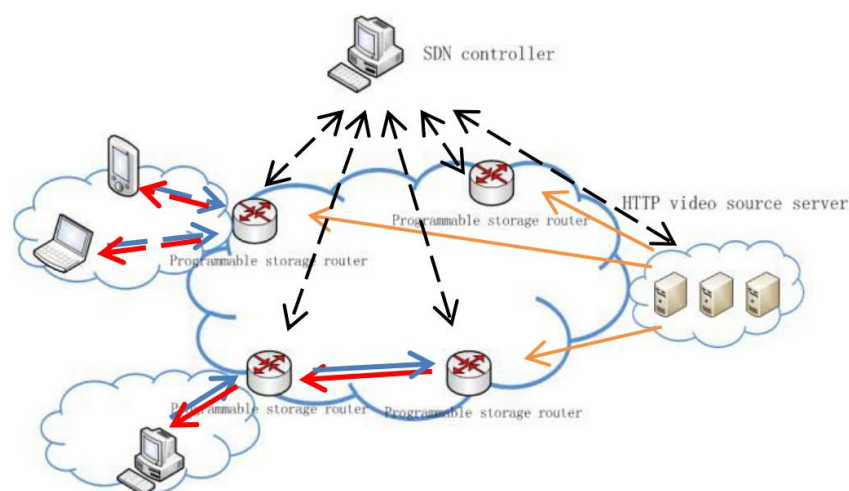


Image 4.6: The SDN-based scheme for HTTP video quality optimization [21]

Work [19] focuses its attention on the two complementary technologies SDN and NFV, which it explains very thoroughly in the introductory sections. The network functions which are virtualized using NFV are called VNFs. The authors use these two technologies to introduce a proof-of-concept SDN/NFV-enabled experimental network domain implementation in order to provide an agile video transcoding process for maintaining the QoE level of a media service. The experimental topology is depicted in Image 4.7 and contains two OpenFlow switches, an OpenDaylight Controller and an Openstack cloud platform to support a NFV Installation Point of Presence (NFVI-PoP), which is able to instantiate VNFs. The video service steering is performed through the following procedure: When an end user requests a unicast media service whose traffic exceeds the available bandwidth, the controller instantiates at the NFVI-PoP a VNF-based transcoder, which is implemented based on FFMPEG, and defines the appropriate SDN rules for the traffic steering. As a next step, the delivered media stream is transcoded in real time at a lower rate, so as to fit in the available bandwidth, and is subsequently transmitted, causing the reinstatement of the QoE level which had faced degradation.

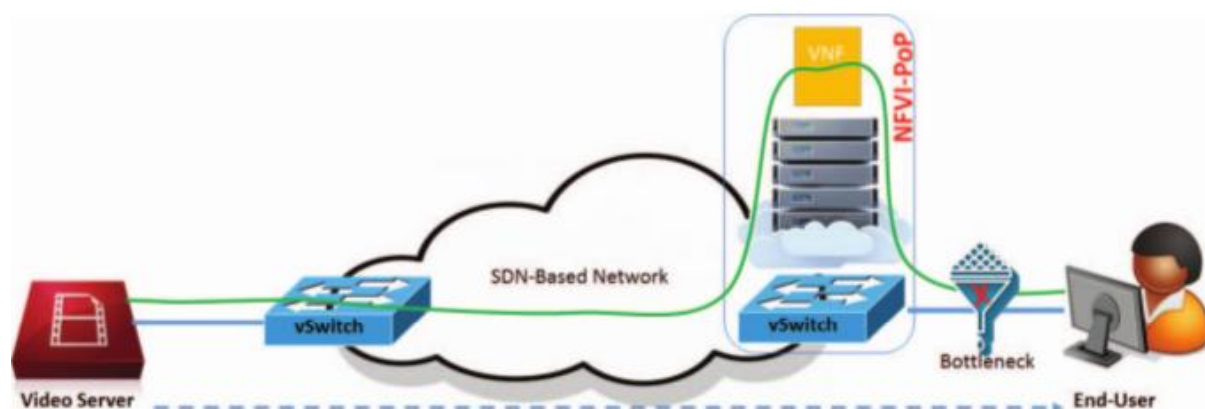


Image 4.7: Topology of the experimental testbed of [19]

The authors of [73] examine the tradeoff between user experience and resources cost and make an introduction of an SDN-enabled cloud video distribution architecture (Image 4.8) aiming at enhancing user QoE and diminishing the operators' cost. To this direction, they also propose a joint resource allocation and traffic management mechanism for Video Service Providers (VSPs) which provides a solution to determine

the optimal resource (i.e. bandwidth) allocation combined with the optimal strategy for request dispatching and response routing.

The proposed architecture contains multiple geographically distributed cloud datacenters being connected with each other over an SDN-enabled network through OF switches. Each datacenter is used to serve a number of users, also spread over multiple regions. The network's centralized control logic is gathered in a control center, consisting of an SDN controller and a cloud management server.

The whole system is mathematically modeled using graph theory and the joint optimization problem is formulated as maximizing the total utility of serving all the requests minus the bandwidth cost, introducing a number of constraints. The system's advantage is that unlike the conventional cloud distribution systems, it does not solely dispatch a user's request to the closest datacenter, but also takes into account the underlying network conditions instead.

The proposed model's evaluation was conducted through an experiment with 10 SDN router nodes, 10 geographical regions, 4 datacenters, up to 3-hop path selection between datacenters and user regions (i.e. 42 paths) and 500 video contents. It is proved that compared to the shortest path strategy, all the users' requests were satisfied and that lower link congestion, higher end-to-end capacity, higher traffic support and lower total cost are achieved by using multiple network paths.

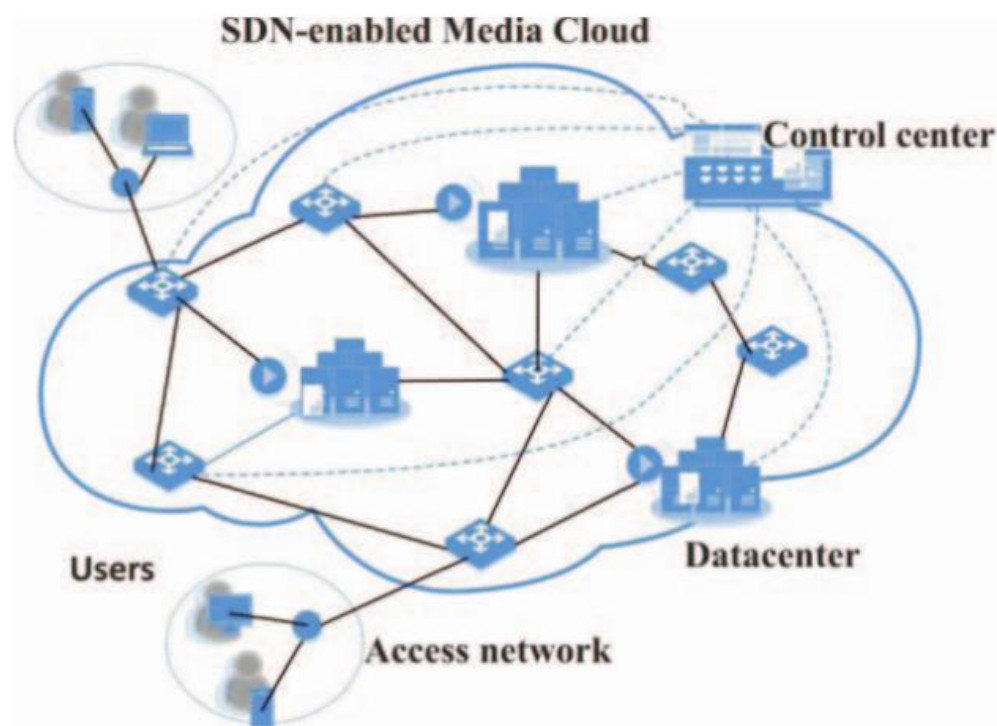


Image 4.8: SDN-enabled cloud video distribution system [73]

A very well-presented and documented work is [29], which proposes an adaptive routing approach for video streaming with QoS support using SDN networks, called ARVS. The authors study the case of high-quality videos being encoded with Scalable Video Coding (SVC), containing one or more subset bit streams such as MPEG4 SVC, which encodes a video into a base layer and one or more enhancement layers. The base layer packets are considered as level-1 QoS flows and should be transmitted without any packet loss or minimized delay variation, whereas the enhancement layer packets are regarded as level-2 QoS flows or best-effort flows and are more tolerant to packet loss. The authors employ SDN technology with the OF protocol to the end of achieving better QoS performance by offering traffic differentiation. Specifically, based

on these two layer types and their needs, they propose the transfer of either the base layer packets or the enhancement layer packets to other paths, when necessary, in order to improve the user QoS in video streaming applications.

The problem is modeled using graph theory and the goal is to minimize the path cost of a routing path subject to a given constraint (i.e. maximum delay variation). For the problem's solution computation, the Lagrange Relaxation based Aggregated Cost (LARAC) algorithm is employed and an adaptive routing approach is introduced. In SDN networks, the controller calculates the shortest path between two nodes based on path cost and both the level-1 and the level-2 video's QoS flows are streamed through this path. However, according to this proposal, if the shortest path does not satisfy the given constraint, a level-1 or a level-2 QoS flow will be rerouted on the second path (i.e. the feasible path) specified by LARAC after its condition is examined. In particular, if there is not enough available bandwidth for the level-1 QoS flow in the feasible path, a level-2 QoS flow will be rerouted to the feasible path (Image 4.9), otherwise a level-1 QoS flow will be rerouted (Image 4.10), leading to guaranteed level-1 QoS performance and congestion mitigation.

The proposed approach is simulated in Mininet, using 30 nodes connected to a remote Floodlight SDN controller, and proves that it achieves up to 77.3% improvement of base layer packets loss rate when the load level of the shortest path is raised to 0.7, as well as it increases the coverage at least 51.4% under various network loads both for the shortest and for the feasible paths.

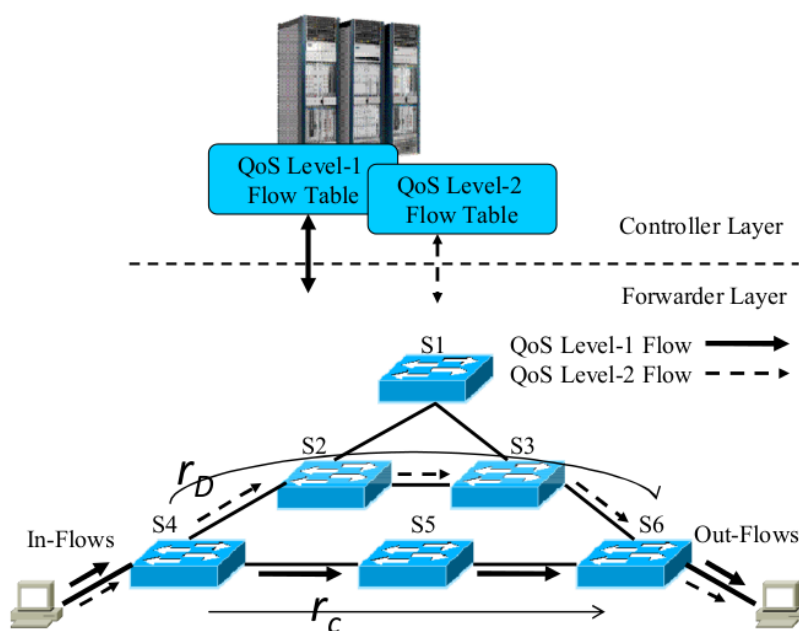


Image 4.9: Level-2 QoS flows rerouting in ARVS [29]

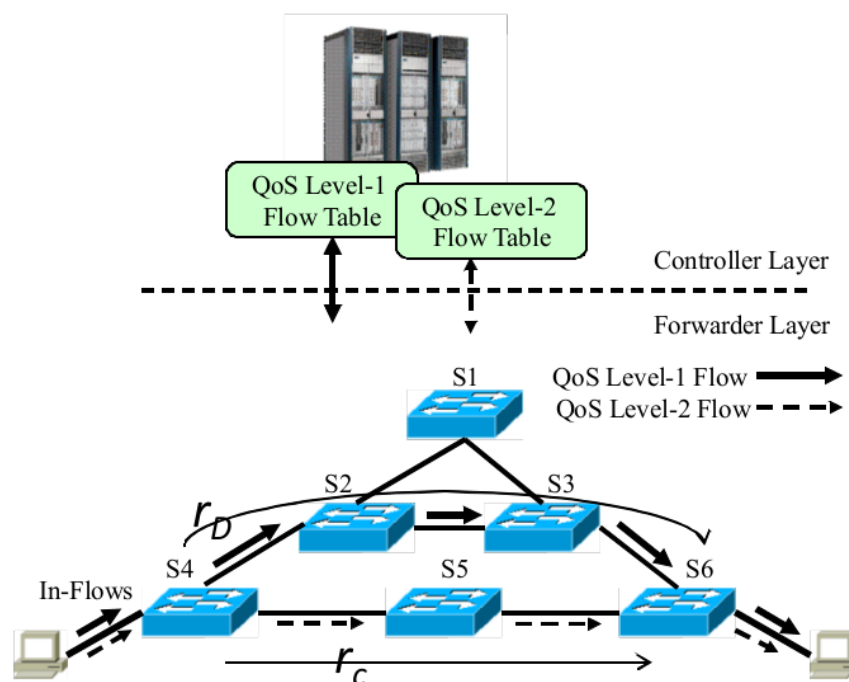


Image 4.10: Level-1 QoS flows rerouting in ARVS [29]

Paper [25] presents a complete and very interesting approach which takes into account the network traffic and uses SDN techniques in order to promptly and correctly identify video streaming flows without inspecting the packets. Existing filtering techniques have several drawbacks, as Stateful Packet Inspection (SPI) only examines the packet's header and therefore is not in a position to detect packets which do not include their type in their header (e.g. some HTTP traffic sent from the server), and Deep Packet Inspection (DPI) leads to long inspection delay, high energy consumption and significant processor overhead, while at the same time being complex and thus hard to manage and maintain.

The authors propose an inspection-free, traffic-aware, SDN-based approach which identifies video streaming flows by analyzing the statistics of flows in SDN and benefiting from the fixed duration ON-OFF cycles that video data generate. Specifically, SDN statistics are requested in every statistics retrieval time (SRT) and the amount of data transmitted within SRT ratio (R_{dt}) is computed for each flow, analyzed subsequently for video streaming pattern exhibition. This is achieved by utilizing confidence level (CL), threshold for CL (TH_{cl}) and decay rate (DR) as follows: If R_{dt} shows a pulse, the flow is considered to exhibit video streaming pattern and CL is increased, otherwise it is decreased, and the flow is flagged as video streaming flow only if eventually CL is larger than its threshold TH_{cl} , whereas in different case it is flagged as non-video streaming flow.

The proposed technique was implemented in an experimental scenario (Image 4.11) where a user has access to the Internet by a TP-Link WR1043ND SDN switch which is running OF 1.3 and is connected to Floodlight SDN Controller, according to which it handles the traffic flows. The user is watching a randomly selected video on YouTube, downloading a large file which exceeds 100MB and browsing Facebook simultaneously, using Safari 8.0 browser and HTML5 for video playback. The implementation results, which are affected both from SRT and DR, show that the proposed approach achieves 75% lower latency or 138% higher success rate compared to DPI and the large file is 100% flagged as non-video streaming flow.

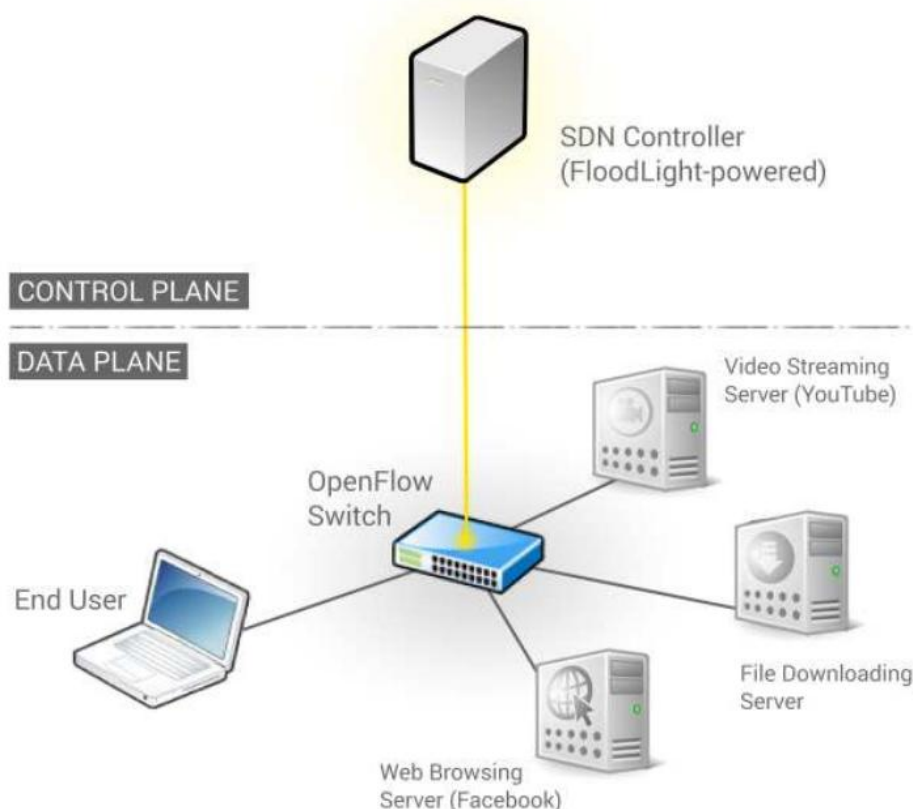


Image 4.11: Traffic-aware SDN-based topology for video streaming flows identification [25]

The in-network bit rate adaption infeasibility problem for video transmission in traditional TCP/IP networks is addressed by [74], which proposes a joint algorithm to decide the number of video layers to be transmitted as well as determine each layer's transmission path at the same time in order to enhance the video quality each user receives, and thus the user QoE. The traffic in each link is divided into SVC traffic, which consists of all scalable-video traffics and is transmitted as QoS traffic with no packet loss, and non-SVC traffic, which contains the rest of the traffic and is transmitted as best-effort traffic. The joint decision problem is formulated as Markov Decision Process (MDP), taking into consideration the non-SVC traffic as well, and solved with the employment of Q-Learning algorithm.

The system's architecture is shown in Image 4.12. There have been designed five modules in the controller: the N-Shortest Paths Calculation Module, which calculates the N shortest paths between a source and a destination node, the Network State Observation Module, which observes the network's state, the Decision Module, which jointly determines the optimal number of video layers to be transmitted as well as their routing paths, the Flow Table Module, which establishes flow tables for switches and the Layers Information Module, which establishes a communication between the SVC Content Server and the SDN controller. The system's goal is to achieve a good tradeoff between the visual quality of the transmitted SVC video and the packet loss of non-SDN traffic, and thus improve the user QoE.

The proposed system is implemented in Mininet using Open vSwitches and POX controller. The results are compared to other benchmark approaches and it is shown that the approach achieves lower loss rate of non-SVC flows caused by SVC flows (2.442% opposed to 2.941% and 4.069% of the benchmark approaches), higher reward value (0.74164 opposed to 0.70589 and 0.59308) and lower Peak Signal-to-Noise Ratio (PSNR) than one of the benchmark approaches (39.939dB opposed to 40.511 dB - the

other approach achieves lower PSNR 39.550 dB), achieving an enhancement for the user QoE.

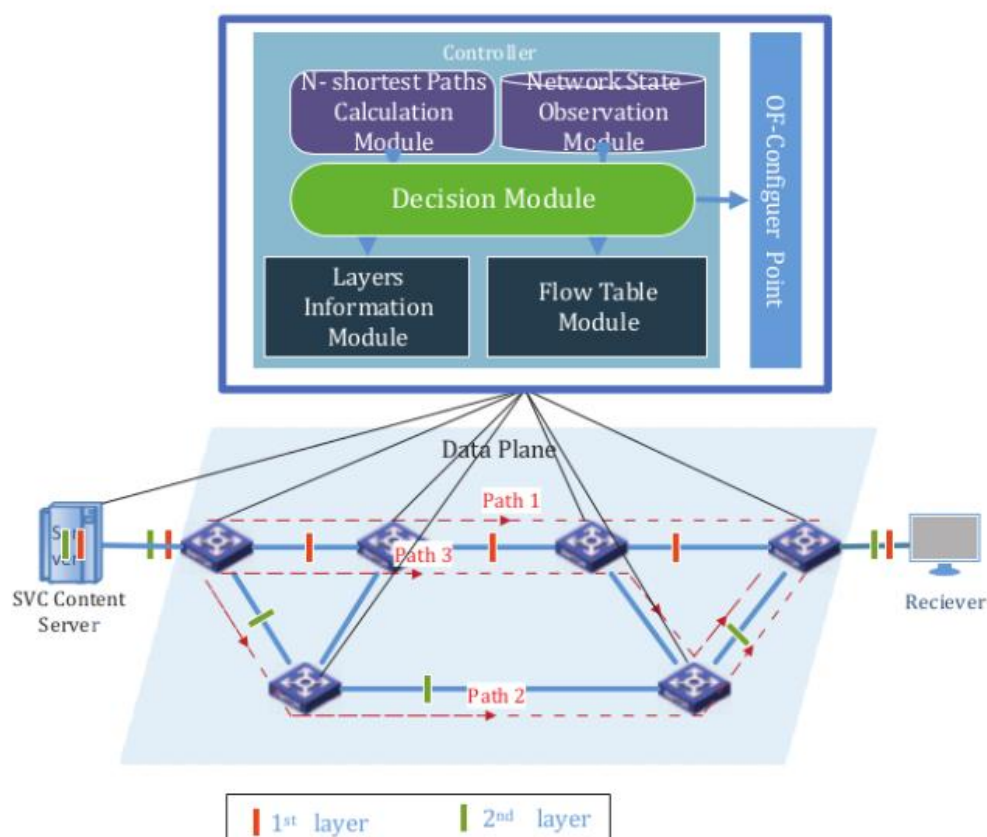


Image 4.12: The architecture of joint routing and layer selecting system [74]

Another paper which examines multimedia transmission, such as UHD video, especially focusing its attention on the next generation 5G networks is [20]. A holistic QoE- and context-aware SDN control plane approach is proposed, employing the H.265 video standard for scalable video encoding and taking into consideration not solely QoS metrics and contextual information extracted from multimedia flows, but also important QoE metrics such as content type. The authors focus on low-latency multimedia traffic, such as real-time video delivery, video surveillance and teleconferencing, and adopt version 2 of H.265 standard which makes a layered video solution feasible. The proposed system's architecture is depicted as a block diagram in Image 4.13 and contains two new network entities, the Video Quality Assurance Manager (VQAM) and the SDN Video Quality Orchestration (SDN-VQO).

The VQAM collects flow and network paths statistics and topology discover data, and also derives the video content type from the compressed H.256 stream and combines it with QoS metrics such as bandwidth, delay and packet loss, which all aid in estimating the QoE utility of each path. After the QoE utility estimation, the VQAM selects the two best paths between source and destination and chooses the best path as the primary and the second as the fallback path. In case congestion is detected, the VQAM re-estimates the two paths' utilities and either performs an action in order to ensure that the congestion is alleviated (e.g. makes the secondary path primary, switches enhancement layers to secondary path etc.) or reports the issue to the SDN-VQO if it is unable to solve it.

The SDN-VQO, which is collocated with the VQAM, has a global view of the entire SDN domain and monitors the QoE utility levels across all multimedia streams in the

network, being thus able to construct a global QoE utility map. Its role is to intervene in congestion situations in order to alleviate them and ensure that the QoE utility is fairly distributed across the network.

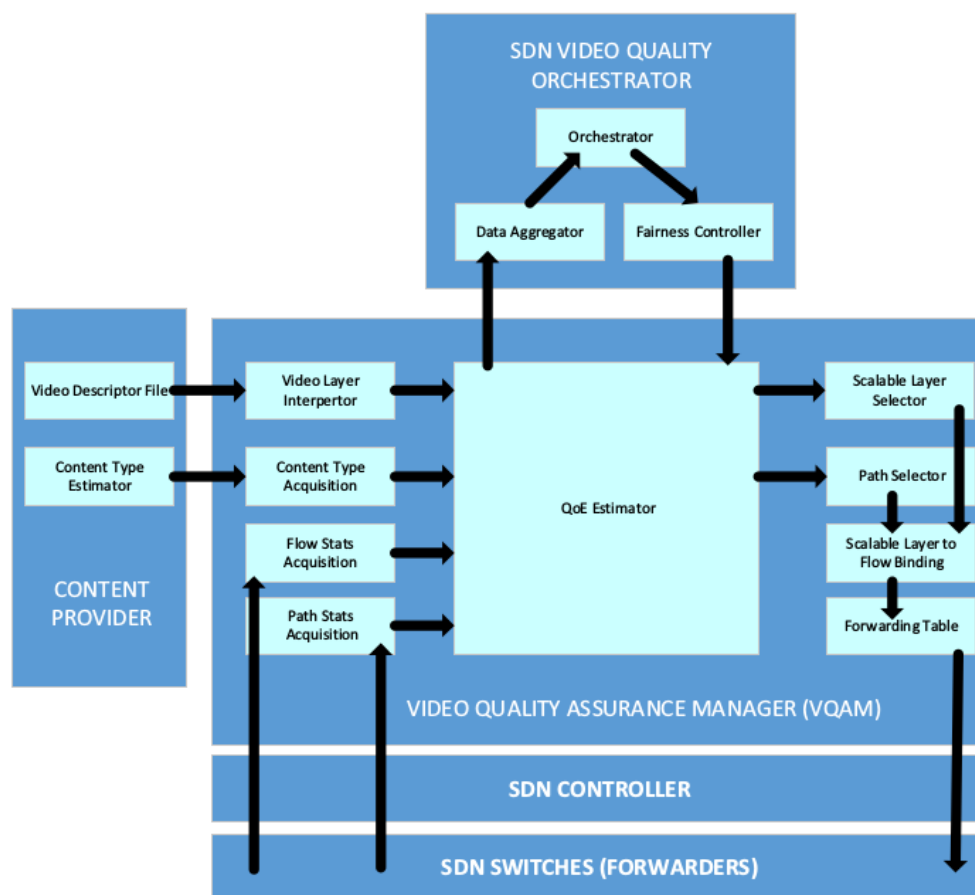


Image 4.13: The block diagram for the system proposed in [20]

The emerging field of big data is also involved and combined with the SDN field, and [75] examines this case with regard to QoS provisioning. This paper applies big data technologies to SDN and its purpose is to extend tensor, a mathematic model (type of high-dimension matrix) with wide use in big data applications, in order to introduce a new tensor-based SDN (TSDN) model for efficient QoS provisioning in SDN networks. The proposed TSDN model is depicted in Image 4.14 and consists of the data plane, the control plane and the application plane, as SDN architecture suggests.

A forwarding tensor model is introduced in the data plane to formalize the packet routing function of the network and to aid in constructing a global controlling tensor model. This is achieved through the combination of all the valuable core forwarding information and the employment of the incremental tensor decomposition approach to generate the core tensor. By using this method, the network devices can update the core forwarding tensor and submit the updates to the control plane for combination. The forwarding functions of the network devices are formalized as forwarding tensor models.

A controlling tensor model is also proposed in the control plane in order to globally compute routing paths and recommend the optimal routing paths for data scheduling in SDN. Finally, a transition tensor model is introduced and located in the application layer in order to predict network traffic and group the traffic flows according to QoS requirements of high-level applications.

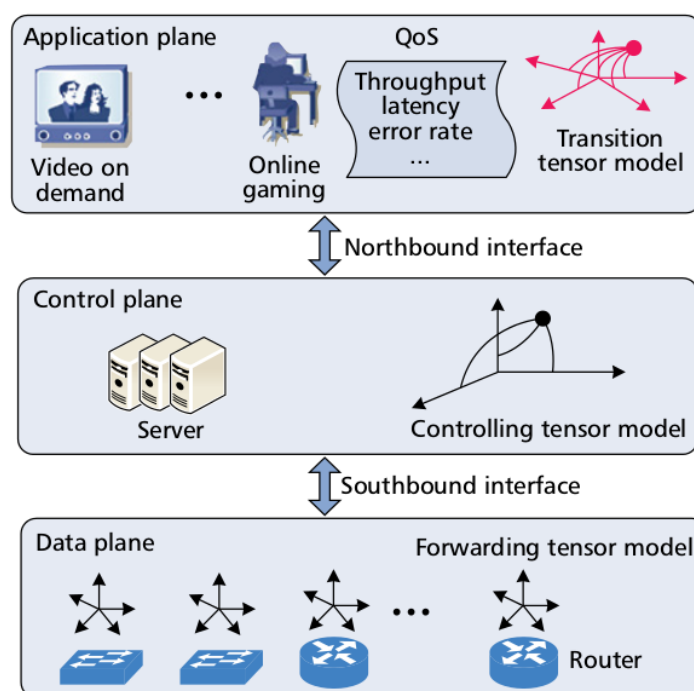


Image 4.14: An overview of the TSDN model proposed in [75]

[30] is another research work that concentrates on QoE optimization in adaptive video streaming cases and further pays great attention to ensuring QoE fairness. Therefore, it proposes an OpenFlow-assisted QoE Fairness Framework (QFF) which employs MPEG-DASH standard and aims to fairly maximize the QoE for all video streaming devices in a multimedia network, taking into account the various device and network requirements, as well as two algorithms (namely, Promote and Boost) for enforcing different optimization policies. QFF allocates network resources dynamically to each device by monitoring each DASH video application's status. An overview of QFF is presented in Image 4.15, from where it occurs that the main entity in its core is an OpenFlow Module (OM) which runs on the OpenFlow controller and consists of three parts:

- Input, which is the network and clients' status, provided to OM by the Network Inspector and the Media Presentation Description (MPD) Parser,
- Intelligence, incarnated by the Utility Functions and the Optimization Function which interact dynamically with the OM to ensure QoE fairness optimization,
- Output, constituted by the Flow Tables Manager and the DASH Plugin which ensure that the OM's decisions are appropriately propagated to the network.

