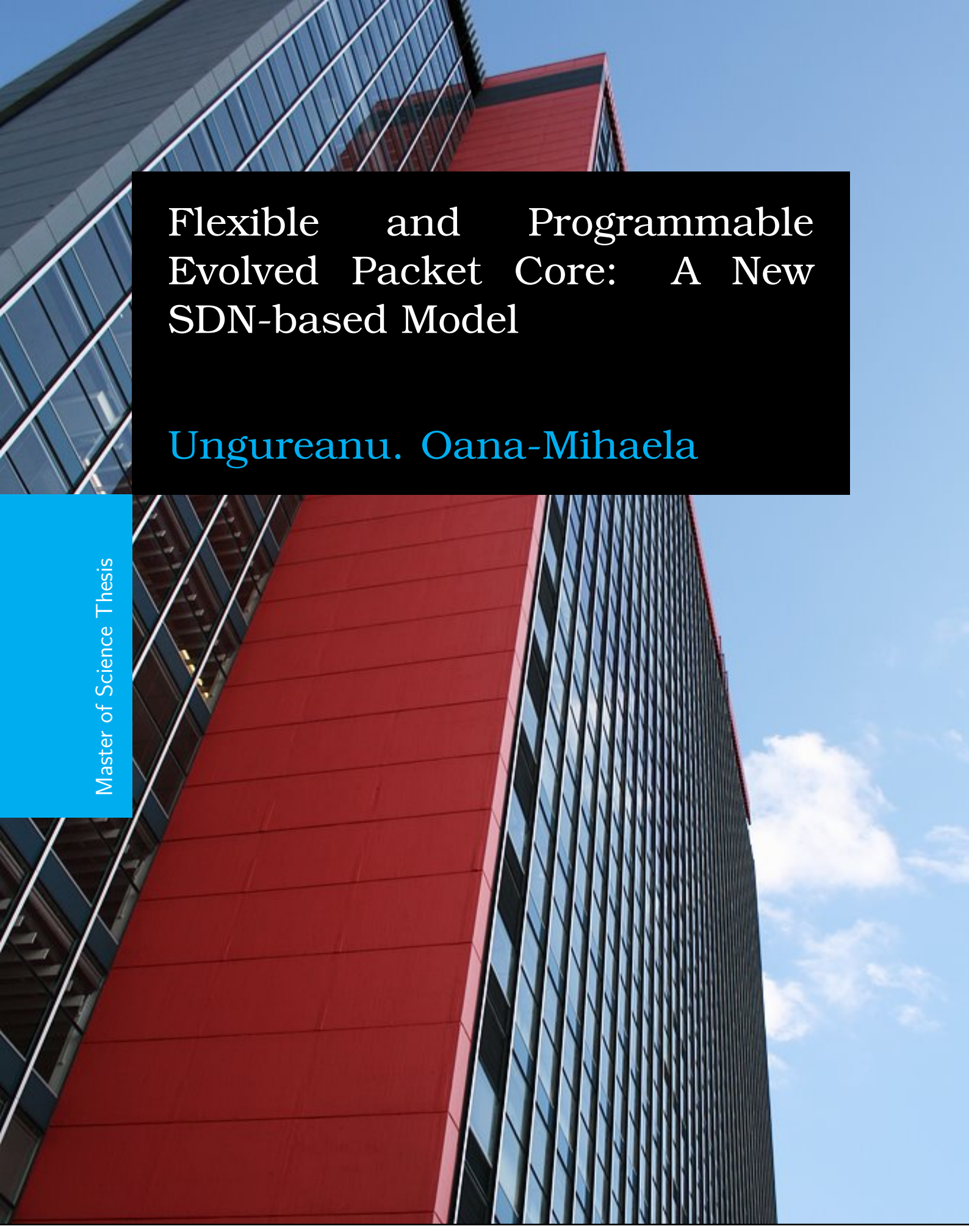


# Flexible and Programmable Evolved Packet Core: A New SDN-based Model

Ungureanu. Oana-Mihaela

Master of Science Thesis

TU Delft Delft University of Technology

Network Architectures and Services (NAS) Group

# Flexible and Programmable Evolved Packet Core: A New SDN-based Model

Master of Science Thesis

For the degree of Master of Science in Electrical Engineering,
Telecommunications track at Delft University of Technology

Ungureanu. Oana-Mihaela

July 7, 2014

Faculty of Electrical Engineering, Mathematics and Computer Science (EWI) · Delft University of Technology

The work in this thesis was supported by Koninklijke KPN N.V. and Telecom Bretagne. Their cooperation is hereby gratefully acknowledged.



Delft University of Technology
Department of
Network Architectures and Services (nas)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science (EWI) for acceptance a thesis entitled

Flexible and Programmable Evolved Packet Core: A New SDN-based Model

by

Ungureanu. Oana-Mihaela

in partial fulfillment of the requirements for the degree of

Master of Science Electrical Engineering, Telecommunications track

Dated: July 7, 2014

Supervisors:

prof.dr.ir. Robert Kooij

Dr.ir. Jos Adema

Committee:

prof.dr.ir. Robert Kooij

dr.ir. Jos Adema

prof.dr.ir. Jos Weber

# Abstract

The mobile core network experienced many transformations during the past decade and at this moment virtualisation constitutes the next big phase in the operators strategies. Another technology which might step in the coming years is Software Defined Networking (SDN), a more mature technology in data centers, which strives to revolutionize the current Long-Term Evolution (LTE) architecture. However, the integration of SDN components into the legacy network and the transition between phases need to be carefully addressed.

This thesis aims to develop a new Evolved Packet Core (EPC) (the core network corresponding to the LTE radio network) SDN-based architecture and to take into consideration the challenges that this technology might bring to the mobile core. Therefore, a thorough assessment of the interfaces and protocols that should be preserved or modified on both the user and control planes is investigated.

By describing the simulation details of the proposed architecture, one of this thesis goals is to present the trade-off between preserving the traditional 3rd Generation Partnership Project (3GPP) architecture and the actual benefits, along with the challenges for mobile operators in adopting SDN and Network Function Virtualisation (NFV) in their deployments. To achieve this, two testbeds are used: one based on a network emulation tool and another one using physical elements. The obtained results emphasize the flexibility in deploying such solutions. Furthermore, a detailed comparison of two SDN controllers performances is presented.

In order to fully benefit of the flexibility of NFV and SDN in their networks, mobile operators should have an overview of the different tools and technologies, which might allow SDN integration in their networks. Another thesis goal is to provide the necessary information in order to give operators the input for building their roadmap to the upcoming technologies.

# Contents

# List of Figures

# List of Tables

# Preface

This document is part of my Master of Science Graduation Thesis. The idea of adopting this subject emerged during the summer internship at KPN Mobile, in the Innovation Department. The major telecom events and presentations (i.e. "LTE World Summit 2013", "IP Gala", etc.) triggered my interest in the area of SDN and NFV. The discussions with the core architects, followed by vendors presentations helped me to take a decision over where can these technologies be applied in the network.

Therefore, I would like to express my gratitude to my daily supervisor at KPN, dr.ir. Jos Adema, who constantly supported and advised me over my work. During our regular meetings, he tried to understand my explanations and always gave me very good feedback. I would also like to thank to my colleagues at KPN for their unconditioned help and interest in my work.

The physical testbed carried in this paper was set up using hardware held by SURFnet company, in Amsterdam. Hereby, I am very thankful to ir. Ronald van der Pol for allowing me to build up this testbed and especially for his interest and patience in answering my questions.

The internship at Telecom Bretagne in France gave me the insight into the most recent developments and challenged me to improve my work. Moreover, I gained experience in the interaction with different operators' perspectives (Orange France and KPN) regarding SDN and NFV. I am very thankful to professors from Telecom Bretagne, prof. Xavier Lagrange and prof. Gwendal Simon, who made this possible, for their preoccupation and advice in my approach. Furthermore, I am very grateful to the Ph.D. student from Orange Labs, Malla Reddy Sama, for his good comments and willingness to help me whenever I faced difficulties.

I would like to thank to my coordinator from TU Delft, prof. dr. Robert Kooij for his constant encouragement, understanding and help whenever I asked for. It would not have been possible for me to complete this thesis without his guidance.

Finally, I am grateful to my family, especially my mother and my grandmother for their love, moral and financial support throughout the years. Without them, none of my achievements so far would have been possible.

"Make everything as simple as possible, but not simpler".