The proposed framework was evaluated in a home networking scenario simulation, where users are connected to a home gateway and access video content on the Internet through an OpenFlow switch, using three different DASH-enabled devices (i.e. an HD TV, a tablet and a smartphone). The results comparing Promote with DASH-JS (unmodified DASH client) and EqualBW (equal allocation of available bandwidth among active users) showed that the proposed approach achieves user QoE fairness and improved network stability.

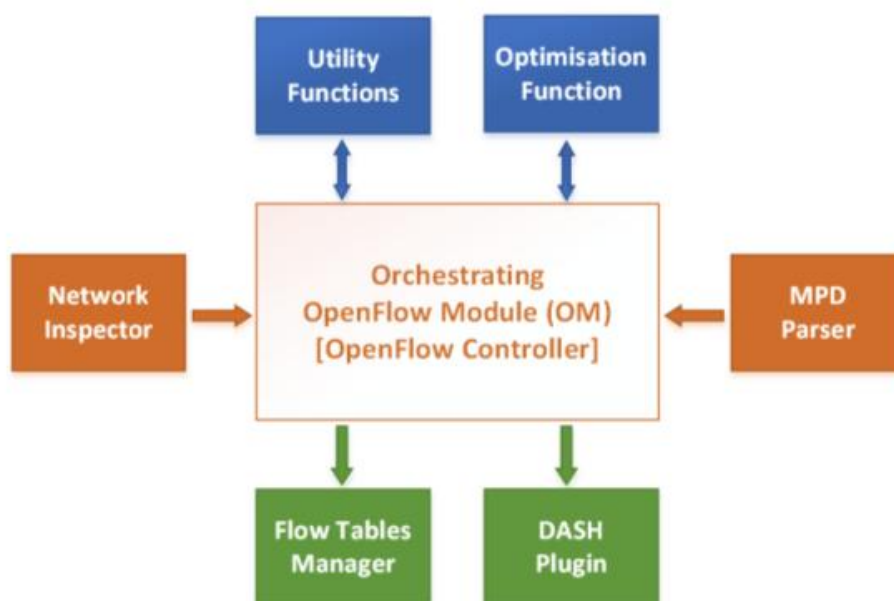


Image 4.15: OpenFlow-assisted QoE Fairness Framework [30]

One can read in [22] that an SDN-based architecture is proposed that can be used to jointly optimize the multimedia flows' path assignment and the service utility measured as QoE, exploiting a global network view. This is achieved with the help of two entities: the QoS Matching and Optimization Function (QMOF), which resides in the SDN application layer, and the Path Assignment Function (PAF), which is located at the SDN control layer. QMOF calculates the optimal (and alternative sub-optimal) service configurations. PAF then uses OF to impose the network paths that will meet the resource requirements of each service. The requirements may concern flow operating parameters (e.g., frame rate, codec), resource requirements (e.g., bit rate), etc. The proposed architecture is shown in Image 4.16.

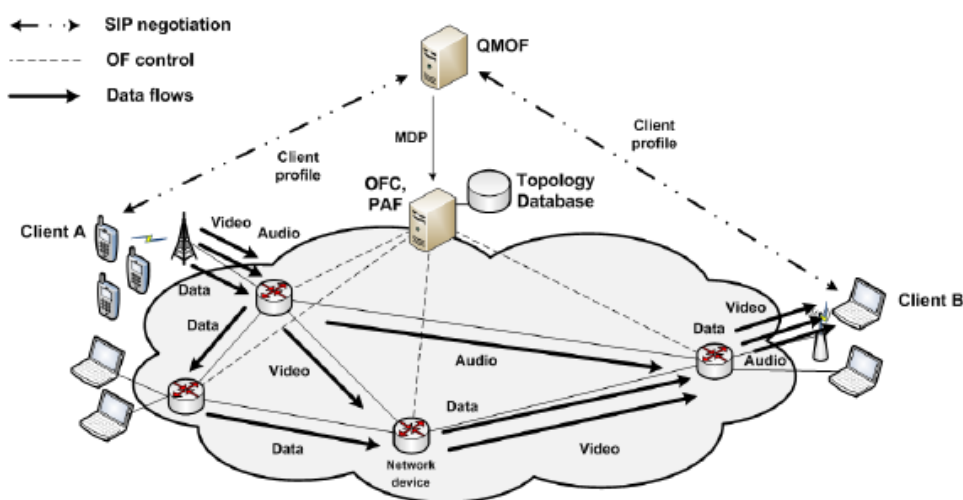


Image 4.16: SDN architecture for QoE-driven service optimization and path assignment [22]

4.2 SUMMARIZING TABLE

Having presented in Section 4.1 several significant research works that constitute the state-of-the-art in SDN networks and specifically in the QoE field, presented here is the summarizing table Table 4.1 which compares all the previously mentioned works against various parameters. Such a table is of great importance as it constitutes a valuable aggregated report containing a large number of papers and aids in obtaining a

thorough view of already proposed solutions. Therefore, it can be used as a future reference.

Table 4.1: Comparison between state-of-the-art works on QoE for SDN networks

Paper Number	QoE Management Use Case	Prototype or Abstract	QoE Monitored Factors	QoE Management Approach Employed	Single or Multiple Clients Scenario Considered
[31]	Video on Demand	Prototype	Startup time, bitrate changes, average bitrate, minimum required bitrate during playback	Content caching	Both scenarios
[24]	Video streaming	Prototype	Bandwidth utilization in relation to users number	Multiple paths routing	Multiple clients
[69]	Video applications in home networks	Abstract	N/A	Dynamic bandwidth allocation, adaptive resources management	N/A
[70]	Video streaming	Prototype	PSNR and Stream Video Quality (SVQ) in relation to users number	Dynamic bandwidth allocation	Multiple clients
[71]	Vehicular networks	Prototype	Percentage of flows satisfied	Dynamic flow and power management and resource allocation	Multiple clients
[72]	HTTP adaptive video streaming	Prototype	Number of bitrate oscillations	Dynamic bandwidth and throughput allocation	Multiple clients
[21]	HTTP video	Prototype	Initial buffering time, Rebuffering frequency, Mean rebuffering	Content caching	Single client

			duration		
[19]	Video streaming	Prototype	System load, HDD utilization, memory usage	Adaptive transcoding	Single client
[73]	Video streaming	Prototype	Average delay, traffic routing ratio, link utilization	Multiple paths routing, adaptive resource allocation	Multiple clients
[29]	Video streaming	Abstract	Packet loss rate	Multiple paths routing, Adaptive flow rerouting	N/A
[25]	Video streaming	Abstract	Time for recognizing video data	Traffic identification	Single client
[74]	Video streaming	Prototype	Packet loss rate, PSNR, reward value	Dynamic adjustment of the video layers for transmission, Dynamic routing path selection	Single client
[20]	Video streaming	Abstract	N/A	Multiple paths routing	N/A
[75]	Multimedia transmission	Prototype	Traffic congestion, incremental updating, recovery of damaged routing path	Dynamic optimal routing path computation	N/A
[30]	Video streaming	Prototype	Video stream bitrate	Dynamic network resources allocation	Single client
[30]	Video streaming	Prototype	Video bit rate, Network utilization	Dynamic network resources allocation	Multiple clients

5. ENVIRONMENT SETUP

This chapter presents in detail all the necessary steps one must follow in order to be fully capable of developing applications for SDN environments. These steps were followed to develop the SDN framework of the current thesis. There are certain system requirements which must be satisfied, in addition to the SDN Controller and Mininet (the network simulator) deployment. As a last step, an Integrated Development Environment (IDE) is needed for the Controller programming and consequently, for the applications development.

5.1 SYSTEM REQUIREMENTS

- **Operating System (OS):** OpenDaylight Controller, used for this thesis, runs in a JVM. Being a Java application, it can potentially be run from any operating system and hardware as long as it supports Java. However, for best results a recent Linux distribution is recommended [76]. The simulation in the scope of the current thesis uses the **Ubuntu 14.04** OS.
- **Java:** Due to the fact that OpenDaylight is a project written primarily in Java project, a Java 8-compliant JDK is required for project development with OpenDaylight [77]. In order to get **Java 8 JDK**, which was used in the current thesis, one should execute the following steps:

- First of all, the following commands must be executed from a terminal, in order to install Java 8 JDK on the system:

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

- Then, the `JAVA_HOME` variable must be set. This can be done by adding the following line at the end of the file `etc/environment`.

```
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

where `JAVA_HOME` is the path to the JDK.

- Finally, by executing the command `echo $JAVA_HOME`, one should be able to see the path of the variable `JAVA_HOME`.
- **Maven:** OpenDaylight primarily uses Apache Maven as a build tool. Consequently, one needs to have Maven version 3.3.1 or later installed on his system [77], [78]. For the simulation's needs, **Maven 3.3.9** was downloaded as follows:
 - The Binary tar.gz archive file, under the name **apache-maven-3.3.9-bin.tar.gz**, was downloaded from the Maven Installation Page (<https://maven.apache.org/download.cgi>).
 - After ensuring that the `JAVA_HOME` environment variable was truly set and pointed to the JDK installation, the distribution archive was extracted using the command `tar xzvf apache-maven-3.3.9-bin.tar.gz`.
 - Finally, the bin directory of the created directory, named **apache-maven-3.3.9**, was added to the `PATH` environment variable, in the first line of the file `etc/environment`.

The success of the installation was confirmed typing the command `mvn -v` in a new shell. The result should look similar to Image 5.1 [79].

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T18:41:47+02:00)
Maven home: /usr/share/maven3
Java version: 1.8.0_111, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-66-generic", arch: "amd64", family: "unix"
```

Image 5.1: The result of `mvn -v` command after Apache Maven successful installation

5.2 SDN CONTROLLER DEPLOYMENT

Before starting developing any ODL projects, one can familiarize with the ODL Controller by installing and configuring it, independently as a separate application. It can be used as a remote SDN controller, but it will not have any extra functionality created by the user. Only the default features can be installed and used. This can be achieved by executing the following steps:

- Visit the OpenDaylight downloads page (<https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation>) and download the OpenDaylight Controller. More specifically, the version used in the current thesis is **Boron SR1**.
- Once the file is downloaded, unzip the zipped file in any directory and open a terminal in this directory.
- Navigate to the bin directory and execute the karaf file (`./bin/karaf`), in order to start the ODL controller.
- In order to shutdown OpenDaylight, one can type `shutdown -f` or `logout` [79].

5.2.1 Karaf features

The OpenDaylight controller is deployed on the concept of Apache Karaf. Apache Karaf is a runtime environment which provides a lightweight container onto which various components and applications can be deployed. It can be thought of as an environment providing an "ecosystem" for an application, as it actually provides a way to provision applications and modules, and supports this using the concept of Karaf Features. OpenDaylight on Apache Karaf is an effort to deploy key OpenDaylight projects onto the Apache Karaf container environment [80], [81], [82].

Apache Karaf features describe applications. A feature defines different resources to resolve and 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

When one installs a feature, Apache Karaf installs all resources described in the feature. It means that it will automatically resolve and install all bundles, configurations,

and dependency features described in the feature [81], [82]. Therefore, the next step after installing the ODL controller is to install all the necessary features. In the case of the present thesis, a number of features was installed using the command:

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

in the karaf console. One of the most important features 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>

giving the result shown in Image 5.2.

Image 5.2: The ODL DLUX Login Page

OpenDaylight's default credentials are **admin** for both the username and password. It is important to note that the DLUX UI is only available when the controller is running, or in other words when karaf is executed. Otherwise, the webpage will not load.

Having installed the *odl-dlux-all* feature, the features *odl-dlux-core*, *odl-dlux-node* and *odl-dlux-yangui* have also been implicitly installed and can be found in the pane at the left of DLUX's page.

- Topology tab is provided by the *odl-dlux-core* feature and shows the graphical representation of the network topology (if there is one connected to the controller) on the right pane. Switches are represented by blue boxes, available hosts by black boxes, the way switches and hosts are connected by lines. By hovering on hosts, links, or switches one can view source and destination ports.
- Nodes tab is provided by the *odl-dlux-node* feature and displays a table that lists all the nodes, node connectors and the statistics at the right pane. Again, this is only possible if there is a topology connected to the controller. One can perform the following actions:
 - Enter a node ID in the Search Nodes tab to search by node connectors.
 - Click on the Node Connector number to view details such as port ID, port name, number of ports per switch or MAC Address.
 - Click Flows in the Statistics column to view Flow Table Statistics for the particular node like table ID, packet match or active flows.
 - Click Node Connectors to view Node Connector Statistics for the particular node ID.

- Yang UI tab is provided by the *odl-dlux-yangui* feature. It is an ODL DLUX-based application designed to simplify and facilitate application development and testing.
 - The Yang UI module enables the interaction with the YANG-based MD-SAL datastore [83].
 - It also generates and renders a simple UI based on YANG models loaded into ODL [84].

5.3 MININET DEPLOYMENT

In order to deploy SDN applications for various network topologies, one must use a network simulation tool. The tool chosen for the present thesis is Mininet. The easiest and most foolproof way of installing Mininet is to use a Virtual Machine (VM) installation, according to the following description:

- Download the Mininet VM image from the Mininet Releases Downloads link, <https://github.com/mininet/mininet/wiki/Mininet-VM-Images>. The current thesis uses version **2.2.1** of Mininet.
- Download and install a virtualization system. The recommended system from the official Mininet webpage is VirtualBox due to the fact that it is a free tool and works on most operating systems. Other suggested systems are Qemu, VMware Workstation, VMware Fusion and KVM.
- Once the VM image file has been downloaded, right click on the *.ovf* file and select *Open With Oracle VM VirtualBox*. This will open a window which is shown in Image 5.3.

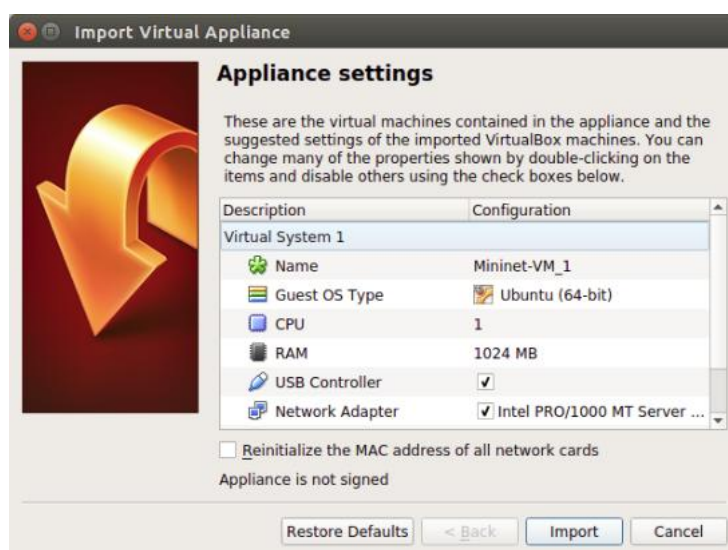


Image 5.3: VirtualBox import window

After clicking on *Import*, the image file will be imported into VirtualBox.

- Select *Settings* → *Network*, and add an additional host-only network adapter that can be used in order to log in to the VM image. In order to select one by name, create a new host-only adapter by clicking *File* → *Preferences* → *Network* → *Host-Only Networks* → *Add*.
- Start the VM. This will launch the network simulator into a Mininet VM console, prompting the user for login credentials. The console is displayed in Image 5.4.

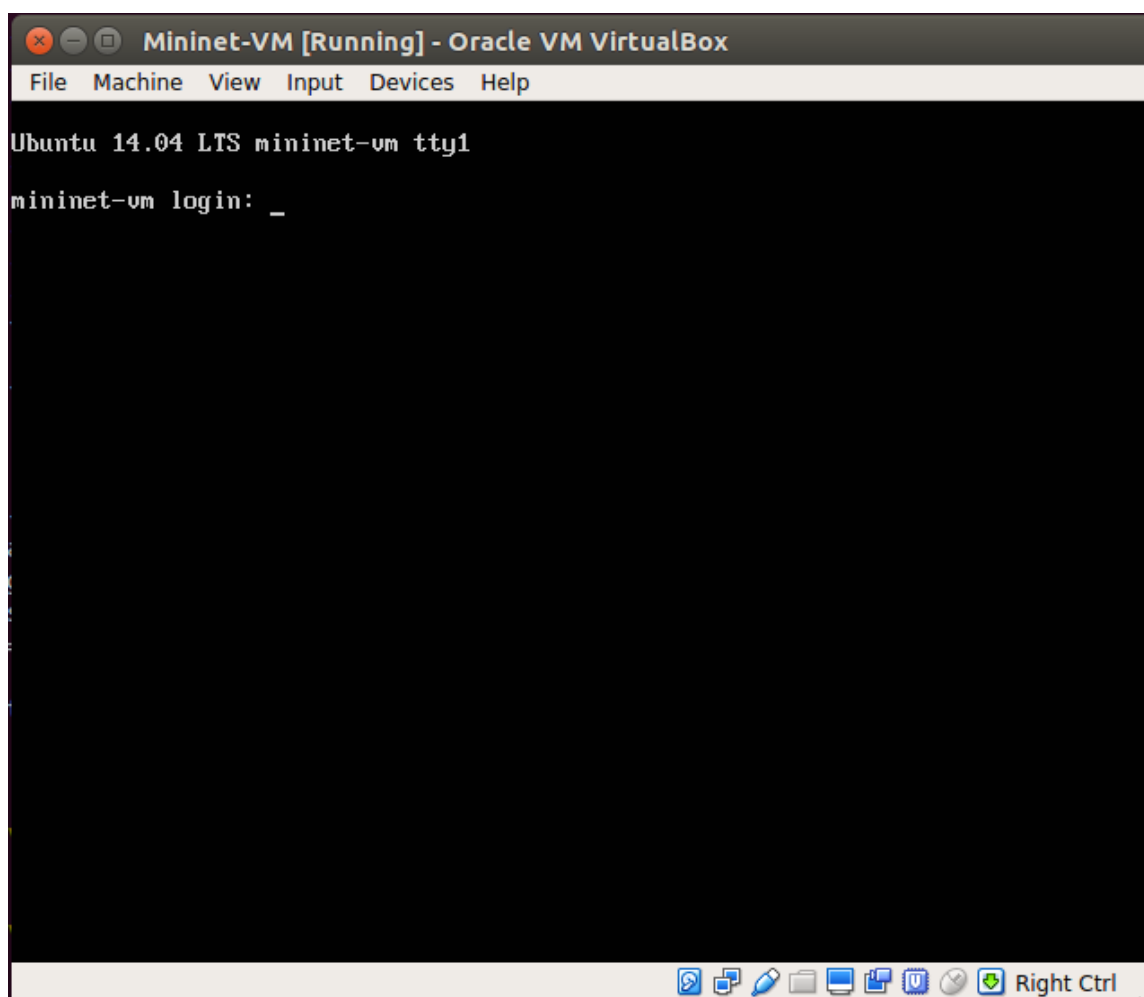


Image 5.4: Mininet login console

The credentials are *mininet* for both username and password. It would be useful to note that the user cannot paste commands in the Mininet console, therefore a proposed solution is to run the command:

```
ssh mininet@[MININET_VM_IP]
```

in a terminal, and connect to Mininet directly from his environment. In order to find the Mininet VM's IP, the command `sudo dhclient eth1` followed by the command `ifconfig` could be used. [85], [86], [88].

Mininet contains an Open vSwitch (OVS) version, which is used to instantiate the SDN switches. In the context of the current thesis, version 2.4.0 of OVS was used. The default OVS version in Mininet can be found to be 2.0.2 by typing the command `sudo ovs-vsctl show`, therefore the following commands were used to switch version [87]:

- `sudo -s`
- `apt-get remove openvswitch-common openvswitch-datapath-dkms openvswitch controller openvswitch-pki openvswitch-switch`
- `cd /root`
- `wget http://openvswitch.org/releases/openvswitch-2.4.0.tar.gz`
- `tar zxvf openvswitch-2.4.0.tar.gz`
- `cd openvswitch-2.4.0/`

- `./configure --prefix=/usr --with-linux=/lib/modules/`uname -r`/build`
- `make`
- `make install`
- `make modules_install`
- `rmmod openvswitch`
- `depmod -a`
- `/etc/init.d/openvswitch-controller stop`
- `update-rc.d openvswitch-controller disable`
- `/etc/init.d/openvswitch-switch start`

5.3.1 Mininet default topology

After the user has finished the installation of both ODL Controller and Mininet, the topology creation in Mininet and connection to ODL is possible.

The default topology can be started if the user types in Mininet console the following command:

```
sudo mn
```

The default topology is the minimal topology, which includes one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller.

- The nodes can be displayed with the command `nodes`
- The links can be displayed with the command `net`
- The command `dump` shows information about all the topology nodes
- The Mininet Command Line Interface (CLI) built-in command `pingall` tests the connectivity between all nodes in pairs.
- In order to exit the topology created, the command `exit` is used
- When the command `sudo mn -c` is typed in the Mininet VM console, it is automatically cleaned up [88].

5.3.2 Changing topology size and type

As already mentioned in 5.3.1, Mininet's default topology consists of a single switch connected to two hosts. One has the ability to change this to a different topology, adding the argument `--topo`, and specifying the parameters for the topology's creation. For example, the parameter "*single*", as used in the following command:

```
sudo mn --topo single,[SWITCH_NUMBER]
```

indicates a topology with *SWITCH_NUMBER* switches and a single host, whereas using the parameter *linear* instead:

```
sudo mn --topo linear,[SWITCH_NUMBER]
```

indicates a linear topology with *SWITCH_NUMBER* switches, where each switch has one host and all switches connect in a line [88].

5.3.3 Custom topologies

Custom topologies can be easily defined using a Python API and passing it as an argument for the topology's creation. An example which connects two switches directly, with a single host off each switch, is presented in Image 5.5. The Python script can be given as parameter to Mininet using the following command [88]:

```
sudo mn --custom [PATH_TO_PYTHON_SCRIPT] --topo mytopo
```

```

11  from mininet.topo import Topo
12
13  class MyTopo( Topo ):
14      "Simple topology example."
15
16      def __init__( self ):
17          "Create custom topo."
18
19          # Initialize topology
20          Topo.__init__( self )
21
22          # Add hosts and switches
23          leftHost = self.addHost( 'h1' )
24          rightHost = self.addHost( 'h2' )
25          leftSwitch = self.addSwitch( 's3' )
26          rightSwitch = self.addSwitch( 's4' )
27
28          # Add links
29          self.addLink( leftHost, leftSwitch )
30          self.addLink( leftSwitch, rightSwitch )
31          self.addLink( rightSwitch, rightHost )
32
33
34  topos = { 'mytopo': ( lambda: MyTopo() ) }

```

Image 5.5: Example Python script for a custom topology creation [88]

5.3.4 Using a remote controller in Mininet

The current section is primarily useful for a user who runs a controller running outside of the VM, such as on the VM host, or a different physical PC. In the case of the present thesis, the ODL controller has been deployed in the VM host, outside of the VM. The host IP has to be provided as an argument so that Mininet is connected to the remote controller. In order to run a custom topology connected to the ODL remote controller, the following command should be used:

```
sudo mn --custom [PATH_TO_PYTHON_SCRIPT] --topo mytopo --
controller=remote,ip=[CONTROLLER_IP]
```

For instance, using the python script shown in Image 5.5 and assuming that it has been saved in the `/home/mininet/` folder of the Mininet VM under the name `custom_topo.py`, the command to create this custom topology would be:

Therefore, by typing the command:

```
sudo mn --custom custom_topo.py --topo mytopo --
controller=remote,ip=[CONTROLLER_IP]
```

in Mininet's console, the output is shown in Image 5.6. The nodes connectivity can be tested using the command `pingall`, which gives the results presented in Image 5.7. Finally, one can see the topology inserted in the DLUX Topology tab, as depicted in Image 5.8.

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s1, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> █
```

Image 5.6: Mininet messages after starting a topology

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> █
```

Image 5.7: Mininet pingall result

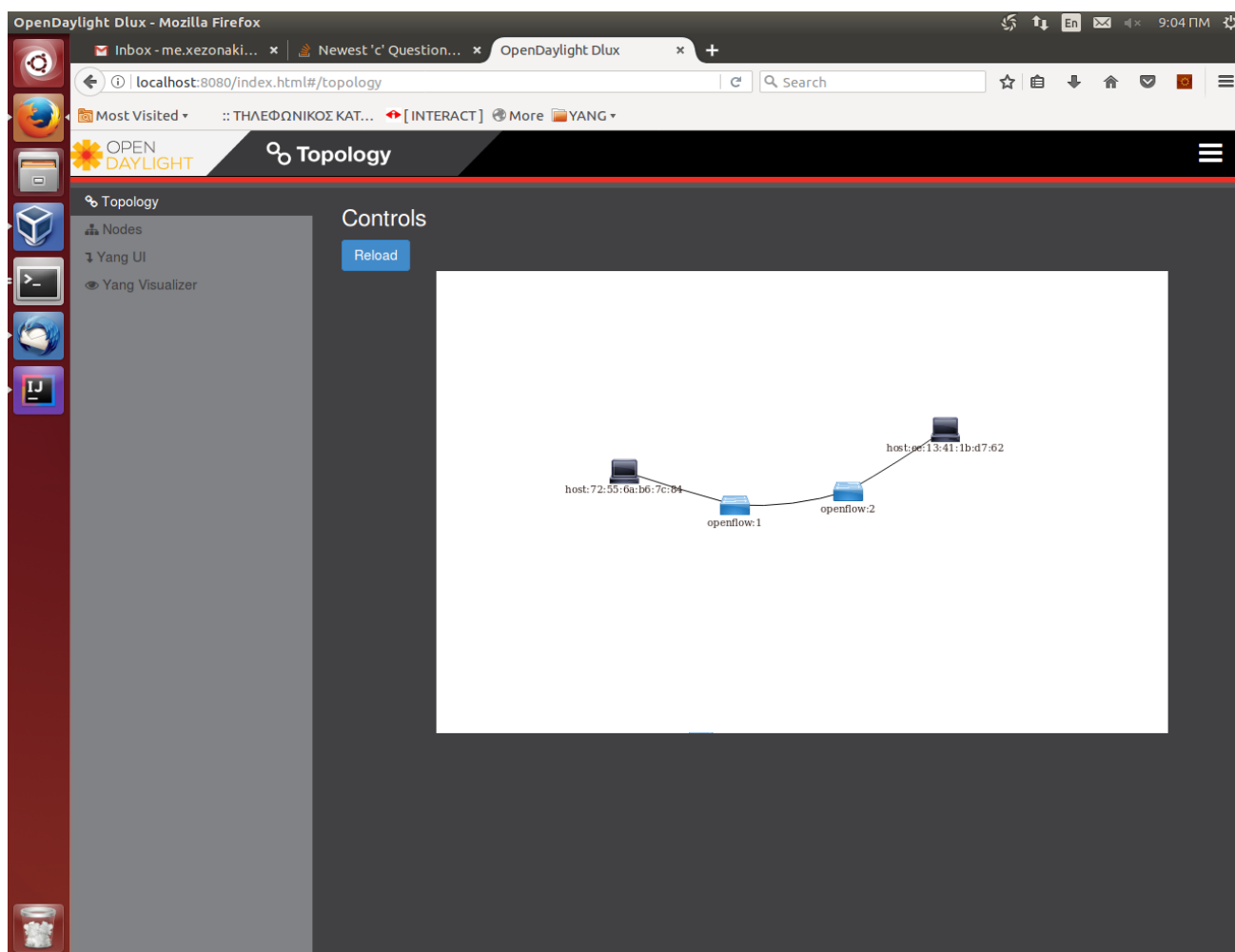


Image 5.8: The created Mininet topology shown in ODL DLUX UI

5.4 FIRST ODL PROJECT CREATION

Just downloading the ODL controller package as shown in 5.2 is not enough in order to create an ODL application. The download will help the user install the ODL controller and use it remotely, but he will not be able to include it in an ODL application. The first step one needs to perform in order to create his first ODL project is to update his Maven “settings.xml” file by using the following commands [89]:

```
cp -n ~/.m2/settings.xml{,.orig};
Wget -q -O -
https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml >
~/.m2/settings.xml
```

Then, a project can be created using Maven and an archetype called *opendaylight-startup-archetype* by typing:

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -
DarchetypeArtifactId=opendaylight-startup-archetype \
-DarchetypeRepository=http://nexus.opendaylight.org/content/repositories/<Snapshot-
Type>/\
```

where one needs to enter the proper *<Archetype-Version>* and *<Snapshot-Type>* that depend on the ODL release he will work on. For the current thesis and version Boron

SR1 the Snapshot-Type=**opendaylight.release** and Archetype-Version=**1.2.1-Boron-SR1** will be used.

Afterwards, the user will be expected to respond to prompts, in the current thesis case completed as follows:

- Define value for property '**groupId**':: org.opendaylight.sqmf
- Define value for property '**artifactId**':: sqmf
- Define value for property '**package**': org.opendaylight.sqmf:
- Define value for property '**classPrefix**':
- Define value for property '**copyright**'::

After completing the above mentioned steps, the archetype will have created a top level directory named $\${artifactId}$, for example, *sqmf/*. When entering the *sqmf/* directory, one can see the following contents:

- *api/*
- *features/*
- *impl/*
- *karaf/*
- *it/*
- *pom.xml*

and build the project using the command `mvn clean install -DskipTests`. Once the project has been built, an OpenDaylight distribution will have been created, which can be tested with the following steps:

```
cd karaf/target/assembly/bin
./karaf
```

During the build process a module called *sqmf* was built, which can now be verified on the console by checking out the log: `log:display | grep sqmf`. A log entry which includes the entry “**SqmfProvider Session Initiated**” will appear. To shutdown OpenDaylight, the command `shutdown -f` has to be used.

In order to understand where the log entry came from, one can navigate to the entry point in the *impl* submodule, and specifically in the *init* method of the class found in *impl/src/main/java/org.opendaylight/odlproject/impl/SqmfProvider.java*:

```
public void init() {
    LOG.info("SqmfProvider Session Initiated");
}
```

In order to create a new RPC or any storage structure, one should edit *api/src/main/yang/sqmf.yang* file [90].

5.5 USE OF AN IDE

As mentioned before, the ODL Controller is actually a Java project. Therefore, anyone wishing to create applications for the controller needs to install a Java IDE, where the

code will be developed and tested. This thesis makes use of the IntelliJ IDE, provided by JetBrains. Other well-known Java IDEs include Eclipse and Netbeans.

The SDN application of the current thesis started with the initial creation of a Maven project as described in 5.4, which was then imported into IntelliJ, following the procedure described below:

- Select to import a project into IntelliJ (Image 5.9).
- Select the project to import (Image 5.10).
- Select to import the project from an external model, specifying Maven as the external model (Image 5.11).
- Click on the button *Environment settings* and configure Maven's home directory, choosing its location (Image 5.12).
- Select the Maven project to import and click *Next* (Image 5.13).
- Click the green cross in order to configure a new JDK (Image 5.14).
- Select the JDK installation folder and click *OK* (Image 5.15).
- Click *OK* after noticing which the JDK resources are (Image 5.16).
- Enter a name for the project and click *Finish* (Image 5.17).

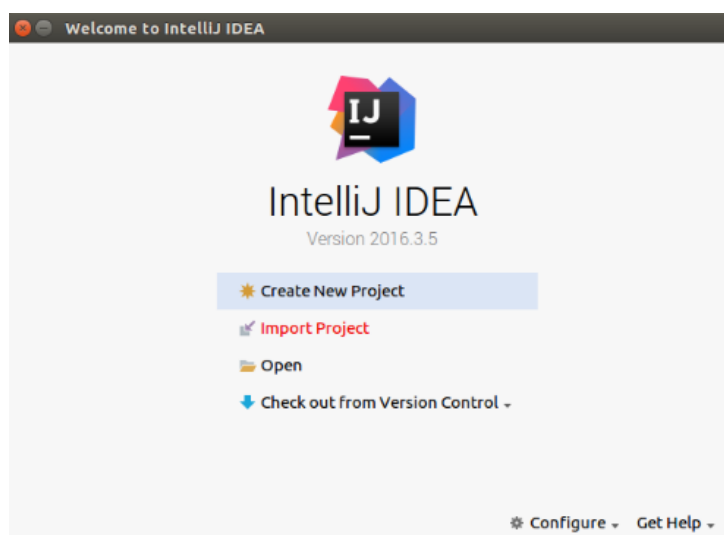


Image 5.9: Import a project into IntelliJ

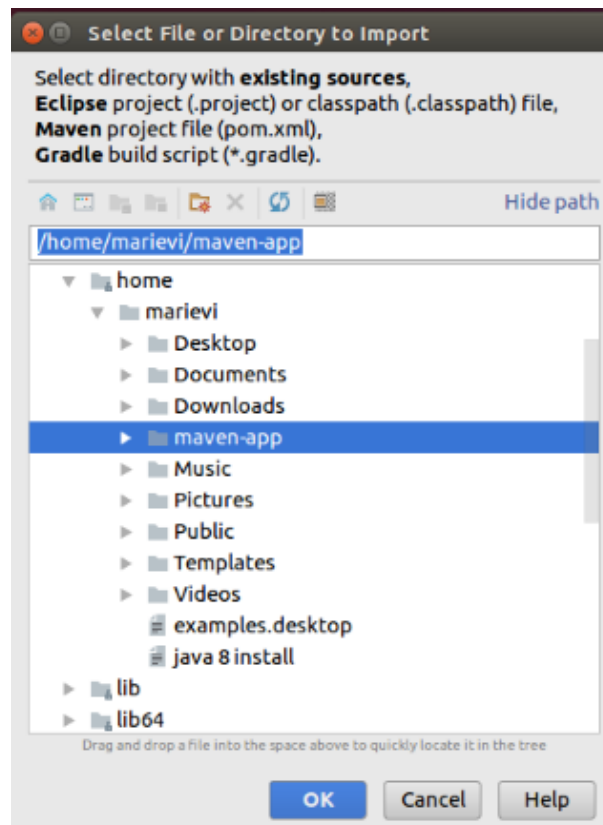


Image 5.10: Select the project to import

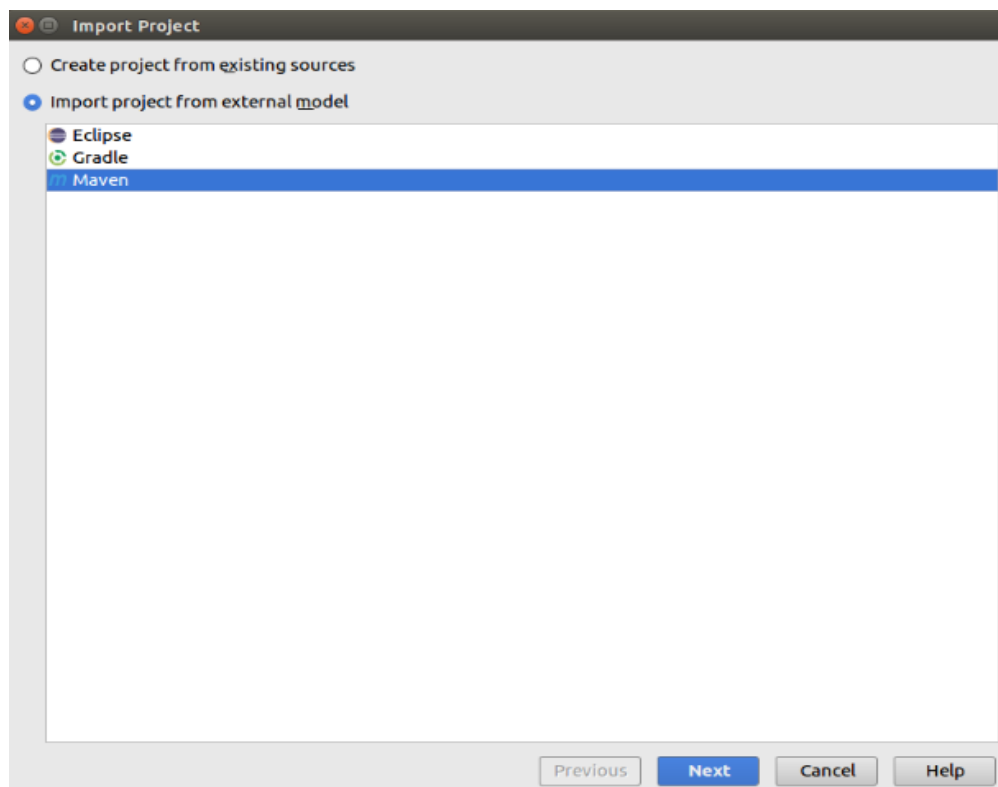


Image 5.11: Select to import a project from an external model, choosing Maven

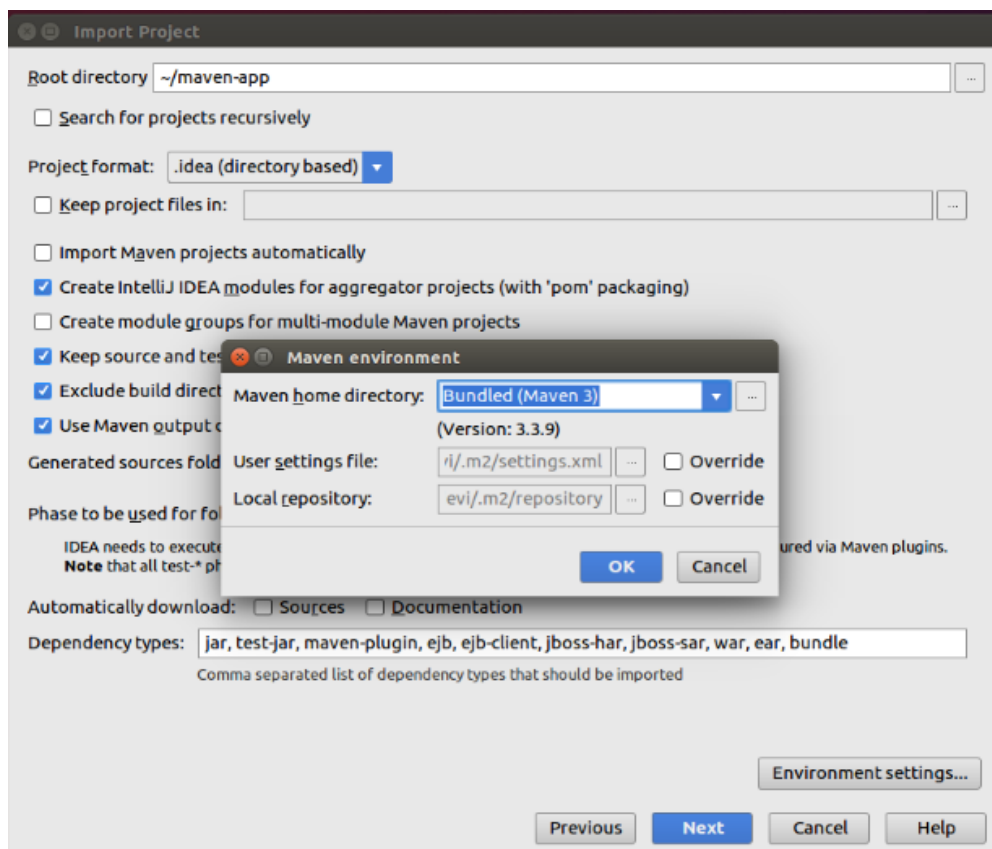


Image 5.12: Click on button *Environment settings* and configure Maven’s home directory

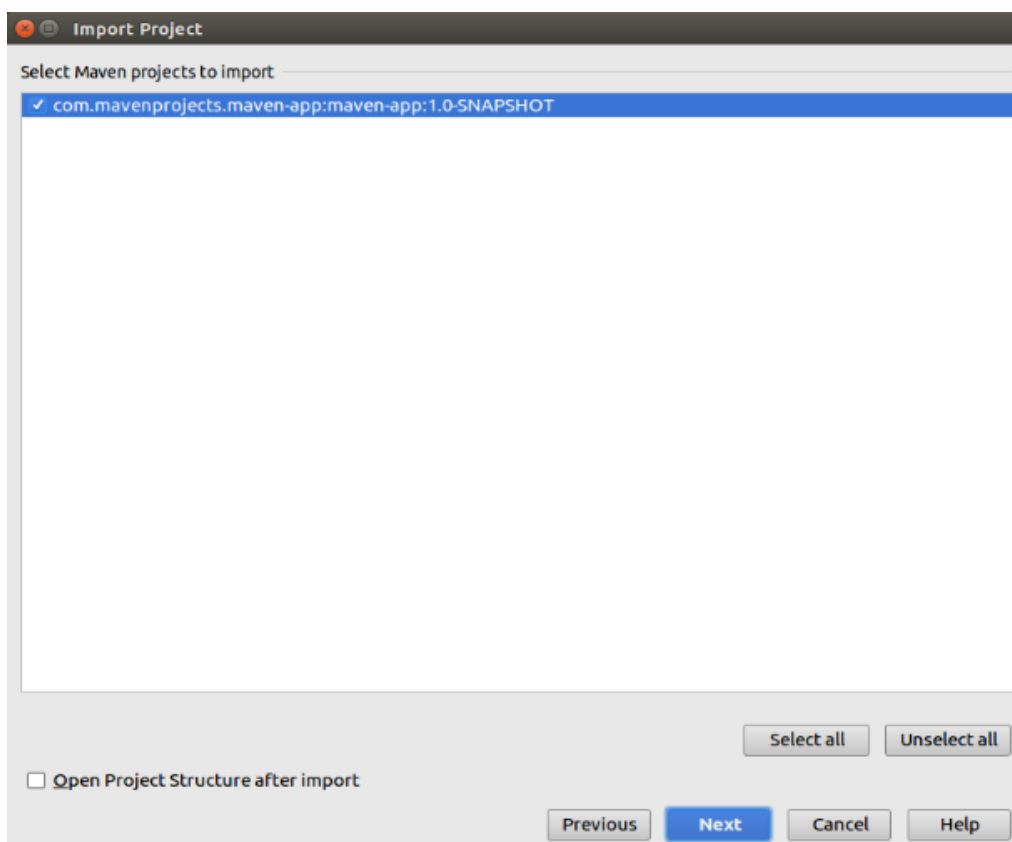


Image 5.13: Select Maven project to import and click *Next*

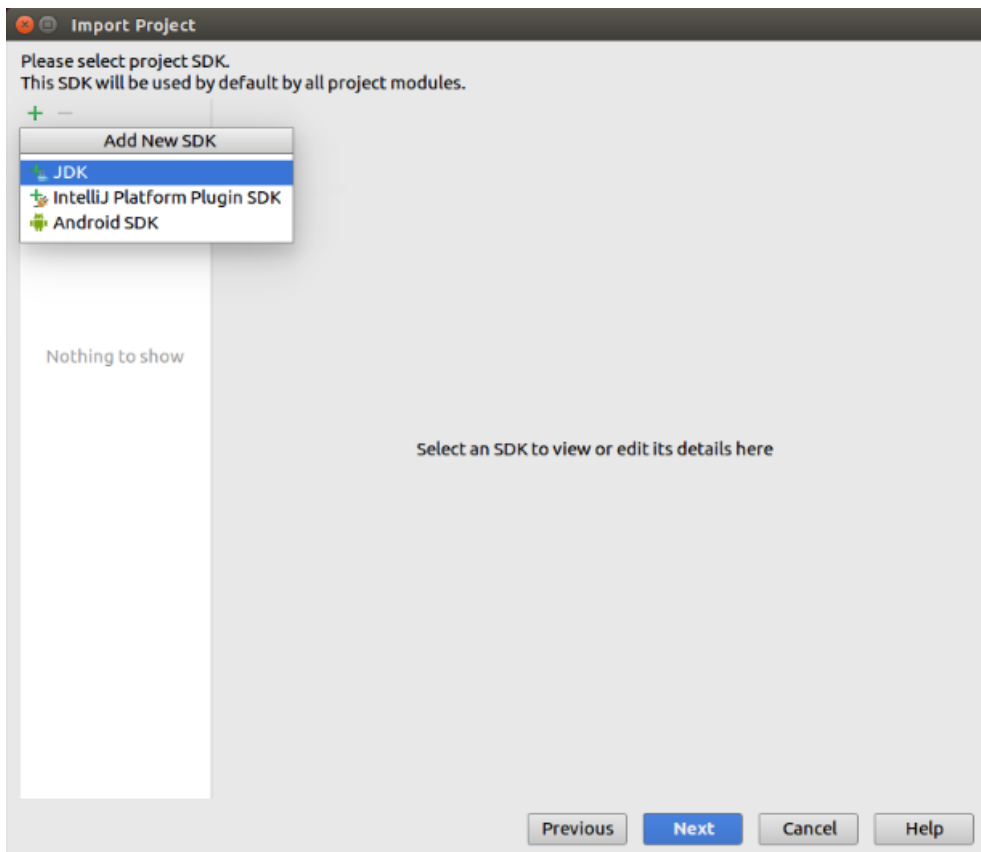


Image 5.14: Configure a new JDK

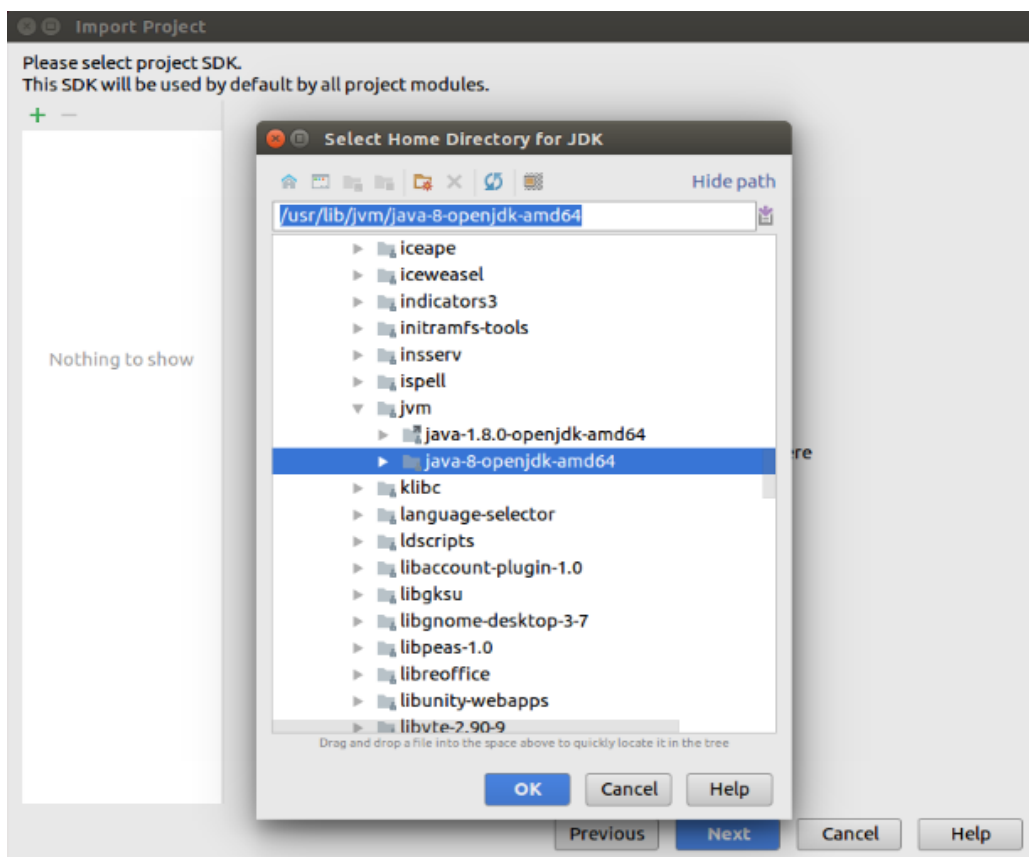


Image 5.15: Choose the JDK installation folder and click OK

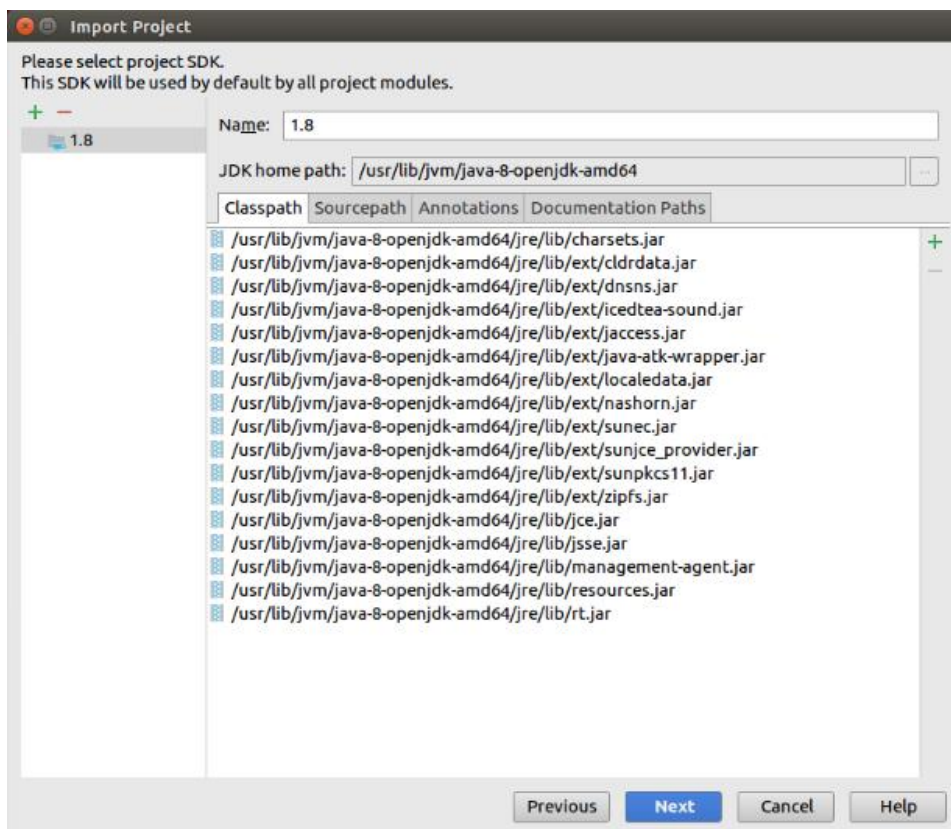


Image 5.16: The JDK resources

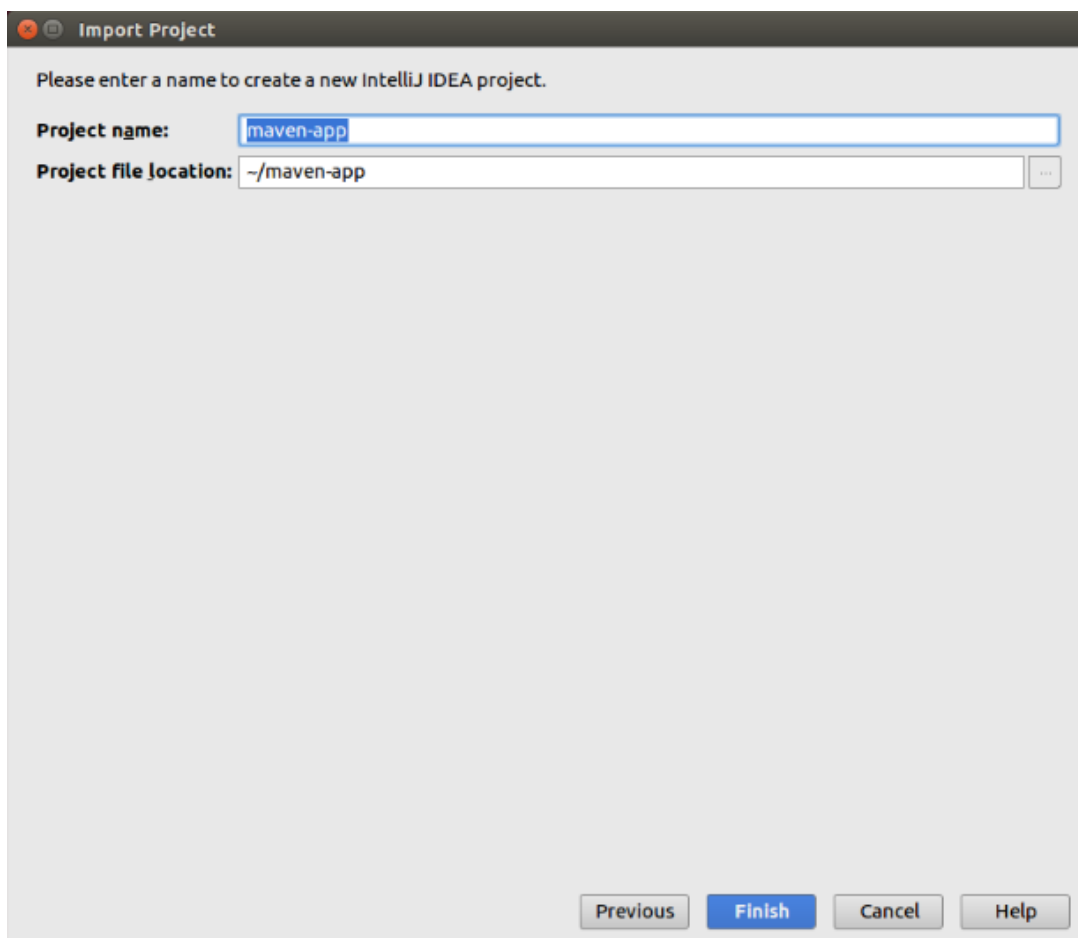


Image 5.17: Enter a name for the project and click *Finish*

5.6 VoIP TRAFFIC CREATION

One of the cases which the framework implemented in the current thesis addresses is the VoIP applications. In order to generate VoIP traffic in the network, the Distributed Internet Traffic Generator (D-ITG) was used in the scope of the current thesis. D-ITG is a platform capable to produce IPv4 and IPv6 traffic at packet level accurately, at network, transport, and application layer [91] by replicating the workload of current Internet applications. At the same time, D-ITG is also a network measurement tool able to measure the most common performance metrics (e.g. throughput, delay, jitter, packet loss) at packet level. It can generate traffic following stochastic models for packet size (PS) and inter departure time (IDT) that mimic application-level protocol behavior. By specifying the distributions of IDT and PS random variables, it is possible to choose different renewal processes for packet generation: by using characterization and modeling results from literature, D-ITG is able to replicate statistical properties of traffic of different well-known applications (e.g. Telnet, VoIP - G.711, G.723, G.729, Voice Activity Detection, Compressed RTP - DNS, network games) [92].

As reported in Image 5.18, the architecture of D-ITG comprises different components.

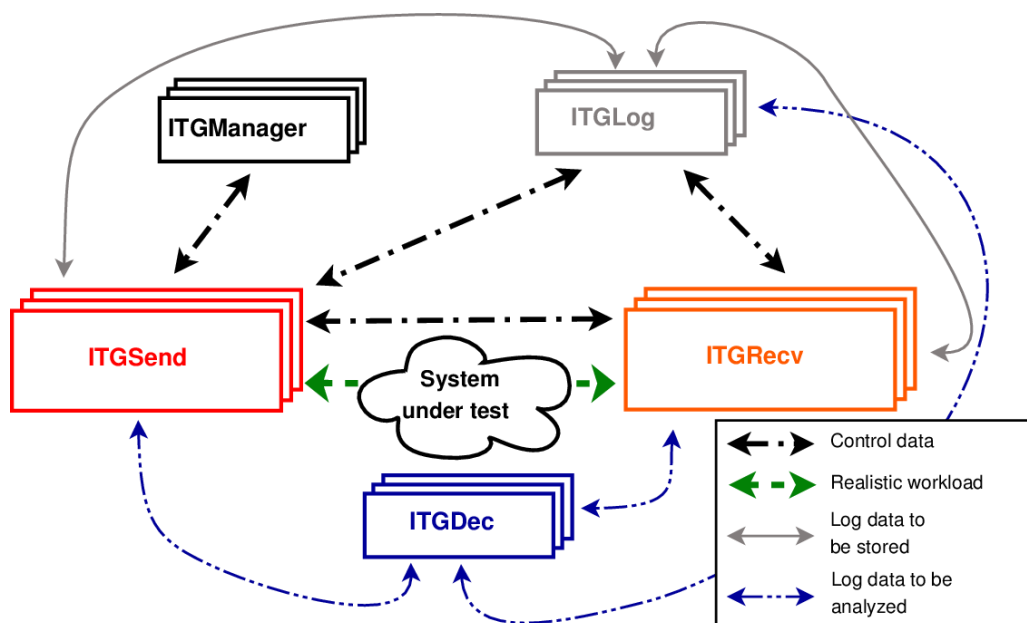


Image 5.18: D-ITG Architecture [92]

The core features of D-ITG are provided by *ITGSend* and *ITGRecv*. *ITGSend* is the component responsible for generating traffic toward *ITGRecv*. Exploiting a multithreaded design, *ITGSend* can send multiple parallel traffic flows toward multiple *ITGRecv* instances, and *ITGRecv* can receive multiple parallel traffic flows from multiple *ITGSend* instances. A signaling channel is created between each couple of *ITGSend* and *ITGRecv* components to control the generation of all the traffic flows between them.

ITGSend and *ITGRecv* can optionally produce log files containing detailed information about every sent and received packet. Such logs can be saved locally or sent - through the network - to the *ITGLog* component (useful to collect all the measures at a single point or in the case of hosts with limited storage capabilities e.g. sensors, embedded devices, smartphones, etc.). The *ITGDec* component is in charge of analyzing the log files in order to extract performance metrics related to the traffic flows.

The experiments (even large-scale ones) can be controlled from a single vantage point: the *ITGRecv* components act as daemons and can be completely configured and controlled by the *ITGSend* components that want to send traffic to them. Also, the

ITGSend components can act as daemons and can be remotely controlled through the D-ITG API. The *ITGManager* component represents an example of how to use the D-ITG API to remotely control *ITGSend*. This way, the user can completely control a large-scale distributed experiment from a single vantage point.

- **ITGSend:** The *ITGSend* component is responsible for generating traffic flows and can work in three different modes:
 - Single-flow - read the configuration of the single traffic flow to generate toward a single *ITGRecv* instance from the command line.
 - Multi-flow - read the configuration of multiple traffic flows to generate toward one or more *ITGRecv* instances from a script file. The script is made of a line for each traffic flow, which includes a set of command-line options as in the single-flow mode.
 - Daemon - run as a daemon listening on a UDP socket for instructions and can be remotely controlled using the D-ITG API.

Every traffic flow generated is described by two stochastic processes relating to PS and IDT, through which well defined traffic profiles can be generated, emulating application protocols such as VoIP, DNS, etc. PS and IDT series can also be loaded from a file for each flow. *ITGSend* can log information about every sent or received packet, when running in One Way or Round Trip mode respectively. In the first case, timestamps (and other information) of sent packets are stored, while in the second case, timestamps (and other information) of sent and received packets are stored. For each flow the source IP address can be specified, which is useful for multi-homed hosts.

- **ITGRecv:** The *ITGRecv* component is responsible for receiving multiple parallel traffic flows generated by one or more *ITGSend* instances. It normally runs as a multi-threaded daemon listening on a TCP socket for incoming traffic reception requests. Each time a request is received from the network, a new thread is created, which performs all the operations related to the new request (e.g. receiving the packets of the flow). The port numbers on which *ITGRecv* will receive each flow and any logging activity required on the receiver side can be remotely controlled by *ITGSend*. A specific signaling protocol, the Tunnel Setup Protocol (TSP), allows *ITGRecv* and *ITGSend* to properly setup and manage the traffic generation process.
- **ITGLog:** The *ITGLog* component is responsible for receiving and storing log information possibly sent by *ITGSend* and *ITGRecv*. It runs as a multi-threaded daemon listening on a TCP socket for incoming log requests. Log information is received over TCP or UDP protocols on port numbers dynamically allocated in the range 9003-10003.
- **ITGDec:** The *ITGDec* component is responsible for decoding and analyzing the log files stored during the experiments conducted by using D-ITG. *ITGDec* parses the log files generated by *ITGSend* and *ITGRecv* and calculates the average values of bitrate, delay and jitter either on the whole duration of the experiment or on variable-sized time intervals. *ITGDec* analyzes the log files produced by *ITGSend*, *ITGRecv*, and *ITGLog* in order to produce results about each flow and about the whole set of flows [92].

One can install the D-ITG tool inside the Mininet VM using the following steps:

- Login into Mininet VM.

- `sudo apt-get install unzip`
- `sudo apt-get install g++`
- `wget http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip`
- `unzip D-ITG-2.8.1-r1023-src.zip`
- `cd D-ITG-2.8.1-r1023/src`
- `make`

In order to be familiarized with this platform, below are presented, as an example of D-ITG use, the steps followed to generate VoIP traffic in a simple case of a topology with two hosts, after the successful installation of D-ITG. The traffic was decided to be sent from h_1 (sender) to h_2 (receiver).

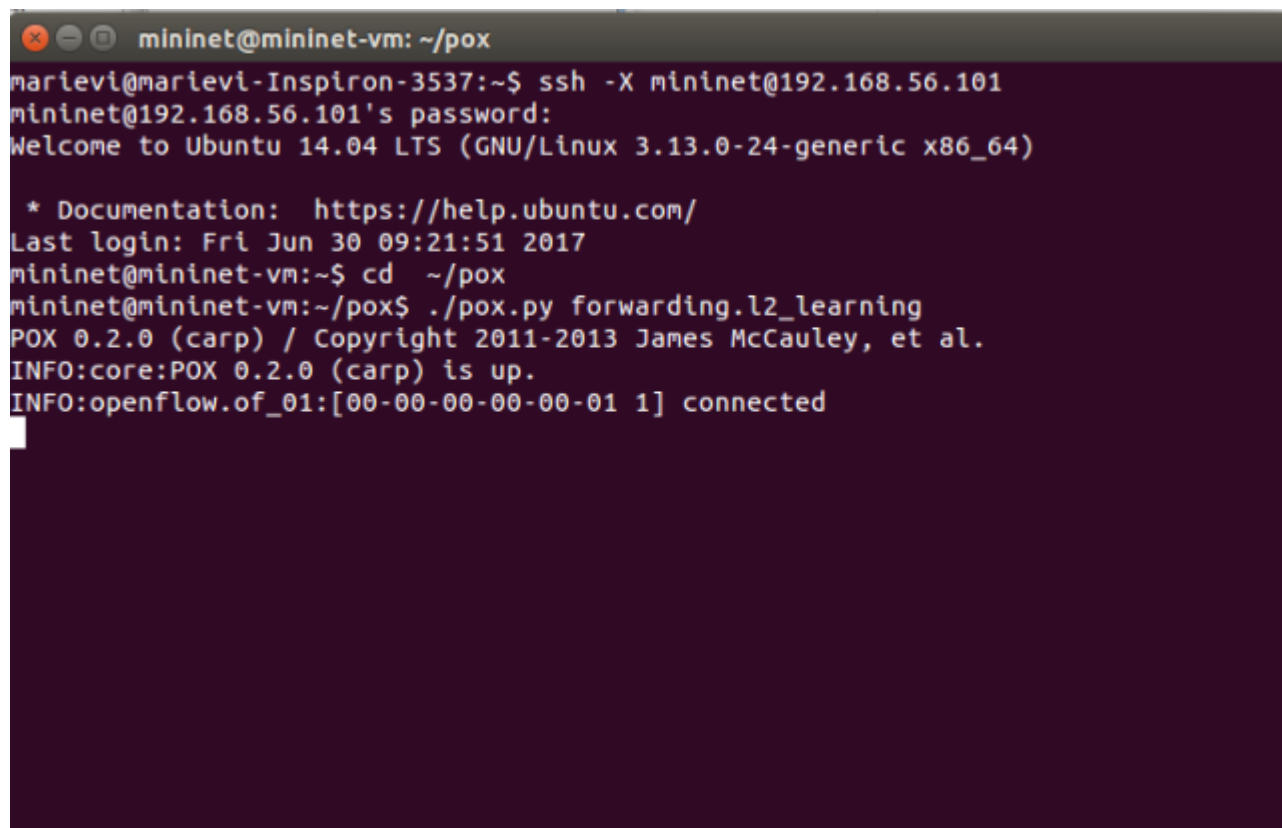
- First, a Mininet session was opened and the commands `cd ~/pox` and `./pox.py forwarding.l2_learning` were given for the controller setup, as shown in Image 5.19.
- Then, a second Mininet session was opened and the command `sudo mn --controller=remote,ip=127.0.0.1,port=6633` was given, as Image 5.20 illustrates.
- As a next step, a console for each of the two hosts was opened separately using the command `xterm h1 h2`.
- After both hosts' consoles opened, the command `cd D-ITG-2.8.1-r1023/bin` was used for both h_1 and h_2 , as shown in Image 5.21.
- The command `./ITGRecv -l receiver_file` was executed in the receiver's console, where `-l` flag enables logging to the specified file and specifically it generates a log file containing timing, ordering and size information about every received packet (Image 5.22).
- A script was created in the sender with the following commands, containing the necessary information about the generation of the VoIP traffic, as depicted in Image 5.23.