— *Albert Einstein*

"Once a new technology rolls over you, if you're not part of the steamroller, you're part of the road".

— *Stewart Brand*

# Chapter 1

# Introduction

## 1-1 Motivation

Mobile broadband networks are expected to experience significant data traffic growth in the future. It is claimed that the current centralized network architecture will face excessive traffic concentration on a single gateway element and possible unoptimized routing. The statistics presented in [17] showed that in 2013 mobile data traffic increased by 18 times the size of the entire global Internet in 2000 (i.e. nearly 18 Exabytes of traffic), whereas mobile video traffic exceeded 50 % of total traffic for the first time in 2012. Moreover, the Cisco's above mentioned forecast estimates that in 2018, the total amount of mobile traffic per month will be almost 15.9 Exabytes. According to [17], almost 526 millions of mobile devices and connections were added in 2013 and furthermore, it is predicted that the mobile devices and connections will grow globally up to 10.2 billion by 2018. The Fourth Generation (4G) connections produced in 2013 nearly 14.5 times more traffic than non-4G connections. Even though 4G connections only account for 2.9 % of mobile connections today, they generate 30 % of mobile traffic.

The emerging increase in data traffic has been one of the most concerning threats for mobile operators in the past years due to the multitude of bandwidth hungry consuming applications, especially in the area of real-time entertainment and video content. The exponential data growth and its unpredictability triggered their interest to look beyond 3rd Generation Partnership Project (3GPP) proposals, which usually imply over-provisioning the network under high costs within long standardization periods, on short term optimizations. Recently, standardization bodies such as Network Function Virtualisation (NFV) Industry Specification Group (ISG) under the European Telecommunications Standards Institute (ETSI) banner [18] focus on operators' demands based on cost effective solutions, which can also bring deployment flexibility in the network architecture under very short time-to-market releases. The ETSI committee gathered operators and vendors to test a series of Proof of Concept (PoC)s [19] with the aim to embrace NFV in their live networks. Although it is a very recent technology, NFV rapidly gained operators' trust and has already become a fact in many vendors solutions.

The NFV technology allows operators to cope with the increasing traffic demands by employing a more flexible network management (easily adapt to software upgrades) and better resource utilization (lowering both Capital Expenditure (CAPEX) and Operating Expenditure (OPEX)). Another benefit of NFV is usage of less hardware with the elasticity of configuring new services, running tests and production on the same infrastructure. It is very interesting to observe the multitude of similarities between Cloud Computing and NFV architecture, in the sense that Network as a Service (NaaS) can be seen as equivalent of Infrastructure as a Service (IaaS) in Cloud computing, whereas Virtual Network Function (VNF) is running on top of it at the same level with Cloud Applications [19].

The NFV came to light at the OpenFlow / Software Defined Networking (SDN) World Congress in Darmstadt, Germany in October 2012, when seven of the largest operators (i.e. Deutsche Telekom, Orange, British Telecom, Telecom Italia, Telefonica, Verizon and China Mobile) published a white paper on NFV and then ETSI Board approved the creation of the Network Functions Virtualisation Industry Specification Group (NFV ISG) in November 2012. The outcome of this meeting was paper [20], which gathers the benefits and the NFV use cases in relation with SDN (Software Defined Networking), a prior technology emerged in data centers.

In contrast to NFV, both OpenFlow protocol and SDN concept emerged in an academic environment in 2007 as a collaboration project between Stanford University and University of Berkeley in USA. Vendors and large enterprises started to produce the equipment and firstly implement SDN in 2011. For example, Google built its own SDN switches and was the first to adopt a global software-driven network [21]. In the same year, 2011, the Open Networking Foundation (ONF) organization has been formed with the purpose of SDN standardization in different group areas.

The SDN is referred as a complementary technology to NFV [20], because it can provide the infrastructure upon which VNF software can run. SDN mainly enhances the benefits of splitting the data from the control plane. Furthermore, it provides better access to real-time information, fast optimized routing and improved user experience. The standardized OpenFlow protocol [8] defines a model of how the traffic is divided into flows and how these flows can be managed remotely by a central controller.

In the next years, network operators will have to adopt a strategy in the evolution of their cellular core network. The most conservative strategy is to preserve the current 3GPP protocols, with arguably successful deployment and limited capabilities to cope with the demands. A more attractive solution can consist of the aforementioned technologies, NFV and SDN, which might allow operators to deploy more services in a more flexible manner with less costs and resources, assuring them a long-term solution compared to the current 3GPP shortages. Nevertheless, there are many open questions that compromises operators decision to fully embrace these technologies, even though they might fully satisfy their requirements.

For traditional major operators like KPN and Orange France, the NFV / SDN approach would imply important changes in their architecture and removal of some legacy hardware. The integration of a new Evolved Packet Core (EPC) SDN-based solution might come up in several swapping phases and requires careful planning. On the other hand, for new emerging market operators, SDN and NFV are considered suitable candidates in terms of cost and easier geographical extension, as well as on-demand capacity fulfillment.

## 1-2 Problem Statement

Nowadays, the Long-Term Evolution (LTE) / EPC architecture experiences a period of rapid and massive changes. It will undergo extensive scale usage as it connects more devices, applications and network traffic. Consequently, the network architecture does not provide any elasticity and on-demand features. The network is typically dimensioned based on the load foreseen in the peak hours. For instance, the mobile network entities in EPC (i.e. Mobility Management Entity (MME), Serving Gateway (SGW), Packet Data Network Gateway (PGW) and Home Subscriber Server (HSS)) are based on custom hardware and need to be statically provisioned and configured. Therefore, to increase the network capacity, it requires the deployment of new entities in specific network sites. Hereby, the operation of such a static network is a costly, cumbersome and time-consuming process.

The NFV and SDN technologies aim to solve these issues, thus a proper integration of these technologies in the existing EPC should be carefully analyzed. One of this thesis goals was to make an analysis of how these concepts will steer the transformation to an SDN/NFV enabled carrier network and propose a new EPC SDN/ NFV-based architecture. In order to be able to adopt such strategies, network operators are seeking to fully understand SDN and NFV benefits along with their impediments. The first step in their approach is to investigate where these two technologies best fit in their network (ranging from access, mobile backhaul, core or service chaining) and adjust their advantages where 3GPP solutions are very complex and less cost-effective. The main issues in current EPC deployment are the high costs, the load balancing in real time and the Quality of Service (QoS) control of different flows. In this context, the following questions arise:

- How can operators maximize their benefits by combining NFV with SDN and in which parts of the network can these technologies be deployed?
- How can SDN / NFV be integrated in the current 3GPP architecture, what are the critical interfaces that should be preserved and where can the extra flexibility be added?
- Apart from lowering the costs, could these technologies provide more reliable traffic management and better QoS?
- What tools can be used and how these tools would affect the network's performances?

To support the ever-growing demands for mobile traffic, many operators try to provide more capacity and over-dimension their network that will lead to a huge increase in CAPEX and OPEX. The most expensive is the telecom infrastructure, though a key strategy to reduce the costs is the virtualisation of the legacy hardware, when this solution is possible. Virtualisation technology allows to have multiple functions running on a single platform and provides the required capacity in a flexible manner. Therefore, it would be easier to scale up both horizontally and vertically the resources while decreasing the cost, power consumption and network size. Carriers need to periodically upgrade their infrastructure in order to keep up with the rapid technology evolution or adopt new services. The costly upgrades and legacy hardware can be easily replaced by virtualized systems, which enable the reuse of legacy software on newer hardware. Another common issue for most of the operators is the maintenance cost of the equipment in several deployment centers, usually due to geo-redundancy

reasons. Using virtualisation techniques, the infrastructure can usually be kept only in one location such as *carrier hoteling* or *collocation*. This is a very common procedure adopted by telecom industry to share resources among each other in a common building and flexibly control those resources, very similar to the "cloud" model. The burden of extra hardware could easily eliminated in this manner, though several considerations are introduced before this choice: a central point of failure in the network and the security issues. Nevertheless, the NFV and SDN technologies also offer solutions to these issues: configuration of monitoring statistics (in the SDN controller), virtualisation back-up mechanisms, Distributed Denial of Service Attacks (DDoS) protection or both hardware and software security isolation.

The SDN technology also provides better cost reduction in terms of the used hardware and extra network "intelligence". For instance, the replacement of mobile gateways with OpenFlow switches might save millions of Euros. SDN's most attractive benefit is not mainly the cost reduction, but the "intelligent" way of performing routing, controlling the load, resiliency and robustness and especially the appliance of different traffic treatment.