```
cat > script <<END
-a 10.0.0.2 -rp 10001 VoIP -x G.711.2 -h RTP -VAD
END
```

where:

- `-a` option sets the destination address of the flow's packets
 - `-rp` option sets the destination port of the flow's packets
 - **VoIP** option emulates VoIP traffic
 - `-x` option is a VoIP sub-option indicating the audio codec
 - `-h` option is a VoIP sub-option indicating the audio transfer protocol
 - `-VAD` option is a VoIP sub-option indicating that voice activity detection is enabled.
- Finally, the command `./ITGSend -l sender_file` was executed in the sender's console, where `-l` flag enables logging to the specified file and specifically it generates a log file containing timing, ordering and size information about every sent packet (Image 5.24).

After having completed the VoIP traffic generation from h_1 to h_2 , the results which occurred and were stored in the log files were decoded. The receiver's results are presented in Image 5.25 and the sender's in Image 5.26 [92], [93].



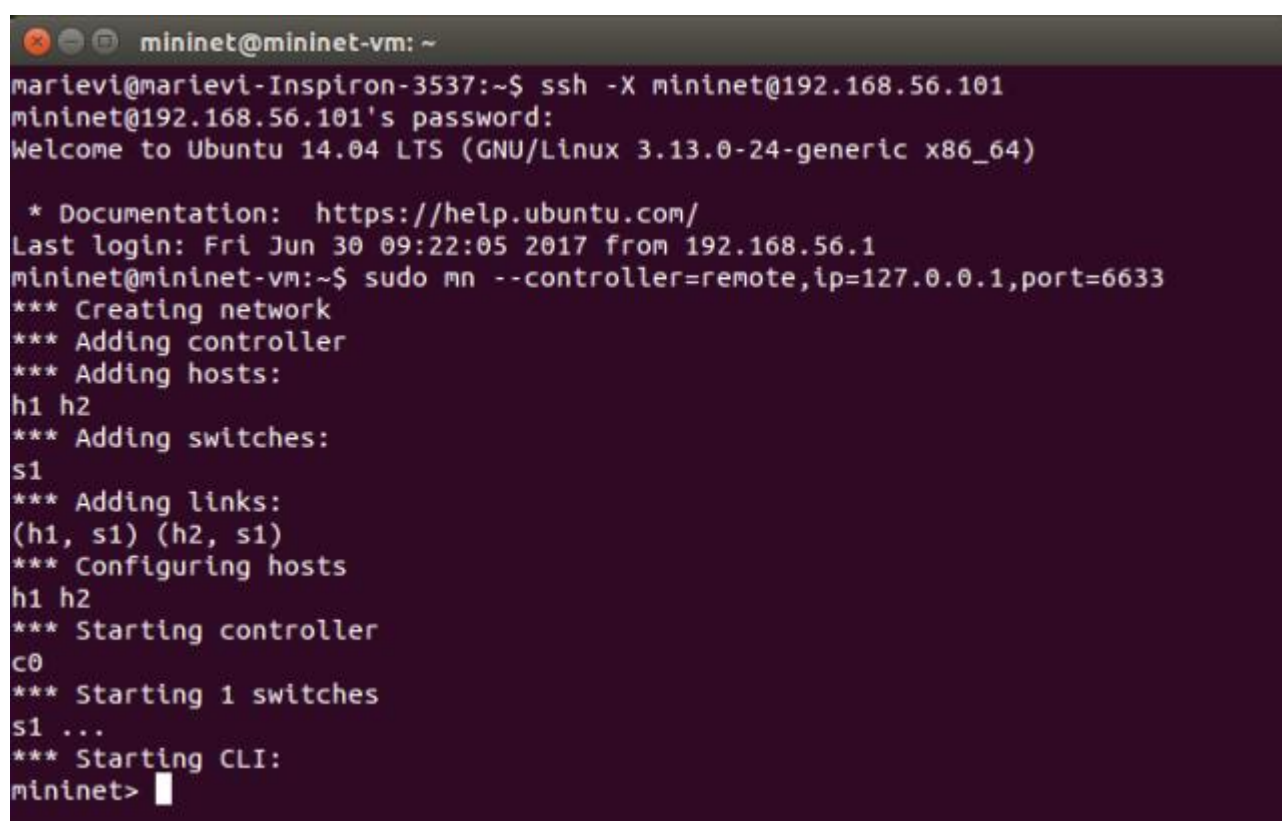
```

mininet@mininet-vm: ~/pox
marievi@marievi-Inspiron-3537:~$ ssh -X mininet@192.168.56.101
mininet@192.168.56.101's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Jun 30 09:21:51 2017
mininet@mininet-vm:~$ cd ~/pox
mininet@mininet-vm:~/pox$ ./pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected

```

Image 5.19: First Mininet session for using D-ITG



```

mininet@mininet-vm: ~
marievi@marievi-Inspiron-3537:~$ ssh -X mininet@192.168.56.101
mininet@192.168.56.101's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Jun 30 09:22:05 2017 from 192.168.56.1
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
*** 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 ...
*** Starting CLI:
mininet>

```

Image 5.20: Second Mininet session for using D-ITG, with topology deployment

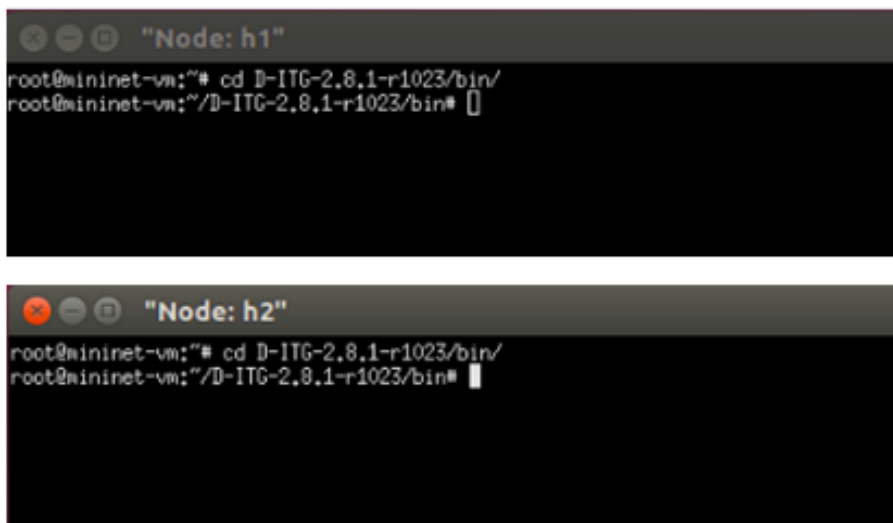


Image 5.21: Navigation in the bin folder of the D-ITG installation for both hosts

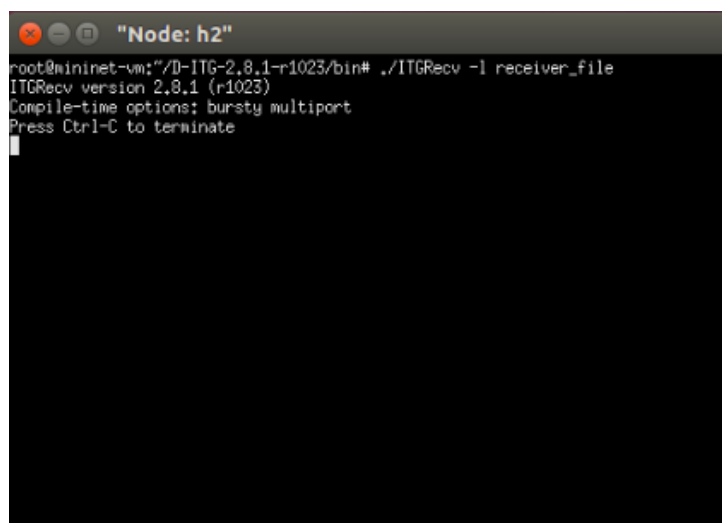


Image 5.22: The command starting the receiver host (h₂)

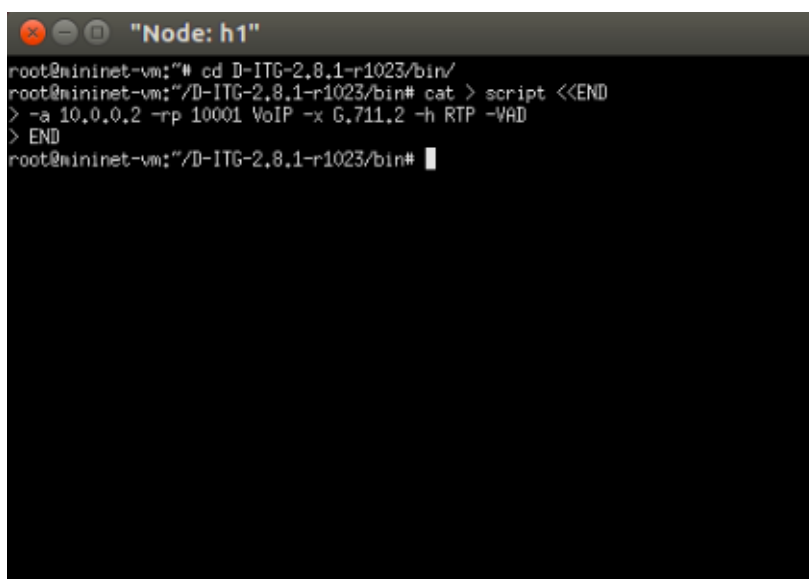


Image 5.23: The commands for the creation of script describing the traffic characteristics, in the sender host (h₁)

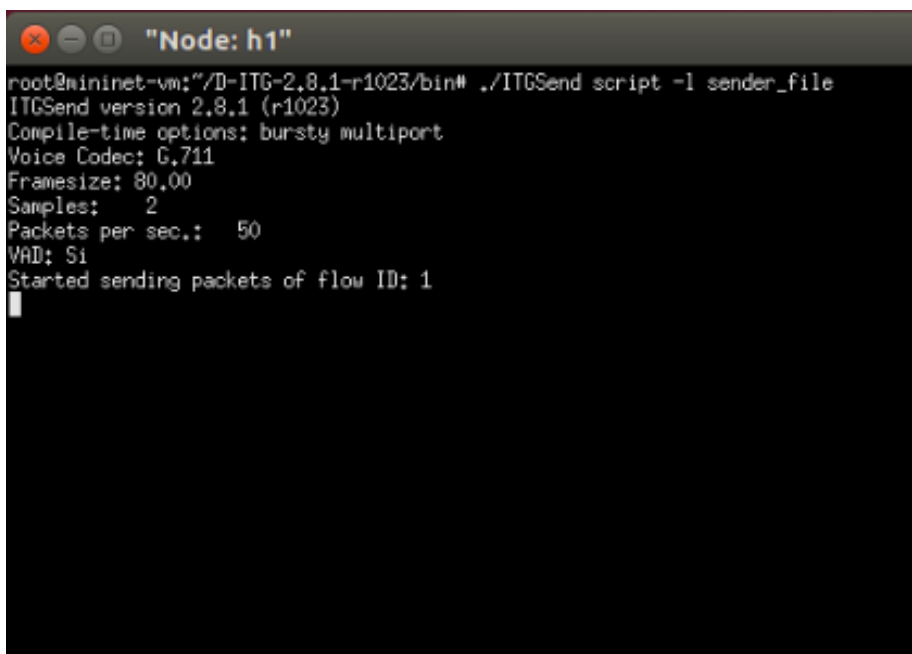


Image 5.24: The command starting the sender host (h₁)

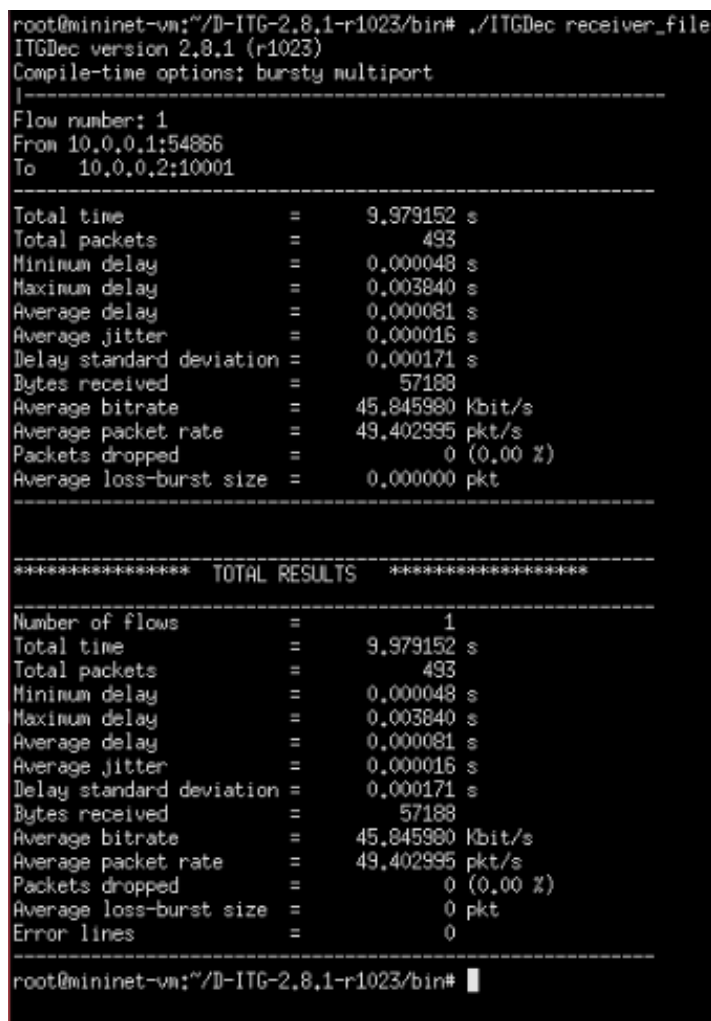


Image 5.25: The receiver host's log file, with information about the received packets

```

root@mininet-vn:~/D-ITG-2.8.1-r1023/bin# ./ITGDec sender_file
ITGDec version 2.8.1 (r1023)
Compile-time options: bursty multiport
-----
Flow number: 1
From 10.0.0.1:54866
To 10.0.0.2:10001
-----
Total time           = 9.982917 s
Total packets       = 493
Minimum delay       = 0.000000 s
Maximum delay       = 0.000000 s
Average delay       = 0.000000 s
Average jitter      = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received      = 57188
Average bitrate     = 45.828689 Kbit/s
Average packet rate = 49.394363 pkt/s
Packets dropped     = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
***** TOTAL RESULTS *****
-----
Number of flows     = 1
Total time         = 9.982917 s
Total packets      = 493
Minimum delay      = 0.000000 s
Maximum delay      = 0.000000 s
Average delay      = 0.000000 s
Average jitter     = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received     = 57188
Average bitrate    = 45.828689 Kbit/s
Average packet rate = 49.394363 pkt/s
Packets dropped    = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines        = 0
-----
root@mininet-vn:~/D-ITG-2.8.1-r1023/bin# █

```

Image 5.26: The sender host's log file, with information about the sent packets

5.7 VIDEO TRAFFIC CREATION

Apart from the VoIP applications, the framework implemented in the current thesis also addresses the case of video applications. Therefore, it was necessary to generate video traffic in the network and this can be achieved with the following steps:

- A media player must be installed inside Mininet, so that the video file can be played. The current thesis uses VLC media player, which can be installed using the command `sudo apt-get update` followed by `sudo apt-get install vlc`.
- The sender host must be equipped with a video file to stream. A sample mp4 video file was transferred from the host machine to the Mininet VM using the command `scp [FILENAME] mininet@[MININET_VM_IP]:[DESTINATION_PATH]`. This way the sender host is also granted access to the video file and therefore can stream it to the receiver host.
- By opening a terminal for the sender host using the command `xterm [HOST_NAME]`, the sender host can stream the video file using the command `vlc-wrapper [VIDEO_FILENAME] --sout '#rtp{dst=[RECEIVER_HOST_IP],port=1234}'`.

5.7.1 Video adjustment to required specifications

As already mentioned in 3.3.2, the video QoE estimation formula contains 12 coefficients which are derived from the video characteristics. The current thesis covers

the five cases given in Table 3.2 and their respective coefficients in Table 3.3. Therefore, in order to create video traffic the video file to be streamed must be adjusted to the required configurations. Example steps are described below to adjust a video to the requirements of case #5 of Table 3.2.

- It would be useful to install *ffmpeg* as it easily shows a video's specifications, using the following commands on Ubuntu 14.04 [94]:
 - `sudo apt-get remove --purge ffmpeg`
 - `sudo apt-add-repository ppa:mc3man/trusty-media`
 - `sudo apt-get update`
 - `sudo apt-get install ffmpeg`
- A Youtube video must be selected for download.
- By clicking *Share* → *Copy*, the video's link is copied to the clipboard and it can be pasted to an online video downloader so that the video is downloaded. The tool used for the current thesis can be found in <https://www.onlinevideoconverter.com/video-converter>.
- In order to adjust the **video codec** and the **video format**, an online converter can be used. The tool used for the current thesis can be found in <https://video.online-convert.com/convert-to-mp4>. Once the video to be processed has been uploaded, the desired codec and format are specified. For the current thesis h264 codec and VGA format (640x480) were selected.
- In order to verify that the video has obtained the desired codec and format, the command `ffmpeg -i [VIDEO_PATH] -hide_banner` can be used, which will give an output like the one depicted in Image 5.27.

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'video.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder         : Lavf57.71.100
Duration: 00:01:52.08, start: 0.000000, bitrate: 1679 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 640x480 [
SAR 4:3 DAR 16:9], 1606 kb/s, 23.98 fps, 23.98 tbr, 24k tbn, 47.95 tbc (default)
Metadata:
  handler_name     : VideoHandler
Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, mono, fltp,
69 kb/s (default)
Metadata:
  handler_name     : SoundHandler
At least one output file must be specified
```

Image 5.27: The output of *ffmpeg* command

- In order to adjust the **video key frame interval to 1**, the command `ffmpeg -i [VIDEO_PATH] -qscale 0 -g 1 [OUTPUT_VIDEO_PATH]` can be used [95].
- In order to verify that the video has obtained the desired key frame interval, the command `ffprobe -show_frames [VIDEO_PATH] | grep key_frame` can be used, which will give an output like the one depicted in Image 5.28.

```
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1  
key_frame=1
```

Image 5.28: The output of ffprobe command

6. QoE DEGRADATION CASES EXAMPLES

In this chapter, various cases where QoE degradation can happen during multimedia applications streaming are presented by conducting experiments on a custom network topology and computing the QoE. The QoE degradation is observed due to limitations imposed by network conditions, as well as network instabilities such as link failures. The experiments aim to illustrate the plethora of the cases where the QoE in a network suffers from degradation due to network conditions, as well as point out the necessity for a QoE monitoring framework.

The first step was to create a custom network topology using Mininet, following the instructions of 5.3.3 and an initial ODL SDN controller as indicated in 5.4, making the connection between them as shown in 5.3.4. Subsequently, different network conditions were applied and both **VoIP and video traffic was generated inside the network from h_1 to h_2** , using the D-ITG tool as shown in 5.6 or streaming a video file as shown in 5.7, respectively.

- For the experiments on **VoIP** applications, traffic was created using D-ITG as shown in 5.6. The **G.107** E-model was used for the QoE evaluation. Each produced flow used the G.729.2 codec of VoIP, as required for the evaluation of the QoE using the ITU G.107 E-model. Also, due to the VoIP traffic type, D-ITG only allowed 50 packets per second to be sent (packet rate). The necessary delays and packet losses for the E-model were obtained from the D-ITG receiver's log file, which presented statistics at the end of the experiment, as described in 5.6.
- For the experiments on **video** applications, traffic was created by streaming a video as shown in 5.7. The **G.1070** E-model was used for the QoE evaluation. For these experiments the necessary packet losses for the ITU G.1070 E-model could not be obtained from a tool such as D-ITG, therefore they were computed using the video QoE monitoring functions created in the context of the current thesis, which will be described later in Chapter 7. The whole implementation's functionality was of course not used at this point. Only the parts computing the packet losses were used.

The packet losses and delays which occurred in each experiment due to the network conditions were used to compute the QoE values and construct relative graphical representations.

The created topology for the experiments is depicted in Image 6.1. It consists of 9 switches, s_1 to s_9 , connected linearly between them, as well as two hosts, h_1 and h_2 , connected to s_1 and s_8 respectively.

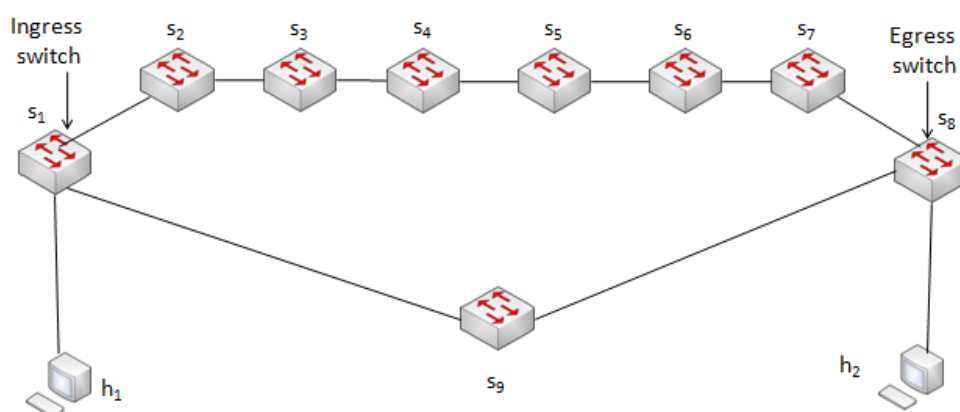


Image 6.1: Topology with nine switches and two hosts, used for the current thesis

6.1 EXPERIMENTS ON VoIP APPLICATIONS

6.1.1 Experiment N° 1: Manual network limitations on VoIP

The first experiment studies the impact of network limitations, such as packet losses in the links, on the QoE of a VoIP application. These limitations were set manually to the links. The parameters used for the current experiment are shown in Table 6.1. VoIP flows were generated in the network, each of which contained 10000 packets, used the G.729.2 model of VoIP and had a packet rate of 50 packets per second.

Table 6.1: Parameters used for experiment N° 1 on VoIP traffic generation

Packets per second	50
VoIP codec	G.729.2
Packets of flow	10000

After the topology creation and in each execution of the experiment, selected links were manually assigned a packet loss value in order to emulate the actual losses that a network may suffer from. Table 6.2 shows the R and MOS values computed per total network packet loss case and occurring delay.

Table 6.2: Delay, R and MOS according to the total network packet loss

Total Network Packet Loss (%)	Delay (sec)	R	MOS
1	0.000184	79.38317681	4.000472662
2.17	0.000173	75.34029544	3.836412202
3.17	0.000174	72.18156709	3.697577086
3.85	0.000173	70.16784241	3.604861916
5.01	0.000172	66.95060989	3.45092752
5.98	0.000062	64.44839811	3.327040486
7.05	0.000168	61.85336357	3.195478775
8.36	0.000168	58.89239631	3.04246389
9.26	0.000171	56.97808346	2.942379419
9.96	0.000170	55.55011288	2.867340439

The above presented data are summarized in Figure 6.1, where MOS decrease - and therefore QoE degradation - are depicted as a graphical representation. It can easily be observed that the higher the level of total packet loss in the network during the packet transmission period, the lower the MOS values, causing poor QoE to the VoIP participants.

VoIP Quality in relation to total network packet loss

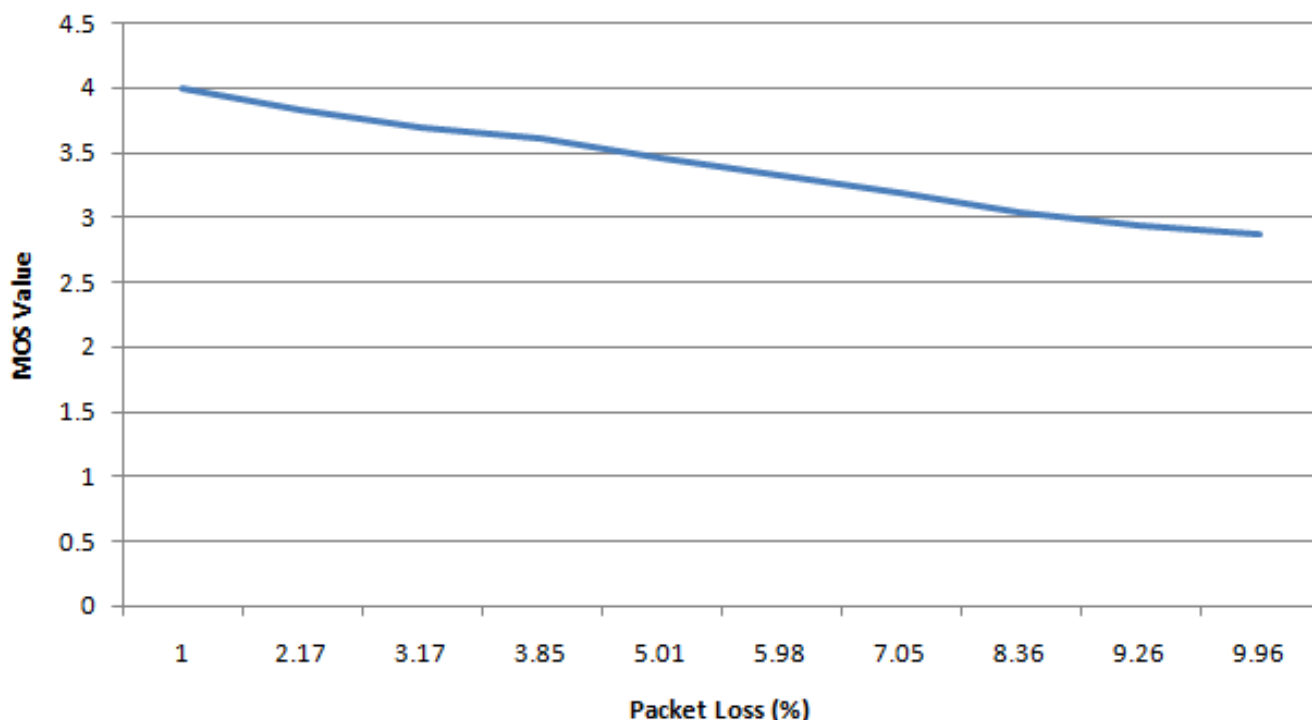


Figure 6.1: VoIP quality decrease in relation to total network packet loss

6.1.2 Experiment N° 2: Multiple sources of traffic on VoIP

The second experiment expands the first one, involving also other sources of traffic in the network – apart from the created VoIP traffic – that can overload the network and cause packet losses. This means that instead of manually inserting packet losses in various links, other sources of traffic were used for additional traffic generation and therefore packet losses were caused naturally. The parameters used for the second experiment are presented in Table 6.1, only this time the number of packets per flow varies. The additional traffic was generated using **iperf**, which is a tool to produce traffic in the network, while the topology shown in Image 6.1 was used again.