As an example, the current load balancing mechanism in the mobile core network is done proactively and does not take into account the instantaneous load or capacity of the gateways (i.e. offline weight factors), which might lead to overload in one or more nodes of a gateway cluster, while there is still remaining capacity on other nodes of the same cluster. The GPRS Tunneling Protocol Control Plane (GTP-C) load balancing based on Dynamic Domain Name System (DNS) updates as proposed by [4] requires a new standard interface for configuration and implies modifications to the operator DNS infrastructure, which is very sensitive for operators. Although 3GPP standards do not specify how many Serving/Packet Data Network Gateway (S/PGW) should be deployed in LTE's core network, in practice, the LTE core network usually uses centralized and integrated S/PGW, which serves a large area of Evolved Node B (eNodeB)s, such as one province or a state with millions of subscribers. This results in a lot back and forth IP traffic in the S/PGW [22].

By introducing SDN controller in the EPC control plane and implement forwarding elements in the user plane, such as OpenFlow (OF) switches, it will bring more flexibility and dynamic load balancing control mechanism based on real-time statistics. This approach might be more feasible to reduce the aforementioned back-and-forth traffic. The proposed solution will lead to distributed S/PGW in the data plane with a centralized controller that uses SDN / OpenFlow to dynamically manage those S/PGW local/mobile Internet Protocol (IP) traffic. Nevertheless, the GPRS Tunneling Protocol (GTP) main role to provide QoS bearers has to also be addressed in the SDN context.

As it is predicted that mobile traffic will increase substantially, congestion in the network will become a serious problem. Therefore, QoS mechanisms for both congestion avoidance and different traffic treatment will become an important investment source for operators.

Support for higher quality services at lower ownership costs is a key requirement for all the operators. Different traffic session types have to be supported with different delay and guaranteed low-latency (e.g. voice and video). This has been usually achieved through differentiated service models. Efficient bandwidth allocation and management of differentiated QoS is an important target for mobile operators. Although they increase network capacity, that will not overcome bandwidth-hungry applications such as video and Peer-to-Peer (P2P) services. The radio spectrum is finite and does not provide too many options, thus, overprovisioning

to increase network capacity will lead to high costs and more complex deployments, lowering the network robustness.

Another way of performing QoS is through a central SDN controller, which can install and control policies on the path configured with bandwidth limitation for different session types. This can be achieved even without GTP in the user plane, only based on Layer 4 forwarding. The flexibility of such SDN controller in the Differentiated Services (DiffServ) context would help to easier adopt and implement the QoS in current EPC.

## 1-3 Related Work

A major concern for mobile operators is how to integrate the SDN and NFV architecture into the legacy hardware. In the area of NFV, the recent ETSI NFV standardization body has defined several use cases for a virtualized EPC in paper [5]. The proposals consists of both fully or partially EPC deployments along with the coexistence of virtualized functions and the legacy elements. On the other hand, proposals to integrate the SDN architecture in the mobile core have been carried under the ONF Wireless & Mobile Group. The load balancing and QoS routing mechanisms are carefully addressed by many researchers, starting from the data centers, where SDN is already a mature technology, to considerable solutions that might also be applied to the mobile core.

Several recent attempts on both control and user plane have been done to optimize the existing mobile architecture for the upcoming threats in terms of huge data increase and signaling storms. One line of research focuses on the development of a new architecture and further goes into the implementation details. For instance, the authors of paper [23] propose to lift up the control plane entities into the Cloud and preserve the Tunnel Endpoint Identifier (TEID)s for data plane by extending the OpenFlow protocol in order to support GPRS Tunneling Protocol User Plane (GTP-U) encapsulation / decapsulation. Paper [24] addresses the same challenge of implementing the user plane of the GTP protocol in an SDN-based switch-Open vSwitch. Under the same assumption of decoupling the control from user plane, the performance of GTP-U was analyzed in comparison to the existing tunneling protocols supported by Open vSwitch. Nevertheless, the approach mentioned in [24] does not take into account the case when a large number of header encapsulations is required, neither the extra overload in the packet header.

Another publication [25] proposes a new encapsulation / decapsulation on top of the IP Layer referred to as vertical forwarding implemented in the Forwarding Element (FE)s and managed by a central controller. The tunneling process is attributed to a line card incorporated in the FE. The authors of paper [26] present a similar solution, consisting of an enhanced OF network element Networking Element (NE)+. The functionality of GTP module is presented under four different frameworks: it can be incorporated in the SDN controller, run on a middlebox, modify OF switch hardware or being integrated via software platform in the OF switch. However, the complexity introduced by each of the solutions presented in [25, 26] in both hardware and software is not thoroughly analyzed.

The solutions presented in [23, 24, 25, 26] address the challenge of modifying the user plane in order to be compliant to the 3GPP standards, whereas papers [27, 28, 29] propose an innovative approach of both data and control planes. For instance, in paper [27] a completely new

mobile backhaul is employed based on OF switches and multiple controllers corresponding to multiple access domains. This solution requires changes on the MME and SGW, whereas the mobility management functionality is outsourced to the OF controller. Paper [28] introduces new OF-based control plane architecture to cope with the resiliency and load balancing issues in the existing 3GPP architecture. The centralized control plane which manages the forwarding plane insures the on-demand connectivity service. On the other hand, paper [29] proposes a mobile architecture based on OF switches and commodity middleboxes which completely replaces the standard 3GPP functions. The employed routing mechanism relies on a multi-dimensional aggregation based on policy tags, location (base station ID) and user identifier. The idea of a local controller at the access edge has been considered in order to perform packet classification for each User Equipment (UE). Due to its revolutionary concept to change both the data and control planes, the open question in paper [29] still remains the interoperability with legacy networks.

As a matter of fact, issues such as load balancing and congestion avoidance in the context of SDN have also drawn researchers' attention. There is meanwhile an extensive literature on load balancing mechanism using SDN technology. Article [30] addresses the impact of colliding flows on effective Equal Cost Multipath (ECMP) cross sectional bandwidth in a fat-tree topology. As a consequence, a bottleneck for both uplink and downlink directions might occur. This paper analyses the performance of two dynamic scheduling algorithms (Global First Fit and Simulated and Simulated Annealing) in comparison to ECMP. Global First Fit scheduler linearly searches all possible paths to find one whose link components can accommodate the flow. The algorithm can be applied in normal network load conditions, otherwise the links become saturated and the number of available paths is substantially decreased. The particularity of Simulated Annealing scheduler consists of assigning a single core switch for each destination host rather than a core switch for each flow, which reduces the number of available choices. The complexity of this algorithm is higher than Global First Fit and aims to reach a converged minimum by reducing the searching space.

The flexibility of SDN technology allows to apply the load balancing algorithms also outside the data centers, in the mobile domain. A solution in this direction is presented in paper [31]. The authors propose an efficient Wireless Local Area Network (WLAN) and eNodeB handover mechanism to retrieve load status through the southbound interface. In this paper, the flow capacity is defined as the number of requested physical Physical Resource Blocks (PRB)s. In the simulation model two types of users are defined: unsatisfied users who only receive the minimum capacity for a flow and angry users who do not get access to their flows. In this article, the authors employ both a probabilistic as well as an analytical approach to validate the proposed solution. The results reveal a drastic reduction of the number of unsatisfied and angry users in the network and a substantial improvement of resources allocated per user.

A sensitive issue in load balancing algorithms is to take into account the load statistics. The question in the SDN statistics retrieval is how to maximize the processing power of the controller while decreasing the overload in the switches. Paper [32] addresses the challenges of implementing monitoring rules in the network in order to determine the load on each switch. An adaptive algorithm based on linear prediction is presented in order to mitigate the amount of processed data by the controller and the overhead in the network produced by the load information. The primary goal is to minimize the load on the switch flow tables and prepare the data for anomaly detection applications. Finally, a trade-off between short delay in detecting the traffic anomalies and the operator need has to be established. The results

were compared to a static aggregation method extracted on the interface between Gateway GPRS Support Node (GGSN) and Serving GPRS Support Node (SGSN) for a European mobile operator and showed improved efficiency in detecting traffic anomalies.