More specifically, in each execution of the experiment iperf was constantly running on the background, generating TCP packets, as different numbers of VoIP packets were generated in parallel. h_2 was set to be an iperf server, using the command `iperf -s -p [PORT_NUMBER]` in a h_2 terminal while h_1 was set to be an iperf client generating TCP traffic, using the command `iperf -c [SERVER_ADDRESS] -p [PORT_NUMBER] -t [TIME]` in a h_1 terminal. As port number, 5566 was used and the server address was h_2 's address, 10.0.0.2. After the traffic generation with iperf, the VoIP traffic was also generated with the way presented in 5.6.

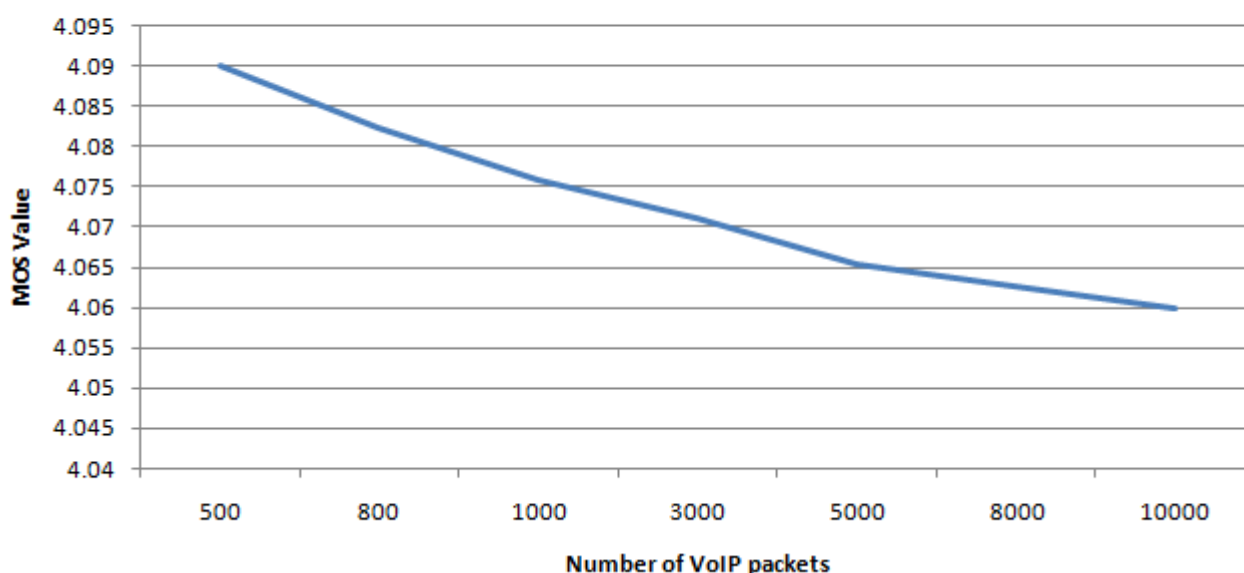
Table 6.3 shows the computed delay, packet loss and corresponding R and MOS values according to the number of VoIP packets sent in each execution of the experiment.

Table 6.3: Delay, Packet Loss, R and MOS according to number of VoIP packets sent simultaneously with iperf

Number of VoIP packets sent	Delay (sec)	Packet Loss (%)	R	MOS
500	0.025556	0.2	81.79455091	4.089990139
800	0.026327	0.25	81.5804475	4.082314506
1000	0.025828	0.3	81.39777591	4.075722697
3000	0.026229	0.33	81.27181639	4.07115445
5000	0.027902	0.36	81.11566625	4.065465425
8000	0.027803	0.38	81.04089661	4.062731257
10000	0.027847	0.4	80.96284347	4.059870077

The presented data are graphically summarized in Figure 6.2, where it can easily be observed that as the number of VoIP packets sent simultaneously with the random TCP packets generated by iperf increases, the MOS factor gets lower, and therefore the QoE suffers from progressive degradation. It is noted that the total QoE degradation for this experiment is not larger than 0.4%. This is expected, as the experiment is not large-scale and intends to just sufficiently depict the impact of multiple sources of traffic on QoE. Therefore, the total network traffic load is not extreme and consequently the QoE does not face a dramatic degradation.

VoIP Quality in relation to generated number of packets simultaneously with TCP traffic generation

**Figure 6.2: VoIP quality decrease in relation to number of VoIP packets, in parallel with iperf**

6.2 EXPERIMENTS ON VIDEO APPLICATIONS

6.2.1 Experiment N° 1: Manual network limitations on video

This experiment is the same as 6.1.1, but now conducted with video traffic. It studies the impact of network limitations, such as packet losses, on the QoE of a video application. These limitations were set manually to the links. The video parameters used for the current experiment are shown in Table 6.4. The created topology for the experiments is depicted in Image 6.1.

Table 6.4: Parameters used for experiments on video streaming

Frame rate	23.98 frames/sec
Bit rate	1679000 bits/sec
Video codec	H264
Video format	VGA (640x480)
Video key frame interval	1

After the topology creation and in each execution of the experiment, selected links were manually assigned a packet loss value in order to emulate the actual losses that a network may suffer from. Then, the video started being streamed as shown in 5.7. Table 6.5 shows the Vq value computed per total packet loss case.

Table 6.5: Vq according to the total network packet loss

Total Network Packet Loss (%)	Vq
1.0265242	4.410821547
2.3940452	4.348032088
2.8251303	4.328479531
3.8654728	4.281762134
5.3094906	4.218002119
6.0782496	4.184564713
6.946294	4.147226316
8.2307905	4.092776194
9.1012839	4.056412297
9.9193799	4.022627062

The above presented data are summarized in Figure 6.3, where Vq decrease - and therefore QoE degradation - are depicted as a graphical representation. It can easily be observed that the higher the level of total packet loss in the network during the packet transmission period, the lower the MOS values.

Video Quality in relation to total network packet loss

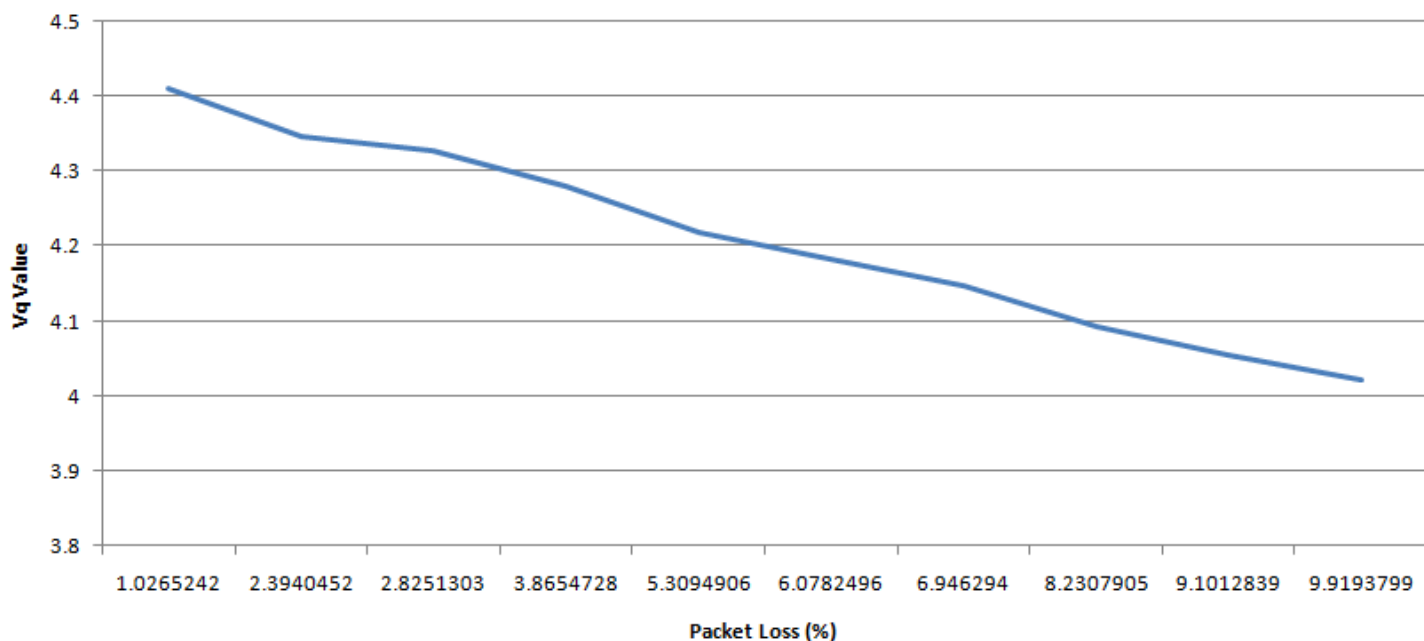


Figure 6.3: Video quality decrease in relation to total network packet loss

6.2.2 Experiment N° 2: Multiple sources of traffic on video

This experiment is the same as 6.1.2, but now conducted with video traffic. It involves other sources of traffic in the network as well – apart from the created video traffic – that can overload the network and cause packet losses. Therefore, additional TCP traffic was generated using iperf as a video session with duration of 115 seconds was being streamed in parallel, while the topology shown in Image 6.1 was used again. The parameters used for the second experiment are presented in Table 6.4.

Table 6.6 shows the computed packet loss and corresponding Vq value during the video streaming period. Results are reported every 5 seconds.

Table 6.6: Packet loss and Vq during video streaming simultaneously with iperf

Time (sec)	Packet Loss (%)	Vq
5	0.03235294117647059	4.309764233651093
10	0	4.4584992730265105
15	0.009302325581395349	4.415062278263543
20	0	4.4584992730265105
25	0.006385696040868455	4.4286223779326495
30	0.18448023426061494	3.691725039366696
35	0.01217391304347826	4.401763984787847
40	0.22203098106712565	3.557837641420684
45	0	4.4584992730265105
50	0.213768115942029	3.5867155645146807
55	0	4.4584992730265105

60	0.22790697674418606	3.5374979600768506
65	0.025500910746812388	4.340721435447504
70	0.10323886639676114	4.005862534614934
75	0.04468412942989214	4.254773299818181
80	0.11788617886178862	3.946633464423855
85	0.053763440860215055	4.214868984433865
90	0.04251386321626617	4.264384915690982
95	0.13070539419087138	3.8957551058669897
100	0.05411255411255411	4.213344409404942
105	0.07098765432098765	4.140506653668701
110	0.15007215007215008	3.820551352876892
115	0.009259259259259259	4.41526211314836

The presented data are graphically summarized in Figure 6.4, where it can be observed that the simultaneous TCP packet generation by iperf affects the video quality of the streamed video, making it extremely instable with continuous fluctuations.

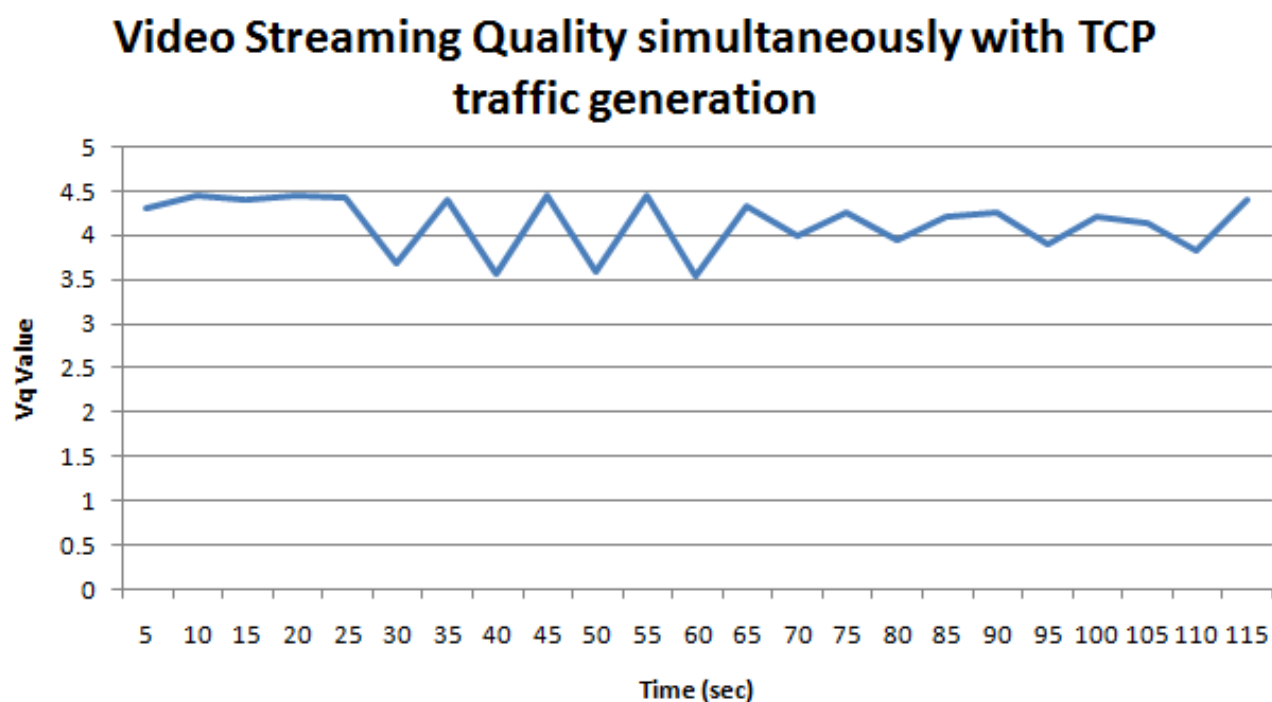


Figure 6.4: Video quality instability for video streaming simultaneously with using iperf

7. IMPLEMENTATION ANALYSIS

Chapter 6 presented cases where the QoE of a VoIP or video application can face degradation due to network conditions, e.g. packet losses due to a link failure or traffic load. This chapter presents the **SDN QoE Monitoring Framework (SQMF)**, an SDN framework implemented in order to overcome such situations and preserve the QoE in VoIP and video applications. This is achieved by using an SDN Controller and implementing extra functionality on top of it in order to change the traffic's transmission path to an alternative one, when QoE falls below a specified threshold.

The SDN Controller used for the current thesis is version Boron SR1 of ODL. The project was created following the instructions of 5.4 and extended the SDN Controller functionality by implementing an extra SDN module, named *sqmf*. The topology used for validation and experiments is the one depicted in Image 6.1.

7.1 DESIGN PRINCIPLES AND WORKFLOW

The approach used in order to ensure that the QoE remains in satisfactory levels is the periodical link monitoring and QoE estimation based on their statistics. In particular, the application computes the shortest path between the source and destination hosts, which will be the main transmission path, as well as the second shortest path (if exists) which will assist as a failover path. Then, rules are inserted to forward the traffic to the main path. Afterwards, the QoE monitoring process starts; the SDN controller periodically collects statistics from the switches (different statistics for each application type) and uses them to compute the QoE level. If the estimated value is lower than a specified threshold, then appropriate rules are inserted to redirect traffic to the failover path. The workflow described above for QoE Monitoring is graphically illustrated in Figure 7.1.

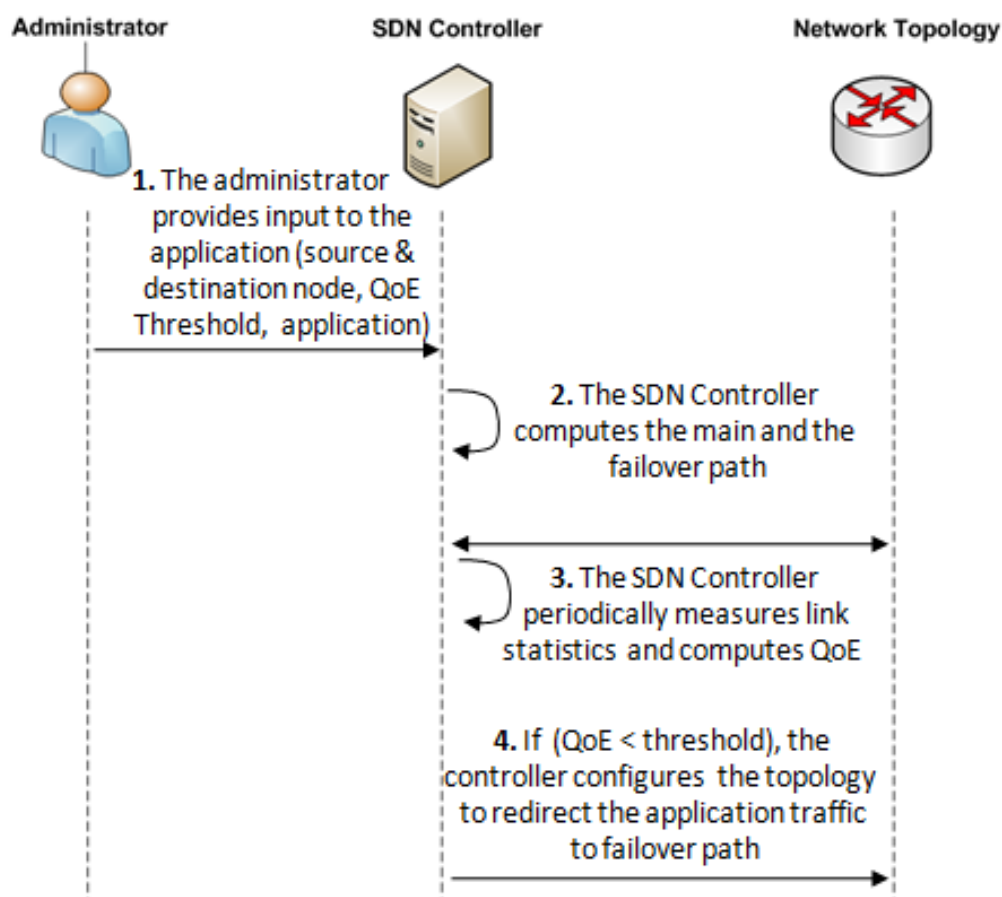


Figure 7.1: QoE Monitoring workflow

The collected statistics according to the streamed application type are:

- For VoIP applications, the **delay** and **packet loss** are necessary, in order to be able to use the G.107 E-model (3.3.1).
 - Each time the SDN controller needs to measure the delay, it creates a packet with a specific source MAC address – **00:00:00:00:00:09** in particular – and sends it on behalf of each switch of the path (except for the egress switch) to the output interface, so that the next switch of the path receives it. Each switch (except for the ingress switch, as it has no previous switch to receive a packet from) is configured with an appropriate flow rule to forward to the Controller any packet with the specific MAC address. The difference between the time that a switch receives a packet and the time that the previous switch had sent the packet is the delay of a particular link. The addition of all the path links' delays results in the path delay.

Each switch is configured with a rule of the following format:

```
priority=1000,dl_src=00:00:00:00:00:09 actions=CONTROLLER:65535
```

- In order to compute the packet loss rate, and given that VoIP traffic generates UDP packets, the SDN controller periodically monitors the number of UDP packets sent from the sender (h_1) and the number of UDP packets received by the receiver (h_2) and computes their difference divided by the number of sent packets.

To achieve packet loss monitoring, the ingress and the egress switches are configured appropriately so as to forward to the Controller – apart from the predefined output interface to the next node - any UDP packet they receive (the ingress receives UDP packets from the sender host and the egress from the previous path node). In its turn, the Controller counts the path's total incoming and outgoing UDP packets and is able to determine the packet loss. The ingress and the egress switch are configured with rules of the following format:

```
priority=1000,udp,in_port=x actions=CONTROLLER:65535,output:y
```

For example, the appropriate rules for s_1 and s_8 of Image 6.1 are:

```
 $s_1$ : priority=1000,udp,in_port=1 actions=CONTROLLER:65535,output:3
```

```
 $s_8$ : priority=1000,udp,in_port=3 actions=CONTROLLER:65535,output:2
```

- For video applications, the **bit rate**, **frame rate** and **packet loss** are necessary, in order to use the G.1070 E-model (3.3.2).
 - In order to compute the bit rate, the command `ffmpeg -i [VIDEO_PATH] -hide_banner` is executed through the Java code and the output is parsed until the bit rate value is accessed.
 - In order to compute the frame rate, the command `-i [VIDEO_PATH] -hide_banner` is executed through the Java code and the output is parsed until the frame rate value is accessed.
 - The packet loss is computed in the same way as for VoIP applications.

The final traffic forwarding scheme based on QoE monitoring is shown in Image 7.1, using the topology of Image 6.1 which was used for the current thesis.

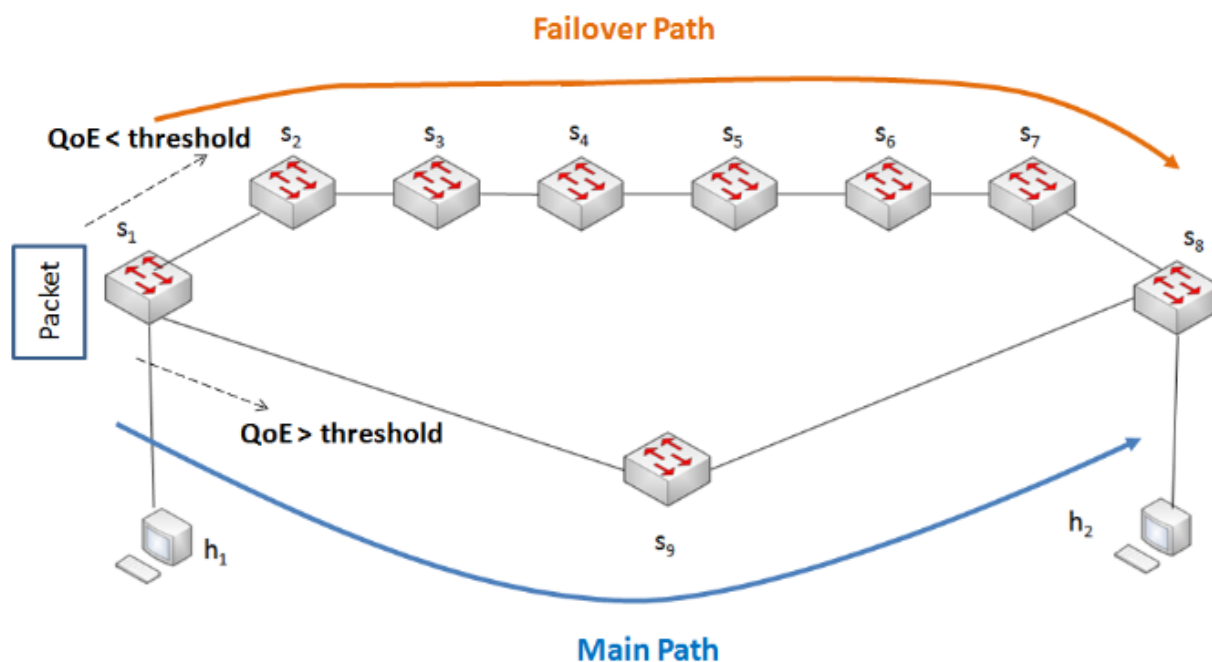


Image 7.1: Traffic forwarding according to QoE Monitoring

7.2 IMPLEMENTATION STRUCTURE

The SQMF framework was implemented following the instructions presented in Chapter 5. The project developed in IntelliJ IDEA is organized in Maven modules with the structure shown in Image 7.2:

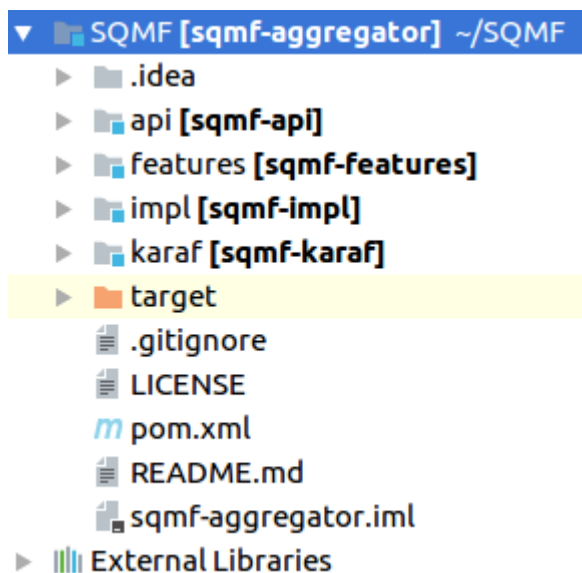


Image 7.2: SQMF implementation structure

where:

- **api** is the created API for the user to the SDN Controller. It contains the YANG files which specify the ODL modules that the project will create – in the case of the current thesis, *sqmf*
- **features** contains the SDN controller features to be used and the SDN features created by the project

- **impl** is the core implementation component, which contains all the developed code for SQMF
- **karaf** is the container provided by the Maven archetype used to create the SDN application, which is necessary so that the ODL modules can run.

When the ODL controller is launched by using the command `./karaf/target/assembly/bin/karaf`, all its modules – including *sqmf*, the one created in the context of the current thesis – are available at the YANG UI tab of the ODL DLUX, as shown in Image 7.3.

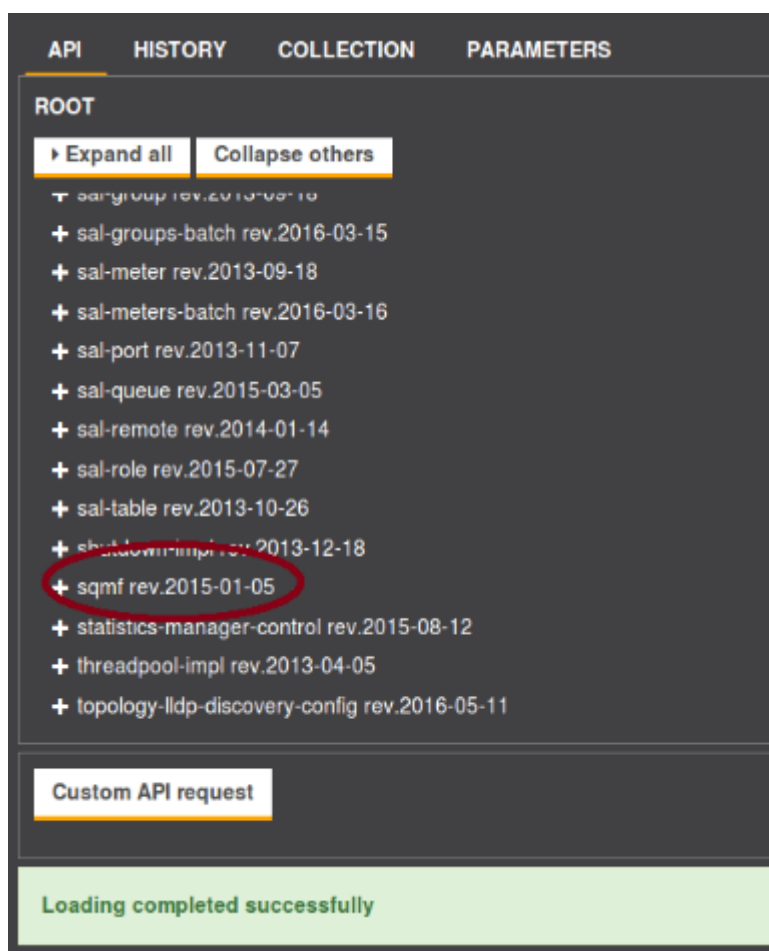


Image 7.3: ODL modules, including *sqmf*

By selecting *sqmf* → *operations*, the user has access to the application’s created RPCs, which are the core implementation methods and constitute the framework’s implemented functionalities. Two RPCs were created, as depicted in Image 7.4.

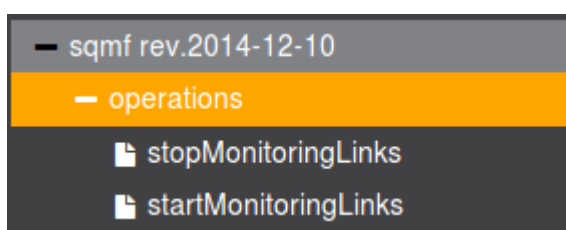


Image 7.4: The created RPCs

- **startMonitoringLinks**: The RPC which starts the process of the periodic link monitoring in the network. More specifically, this process indicates that the links of the path between the source and the destination are monitored periodically and the

QoE is computed based on their delay and packet loss. In case the QoE is lower than a predefined threshold, the controller performs corrective actions. The requested input fields are shown in Image 7.5 and must be filled as following:

- **srcNode** is the name of the switch connected to the sender host. The implementation in the current thesis uses the topology depicted in Image 6.1: Topology with nine switches and two hosts, used for the current thesis, therefore the srcNode *openflow:1*, as the sender host is h_1 .
- **dstNode** is the name of the switch connected to the receiver host. In the topology used for the current thesis the receiver host is h_2 so the dstNode is *openflow:8*.
- **QoEThreshold** is the minimum accepted value for the QoE. If QoE gets lower than QoEThreshold, corrective actions are performed.
- **ApplicationType** is a dropdown menu where the type of the application which will be used (VoIP or video) must be filled.

Image 7.5: Requested input for *startMonitoringLinks* RPC

- **stopMonitoringLinks**: The RPC which stops the periodical monitoring of links, so that the execution is terminated gracefully without throwing an exception for forcing the periodic task to stop. No input is required for this RPC.

Some important notes on the application's behavior are the following:

- In order to select an RCP, a network topology must have been created in Mininet first. Otherwise, the implemented function will not take place.
- All the input fields are required to be filled in. The case where a field is left empty is handled successfully by the application, but the implemented function will not take place.
- Fields *srcNode* and *dstNode* need to be given values that exist as nodes and are connected to hosts. Otherwise, the implemented function will not take place and undefined behavior will be invoked.
- In case *startMonitoringLinks* is selected and the created traffic type does not match the specified *ApplicationType*, the QoE will be computed based on the formula for other application type. This happens because there is no way to determine if the generated traffic type matches the specified *ApplicationType*, as both VoIP and video traffic generate UDP packets.

7.3 DETAILED IMPLEMENTATION ANALYSIS

The source code of SQMF is located in two of the project's Maven modules; the YANG file creating the generated SDN module for the current thesis can be found in *api* and the Java code can be found in *impl*. Moreover, each module contains a *pom.xml* file, where all the dependencies to other modules are declared. The code files are now described per module in more detail. Only the most significant parts of code are shown in this chapter, whereas the whole code can be found in the URL provided in the Annex.

1. *api*

- *pom.xml*: Contains the dependencies for *api* module, which are some default SDN Controller features.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0">

  <parent>
    <groupId>org.opendaylight.mdsal</groupId>
    <artifactId>binding-parent</artifactId>
    <version>0.9.1-Boron-SR1</version>
    <relativePath/>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>sqmf</groupId>
  <artifactId>sqmf-api</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>bundle</packaging>
  <properties>
    <ietf-inet-types.version>2010.09.24.8.1-Beryllium-SR1</ietf-inet-types.version>
    <ietf-yang-types.version>2010.09.24.8.1-Beryllium-SR1</ietf-yangtypes.version>
    <yang-ext.version>2013.09.07.8.1-Beryllium-SR1</yang-ext.version>
    <controller-model.version>1.3.1-Beryllium-SR1</controller-model.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.opendaylight.controller.model</groupId>
      <artifactId>model-inventory</artifactId>
      <version>${controller-model.version}</version>
    </dependency>
    <dependency>
      <groupId>org.opendaylight.mdsal.model</groupId>
      <artifactId>ietf-yang-types</artifactId>
      <version>${ietf-yang-types.version}</version>
    </dependency>
    <dependency>
      <groupId>org.opendaylight.mdsal.model</groupId>
      <artifactId>ietf-inet-types</artifactId>
      <version>${ietf-inet-types.version}</version>
    </dependency>
  </dependencies>

```

```

<dependency>
  <groupId>org.opendaylight.mdsal.model</groupId>
  <artifactId>yang-ext</artifactId>
  <version>${yang-ext.version}</version>
</dependency>
</dependencies>
</project>

```

2. api/src/main/yang

- *sqmf.yang*: The YANG file which generates the thesis's SDN module, named *sqmf*. The module contains two RPCs defined in the YANG file: *startMonitoringLinks* and *stopMonitoringLinks*. For each RPC, the required input is specified. Each input field required is denoted as *leaf* and a name and type must be specified for it.

```

module sqmf {
  yang-version 1;
  namespace
  "urn:opendaylight:params:xml:ns:yang:sqmf";
  prefix "sqmf";
  revision "2014-12-10" {
    description "Initial revision of sqmf model";
  }
  typedef ApplicationType {
    type enumeration {
      enum "VoIP";
      enum "Video";
    }
  }
  rpc startMonitoringLinks{
    input{
      leaf srcNode {
        type string;
      }
      leaf dstNode {
        type string;
      }
      leaf QoEThreshold {
        type string;
      }
      leaf application {
        type ApplicationType;
      }
    }
  }
  rpc stopMonitoringLinks{
  }
}

```

When the project is compiled, the YANG file generates one Java class per defined RPC and defined entity (e.g. typedef), which are used by the Java code in order to

implement the RPCs' desired functionalities. For each RPC, Java classes corresponding to its input and output are generated. Also a service class is generated, which contains the prototypes of all the defined RPCs and enables their implementation. These classes can be found in the path *api/target/classes/org/opendaylight/yang/gen/v1/urn/opendaylight/params/xml/ns/yang/sqmf/rev141210* and are:

- *ApplicationType*: An enumeration containing the VoIP and video application types. It is used in the Java code to specify the user's selection.
- *SqmfService*: A class gathering all the implemented RPCs together. It is used in the Java code to implement the body of the created RPCs.

```
public interface SqmfService extends RpcService {
    Future<RpcResult<Void>> startFailover(StartFailoverInput var1);
    Future<RpcResult<Void>> stopMonitoringLinks();
    Future<RpcResult<Void>>startMonitoringLinks(StartMonitoringLinksInput var1);
}
```

- *StartMonitoringLinksInput*: A class representing the required input for *startMonitoringLinks*, It is used in the Java code to get the user's input for the RPC.

```
public interface StartMonitoringLinksInput extends DataObject,
    Augmentable<StartMonitoringLinksInput> {
    QName QNAME = QName.create("urn:opendaylight:params:xml:ns:yang:sqmf",
        "2014-12-10", "input").intern();
    String getSrcNode();
    String getDstNode();
    String getQoEThreshold();
    ApplicationType getApplication();
}
```

3. *impl*

- *pom.xml*: Contains the dependencies for *impl* module, which are some default SDN Controller features and the *api* module.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://maven.apache.org/POM/4.0.0">
  <parent>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>config-parent</artifactId>
    <version>0.5.1-Boron-SR1</version>
    <relativePath/>
```

```

</parent>
<modelVersion>4.0.0</modelVersion>
<groupId>sqmf</groupId>
<artifactId>sqmf-impl</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>bundle</packaging>
<properties>
  <openflow.plugin.version>0.3.1-Boron-SR1</openflow.plugin.version>
  <l2switch.version>0.4.1-Boron-SR1</l2switch.version>
  <jung2.version>2.0.1</jung2.version>
  <mdsal.version>1.4.1-Boron-SR1</mdsal.version>
</properties>
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>sqmf-api</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.openflowplugin.model</groupId>
    <artifactId>model-flow-service</artifactId>
    <version>${openflow.plugin.version}</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.controller.model</groupId>
    <artifactId>model-topology</artifactId>
    <version>${mdsal.version}</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.openflowplugin</groupId>
    <artifactId>openflowplugin-api</artifactId>
    <version>${openflow.plugin.version}</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.l2switch.addresstracker</groupId>
    <artifactId>addresstracker-impl</artifactId>
    <version>${l2switch.version}</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.controller.thirdparty</groupId>
    <artifactId>net.sf.jung2</artifactId>
    <version>${jung2.version}</version>
  </dependency>
  <dependency>
    <groupId>org.jgrapht</groupId>
    <artifactId>jgrapht-core</artifactId>
    <version>1.0.1</version>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>libldp</artifactId>
    <version>0.12.0-SNAPSHOT</version>
  </dependency>

```



```

</dependency>
<dependency>
  <groupId>org.opendaylight.openflowplugin.applications</groupId>
  <artifactId>topology-ldp-discovery</artifactId>
  <version>0.4.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.opendaylight.controller</groupId>
  <artifactId>liblldp</artifactId>
  <version>0.11.1-Boron-SR1</version>
</dependency>
<dependency>
  <groupId>org.opendaylight.openflowplugin.applications</groupId>
  <artifactId>topology-ldp-discovery</artifactId>
  <version>0.3.1-Boron-SR1</version>
</dependency>
<dependency>
  <groupId>sqmf</groupId>
  <artifactId>sqmf-api</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
</dependencies>
</project>

```

4. impl/src/main/java/sqmf/impl

- *DomainLink.java*: The class modeling a link of the graph using *jgrapht* library (added as Maven dependency in *impl's pom.xml*). It contains an ODL link as well its ODL id (e.g. openflow:1:1).
- *DomainNode.java*: The class modeling a node of the graph using *jgrapht* library. It contains the node's id inside the graph (e.g. 1), the node's ODL ID (e.g. openflow:1) and the graph's id which the node belongs to.
- *ExecuteShellCommand.java*: The class which executes a shell command through the Java code. This is used with video application type to run commands the `ffmpeg -i VIDEO_PATH -hide_banner` and `ffprobe -show_frames [VIDEO_PATH] | grep key_frame` in order to parse their outputs and extract the video frame rate, format, bit rate, codec and key frame interval respectively.
- *GraphOperations.java*: The class which handles the network topology and translates it into a Java graph. It contains methods to add links and nodes to the graph, as well as to remove links. Nodes, once added, are not removed from the graph; only their links can be removed, meaning that the node is actually not used in the graph.
- *LatencyMonitor.java*: The class which measures the latency of a link. For each link, the core method of the class sends a packet from the source node so that the destination can receive it, and forward it to the controller where the interval can be measured. The method then waits until a global variable, *latency*, is updated with the computed interval, and then returns it. The core method is:

```

public Long MeasureNextLink(Link link, String srcMac, String nextNodeConnector)
{
    MonitorLinksTask.packetReceivedFromController = false;
    latency = -1L;
    String nodeConnectorId = link.getSource().getSourceTp().getValue();
    String nodeId = link.getSource().getSourceNode().getValue();
    packetSender.sendPacket(0, nodeConnectorId, nodeId, srcMac,
nextNodeConnector);
    while (latency == -1) {
    }
    return latency;
}

```

- *LLDPUtils.java*: The utility class dealing with LLDP packets. It creates the payload for the packet which is created and sent by *PacketSender.java* (explained later) for delay monitoring. The code for this class was not created in the context of the current thesis, but instead was found on the URL [https://github.com/opendaylight/openflowplugin/blob/master/applications/llpspeaker/src/main/java/org/opendaylight/openflowplugin/applications/llpspeaker/LLDPUtil.java](https://github.com/opendaylight/openflowplugin/blob/master/applications/lldp-speaker/src/main/java/org/opendaylight/openflowplugin/applications/llpspeaker/LLDPUtil.java).
- *MonitorLinksTask.java*: The core class of the QoE monitoring functionality, which is used periodically. It measures the delay and packet loss for VoIP or frame rate, bit rate and packet loss for video and computes the QoE for the application. Its core function is presented:

```

public void run() {

    double pathMOS = -1;

    // if application streamed is VoIP
    if (SqmfImplementation.applicationType.equals(VoIP.getName())){
        Long delay = monitorDelay(SqmfImplementation.mainGraphWalk);
        double packetLoss = monitorPacketLoss();
        System.out.println("Total delay is " + delay + " ms");
        System.out.println("Total loss is " + packetLoss + "%");
        pathMOS = VoIP.estimateQoE(delay, packetLoss);
    }
    // if application streamed is Video
    else if (SqmfImplementation.applicationType.equals(Video.getName())){
        double packetLoss = monitorPacketLoss();
        int bitsReceivedCount = findBits();
        //float frameRate = computeVideoFPS(videoAbsolutePath);
        float frameRate = videoFPS;
        float N = computeN(frameRate);
        float BR = computeVideoBitRate(videoAbsolutePath);

        float bitRate;
    }
}

```

```

        if (bitsReceivedCount == 0){
            BR = 0;
        }
        if (frameRate != -1){
            pathMOS = Video.estimateQoE(frameRate, BR, packetLoss, videoCase);
        }

        System.out.println("FPS is " + frameRate);
        System.out.println("BR is " + BR);
        System.out.println("PLR is " + packetLoss);

    }

    System.out.println("MOS is " + pathMOS);
    if ( linkFailure || ((pathMOS >= 0) && (pathMOS <
    SqmImplementation.QoEThreshold)) ) {
        System.out.println("MOS is lower than the threshold.");
        if (!isFailover && PacketProcessing.videoHasStarted) {
            if (!SqmImplementation.fastFailover) {
                SqmImplementation.changePath();
            }
        }
    }
    else{
        System.out.println("Cannot change path although QoE low.");
    }
}
System.out.println("-----");
}
}

```

It is important to note that the highlighted line is the key to the framework's mechanism; if commented, the path will not change even if low QoE is detected and the framework will implement a monitoring function, without taking corrective actions.

- *NetworkGraph.java*: The class modeling the network topology. It contains an instance of the network graph, as well as methods to update this instance with nodes and links additions or removals.
- *PacketParsingUtils.java*: The class containing the functions to parse an incoming packet to the SDN Controller. The code for this class was not created in the context of the current thesis, but instead was found on the URL https://github.com/sdnhub/SDNHub_Openaylight_Tutorial/blob/master/commons/Utils/src/main/java/org/sdnhub/odl/tutorial/Utils/PacketParsingUtils.java.
- *PacketProcessing.java*: The class which listens for packets received by the controller and examines them.
 - If the packet received is a UDP packet received from the ingress node or the egress node, it is counted to the packets sent from the source or received from the destination respectively, so that packet loss is computed.
 - If the packet received is a UDP packet having the value 00:00:00:00:00:09 as its source MAC address, it is understood that it was sent to the

controller by a switch which received the packet from its previous switch, therefore delay is measured.

- *PacketSender.java*: The class creating a packet with a specific MAC address - 00:00:00:00:00:09 in particular - and sending it to the output interface of a node, which leads to the next node of the path. The method sending the packet is:

```
public boolean sendPacket(String outputNodeConnector, String nodeId, String
srcMac, String nextNodeConnector) {

    MacAddress srcMacAddress = new MacAddress(srcMac);
    String nodeConnectorId = outputNodeConnector.split(":")[2];

    NodeRef ref = createNodeRef(nodeId);
    NodeConnectorId ncid = new NodeConnectorId(outputNodeConnector);
    NodeConnectorKey nodeConnectorKey = new NodeConnectorKey(ncid);
    NodeConnectorRef nEgressConfRef = new
    NodeConnectorRef(createNodeConnRef(nodeId, nodeConnectorKey));

    byte[] lldpFrame = LLDPUtils.buildLldpFrame(new NodeId(nodeId),
        new NodeConnectorId(outputNodeConnector), srcMacAddress,
        Long.parseLong(nodeConnectorId));

    ActionBuilder actionBuilder = new ActionBuilder();
    ArrayList<Action> actions = new ArrayList<>();

    Action outputNodeConnectorAction = actionBuilder
        .setOrder(0).setAction(new OutputActionCaseBuilder()
            .setOutputAction(new OutputActionBuilder()
                .setOutputNodeConnector(new Uri(nodeConnectorId))
                .build())
            .build())
        .build();
    actions.add(outputNodeConnectorAction);

    TransmitPacketInput packet = new TransmitPacketInputBuilder()
        .setEgress(nEgressConfRef)
        .setNode(ref)
        .setPayload(lldpFrame)
        .setAction(actions)
        .build();
    sentTimes.put(nextNodeConnector, System.currentTimeMillis());

    Future<RpcResult<Void>> future =
    packetProcessingService.transmitPacket(packet);
    try {
        if (future.get().isSuccessful()) {
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

        return false;
    }
}

```

- *SqmflImplementation.java*: The core class of SQMF, implementing its functionality. This class is based on the RPCs generated by the YANG module of *api* and implements their bodies. The RPC which implements the QoE monitoring gets the user input and checks its validity, configures the paths and starts the monitoring task. The code is:

```

@Override
public Future<RpcResult<Void>> startMonitoringLinks(StartMonitoringLinksInput
input) {

    if (input != null){
        if (input.getSrcNode() != null && input.getDstNode() != null &&
input.getQoEThreshold() != null && input.getApplication() != null){
            srcNode = input.getSrcNode();
            dstNode = input.getDstNode();
            try {
                Float QoE = Float.parseFloat(input.getQoEThreshold());
                QoEThreshold = QoE.doubleValue();
            }
            catch (NumberFormatException e){
                LOG.info("Wrong number format for QoE threshold, try again.");
                return
                Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
            }
            applicationType = input.getApplication().getName();
        }
    }
    else{
        LOG.info("A field of the input is empty, try again.");
        return Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
    }

    // if the application type is video, launch a file chooser to select a video file to
    be streamed
    if (applicationType.equals(Video.getName())){
        FileDialog dialog = new FileDialog((Frame)null, "Select File to Open");
        dialog.setMode(FileDialog.LOAD);
        dialog.setVisible(true);
        videoAbsolutePath = dialog.getDirectory() + dialog.getFile();
        videoCase = findVideoCase(videoAbsolutePath);
        if (videoCase == 0){
            LOG.info("Video required specifications not met, choose another video.");
            return
            Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
        }
    }

    //first, add rules to ingress and egress nodes to forward their packets to

```

```

controller
if (NetworkGraph.getInstance().getGraphNode() != null &&
NetworkGraph.getInstance().getGraphLinks() != null) {
    Hashtable<String, DomainNode> domainNodes =
    NetworkGraph.getInstance().getDomainNodes();
    DomainNode sourceNode = domainNodes.get(srcNode);
    DomainNode destNode = domainNodes.get(dstNode);

    //check if the given switches are edge switches, therefore connected to
    hosts
    if (!checkIfEdgeSwitches(sourceNode, destNode)){
        LOG.info("Not edge switches given, returning...");
        return
        Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
    }

    List<GraphPath<Integer, DomainLink>> possiblePaths =
    createPaths(NetworkGraph.getInstance(), sourceNode.getNodeID(),
    destNode.getNodeID());
    if (possiblePaths.size() > 1) {
        GraphPath<Integer, DomainLink> mainPath = possiblePaths.get(0);
        GraphPath<Integer, DomainLink> failoverPath = possiblePaths.get(1);

        //determine main and failover path
        mainGraphWalk = mainPath;
        failoverGraphWalk = failoverPath;
        findPorts(mainGraphWalk, sourceNode, destNode);

        //register packet processing listener
        PacketProcessing packetProcessingListener = new
        PacketProcessing(srcNode, dstNode, srcMacForDelayMeasuring);
        if (notificationService != null) {
            notificationService.
            registerNotificationListener(packetProcessingListener);
        }
        SwitchConfigurator switchConfigurator = new SwitchConfigurator(db);
        //configure ingress and egress switches to send their packets to controller
        (for packet loss monitoring)
        switchConfigurator.configureIngressAndEgressForMonitoring(srcNode,
        dstNode, inputPorts, outputPorts);
        /* then, add rules to all nodes of both main and failover path to forward
        packets with specific MAC to controller (for delay monitoring) */
        switchConfigurator.
        configureNodesForDelayMonitoring(mainGraphWalk.getEdgeList(),
        srcMacForDelayMeasuring);
        switchConfigurator.
        configureNodesForDelayMonitoring(failoverGraphWalk.getEdgeList(),
        srcMacForDelayMeasuring);
        switchConfigurator.
        configureNodesForUDPTrafficForwarding(mainGraphWalk.getEdgeList(),
        inputPorts, outputPorts);
    }
}

```

```

    }
    else{
        return Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
    }

    //finally, start monitoring links
    timer = new Timer();
    monitorLinksTask = new MonitorLinksTask(db, rpcProviderRegistry,
    srcMacForDelayMeasuring, videoAbsolutePath,
    Video.getVideoFPS(videoAbsolutePath), videoCase);
    timer.schedule(monitorLinksTask, 0, 5000);

    return Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
}

```

The RPC which stops the monitoring of links is:

```

@Override
public Future<RpcResult<Void>> stopMonitoringLinks() {
    System.out.println("Stopping the monitoring of links.");
    if (monitorLinksTask != null) {
        monitorLinksTask.cancel();
    }
    if (timer != null) {
        timer.cancel();
        timer.purge();
    }
    return Futures.immediateFuture(RpcResultBuilder.<Void>success().build());
}

```

- *SqmfProvider.java*: The class which initiates the created SDN module *sqmf*. It initializes services which will provide access to the SDN Controller's MDSAL and to notifications and starts the core class *SqmfImplementation.java*
- *SwitchConfigurator.java*: The class which contains all the necessary methods for any switch configuration. Its methods insert the desired OF rules in the appropriate switches.
- *TopologyListener.java*: The class which detects for topology changes in the network and updates the graph respectively. Each time a link change happens, it keeps a list with all the links which changed their status and asks for either addition to or removal from the network graph.
- *Video.java*: The class modeling the video application type. It contains methods for QoE estimation as well as for video characteristics detection, such as frame rate and codec. One of the most significant methods is the one estimating the QoE of a video application, using the G.1070 E-model formula:

```

public static double estimateQoE(float frameRate, float bitRate, double packetLoss,
int videoCase) {

    assignCoordinatesValues(videoCase);
    double OFr = v1 + v2*bitRate;

```

```

double IOfr = v3 - v3/(1 + (Math.pow(bitRate, v5)/v4));
double DFrv = v6 + v7*bitRate;
double DPpIV = v10 + v11*Math.exp(-frameRate/v8) + v12*Math.exp(
bitRate/v9);
double numeratorIcoding = -Math.pow((Math.log(frameRate)-Math.log(OFr)), 2);
double denominatorIcoding = 2*Math.pow(DFrv, 2);
double Icoding = IOfr*Math.exp(numeratorIcoding/denominatorIcoding);
double Itransmission = Math.exp(-(packetLoss/DPpIV));
double MOS = 1 + Icoding*Itransmission;
return MOS;
}

```

Other important methods in the video modeling class are the ones obtaining the video codec, frame rate, key frame interval and format by running a shell command through Java and parsing its output. Finally, the last method of the class is the one assigning values to the 12 coefficients needed for the video quality computation, as shown in 3.3.2. The five cases presented in Table 3.2 are supported.

- *VoIP.java*: The class modeling the VoIP application type. The most significant method is the one estimating the QoE of a VoIP application, using the G.107 E-model:

```

public static double estimateQoE(Long delay, double packetLoss)
{
    int h;
    if (delay - 177.3 > 0){
        h = 1;
    }
    else {
        h = 0;
    }
    double R = 94.2 - 0.024*delay - 0.11*h*(delay-177.3) - 11 -
40*Math.log(1+10*packetLoss);
    double MOS;
    if (R < 0){
        MOS = 0;
    }
    else{
        MOS = 1 + 0.035*R + R*(R-60)*(100-R)/1000000;
    }
    return MOS;
}

```

An overall correlation between the created classes is shown in Figure 7.2.

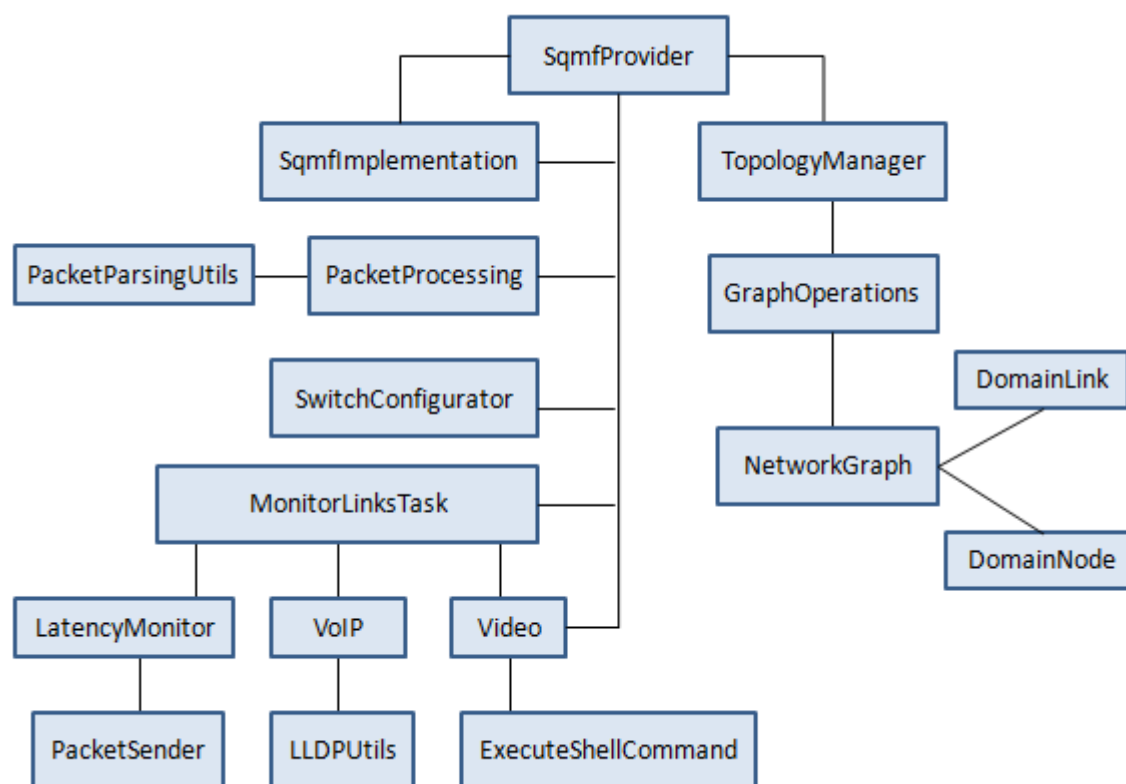


Figure 7.2: SQMF classes correlation

7.4 SQMF DEMONSTRATION

In order to give a more vivid description of the implemented framework, this subchapter presents a use case of SQMF: a VoIP or video streaming case where a link failure occurs and corrective actions are made so that the QoE is preserved at high levels. The provided input, the established OF rules at the beginning of the QoE monitoring, the monitoring messages, the established OF rules when the link failure occurs and Wireshark captures will be presented.

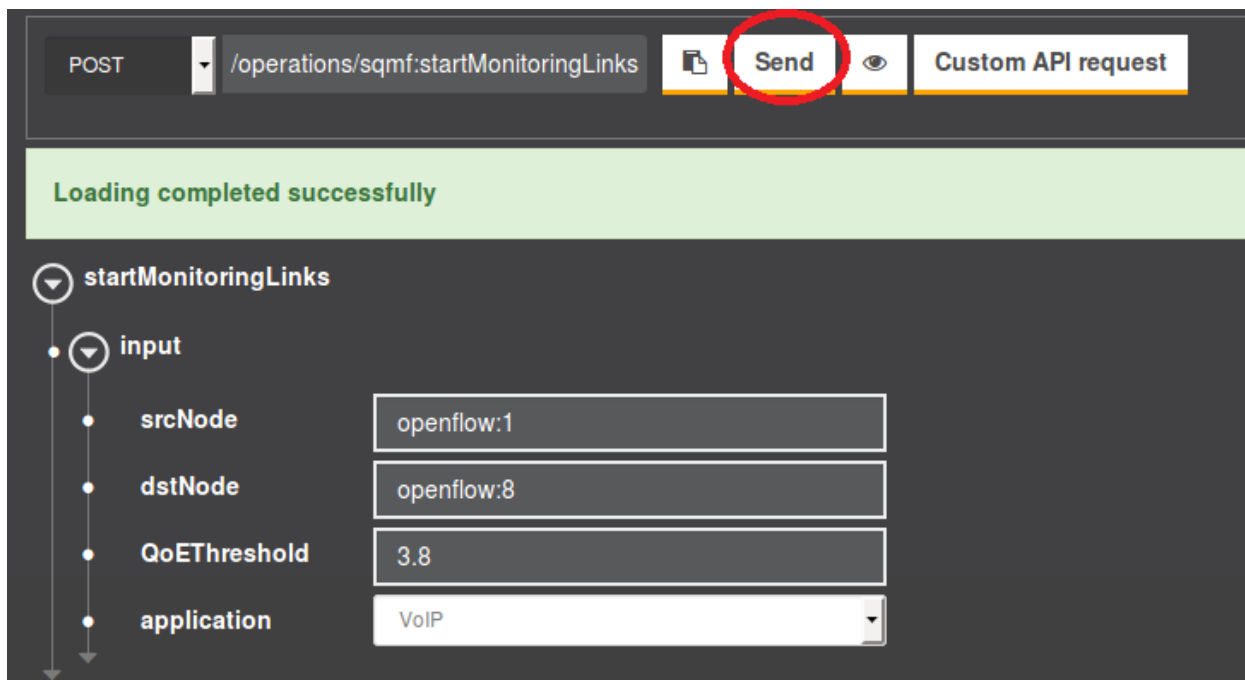
To simulate the use case, the first step is to launch the SDN controller, create the network topology shown in Image 6.1 and connect it to the controller. It is important to note that the command `pingall` must be typed in the Mininet console in order for the framework to operate, as it makes the hosts and their links to the topology visible to the controller. The controller's message when notified about the added links is shown in Image 7.6.

```

opendaylight-user@root>Added graph node with id openflow:8
Added graph node with id openflow:9
Added graph node with id openflow:7
Added graph node with id openflow:2
Added graph node with id openflow:3
Added graph node with id openflow:6
Added graph node with id openflow:5
Added graph node with id openflow:4
Added graph node with id openflow:1
Added graph node with id host:00:00:00:00:00:02
Added graph node with id host:00:00:00:00:00:01
  
```

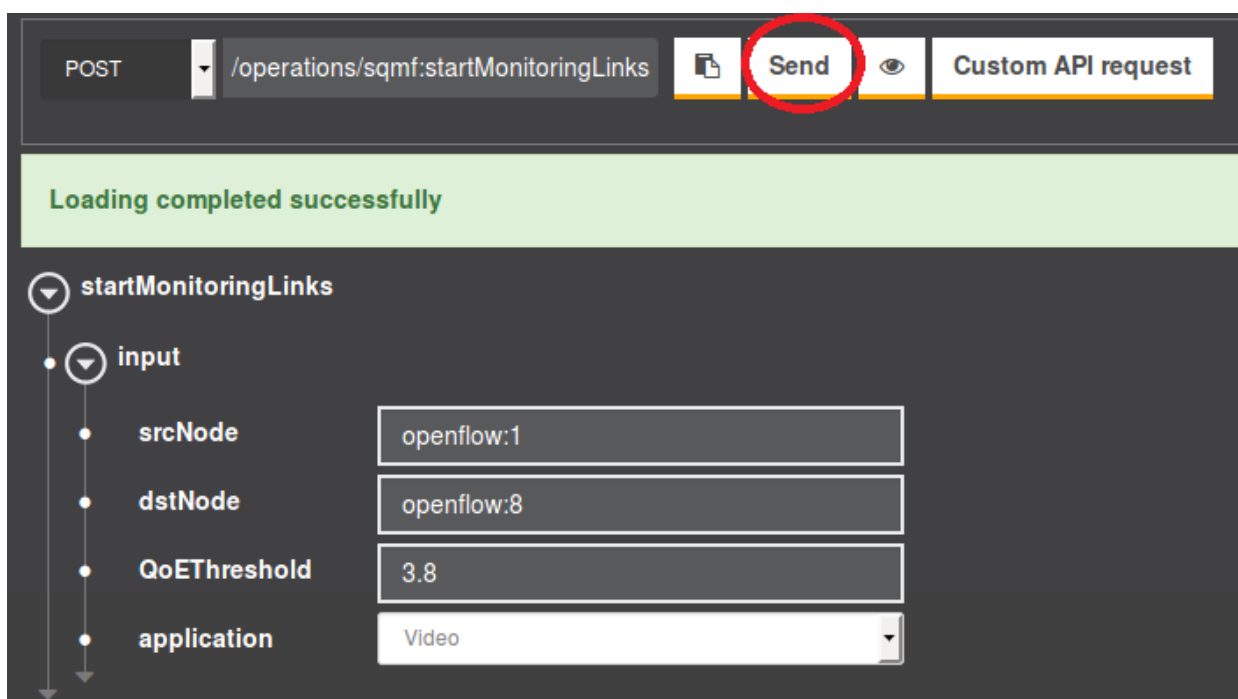
Image 7.6: The controller message when the new topology is created

After this, the user browses to the DLUX page, selects *sqmf* → *operations* → *startMonitoringLinks* and provides the necessary input. Image 7.7 depicts the provided input in case of VoIP traffic, whereas Image 7.8 shows the video traffic case's input, while the prompt for video selection is shown in Image 7.9.