Another monitoring method which mitigates the extra load information in the packet header mentioned in [32] is to send the traffic to an external analyzer. Recent paper [33] presented at OpenDaylight Summit 2014 proposes to use sFlow standard in combination with SDN architecture in order to detect large flows in real-time and send this information to the Load Balancing Application on the top of the controller.

Papers [30, 31, 32, 33] deal with an SDN-based load balancing. Nevertheless, there are several drawbacks which should be taken into consideration when adopting these solutions such as: the complexity of the proposed algorithms or the extra delay in the controller and switches processing which does not report real time congestion problems. Moreover, these solutions do not address the mobile core directly, but they are rather solutions dedicated to data centers.

Another area of interest for SDN research is the support for QoS routing and different service treatment. One of the main papers which focuses on the QoS mechanisms using an SDN controller is article [34]. The authors of this paper propose an OpenFlow controller design with optimized routing for QoS deployments. In their approach the data traffic is separated from multimedia flows which are sent throughout QoS guaranteed routes, whereas data flows follow shortest-path algorithm. The employed QoS is different than the existing Integrated Services (IntServ) or DiffServ mechanisms. An interesting paper which actually focuses on DiffServ and Traffic Shaping classification using an SDN controller is [35]. The authors developed a QoS module which communicates with the application layer via an open Application Programming Interface (API). Nevertheless, the two proposed applications: the "class of services" and the "policy path" do not present any interdependencies. The relationship between these modules has been further developed in this thesis.

Recent demands for QoS in the mobile networks question the possibility of implementing service differentiation using SDN. Paper [36] targets to apply the QoS mechanisms mapped to the subscribers attributes instead of network addresses and locations. The continuity of paper [36] is presented in [29]. The authors of these two articles propose an innovative EPC SDN-based architecture while taking into account the flow aggregation and paths installation based on several attributes: policies, location and UE identifier. The work in papers [29, 36, 37] is mainly based on policy rule minimization and the results are presented in the network context. The UE QoS experience in terms of throughput, jitter and loss is still open to exploration.

## 1-4 Contributions

In contrast to the aforementioned papers which are either conservative, preserving the traditional 3GPP interfaces [23, 24, 25, 26] or innovative in adopting a complete new SDN architecture in the mobile core [27, 28, 29, 36, 37], this thesis sets a trade-off between the two approaches. The employed architecture maintains the S1-MME interface from the current EPC, which is essential to procedures like *Initial Attachment*, while it introduces a new way of routing in the data plane without tunneling. The main advantage of the proposed architecture is that it does not require any changes to the OpenFlow protocol, neither in the OpenvSwitch. Paper [24] compares the GTP-U implementation performance with the

non-encapsulation alternative and shows that the performance of the second solution is better in terms of throughput, latency and jitter. Nevertheless, the addition of another type of encapsulation such as Generic Routing Encapsulation (GRE) or Multiprotocol Label Switching (MPLS) can be very easily configured. The last option also brings continuity towards the metro network where usually MPLS is deployed.

This thesis addresses the challenges of integrating SDN and NFV in the current LTE EPC architecture, as well as the interoperability with legacy networks. The main contributions of this thesis consists of a new EPC SDN-based architecture proposal, which aims to separate the control from the data plane in a flexible data and control plane traffic management. The 3GPP control plane functions such as MME, S/PGW Control Plane (S/PGW-C) can be virtualised, whereas the user plane consists of OF switches managed by a central entity called SDN controller. An important aspect of this architecture is the impact to the existing systems, therefore a careful analysis investigates which standard interfaces are being preserved in the control plane and, on the other hand, the possibility of replacing the GTP-U tunneling in the user plane. The EPC SDN-based architecture addresses several challenges: the protocol stack modification, a dynamic flow aggregation and policy path implementation. This thesis aims to propose solutions to these challenges in order to satisfy the mobility requirement. Apart from the system design, possible EPC SDN-based enhancements have been described in terms of flexibility, deployment complexity and performances.

The system design also takes into account the mobility issues applied on two main use cases: *UE-triggered request* and *Network-triggered request*. In terms of QoS, this thesis aims to modify the QoS module available in [34] in order to support different types of traffic (data, voice, video) and create dependencies between the applications on top of the controller. In this manner, the routing decisions are mainly taken based on the configured service class. In addition to that, a new congestion avoidance mechanism is employed in the access part of the network. A round-robin load balancing mechanism is capable to equally distribute the flows into different clusters of Evolved Node B (eNB)s and it also manages the handovers between base stations. One of the major targets is to measure the network QoS parameters such as the obtained throughput, jitter and packet loss.

The practical work consisted of two testbeds: one using physical OpenFlow switches and another one in a network simulation environment. A careful assessment of both the user and the control plane performances has been conducted for different scenarios and topologies. Several tools and technologies for both SDN and NFV have been presented along with the implementation means. The mobile traffic was captured using OpenEPC, an LTE simulator in a virtualized environment. The physical testbed was meant to validate the reactive flow instantiation mode and GRE tunneling. In the simulation testbed with virtual switches two use cases have been investigated: load balancing and QoS for the core network. These use cases involved the integration of additional modules in the controller and building the proposed topology. In the user plane, the use cases have been tested under several scenarios with different loads. In the control plane, two controllers have been compared in terms of processing capabilities. The analysis of the obtained results helped to decide over the simulation tool capabilities and which of the two SDN controllers is performing better under certain configurations.

## 1-5 Thesis Outline

This thesis approaches the aforementioned issues starting from a broader definition of technologies and introduction to the context, followed by a proposed system design, description of the implementation tools and measurements analysis. Hereby, the work has been structured in five main chapters as follows:

**Chapter 2** presents the standardized EPC architecture and its functionality along with the main protocols. Some of the current LTE drawbacks are investigated, as well as the possible 3GPP solutions to these issues. New concepts like Cloud Computing, NFV and SDN are explored along with their interdependencies and common capabilities. The OpenFlow protocol as well as the OpenFlow switch and OpenvSwitch architecture are presented.

In **Chapter 3**, a thorough analysis of the involved procedures (*Attach Procedure, Default Bearer Establishment*), the protocol stack modification and the challenges of the new EPC SDN-based architecture are investigated. The mobility issue and the *UE triggered Request / Network triggered procedures* are explained in details along with the call flows. Possible optimizations of this new architecture are also presented.

**Chapter 4** describes the tools and technologies used in the implementation phase. Both NFV and SDN tools were used and parts of the configuration of the two testbeds are included together with explanation of the commands. The two controller architecture is presented and different extensions of these controllers are investigated.

**Chapter 5** introduces the expectations regarding the NFV / SDN benefits to the applied use case (i.e. EPC). Several use cases are proposed and their performance is analyzed for different scenarios. This chapter consists of an evaluation of the obtained results and a comparison between the two controllers.

**Chapter 6** aims to draw the final remarks and conclusions of the presented work. Proposed optimizations and complementary future work are also presented.

# Chapter 2

# Overview of the Evolved Packet Core (EPC) and Network Function Virtualisation (NFV) / Software Defined Networking (SDN) Concepts

The rapid growth of internet and packet data services in last few years triggered the demand for the mobile core network evolution, the current EPC. This chapter presents the EPC architecture and the functionality of the main core elements. Moreover, the possible drawbacks in current EPC are identified along with the proposed 3rd Generation Partnership Project (3GPP) solutions. New technologies like Cloud Computing, NFV and SDN are foreseen to provide more reliable long-term solutions to these issues. Therefore, a study of the benefits of employing these technologies is presented, as well as the interdependencies between them, which might even enhance the advantages to the applied use cases.

## 2-1 Evolved Packet Core (EPC)

The progression of the mobile core network is called System Architecture Evolution (SAE) and resulted in EPC, whereas the radio access part is called Evolved UMTS Terrestrial Radio Access Network (E-UTRAN). The Long-Term Evolution (LTE) and the EPC together are called the Evolved Packet System (EPS), where both the core network and the radio access are fully packet-switched. There are many benefits of the SAE including flat architecture with less network nodes, smaller delays and bigger data rate support. The LTE / EPC architecture has been designed to provide Internet Protocol (IP) connectivity between User Equipment (UE) and external Packet Data Networks (PDN)s.