The screenshot shows a web interface for configuring monitoring links. At the top, there is a header with a dropdown menu set to 'POST', a text input field containing '/operations/sqmf:startMonitoringLinks', a 'Send' button circled in red, an eye icon, and a 'Custom API request' button. Below the header is a green notification bar that says 'Loading completed successfully'. The main content area is titled 'startMonitoringLinks' and contains an 'input' section with four fields: 'srcNode' (text input with 'openflow:1'), 'dstNode' (text input with 'openflow:8'), 'QoEThreshold' (text input with '3.8'), and 'application' (dropdown menu with 'VoIP' selected).

Image 7.7: The provided input for VoIP traffic



The screenshot shows the same web interface as Image 7.7, but with the 'application' dropdown menu set to 'Video'. The 'Send' button is also circled in red. The notification bar and other fields remain the same.

Image 7.8: The provided input for video traffic

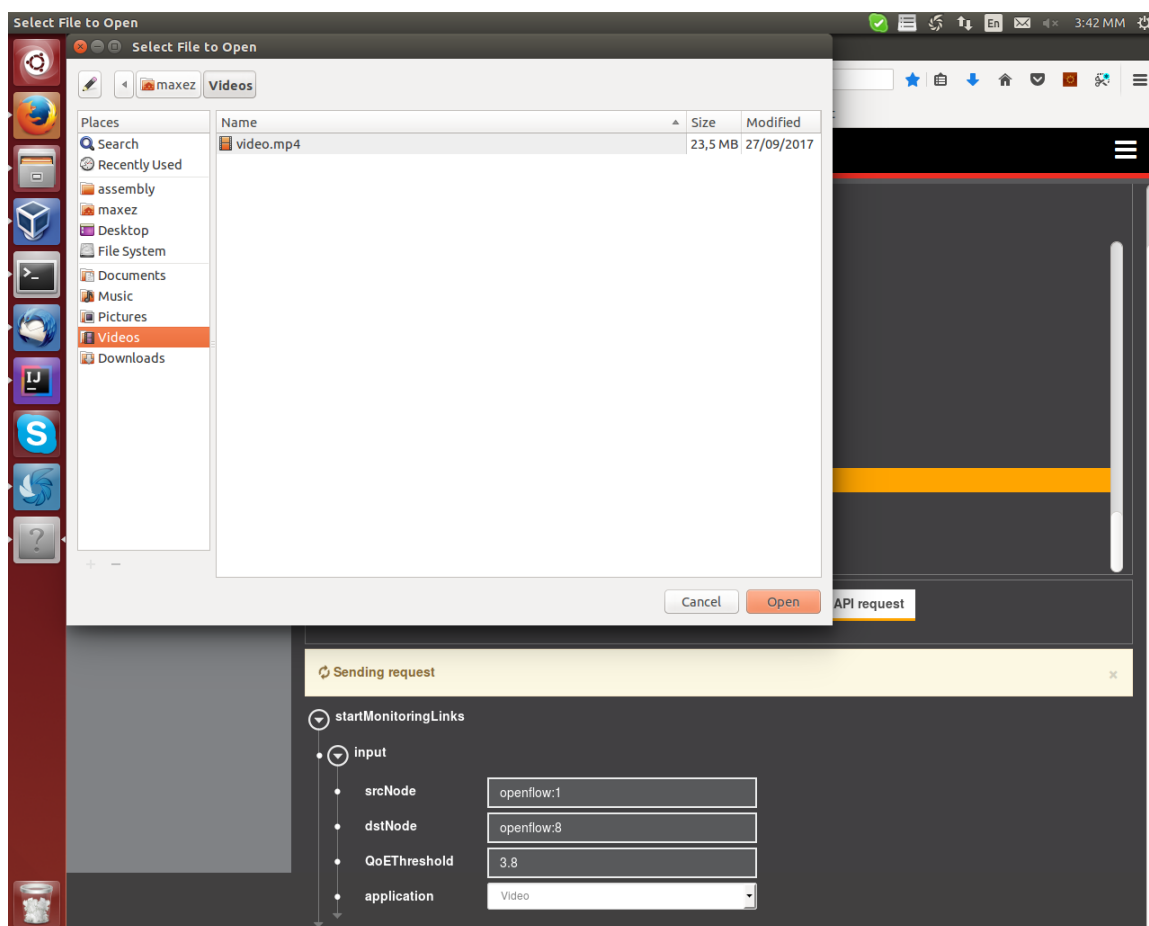


Image 7.9: Prompt for video selection, after input for video application type

By clicking *Send*, the RPC begins its functionality by computing the transmission paths; the **main path will be** $s_1 \rightarrow s_9 \rightarrow s_8$, as it is the shortest, and the **backup path will be** $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7 \rightarrow s_8$. Then the RPC installs the appropriate OF rules in the switches:

- For the path's ingress switch s_1 , the inserted rule is depicted in Image 7.10 with priority 1000, so that it is matched first. The rule is used both for the path establishment and for packet loss monitoring; it matches UDP traffic as well as the input port, and if the packet is coming from the sender host, it is forwarded both to the controller, so that packet loss is measured, and to the next node of the path. The other rules are inserted by default from the OF switch.

```
cookie=0x2a00000000000000, duration=871.160s, table=0, n_packets=0, n_bytes=0, priority=1000,udp,in_port=2 actions=CONTROLLER:65535,output:3
cookie=0x2b0000000000000007, duration=988.855s, table=0, n_packets=77, n_bytes=6545, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b000000000000000f, duration=985.013s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=3 actions=output:2,output:1
cookie=0x2b0000000000000010, duration=985.012s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
cookie=0x2b0000000000000011, duration=985.012s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:3,output:2
```

Image 7.10: OF rules established in ingress switch for QoE monitoring

- In the core switches, i.e. only s_9 in the case of the topology used for this thesis, the inserted rules are depicted in Image 7.11 with priority 1000. The first rule with priority 1000 is used for delay measurement; if a packet with 00:00:00:00:00:09 as source MAC address is received, it is assumed that it was sent from the previous node (s_1 in this case) and is forwarded to the controller. The second rule with

priority 1000 is used for the path establishment; if a UDP packet is received from the input port, it is forwarded to the next node. The other rules are inserted by default from the OF switch.

```
cookie=0x2a00000000000002, duration=940.522s, table=0, n_packets=13, n_bytes=1105, priority=1000,dl_src=00:00:00:00:00:09 actions=CONTROLLER:65535
cookie=0x2a0000000000000b, duration=940.302s, table=0, n_packets=0, n_bytes=0, priority=1000,udp,in_port=1 actions=output:2
cookie=0x2b00000000000004, duration=1058.510s, table=0, n_packets=78, n_bytes=6630, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b00000000000003, duration=1054.537s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=2 actions=output:1
cookie=0x2b00000000000004, duration=1054.537s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=1 actions=output:2
```

Image 7.11: OF rules established in core switches for QoE monitoring

- In the egress switch s_8 the inserted rules are depicted in Image 7.12 with priority 1000. The first rule with priority 1000 is used both for the path establishment and for packet loss monitoring; it matches UDP traffic as well as the input port, and if the packet is coming from the previous node, it is forwarded both to the controller, so that packet loss is measured, and to the destination host. The second rule with priority 1000 is used for delay measurement; if a packet with 00:00:00:00:00:09 as source MAC address is received, it is assumed that it was sent from the previous node (s_9 in this case) and is forwarded to the controller for delay monitoring. The other rules are inserted by default from the OF switch.

```
cookie=0x2a00000000000001, duration=1043.294s, table=0, n_packets=0, n_bytes=0, priority=1000,udp,in_port=3 actions=CONTROLLER:65535,output:2
cookie=0x2a0000000000000a, duration=1043.130s, table=0, n_packets=0, n_bytes=0, priority=1000,dl_src=00:00:00:00:00:09 actions=CONTROLLER:65535
cookie=0x2b00000000000006, duration=1161.060s, table=0, n_packets=76, n_bytes=6460, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b00000000000000, duration=1157.154s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=3 actions=output:2,output:1
cookie=0x2b00000000000001, duration=1157.153s, table=0, n_packets=4, n_bytes=280, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
cookie=0x2b00000000000002, duration=1157.152s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:3,output:2
```

Image 7.12: OF rules established in egress switch for QoE monitoring

- The core switches of the failover path are also configured for delay monitoring, just as the core switches of the main path. The OF rules for $s_2 - s_7$ are identical to the first rule shown in Image 7.11.

After the OF rules establishment, a periodic task is started and part of the output in the controller console is depicted in Image 7.13. The pattern is repeated every 5 seconds.

```
Packets 0 0
Total delay is 3 ms
Total loss is 0.0%
QoE is 3.9419178437268485
-----
Packets 0 0
Total delay is 5 ms
Total loss is 0.0%
QoE is 3.9402438698880005
-----
Packets 0 0
Total delay is 6 ms
Total loss is 0.0%
QoE is 3.939406728720384
-----
```

Image 7.13: Part of the controller output after QoE monitoring has started

The next step is to create VoIP or video traffic between h_1 and h_2 . The OF rules inserted in the nodes ensure that the traffic reaches h_2 , as depicted in the following Wireshark capture (Image 7.14):

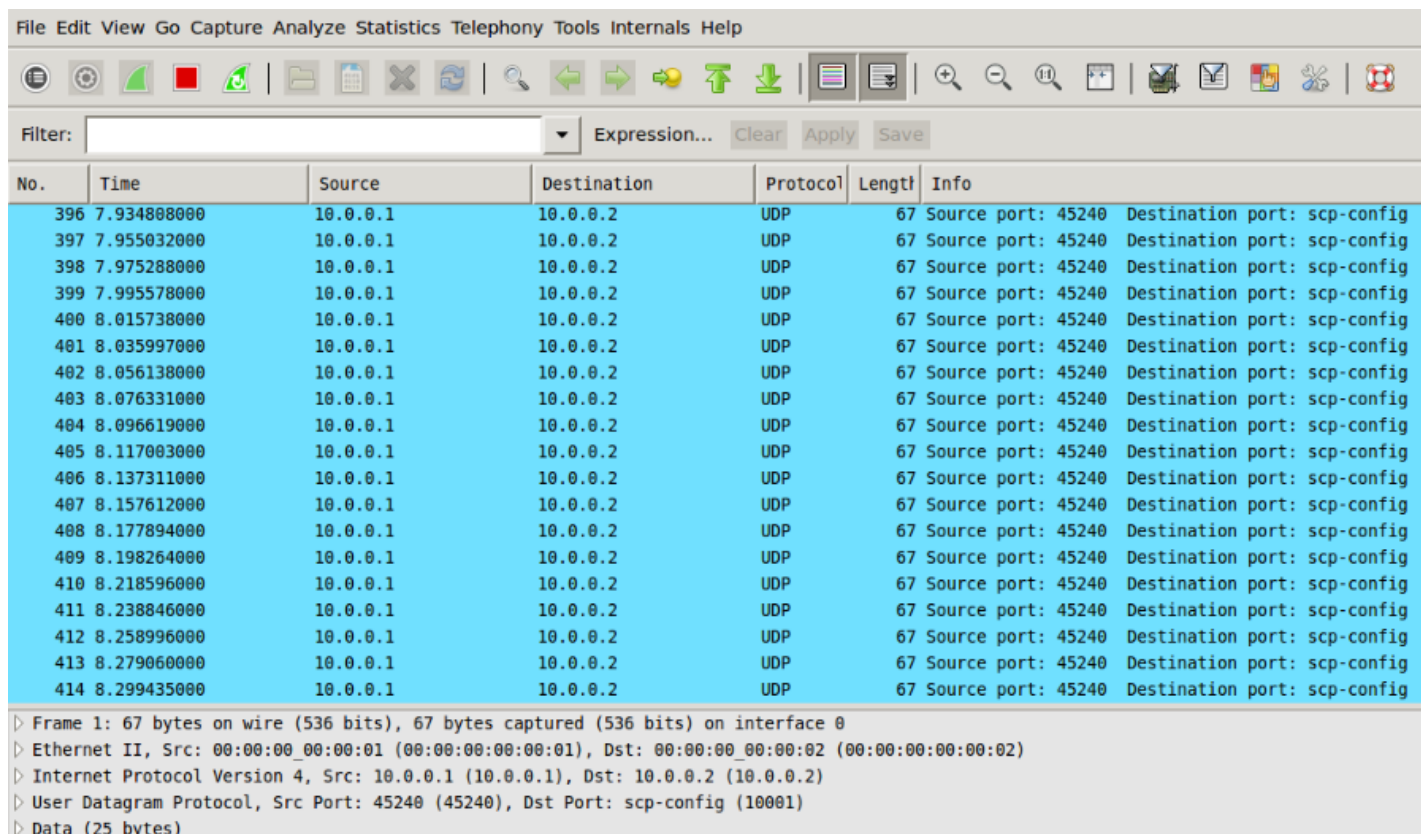


Image 7.14: The traffic received by the destination (h_2)

Let's now assume that the link between s_9 and s_8 faces a failure. In this case, without SQMF, the destination would stop receiving traffic as shown in Image 7.15:

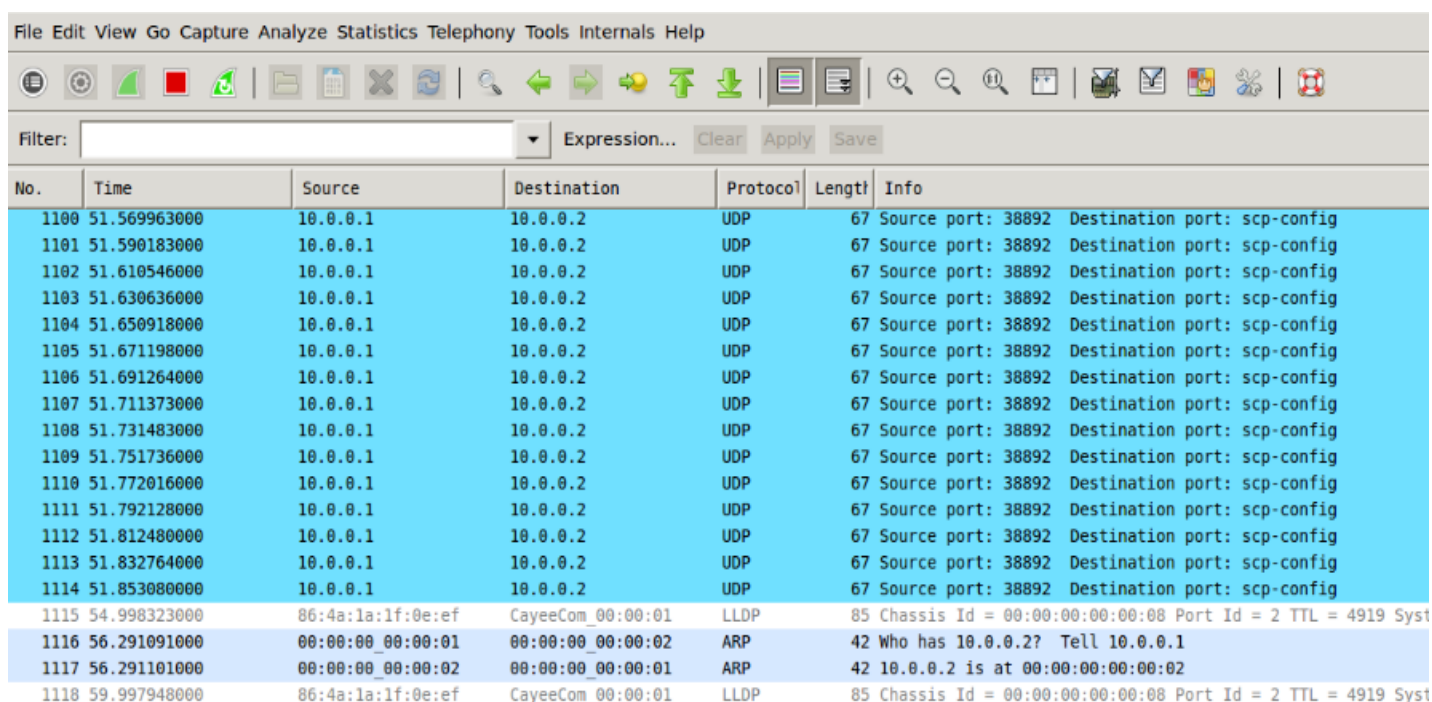


Image 7.15: Traffic stops being delivered at the destination when link failure occurs and no corrective actions are made

However, using the SQMF functionality, the controller inserts new OF rules in the switches when low QoE is detected. More specifically:

- The ingress switch is configured to now *output the incoming UDP packets to the backup path (instead of the main path) and the controller*, as depicted in Image 7.16 with the first rule. The output port has changed to 1, compared to output 3 of the first rule of Image 7.10.

```
cookie=0x2a00000000000000, duration=3.722s, table=0, n_packets=847, n_bytes=56749, priority=1000,udp,in_port=2 actions=CONTROLLER:65535,output:1
cookie=0x2b0000000000000007, duration=926.024s, table=0, n_packets=371, n_bytes=31535, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b0000000000000023, duration=4.468s, table=0, n_packets=10, n_bytes=644, priority=2,in_port=3 actions=output:2,output:1
cookie=0x2b0000000000000024, duration=4.468s, table=0, n_packets=12, n_bytes=796, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
cookie=0x2b0000000000000025, duration=4.468s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=1 actions=output:3,output:2
```

Image 7.16: The OF rules established in the ingress switch when low QoE is detected

- The egress switch is configured to now detect packets coming *from the backup path instead of the main path*, as depicted in Image 7.17 with the first rule. The in_port has changed to 1, compared to in_port 3 of the first rule of Image 7.12.

```
cookie=0x2a0000000000000001, duration=71.233s, table=0, n_packets=3490, n_bytes=233830, priority=1000,udp,in_port=1 actions=CONTROLLER:65535,output:2
cookie=0x2a000000000000000a, duration=159.336s, table=0, n_packets=0, n_bytes=0, priority=1000,dl_src=00:00:00:00:00:09 actions=CONTROLLER:65535
cookie=0x2b0000000000000006, duration=993.622s, table=0, n_packets=381, n_bytes=32385, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x2b0000000000000012, duration=71.987s, table=0, n_packets=12, n_bytes=796, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
cookie=0x2b0000000000000013, duration=71.987s, table=0, n_packets=12, n_bytes=728, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
cookie=0x2b0000000000000014, duration=71.987s, table=0, n_packets=2, n_bytes=84, priority=2,in_port=1 actions=output:3,output:2
```

Image 7.17: The OF rules established in the egress switch when low QoE is detected

As a result, the traffic only stops instantly for a very short time and then is redirected to the backup path. The instant traffic flow pause and recovery is depicted in Image 7.18 and the relevant controller message in Image 7.19.

276	5.324633000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
277	5.344858000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
278	5.365076000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
279	5.385391000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
280	5.405652000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
281	5.425801000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
282	5.445968000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
283	5.452107000	d2:77:d6:41:96:1e	CayeeCom 00:00:01	LLDP	85	Chassis Id = 00:00:00:00:00:08	Port Id = 2 TTL = 4919
284	5.466230000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
285	5.486441000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
286	5.506757000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
287	5.527077000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
288	5.547337000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
289	5.567587000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
290	5.587847000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
291	5.608139000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
292	5.628424000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
293	5.648710000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config
294	5.668839000	10.0.0.1	10.0.0.2	UDP	67	Source port: 38164	Destination port: scp-config

Frame 718: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0
 Ethernet II, Src: d2:77:d6:41:96:1e (d2:77:d6:41:96:1e), Dst: CayeeCom_00:00:01 (01:23:00:00:00:01)
 Link Layer Discovery Protocol

Image 7.18: Instant pause and recovery of traffic flow in case of link failure, using SQMF

```

Packets 567 345
Total delay is 2 ms
Total loss is 0.8987854251012146%
QoE is 1.0
MOS is lower than the threshold.
-----
Packets 814 579
Total delay is 17 ms
Total loss is 0.05263157894736842%
QoE is 3.318932936229071
MOS is lower than the threshold.
-----
Packets 1061 826
Total delay is 10 ms
Total loss is 0.0%
QoE is 3.9360571376640006
    
```

Image 7.19 : Controller output when low QoE is detected

Finally, to stop the monitoring process, the user browses to the DLUX page and selects *sqmf* → *operations* → *stopMonitoringLinks* (Image 7.20).

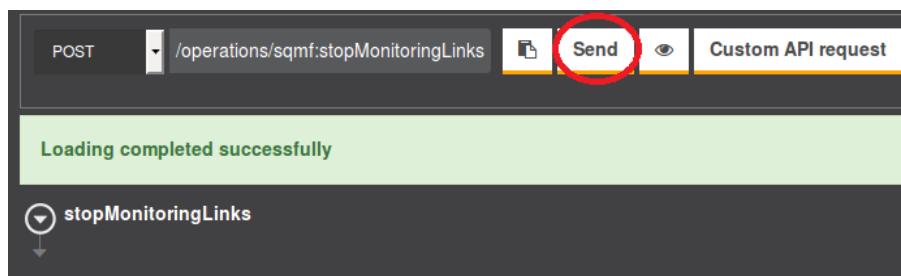


Image 7.20: Calling *stopMonitoringLinks* RPC

By clicking on *Send*, the SDN controller outputs a relevant message, as Image 7.21 shows.

```

-----
Stopping the monitoring of links.
    
```

Image 7.21: The controller message after calling *stopMonitoringLinks*

7.5 SQMF EVALUATION

The current subchapter conducts a proof-of-concept evaluation of SQMF through graphical representations that compare the network’s behavior with and without the SQMF functionality, i.e. comparing the default forwarding with the QoE-based forwarding. The results were extracted from the use cases described in Section 7.4. The evaluation was carried out on the topology depicted in Image 6.1.

7.5.1 SQMF on VoIP traffic

In order to evaluate the SQMF functionality for VoIP applications, **VoIP traffic was generated from h_1 to h_2 for 105 seconds**. The parameters used for the current experiment are shown in Table 7.1.

Table 7.1: Parameters used for SQMF evaluation on VoIP traffic

Packets per second	50
VoIP codec	G.729.2
Traffic generation duration	105 sec

As a first case, the results of which are presented in Table 7.2, the QoE monitoring – based forwarding functionality did not take place and therefore the packets kept being forwarded to the main path as initially configured, even when a link failure occurred. This caused total packet loss after the link failure.

Table 7.2: Delay, R , packet loss and MOS during VoIP traffic generation with link failure, without SQMF

Time (s)	Delay (s)	Packet Loss	R	MOS
5	0.004	0	83.104	4.135726358
10	0.022	0	82.672	4.120869894
15	0.003	0	83.128	4.136544906
20	0.003	0	83.128	4.136544906
25	0.001	87.0445344	-7.727395912	1
30	0.001	100	-12.73981091	1
35	0.001	100	-12.73981091	1
40	0.002	100	-12.76381091	1
45	0.001	100	-12.73981091	1
50	0.001	100	-12.73981091	1
55	0.001	100	-12.73981091	1
60	0.001	100	-12.73981091	1
65	0.0014	100	-13.05181091	1
70	0.0015	100	-13.07581091	1
75	0.001	100	-12.73981091	1
80	0.002	100	-12.76381091	1
85	0.002	100	-12.76381091	1
90	0.005	100	-12.83581091	1
95	0.001	100	-12.73981091	1
100	0.001	100	-12.73981091	1
105	0.001	100	-12.73981091	1

Then, the same experiment was carried out using the SQMF functionality. In this case, the controller efficiently changed the transmission path when the link failure occurred, as it detected QoE levels below the predefined threshold, by configuring the packets to be forwarded to the failover path, so the overall packet losses were much lower. The results of the second case are presented in Table 7.3.

Table 7.3: Delay, R , packet loss and MOS during VoIP traffic generation with link failure, with SQMF

Time (s)	Delay (s)	Packet Loss (%)	R	MOS
5	0.006	0	83.056	4.135726358
10	0.008	0	83.008	4.120869894
15	0.003	0	83.128	4.136544906
20	0.005	0	83.08	4.136544906
25	0.002	83.805668	-6.393607512	1
30	0.009	1.2244898	78.36348452	3.960670612
35	0.02	0	82.72	4.12253203
40	0.008	0	83.008	4.132444968
45	0.013	0	82.888	4.128327083
50	0.012	0	82.912	4.129152092
55	0.005	0	83.08	4.134907089
60	0.022	0	82.672	4.120869894
65	0.01	0	82.96	4.130799964
70	0.018	0	82.768	4.124191323
75	0.017	0	82.972	4.125019902
80	0.013	0	82.888	4.128327083
85	0.016	0	82.816	4.125847767
90	0.011	0	82.936	4.129976386
95	0.017	0	82.972	4.125019902
100	0.01	0	82.96	4.130799964
105	0.01	0	82.96	4.130799964

By illustrating the results of the two cases, with and without QoE monitoring, in the same graphical representation, it is obvious that QoE monitoring-based forwarding performs much better than the default forwarding to a configured main path. Figure 7.3 shows that QoE monitoring-based forwarding achieves much lower packet losses when a link failure occurs, managing to recover immediately after a small period of packet loss detection, in contrast to the default case where all the packets are lost after the link failure.

Packet losses due to link failure with and without SQMF for VoIP traffic generation

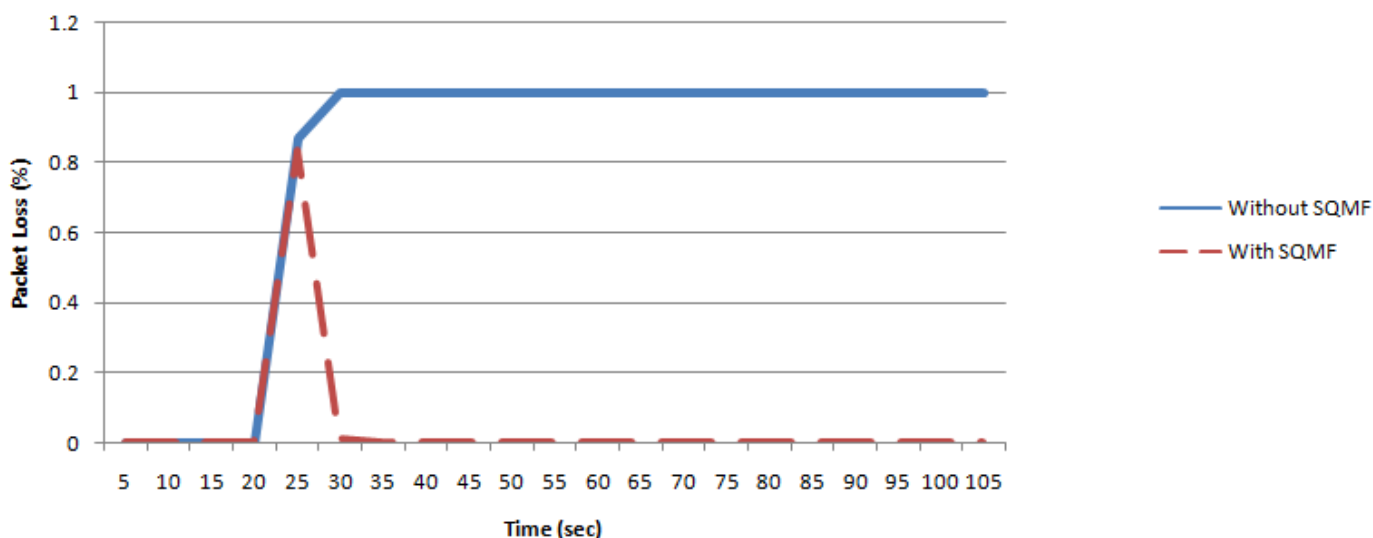


Figure 7.3: Packet loss comparison between cases with and without SQMF in VoIP traffic generation, when a link failure occurs

Therefore, the QoE Monitoring-based forwarding preserves the total QoE and keeps it at high levels even after the link failure, whereas in the default case the QoE faces a permanent degradation after the failure, as depicted in Figure 7.4.

MOS due to link failure with and without SQMF for VoIP traffic generation

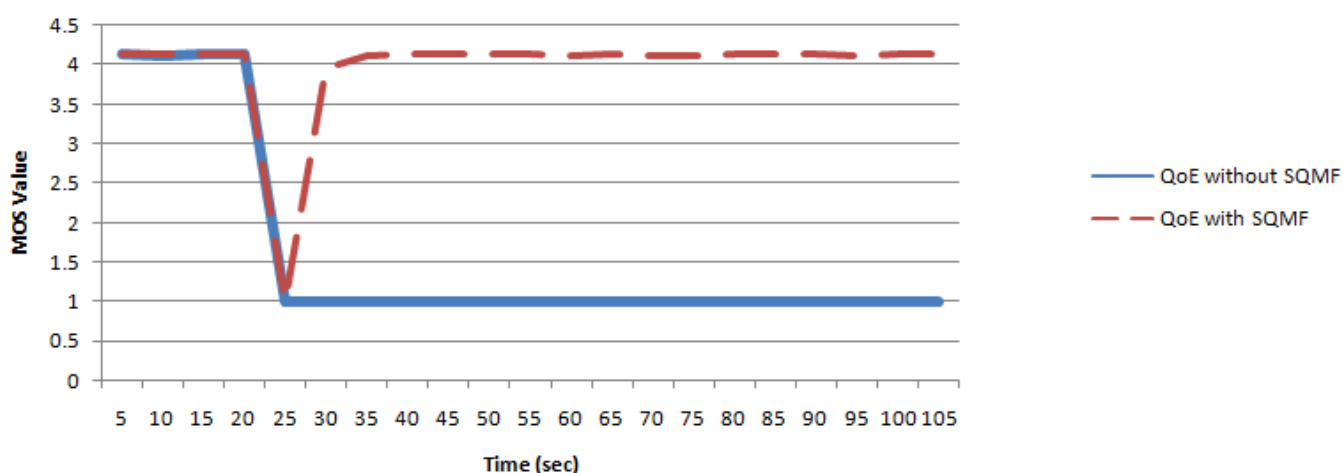


Figure 7.4: MOS comparison between cases with and without SQMF in VoIP traffic generation, when a link failure occurs

7.5.2 SQMF on video traffic

In order to evaluate the QoE monitoring functionality for video applications, **video traffic was streamed from h_1 to h_2 for 105 seconds**. The video used is the same as the one used for Chapter’s 6 experiments, therefore the video parameters are depicted in Table 6.4.