In contrast to Global System for Mobile Communications (GSM) / General Packet Radio Service (GPRS) core network and Wideband Code Division Multiple Access (WCDMA) /

High Speed Packet Access (HSPA), the EPC only provides the packet-switched domain without support for the circuit-switched domain. The EPC is mainly responsible for the control of the UE and the establishment of bearers. In EPS, bearers are used to route IP traffic from the UE to the Packet Data Network Gateway (PGW) and vice versa. A bearer can be seen as an IP packet flow with a defined Quality of Service (QoS) between the gateway and the UE. The basic functionality (or "service") supported by the EPC is essential to provide always-on IP connectivity to mobile subscribers, i.e. to enable wireless mobile terminals UE to get a valid IP address, to send and receive traffic to / from other IP hosts, taking into account their mobility.

The EPC standardized architecture is described in Figure 2-1. The Serving GPRS Support Node (SGSN) is part of the GPRS and Universal Mobile Telecommunications System (UMTS) architecture. The Mobility Management Entity (MME), Serving Gateway (SGW), PGW, Policy and Charging Rules Function (PCRF) and Home Subscriber Server (HSS) represent the EPS elements in the LTE architecture.

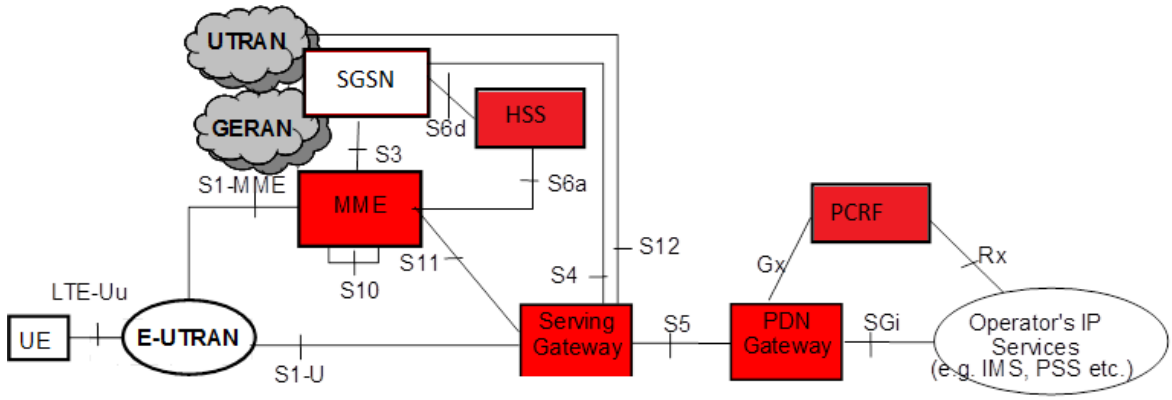



**Figure 2-1:** The EPS network elements, source [1]

## 2-1-1 EPC basic functions and architectural elements

### 2-1-1-1 Mobility Management Entity (MME)

The MME is the key control node for LTE access network, as it processes the signaling between the UE and the EPC (*S1-MME* interface). The MME's functionality is described in [38]. It is responsible for bearer activation / deactivation procedures and in charge of choosing the SGW for a UE in the process of *Initial Attachment* or when the intra-handover takes place, which involves core network node relocation. MME is involved in Tracking Area (TA) and Paging procedure including retransmissions, and also in managing idle mode of UE.

The MME is responsible for authenticating the UE towards the HSS. In case the user is roaming, MME terminates the *S6a* interface towards user's home HSS. All Non Access Stratum (NAS) signaling terminates at the MME point, which is also responsible for generation and allocation of temporary UE identities such as Globally Unique Temporary ID (GUTI). Among its duties, it is counted user's authorization to Public Land Mobile Network (PLMN) and enforcing UE roaming restrictions. The MME also constitutes the termination point of

ciphering and integrity protection for NAS signaling. Lawful Interception (LI) of signaling could also be supported by MME entity. It also provides the control plane function for mobility between LTE and Second Generation (2G) / Third Generation (3G) networks by the *S3* interface (from SGSN to MME).

#### 2-1-1-2 Serving Gateway (SGW)

The SGW is a user-plane node which connects the E-UTRAN with the EPC (*S1-U* interface). Its main functions are described in [38]. The SGW provides packet transfer along the user plane and it also represents the mobility anchor point when the UE moves from one Evolved Node B (eNodeB) to another in a handover procedure. At a certain point in time, a single SGW serves to each UE associated with an EPS. The SGW duties also consist of taking care of the mobility interface to other networks such as 2G /3G. The SGW monitors and maintains context information related to UE during its idle state and generates paging requests when data arrives in the UE in downlink direction (from Internet towards the UE).

#### 2-1-1-3 Packet Data Network Gateway (PDN-GW)

The PGW is the entity which connects the EPC to the Internet (*SGi* interface). PGW functions are described in [38]. The PGW is responsible to act as an "anchor" for mobility between 3GPP and non-3GPP technologies. The PGW provides connectivity from the UE to external PDN by being the point of entry or exit of traffic for the UE. The PGW also handles policy enforcement, packet filtration for users, charging support and LI.

#### 2-1-1-4 Policy Control Enforcement Function (PCRF)

The PCRF is responsible for policy control, QoS handling and controlling the charging functionalities in the Policy Control Enforcement Function (PCEF), which resides in the PGW. It decides how a certain data flow will be treated. PCRF dynamically controls and manages all data sessions and provides appropriate interfaces towards charging and billing systems, as well as it enables new business models.

#### 2-1-1-5 GPRS Tunneling Protocol (GTP) and Bearer Establishment

The GTP is the main protocol within EPC: it uses User Datagram Protocol (UDP) and it basically adds a small header, in which, among other information, two identifiers are inserted: Source and Destination Tunnel Endpoint Identifier (TEID)s, used by the source and destination nodes to identify tunnels for each user. The reason why the user traffic is encapsulated into GTP tunnels within the EPC, is to easily handle mobility of the subscribers across base stations, i.e. across Evolved Node B (eNB)s (the mobility within the coverage area of a single eNB is handled by the eNB itself and it does not impact the core) (see Figure 2-2).

The GTP protocol has been standardized under different types corresponding to the user and control planes:

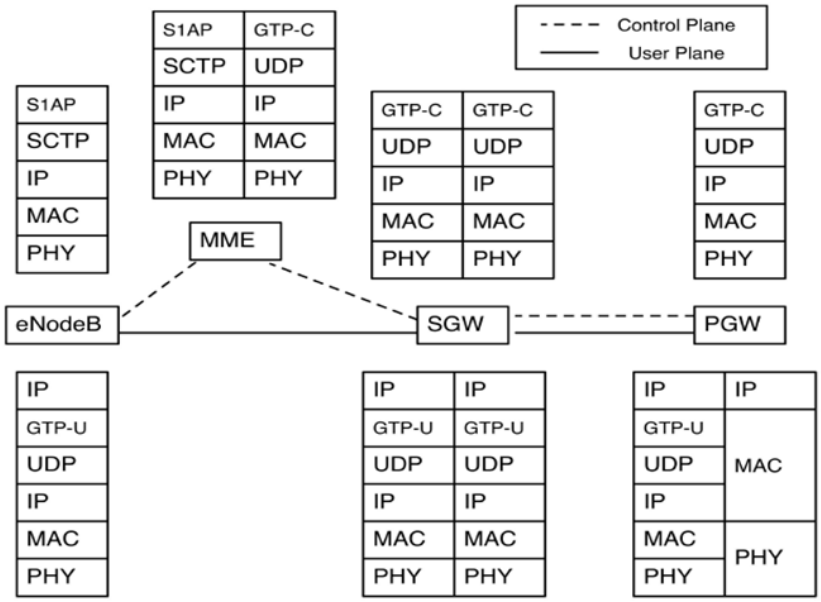



**Figure 2-2:** 3GPP LTE/EPC Protocol Stack, source [2, 3]

- **GPRS Tunneling Protocol Control Plane (GTP-C) (GPRS tunneling protocol version 2 (GTPv2))** is a control plane protocol that supports establishment of tunnels for mobility management and bearers for QoS management that matches wired backhaul and packet core QoS to radio link QoS. It is used for core network specific signaling (bearer activation/ deletion / modification, etc.) and uses UDP as a transport protocol [3].
- **GPRS Tunneling Protocol User Plane (GTP-U)** is a data plane protocol used for implementing tunnels between network elements that act as routers, specifically the SGW, PGW and the wired network side of the radio base station (eNodeB) [2].
- **GPRS Tunneling Protocol Prime (GTP')** consists of the same message structure as GTP-C but it is used for carrying charging data from the Charging Data Function (CDF) to the Charging Gataway Function (CGF) [39].

The user traffic (Data Plane in the 3GPP terminology) is transmitted on the LTE radio interface from the UE to the eNB, which encapsulates it in a GTP tunnel and transmits it over an transport IP network, like a Metro Ethernet, towards one SGW. This, in turn, encapsulates the user traffic in a new GTP tunnel towards a PGW. Finally, the PGW decapsulates the traffic and transmits it "out" of the EPC (the *SGi* interface in Figure 2-1), towards its destination: e.g. the Internet or a service within the mobile operator network. It is important to understand that the PGW is the entity which assigns the IP address to the UE: hence, because of the tunneling, the UE sees the PGW as its first IP hop ( i.e. the PGW is the default IP gateway for the UEs).

One EPS bearer is established when the UE connects to a PDN and that it remains established throughout the lifetime of the PDN connection to provide the UE with "always-on" IP

connectivity to that PDN. That bearer is referred to as the default bearer. Any additional EPS bearer that is established for the same PDN connection is referred to as a dedicated bearer.

An EPS bearer is the level of granularity for bearer level QoS control in the EPC / E-UTRAN. That is, all traffic mapped to the same EPS bearer receives the same bearer level packet forwarding treatment (e.g. scheduling policy, queue management policy, rate shaping policy, etc.). Providing different bearer level packet forwarding treatment requires separate EPS bearers.

### 2-1-2 Differentiated Services (DiffServ) and QoS in LTE

There is a significant distinction between real-time services (i.e. conversational video and voice) and best-effort services (i.e. Internet browsing). Real-time services must reserve a minimum amount of guaranteed bandwidth, and are more sensitive to packet loss and latency / jitter. Policy management allows operators to control the availability and Quality of Experience (QoE) of different services for end users. First, policies are used to dynamically allocate network resources, for instance, a particular bandwidth can be reserved in the radio base station and core network to support a live video conversation. In other cases, policy rules might be used to limit traffic rates on the network in order to limit network abusers (e.g. Distributed Denial of Service Attacks (DDoS)) and provide fair usage of the resources while preventing some users, from negatively impacting the quality of other services.

In LTE networks QoS is implemented between UE and PGW and it is applied to a group of bearers . A "bearer" is the basic traffic separation element that provides differential treatment for traffic with different QoS requirements. Bearers provide a logical, edge-to-edge transmission path with well-defined QoS between the user equipment (UE) and PGW. All flows contained in a single bearer are forwarded and treated in the same manner (e.g scheduling, queue management, rate shaping, link layer configuration, etc.) between the UE and PGW. Each bearer is associated with a set of QoS parameters that describes the properties of the transport channel, including bit rates, packet delay, packet loss, bit error rate and scheduling policy in the radio base station [15].

A bearer is characterized by several QoS parameters mentioned below, depending on whether it is a real-time or best effort service:

1. **QoS Class Indicator (QCI):** specifies the treatment of the IP packets received on a specific bearer. Packet forwarding of traffic traversing a bearer is handled by each functional node (for example, a PGW or eNodeB). The QCI values impact several node-specific parameters, such as link layer configuration, scheduling weights, and queue management.

2. **Allocation and Retention Priority (ALRP):** This specifies the control-plane traffic treatment. The ALRP enables bearer establishment and modification, as well as connection setup and release. For instance, Address Resolution Protocol (ARP) can be used by the EPS to decide which bearer should be released during resource limitations or traffic congestion.

| QCI | Bearer Type | Priority | Packet Delay | Packet Loss | Example |
|---|---|---|---|---|---|
| **1** | GBR | 2 | 100 ms | $10^{-2}$ | Voice over IP (VoIP) call |
| **2** | GBR | 4 | 150 ms | $10^{-3}$ | Video call |
| **3** | GBR | 3 | 50 ms | $10^{-3}$ | Online Gaming (Real Time) |
| **4** | GBR | 5 | 300 ms | $10^{-6}$ | Video streaming |
| **5** | GBR | 1 | 100 ms | $10^{-6}$ | IP Multimedia System (IMS) signalling |
| **6** | Non-GBB | 6 | 300 ms | $10^{-6}$ | Video, TCP based services: e.g. e-mail, chat, FTP, etc. |
| **7** | Non-GBR | 7 | 100 ms | $10^{-3}$ | Voice, Video, Interactive Gaming |
| **8** | Non-GBR | 8 | 300 ms | $10^{-6}$ | Video, TCP based services: e.g. email, chat, FTP, etc. |
| **9** | Non-GBR | 9 | 300 ms | $10^{-6}$ | Video, TCP based services: e.g. email, chat, FTP, etc. |

**Table 2-1:** 3GPP Standardized QCI, source [15]

3. **Guaranteed Bit Rate (GBR) bearers:** are used for real-time services, such as conversational voice and video. A GBR bearer has a minimum amount of bandwidth that is reserved by the network, and it always consumes resources in a radio base station regardless of whether it is used or not. If implemented properly, GBR bearers should not experience packet loss on the radio link or the IP network due to congestion. GBR bearers are also defined with the lower latency and jitter tolerances that are typically required by real-time services (see Table 2-1).

4. **Non-GBR bearers:** do not have specific network bandwidth allocation. A default bearer is always non-GBR. Non-GBR bearers are dedicated to best-effort services, such as file downloads, email, and Internet browsing. These bearers will experience packet loss when a network is congested.

5. **Maximum Bit Rate (MBR):** is applicable only for real-time services and it is defined for GBR bearers. The MBR is the bit rate that the traffic on the bearer should not exceed. A maximum bit rate for non-GBR bearers is not specified on a per-bearer basis. However, an Aggregate Maximum Bit Rate (AMBR) will be specified on a per-subscriber basis for all non-GBR bearers.

#### 2-1-2-1 Policy control and mapping of QoS Rules

The PCRF, also known as the policy controller, filters the packet flows based on an IP five-tuple: source IP address, destination IP address, source port number, destination port number, and protocol identification (Transmission Control Protocol (TCP) or UDP). PCRF determines the QoS requirements upon three criteria: application requirement, subscription information and policy of the operator. Based on the decision made by PCRF, a dedicated EPS bearer may be established.

The 3GPP also defines Traffic Flow Template (TFT) as a classifier that matches on fields on the inner-IP of a GTP-U tunnel. This in turn causes differentiated radio-bearer performance. It can match the following fields: source address (with subnet mask), IP protocol number (TCP, UDP), destination port range, source port range, IPSec Security Parameter Index (SPI), Type of Service (ToS) (IPv4), flow-Label (IPv6 only) [40].

### 2-1-3 Drawbacks in the current 3GPP Architecture

The GTP-C interface is defined for signaling between many core network nodes (MME-SGW-PGW) and services. An inefficient management of this interface would produce: loss of PDN connectivity (IMS, Internet) and associated services, loss of ability to setup and release network bearers (GBR bearers for Voice over LTE (VoLTE)), loss of ability to report to the PGW / PCRF user information's changes (location information for emergency services and lawful intercept, changes in Radio Access Transfer (RAT) or QoS), billing errors and loss of revenue. The GTP-C interface should be enhanced to support overload prevention and dynamic load control as it has currently very limited support in this aspect. Therefore, mechanisms to detect and mitigate overload on GTP-C plane should be investigated [4].

The standardized GTP-C load balancing mechanism relies on the MME using Domain Name System (DNS) weights that are semi-static and "typically set according to the capacity of a gateway node relative to other gateway nodes" [4]. In this case the DNS weights associated with this node become wrong information. When the failed gateway restarts / recovers, the traffic allocation, based on DNS weights, distributes the same amount of traffic to the various gateways, even though the recovering gateway is not loaded at all, while the other gateways support a higher amount of load than usual. For example, in the case of SGW, the weight factor is typically set according to the capacity of a SGW node relative to other SGW nodes serving the same TA. The same applies during network extensions (e.g. when adding a new SGW to a cluster)[4]. The load-balancing mechanisms also proves to be inefficient when a SGW node has a partial failure (e.g. one of its components has failed), or during certain maintenance operations, it may still be working but not under full capacity.

By its nature, the GTP-C protocol has not been designed to cope with sudden overload or congestion issues in both control and user planes. Some of this shortages along with the proposed solutions were discussed in [4]. Nevertheless, these solutions involve configuration of new standard interfaces and imply modifications to the core infrastructure, which is a very sensitive area for operators.