As a first case, the results of which are presented in Table 7.4, the SQMF functionality did not take place and therefore the packets kept being forwarded to the main path as initially configured, even when a link failure occurred. This caused total packet loss after the link failure.

Table 7.4: Packet loss and Vq during video streaming with link failure, without SQMF

Time (s)	Packet Loss (%)	Vq
5	0	4.458499273
10	0	4.458499273
15	0	4.458499273
20	0	4.458499273
25	89.1222806	2.03038674
30	100	1
35	100	1
40	100	1
45	100	1
50	100	1
55	100	1
60	100	1
65	100	1
70	100	1
75	100	1
80	100	1
85	100	1
90	100	1
95	100	1
100	100	1
105	100	1

Then, the same experiment was carried out using the SQMF functionality. In this case, the controller monitored periodically the QoE and when the QoE was detected to be lower than the threshold, due to the link failure, it efficiently changed the transmission path by configuring the packets to be forwarded to the backup path, so the packet losses were much lower and of smaller duration. The results of the second case are presented in Table 7.5.

Table 7.5: Packet loss and Vq during video streaming with link failure, with SQMF

Time (s)	Packet Loss (%)	Vq
5	0	4.458499273
10	0	4.458499273
15	0	4.458499273
20	0	4.458499273
25	85.5957768	2.080959055
30	6.4275037	4.169280356
35	0	4.458499273
40	0	4.458499273
45	0	4.458499273
50	0	4.458499273
55	0	4.458499273
60	0	4.458499273
65	0	4.458499273
70	0	4.458499273
75	0.6728343	4.427026544
80	0	4.458499273
85	0	4.458499273
90	0	4.458499273
95	0	4.458499273
100	0	4.458499273
105	0	4.458499273

By illustrating the results of the two cases, with and without QoE monitoring, in the same graphical representation, it is obvious that QoE monitoring-based forwarding performs much better for video applications than the default forwarding to a configured main path. Figure 7.5 shows that QoE monitoring-based forwarding achieves much lower packet losses when a link failure occurs, managing to recover immediately after a small period of packet loss detection, in contrast to the default case where all the packets are lost after the link failure.

Packet losses due to link failure with and without SQMF for Video Streaming

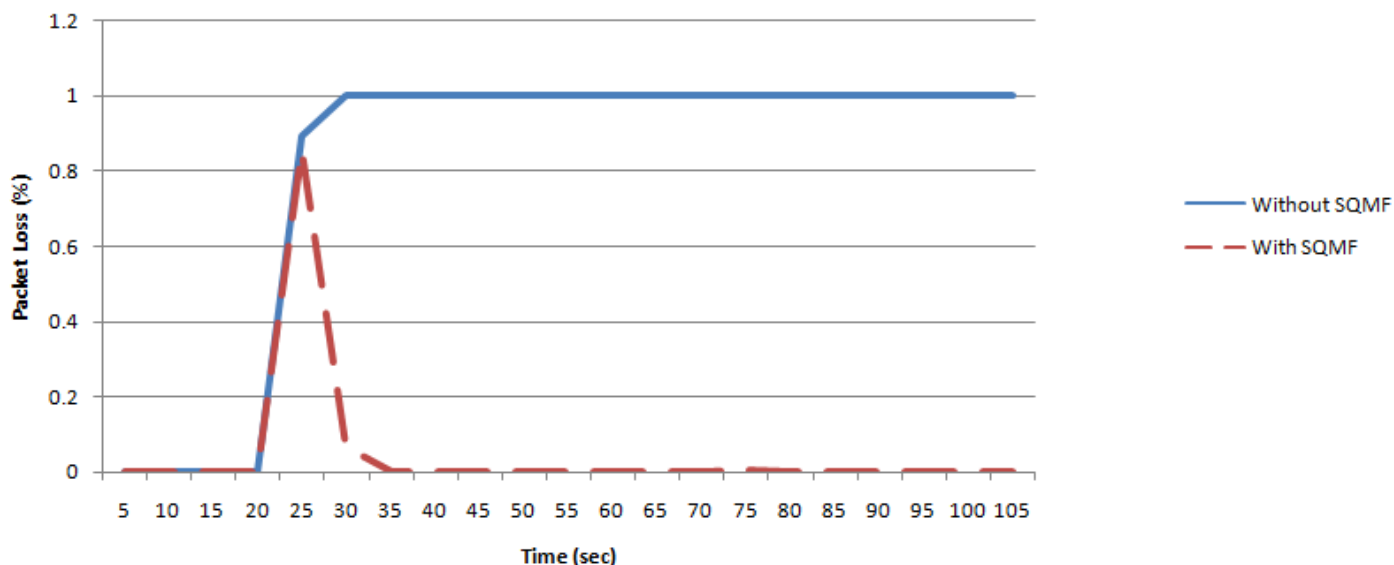


Figure 7.5: Packet loss comparison between cases with and without SQMF in video streaming, when a link failure occurs

Therefore, the QoE monitoring-based forwarding preserves the total QoE and keeps it at high levels even after the link failure, whereas in the default case the QoE faces a permanent degradation after the failure, as depicted in Figure 7.6.

Video Quality due to link failure with and without SQMF for Video Streaming

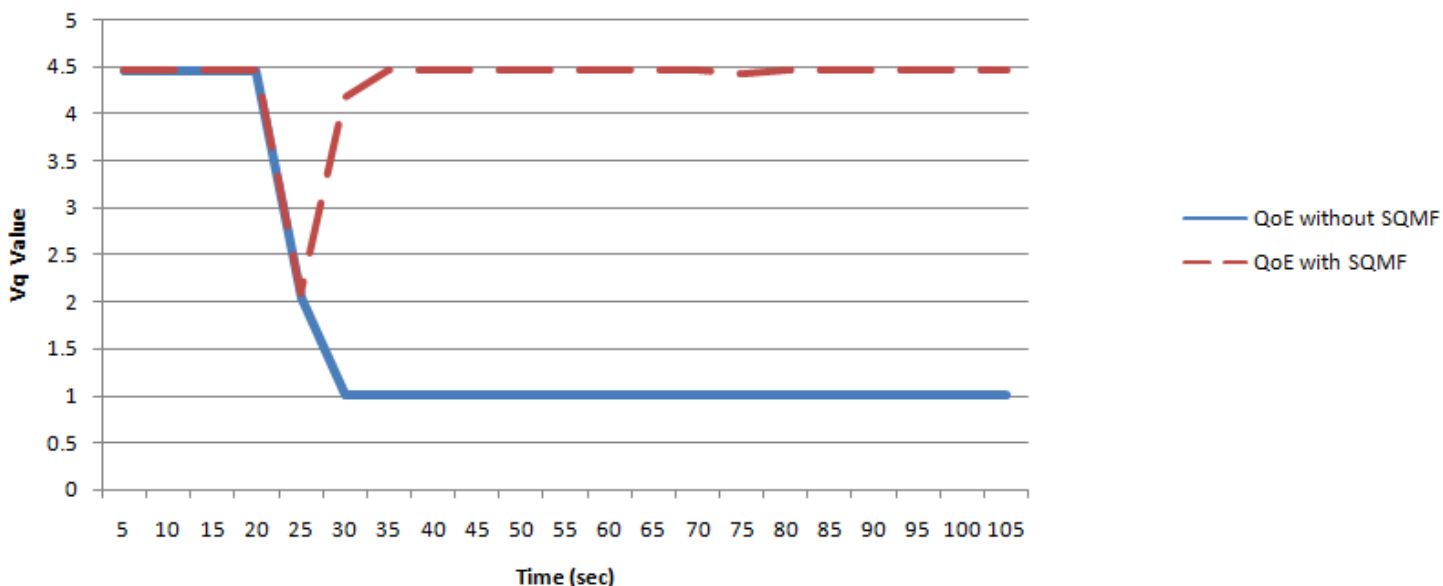


Figure 7.6: Vq comparison between cases with and without SQMF in video streaming, when a link failure occurs

8. CONCLUSION AND FUTURE WORK

This MSc thesis presents the current networking state which is characterized by the explosion of mobile devices and content, server virtualization, and advent of cloud services. Also, it emphasizes that the design of conventional networks is hierarchical, leading to a static architecture which cannot deal with the dynamic computing and storage needs of today's computing environments. The current networks' limitations raise the need for an alternative approach to effectively face these challenges.

This alternative approach is SDN, which decouples the control from the data plane and transforms the network elements to simple forwarding devices, routing the traffic according to rules set to them by the control plane. SDN has been described in terms of architecture, controller, dominant protocol, use cases, advantages and disadvantages.

As in any technology, a very important factor in the evaluation of SDN is the user's satisfaction from the service offered to him, or the QoE, as the appropriate term is. Quality assessment schemes act as translator between a set of technical (QoS) and non-technical (subjective and contextual) key influence factors and user perception, or ultimately, user experience. These can be categorized into subjective and objective quality assessment methods, depending on whether human subjects are involved in the assessment process or not.

To this end, this thesis has developed an SDN framework for monitoring the QoE of VoIP and video applications in real-time that manages to preserve QoE at acceptable levels, despite sudden network problems, such as link failures. This is achieved by periodically monitoring the necessary QoE-related parameters from the network, evaluating the QoE in real time and changing the transmission path in the case that low QoE is detected, as specified by a threshold. This mechanism ensures that the packets will always be transmitted through a path which preserves an acceptable QoE level.

The implemented framework has been presented and evaluated, resulting in much lower packet losses than the default forwarding, and therefore to much higher QoE. This thesis' contribution goes far beyond an abstract framework introduction, as it provides a practical implementation of *real-time QoE monitoring* in SDNs by using real QoE estimation models. It also describes in detail the implementation steps, which can be replicated by any researcher.

Some interesting issues that have been identified as future work points are the following:

- The extension of SQMF to provide more than one backup paths, in case that the QoE is detected again low in the (first) backup path
- The experimentation with different kinds of network problems, not related to link failure (e.g. heavy congestion)
- The dynamic coefficients computation for the video QoE estimation model, so that the application can additionally deal with more challenging video types.
- The integration of different QoE models into SQMF, related to other application types such as TCP-based video streaming, web browsing and IPTV.

ABBREVIATIONS - ACRONYMS

1G	First Generation
2G	Second Generation
4G	Fourth Generation
5G	Fifth Generation
ABR	Adaptive Bit Rate
A-CPI	Application-Control plane Interface
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPU	Average Revenue Per User
ASIC	Application-specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BYOD	Bring Your Own Device
CAGR	Compound Annual Growth Rate
CAPEX	Capital Expenditure
CDN	Content Delivery Network
CLI	Command Line Interface
CRUD	Create, Retrieve, Update, Delete
D-CPI	Data-Control plane Interface
D-ITG	Distributed Internet Traffic Generator
DOM	Document Object Model
DPI	Deep Packet Inspection
DR	Decay Rate
DSL	Domain-Specific Language
ForCES	Forwarding and Control Element Separation
FR	Full Reference
HAS	HTTP Adaptive Streaming
HD	High Definition
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IDE	Integrated Development Environment
IDL	Interface Description Language
IETF	Internet Engineering Task Force

IF	Influence Factor
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
IPTV	Internet Protocol Television
IPv4 / v6	Internet Protocol version 4 / version 6
ITU-T	International Telecommunications Unit – Telecommunications Standardization Sector
JVM	Java Virtual Machine
KPI	Key Performance Indicator
KQI	Key Quality Indicator
LARAC	Lagrange Relaxation-based Aggregation Cost
LTE	Long Term Evolution
LTE-A	Long Term Evolution - Advanced
M2M	Machine - to - Machine
MDP	Markov Description Process
MD-SAL	Model-Driven Service Abstraction Layer
MOS	Mean Opinion Score
MPD	Media Presentation Description
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation
NBI	Northbound Interface
NE	Network Element
NETCONF	Network Configuration
NFV	Network Functions Virtualization
NFVI-PoP	NFV Installation - Point of Presence
NOS	Network Operating System
NR	No Reference
OF	OpenFlow
ODL	OpenDaylight
OM	OpenFlow Module
ONF	Open Networking Foundation
OPEX	Operational Expenditure
OS	Operating System
OTT	Over the Top

OVS	Open vSwitch
PAF	Path Assignment Function
PCE	Path Computation Element
QFF	QoE Fairness Framework
QMOF	QoS Matching and Optimization Function
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RESTCONF	Representational State Transfer Configuration
RPC	Remote Procedure Call
RR	Reduced Reference
RSU	Road-Side Unit
SAL	Service Abstraction Layer
SBI	Southbound Interface
SD	Standard Definition
SDN	Software Defined Networking / Software Defined Network
SDWN	Software Defined Wireless Networking / Software Defined Wireless Network
SPI	Stateful Packet Inspection
SQMF	SDN QoE Monitoring Framework
SVC	Scalable Video Coding
SLA	Service License Agreement
SRT	Statistics Retrieval Time
TCP	Transmission Control Protocol
TSP	Tunnel Setup Protocol
UHD	Ultra High Definition
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VoD	Video on Demand
VoIP	Voice over IP
VQA	Video Quality Application
VQAM	Video Quality Assurance Manager
VQO	Video Quality Orchestrator
VSP	Video Service Provider

WAN	Wide Area Network
WAP	Wireless Access Point
WWRF	Wireless World Research Forum
YANG	Yet Another Next Generation

ANNEX

The SQMF implementation, as well as instructions to download and execute it, are available in a public Github repository at the following URL:

<https://github.com/marievixezonaki/SQMF>

REFERENCES

- [1] L. Sørensen, K. Skouby, "Visions and research directions for the Wireless World", July 2009, pp. 5-9.
- [2] C. Wang et al., "Cellular Architecture and Key Technologies for 5G Wireless Communication Networks", IEEE Communications Magazine, February 2014
- [3] European Commission, Communication from the Commission to the European Parliament, the council, the European Social and Economic committee and the committee of the regions, "Exploiting the employment potential of ICTs", April 2012, p.3.
- [4] "SDN: Transforming Networking to Accelerate Business Agility", <http://opennetsummit.org/archives/mar14/site/why-sdn.html>, March 2014. [Accessed 04/05/2017]
- [5] B. Sotirov, "Why Software Defined Networking (SDN)?", <https://www.slideshare.net/lz1dsb/why-sdn>, 2015.
- [6] D. Kreutz et al., "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE , Volume 103, Issue 1, January 2015.
- [7] D. Athow, "Determining the need for Software Defined Networking", <http://www.techradar.com/news/networking/determining-the-need-for-software-defined-networking-1253463>, June 2014. [Accessed 04/04/2017]
- [8] J. du Toit, "The current state of networking: pitfalls of modern network systems", <https://www.irisns.com/the-current-state-of-networking-pitfalls-of-modern-network-systems>, October 2015. [Accessed 04/05/2017]
- [9] "Why SDN or NFV Now?", <https://www.sdxcentral.com/sdn/definitions/why-sdn-software-defined-networking-or-nfv-network-functions-virtualization-now> [Accessed 04/05/2017]
- [10] CISCO, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019", February 2015.
- [11] "Software Defined Networking", https://en.wikipedia.org/wiki/Software-defined_networking, 2017. [Accessed 04/05/2017]
- [12] J. Porras et al., "User 2020 –A WWRF Vision", September 2014, p. 4.
- [13] J. Andrews et al., "What Will 5G Be?", IEEE JSAC SPECIAL ISSUE ON 5G WIRELESS COMMUNICATION SYSTEMS, May 2014.
- [14] Nokia Solutions and Networks, "White Paper: 5G use cases and requirements", July 2014.
- [15] W. Tong, Z. Peiyong "White Paper: 5G: A Technology Vision", Huawei, 2013.
- [16] C. I et al., "Toward Green and Soft: A 5G Perspective", IEEE Communications Magazine, Volume 52 , Issue 2, February 2014.
- [17] G. Wunder et al., "5GNOW: Non-Orthogonal, Asynchronous Waveforms for Future Mobile Applications", IEEE Communications Magazine, Volume 52, Issue 2, February 2014.
- [18] Ericsson Review, "5G Radio Access", The communications technology journal since 1924, June 2014.
- [19] H. Koumaras et al., "Enabling Agile Video Transcoding over SDN/NFV-enabled Networks", International Conference on Telecommunications and Multimedia (TEMU), July 2016.
- [20] O. Awobuluyi et al., "Video Quality in 5G Networks: Context-Aware QoE Management in the SDN Control Plane", IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, October 2015.
- [21] H. Liu et al., "Software Defined Networking for HTTP Video Quality Optimization", 15th IEEE International Conference on Communication Technology (ICCT), November 2013.
- [22] A. Kessler et al., "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking", 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2012.
- [23] S. Ramakrishnan and X. Zhu, "An SDN Based Approach To Measuring And Optimizing ABR Video Quality Of Experience", Cisco Systems, 2014.
- [24] W. Hsu et al., "The Implementation of a QoS/QoE Mapping and Adjusting Application in software-defined networks", 2nd International Conference on Intelligent Green Building and Smart Grid (IGBSG), 2016.
- [25] C. Hue et al., "Traffic-aware Networking for Video Streaming Service Using SDN", IEEE 34th International Performance Computing and Communications Conference (IPCCC), December 2015.
- [26] Open Networking Foundation (ONF), "Software-Defined Networking (SDN) Definition", <https://www.opennetworking.org/sdn-resources/sdn-definition>, 2017. [Accessed 04/04/2017]
- [27] Open Networking Foundation (ONF), "SDN Architecture", https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf , Issue 1.0. [Accessed 22/05/2017]
- [28] M. Jarschel et al., "SDN-based Application-Aware Networking on the Example of YouTube Video Streaming", 2nd European Workshop on Software Defined Networks (EWSN), September 2013.

- [29] T. Yu et al., "Adaptive Routing for Video Streaming with QoS Support over SDN Networks", International Conference on Information Networking (ICOIN), January 2015.
- [30] P. Georgopoulos et al., "Towards Network-wide QoE Fairness Using OpenFlow-assisted Adaptive Video Streaming", ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking, August 2013.
- [31] P. Georgopoulos et al., "Using Software Defined Networking to enhance the delivery of Video-on-Demand", Computer Communications by Elsevier, September 2015.
- [32] Σ. Γ. Μαστοράκης, "Μέθοδοι εξουσιοδότησης για δέσμευση πόρων σε Ευφυή – Προγραμματιζόμενα - Δίκτυα (Software-Defined-Networks)", May 2014.
- [33] T. Zinner et al., "Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges", IEEE Network Operations and Management Symposium (NOMS), May 2014.
- [34] "OpenDaylight: Open Source SDN Platform", <https://www.opendaylight.org>. [Accessed 21/06/2017]
- [35] "What is an OpenDaylight Controller? AKA: OpenDaylight Platform", <https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller>. [Accessed 21/06/2017]
- [36] "OpenDaylight Project", https://en.wikipedia.org/wiki/OpenDaylight_Project. [Accessed 21/06/2017]
- [37] L. Efremova, "What's in OpenDaylight?", <https://www.mirantis.com/blog/whats-opendaylight>, April 2015. [Accessed 21/06/2017]
- [38] "OpenDaylight Application Developer's Tutorial", <http://sdnhub.org/tutorials/opendaylight/>, 2014. [Accessed 07/07/2017]
- [39] "OpenDaylight Controller: MD-SAL: MD-SAL Document Review: Architecture", https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_Document_Review:Architecture. [Accessed 21/06/2017]
- [40] G. Hunt, "The Quest for Dominance: OpenFlow or NETCONF for Networks Outside the Data Center?", <https://networkmatter.com/2015/02/06/the-quest-for-dominance-openflow-or-netconf-for-networks-outside-the-data-center>, February 2015. [Accessed 21/06/2017]
- [41] S. Wallin, "Tutorial: Netconf and YANG", May 2013.
- [42] "What is NETCONF", <http://sdntutorials.com/what-is-netconf>. [Accessed 21/06/2017]
- [43] "NETCONF Overview", <http://www.brocade.com/content/html/en/user-guide/SDN-Controller-2.1.0-User-Guide/GUID-9E240EBE-223E-4E48-823B-70455037C908.html>. [Accessed 21/06/2017]
- [44] CloudeNablers Inc., "OpenDaylight and YANG", <https://www.slideshare.net/CloudeNablers/opendaylight-and-yang/4>, April 2016. [Accessed 21/06/2017]
- [45] "YANG", <https://en.wikipedia.org/wiki/YANG>. [Accessed 21/06/2017]
- [46] T. Oliveky, "OpenDaylight, Netconf, Restconf & YANG", <http://sdntutorials.com/opendaylight-netconf-restconf-and-yang>. [Accessed 21/06/2017]
- [47] M. Saleem, "SOFTWARE DEFINED NETWORK: USE CASES FROM THE REAL WORLD", <http://www.routerfreak.com/software-defined-network-use-cases-from-the-real-world>, July 2016. [Accessed 07/06/2017]
- [48] M. McNickle, "Five SDN use cases: From video to service orchestration", <http://searchsdn.techtarget.com/news/2240187268/Five-SDN-use-cases-From-video-to-service-orchestration>, July 2013. [Accessed 07/06/2017]
- [49] S. Katukam, "Six Campus Networks SDN Use Cases That You Need to Know About", <https://www.sdxcentral.com/articles/contributed/sdn-use-cases-campus-networks/2013/07/>, July 2013. [Accessed 07/06/2017]
- [50] "SDN & NFV Use Cases Defined", <https://www.sdxcentral.com/sdn-nfv-use-cases>. [Accessed 07/06/2017]
- [51] A. Hakiri et al., "Software-defined Networking: Challenges and Research Opportunities for Future Internet", 2014.
- [52] S. Perrin, "Defining Use Cases & Business Cases for SDN", <http://www.lightreading.com/carrier-sdn/sdn-architectures/defining-use-cases-and-business-cases-for-sdn/a/d-id/716315>, June 2015. [Accessed 07/06/2017]
- [53] J. Cottrell, "What Is SDN? Why Is It Important?", <https://www.mirazon.com/sdn-important>, March 2014. [Accessed 04/05/2017]
- [54] V. Shamugam et al., "Software Defined Networking challenges and future direction: A case study of implementing SDN features on OpenStack private cloud", March 2016.
- [55] A. Shapochka, "4 Challenges Lying in the Wait of SDN", <http://www.nojitter.com/post/240169834/4-challenges-lying-in-the-wait-of-sdn>, April 2015. [Accessed 08/06/2017]
- [56] V. Shamugam et al., "Software Defined Networking challenges and future direction: A case study of implementing SDN features on OpenStack private cloud", <http://iopscience.iop.org/article/10.1088/1757-899X/121/1/012003/pdf>, 10th Curtin University of Technology Science and Engineering International Conference (CUTSE), November 2015.

- [57] "SDN Security Challenges in SDN Environments", <https://www.sdxcentral.com/security/definitions/security-challenges-sdn-software-defined-networks>. [Accessed 08/06/2017]
- [58] V. Bakalov, "Opportunities and Challenges with SDN", <http://www.networkworld.com/article/2973610/software-defined-networking/opportunities-and-challenges-with-sdn.html>, August 2015. [Accessed 08/06/2017]
- [59] A. Farshad et al., "Leveraging SDN to Provide an In-network QoE Measurement Framework", IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), May 2015.
- [60] M. Fiedler et al., "QoE-based Cross-Layer Design of Mobile Video Systems: Challenges and Concepts", 2009 IEEE-RIVF International Conference on Computing and Communication Technologies, July 2009.
- [61] R. Schatz et al., "From Packets to People: Quality of Experience as a New Measurement Challenge", Lecture Notes in Computer Science, Springer, pp 219-263.
- [62] V. Beal, "QoE", <http://www.webopedia.com/TERM/Q/QoE.html>. [Accessed 22/06/2017]
- [63] "Quality of Experience (QoE)", <https://www.techopedia.com/definition/25802/quality-of-experience-qoe>. [Accessed 22/06/2017]
- [64] "Quality of experience", https://en.wikipedia.org/wiki/Quality_of_experience. [Accessed 22/06/2017]
- [65] S. Barakovic and L. Skorin-Kapov, "Survey and Challenges of QoE Management Issues in Wireless Networks", <https://www.hindawi.com/journals/jcnc/2013/165146>, Journal of Computer Networks and Communications Volume 2013, 2013.
- [66] R. G. Cole and J. H. Rosenbluth, "Voice over IP Performance Monitoring", ACM SIGCOMM Computer Communication Review, Volume 31, Issue 2, pp. 9 – 24, April 2001.
- [67] INTERNATIONAL TELECOMMUNICATIONS UNION, TELECOMMUNICATIONS STANDARDIZATION SECTOR, "Opinion model for video-telephony applications", <https://www.itu.int/rec/T-REC-G.1070>, July 2012.
- [68] M. Fiedler et al., "A generic quantitative relationship between Quality of Experience and Quality of Service", IEEE Network, Volume: 24, Issue: 2, March-April 2010
- [69] G. Agapiou, I. Papafili, S. Agapiou, "The role of SDN and NFv for dynamic bandwidth allocation and QoE adaptation of video applications in home networks", Euro Med Telco Conference (EMTC), November 2014.
- [70] S. Ramakrishnan et al., "SDN Based QoE Optimization for HTTP-Based Adaptive Video Streaming", IEEE International Symposium on Multimedia (ISM), December 2015.
- [71] E. Bozkaya and B. Canberk, "QoE-based Flow Management in Software Defined Vehicular Networks", IEEE Globecom Workshops (GC Wkshps), December 2015.
- [72] I. Mustafa and T. Nadeem, "Dynamic Traffic Shaping Technique for HTTP Adaptive Video Streaming using Software Defined Networks", IEEE International Conference on Sensing, Communication and Networking (SECON), June 2015.
- [73] Z. Zhang et al., "Joint Resource Allocation and Traffic Management for Cloud Video Distribution over Software-Defined Networks", 8th IEEE International Conference on Communication Software and Networks (ICCSN), June 2016.
- [74] Y. Yue et al., "Joint Routing and Layer Selecting for Scalable Video Transmission in SDN", IEEE Globecom Workshops (GC Wkshps), December 2015.
- [75] L. Kuang et al., "A Tensor-Based Big Data Model for QoS Improvement in Software Defined Networks", IEEE Network, Volume: 30, Issue: 1, January-February 2016.
- [76] "OpenDaylight User Guide", <https://www.opendaylight.org/sites/opendaylight/files/bk-user-guide.pdf>, 2016, p.2.
- [77] "Release Notes", http://docs.opendaylight.org/en/stable-boron/getting-started-guide/release_notes.html, 2016. [Accessed 20/3/2017].
- [78] "Getting started: Development Environment Setup", https://wiki.opendaylight.org/view/GettingStarted:Development_Environment_Setup [Accessed 20/3/2017].
- [79] "Installing Apache Maven", <https://maven.apache.org/install.html>, March 2017. [Accessed 20/3/2017]
- [80] "CrossProject: Helium Release Vehicle Brainstorming: Pure Karaf", https://wiki.opendaylight.org/view/CrossProject:Helium_Release_Vehicle_Brainstorming:Pure_Karaf [Accessed 20/3/2017]
- [81] "KAR", <https://karaf.apache.org/manual/latest/kar>, April 2016. [Accessed 20/3/2017]
- [82] "Provisioning", <https://karaf.apache.org/manual/latest/provisioning>, April 2016. [Accessed 20/3/2017]
- [83] "Using the OpenDaylight User Interface (DLUX)", <http://docs.opendaylight.org/en/stable-boron/user-guide/using-the-opendaylight-user-interface-dlux.html>, 2016. [Accessed 20/3/2017]
- [84] "YANG User Interface (YANGUI) in OpenDaylight", <http://events.linuxfoundation.org/sites/events/files/slides/YANGUI-metz-malachovsky-sebin-ODL-Summit-final-July29.pdf>, p. 1-10.

- [85] "Download/Get Started with Mininet", <http://mininet.org/download/>, 2017. [Accessed 20/3/2017]
- [86] "Mininet VM Setup Notes", <http://mininet.org/vm-setup-notes/>, 2017. [Accessed 20/3/2017]
- [87] "Installing new version of Open vSwitch", <https://github.com/mininet/mininet/wiki/Installing-new-version-of-Open-vSwitch>, February 2015. [Accessed 19/09/2017]
- [88] "Mininet Walkthrough", <http://mininet.org/walkthrough/>, 2017. [Accessed 20/3/2017]
- [89] "Getting Started: Development Environment Setup", https://wiki.opendaylight.org/view/GettingStarted:Development_Environment_Setup#Edit_your_.7E.2F.m2.2Fsettings.xml. [Accessed 10/07/2017]
- [90] "OpenDaylight Controller: MD-SAL: Startup Project Archetype", https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Startup_Project_Archetype. [Accessed 10/07/2017]
- [91] A. Botta, A. Dainotti, A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios", Computer Networks (Elsevier), 2012, Volume 56, Issue 15, pp 3531-3547.
- [92] A. Botta, W. Donato, A. Dainotti, S. Avallone, A. Pescapè, "D-ITG 2.8.1 Manual", <http://traffic.comics.unina.it/software/ITG/manual/index.html>, October 2013. [Accessed 29/06/2017]
- [93] A. Iqbal, "Using D-ITG Traffic Generator in Mininet", <http://sdnopenflow.blogspot.gr/2015/05/using-of-d-itg-traffic-generator-in.html>, May 2015. [Accessed 29/06/2017]
- [94] "ffmpeg: command not found", <https://askubuntu.com/questions/699502/ffmpeg-command-not-found>, November 2015. [Accessed 02/10/2017]
- [95] "x264 FFmpeg Options Guide", <https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping>. [Accessed 02/10/2017]