Another important drawback is the tunnel establishment (between eNB, SGW and PGW) in the *Initial Attachment* procedure, even though there is no data to be sent. The TEID values are locally allocated by each node. Therefore, new TEID values should be exchanged for each node's relocation which drastically affects network elasticity.

Up till present there are no standardized policy management per-subscriber due to its high complexity and the first phase regards an aggregate-level policy along with congestion mechanisms. The 3GPP standards explain how to build transmission paths between the UE and the PGW with well-defined QoS. Up till present, the 3GPP has defined an extensive "bearer model" to implement QoS [15].

#### 2-1-3-1 Overload Scenarios

The 3GPP paper [4] has defined several mobility and session management scenarios. These scenarios can cause the following GTP-C overload or failure issues in the mobile core network:

1. Frequent *Idle to Connected*, and *Connected to Idle* transitions caused due to e.g. eNB idle timer. Depending on the value of eNB idle timers (which may result in large number of e.g. *SERVICE REQUEST*s from UEs in a busy hour), session overload may occur in either a single SGW or in a set of SGWs managing multiple TAs.

2. An overload of a downstream node (e.g. PGW) may also potentially cause overload of an upstream node (e.g. SGW) due to GTP-C signalling retransmissions.

3. At the overload or failure of an EPC node ( e.g. SGW), where the network would try to re-establish the GTP-C session via a new EPC node (SGW) that would replace the failing one. The risk is that the failure of a node (e.g. SGW) would trigger a spike in GTP-C signalling to restore within the shortest time the PDN connections affected by the failure. These attempts to restore PDN connections affected by the failure would overflow other nodes (e.g. other SGW, PGW) and transform a local failure (e.g. of an SGW) into a complete network issue via a snow ball effect. The same applies upon a failure of a PGW, MME or SGSN.

4. A GTP-C node (e.g. PGW) may encounter issues to handle traffic on a non-overloaded GTP-C interface (e.g. *S5* interface) when another of its (possibly non GTP-C) interfaces (e.g. *Gx*) is overloaded.

5. Application signalling that induces creation of dedicated bearers served by a MME or pool of MME. A large number of users may start application related interactions (e.g. IMS Session Initiation Protocol (SIP) call) simultaneously when some exceptional event occurs (e.g. New Year's Eve, natural disasters like flooding, earthquakes, etc.) which leads to a large amount of almost simultaneous Create / Update Bearer Requests sent from PGW to MME.

#### 2-1-3-2 3GPP load balancing proposed solutions

Several solutions to the issues mentioned above have been introduced in 3GPP paper [4]. For instance, the authors of paper [4] proposed as a solution to introduce gateway load / overload information for enhanced load balancing when selecting an SGW or PGW, e.g. in complement to the existing DNS weights mechanism (which takes into account relative capacities of nodes versus each other's). The other solution implies to introduce a new entity responsible for gateway load balancing.

1. *Using load / overload information for dynamic load balancing of gateway nodes:*

   (a) The "load information" reflects the operating status of the resources in the originating GTP-C node and it should have a value between 0-100 with lower values for less loaded nodes. Moreover, PGW should provide its "Load Information" to

MME / SGSN via SGW, as included over existing signaling messages (e.g. Create Session Request) and further SGW should forward in the same manner it to MME / SGSN.

*Issues*: Paper [4] states that the way in which a node computes its "load information" depends on the implementation and does not guarantee that local configuration allows it. The exact mechanism of transmitting the load information and the exact format of this parameter have not been defined yet, as it is very sensitive to determine how much additional load will add to each peer node and how this might impact the resource utilization of the node. Under these circumstances, one might question how well the load information will be synchronized with the weight factor provided by DNS which currently does not have real time statistics.

(b) The "overload information" reflects an indicator of when the originating node is running above its nominal capacity which may cause severe issues in handling the incoming traffic. The content of this additional information (e.g. validity-period, throttle rate, message scope, etc.) has not been defined yet as it is a very important issue is to coordinate the same "overload action" for different GTP-C nodes in order to lead to the same system level mitigation result.

*Issues*: Node congestion control can run in parallel or independently of existing congestion mechanisms (such as PGW APN back-off timer). One of the investigation directions is how can PGW determine the signal overload associated with a particular Access Point Name (APN). A special situation is when overload information is transmitted from MME / SGSN towards SGW / PGW direction (e.g. inter-MME Tracking Area Update (TAU)) and involves MME address change, but the PGW is not aware of the new MME address and keeps acting on the overload information of the old MME. As in the case of "information load" feature, this solution is dependent on the implementation and there might be cases when the peer node does not support overload information, thus this would be discarded by the peer node.

2. *Use of Gateway Load Manager for DNS based load balancing:* To allow the SGW and PGW selection procedure to also take into account their current load condition, a functional entity called Gateway Load Manager (GLM) is introduced between the MME / SGSN and DNS server. The MME / SGSN will send DNS query to GLM which holds the current load status of the SGW / PGW. The DNS weight might be changed based on this information. Paper [4] proposes two ways for GLM to retrieve load information from SGW / PGW: either to use protocols such as Simple Network Management Protocol (SNMP) or via Operations, Administration and Management (OAM). Another option would imply to standardize a new interface in order to be deployed in a multi-vendor environment (Figure 2-3).

   *Issues:* The major inaccuracy of this solution stated in [4] is that "current load factor" does not imply real time statistics about the load since it can be updated after a short period (e.g. 3 min) to only deal with peak oscillations during that interval. For instance, in the case when both SGW and PGW are heavily loaded and are prone to enter in overload condition resulting in traffic spikes, the GLM is unable to return any SGW / PGW address in response to the MME / SGSN queries towards DNS and cannot prevent further load increase. Another issue to deal with is how the latency introduced

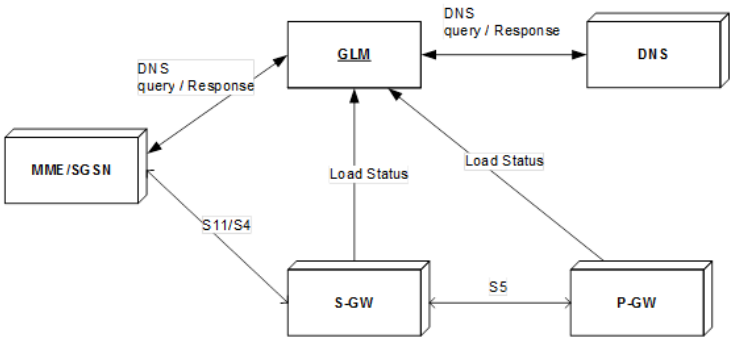



**Figure 2-3:** GLM, source [4]

by the change of SGW / PGW load information will impact the MME / SGSN selection (considering DNS caching in MME/ SGSN) and the amount of extra signaling between MME / SGSN and DNS. Apart from the over-provisioning, GLM introduces a new point of failure in the network. Solutions to deal with GLM failure have not been investigated yet, neither additional redundancy aspects are provided.

#### 2-1-3-3 Possible NFV and SDN solutions

Looking beyond 3GPP proposed solutions, NFV and SDN can provide long-term solutions for the static load-balancing and QoS mechanism challenges. In order to build a fully compliant solution it is very important to understand the evolution and capabilities of these technologies, as well as the relation: Cloud Computing / NFV, or between NFV and SDN. The foreseen benefits of each of these technologies should be accommodated to the upcoming demands.

## 2-2 Cloud Computing

Cloud computing can be seen as a new Internet-based business model which provides shared resources to computers and other devices on-demand. Storage, processing, memory, network bandwidth, and virtual machines are seen as resources. One of the major advantage of cloud computing is that users do not concern at any particular time where the resources are, or how to consume them. Therefore, it is called *on-demand self-service* and it does not require any human interaction with each services provider.

Resource pooling is another characteristic of cloud computing: the customer may not know the exact location of the resource; one can only specify the location at a higher level (country, state or data center). Rapid elasticity means that time and space are not constrains and the queries appear to be unlimited and purchased any time. The service can be measured and resources can be monitored, controlled, as well as reported in a complete transparency between the consumer and service [41]. Cloud services do not only refer to convenient storage, accessible by multiple devices, but also include important benefits such as more accessible communication and instant multi-point collaboration.

### 2-2-1 Service Models and Benefits

Cloud Computing Services are typically offered to customers in one of the three service models as defined by National Institute of Standards and Technology (NIST) 800-146 [42]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS).

The IaaS reflects the capability to offer consumers the processing, storage and fundamental computing resources. The NFV Infrastructure shall provide computation capabilities comparable to an IaaS cloud computing service.

The PaaS adds an additional layer of integration with application development frameworks and functions such as database, messaging and queuing. The consumer only controls the deployed applications and some application hosting and does not have any right to manage the underlying cloud infrastructure: networks, servers, operating systems or storage.

The SaaS is sometimes mentioned as "on-demand software" in which the data and software resources are usually accessed by users through a web browser over the Internet. This includes the content, its presentation, the application and management capabilities integrated in a self-contained operating environment [41].

The success of cloud computing and SDN / OpenFlow in data centres is attributed to the simple all-IP core networks. In comparison to the multi-technology and complexities in Communications Service Providers (CSP)s' costly transport, the solutions provided by cloud computing are very flexible and demand proper adjustment to the CSP's needs.

### 2-2-2 Mapping between Cloud Computing and NFV

The benefits of cloud computing (i.e. virtualisation of standard IT computing and storage) are well understood, therefore it is implemented in data centers worldwide. Cloud computing is the virtualisation of commodity IT hardware (namely x86 servers) and applications / software, which can run at least up to 99 % of the availability level.

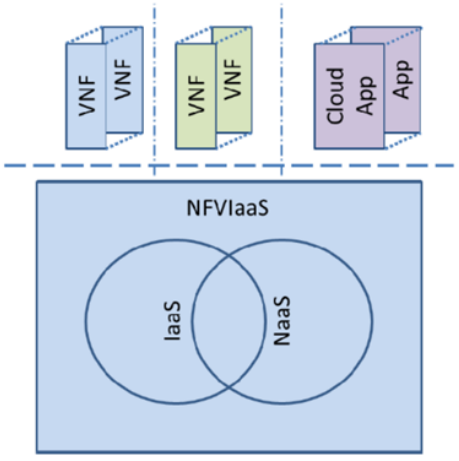
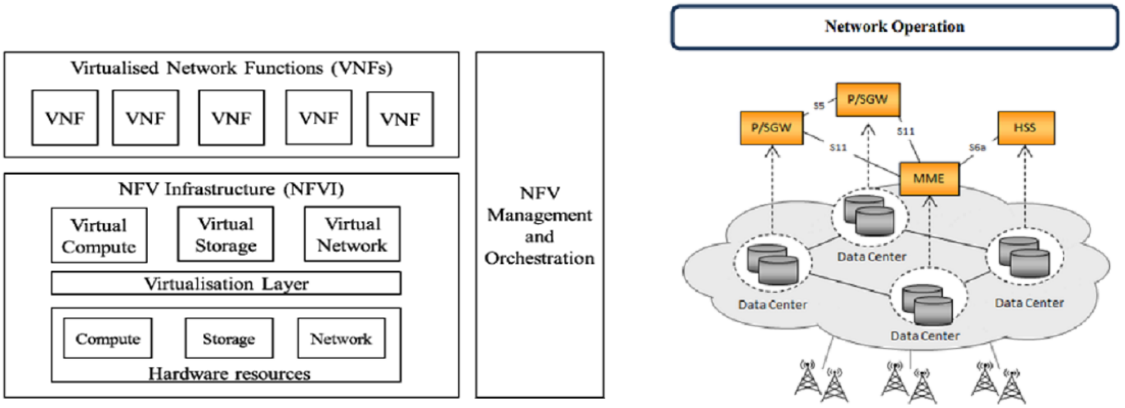



**Figure 2-4:** NFVIaaS Multi-tenant Support, source [5]

On the other hand, NFV is the virtualisation of telecoms-specific network functions into




(a) NFV architecture, source [5] (b) Example of Virtualised EPC, source [5]

**Figure 2-5:** NFV architecture and deployment

applications that will run at least up to 99.999 % of the availability on suitable carrier-grade hardware and software [43].

The PaaS and SaaS cloud computing service models provide software based capabilities that can run over the provided infrastructure (the same infrastructure that may be offering IaaS or Network as a Service (NaaS) service) (Figure 2-4). Major CSPs are now convinced that NFV has matured sufficiently to virtualize the majority of network functions.

## 2-3 Network Function Virtualisation (NFV)

Due to the explosion of mobile devices and content, server virtualisation and onset of cloud services, the networking industry has been driven to re-examine traditional network architectures. The NFV aims to transform the way operators build their network architecture by evolving standard IT virtualisation technology to consolidate many network equipment types. It involves implementation of network functions in software that can run on a range of industry standard server hardware, and can be moved to / instantiated in, various locations in the network as required, without the need for installation of new equipment [6].

The Virtual Network Function (VNF) is the software implementation of a network function which is capable of running over the NFV infrastrucure (NFVI) (Figure 2-5-a)). The NFV Management and Orchestration controls the physical and software resources that support the NFVI and the lifecycle management of VNFs. It allows both horizontal and vertical scalability of the resources based on capacity needs. The VNF which might fulfill the role of Serving/Packet Data Network Gateway (S/PGW), MME functions, can scale up both horizontally and vertically according to their specific resource requirements (e.g. the user plane resources can be increased independently of the control plane and vice versa (Figure 2-5-b)).

### 2-3-1 Benefits of NFV

Some of the main benefits that NFV promises to bring are: lower equipment costs and reduced power consumption through consolidating the equipment and exploiting the scale economies of the IT industry. Apart from these, other benefits have been further identified which can be directly mapped to the EPC applied use case.

For instance, NFV provides increased velocity of time-to-market and mitigates the typical network operator cycle of innovation. Economies of scale required to cover investments in hardware-based functionalities are no longer applicable for software-based development, making feasible other modes of feature evolution. The NFV should enable network operators to significantly reduce the maturation cycle.

The virtualisation of a mobile core network targets at a more cost efficient production environment. This will allow network operators to cope with the increasing traffic demand in mobile networks and acquire better resource utilization (including energy savings). Other advantages are: more flexible network management (no need to change hardware for upcoming upgrades), hardware consolidation, as well as easier multi-tenancy support and faster configuration of new services.

The NFV might help optimizing network configuration and topology in real time based on the actual traffic / mobility patterns and service demands. Another important capability is the possibility of running production, test and reference facilities on the same infrastructure [6].

### 2-3-2 Differences between SDN and NFV

The NFV's main goal is to reduce equipment cost and decrease time-to-market by providing scalability, elasticity and a strong ecosystem. The OpenFlow-enabled SDN proposed by the Open Networking Foundation (ONF) aims to achieve the same benefits. While NFV is intended to optimize the deployment of network functions (such as firewalls, DNS, load balancers, etc.), the OpenFlow-based SDN solution is more focused on routing and optimizing the underlying networks [44].

As shown in Figure 2-6, the NFV is highly complementary to SDN, but not dependent on it (or vice-versa). The NFV can be implemented without a SDN being required, although the two concepts and solutions can be combined and potentially greater value obtained.

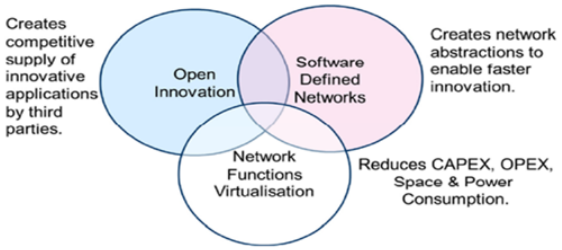



**Figure 2-6:** Network Functions Virtualisation Relationship with SDN, source [6]

## 2-4 Software Defined Networking

SDN is an emerging network architecture where network control is decoupled from forwarding plane and it is directly programmable. This technology has been promoted by the ONF which is a nonprofit, mutually beneficial trade organization, founded by some of the largest IT companies in the world: Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo.

The ONF was organized for the purpose of standardizing a protocol between the controllers and the network elements. This protocol is called OpenFlow protocol and runs on the *Southbound* interface. With SDN, enterprises and carriers main benefit is to gain control over the entire network from a single entity, which reduces the network complexity and operation. The SDN also simplifies the network devices themselves; since they no longer need to understand and process thousands of protocols standards, but rather accept instructions from the SDN controller [45].

### 2-4-1 SDN Architecture

Figure 2-7 depicts a logical view of the SDN architecture which consists of three main layers: *infrastructure, control and application layer*. Network intelligence is (logically) centralized in software-based SDN controller, which maintains a global view of the network:

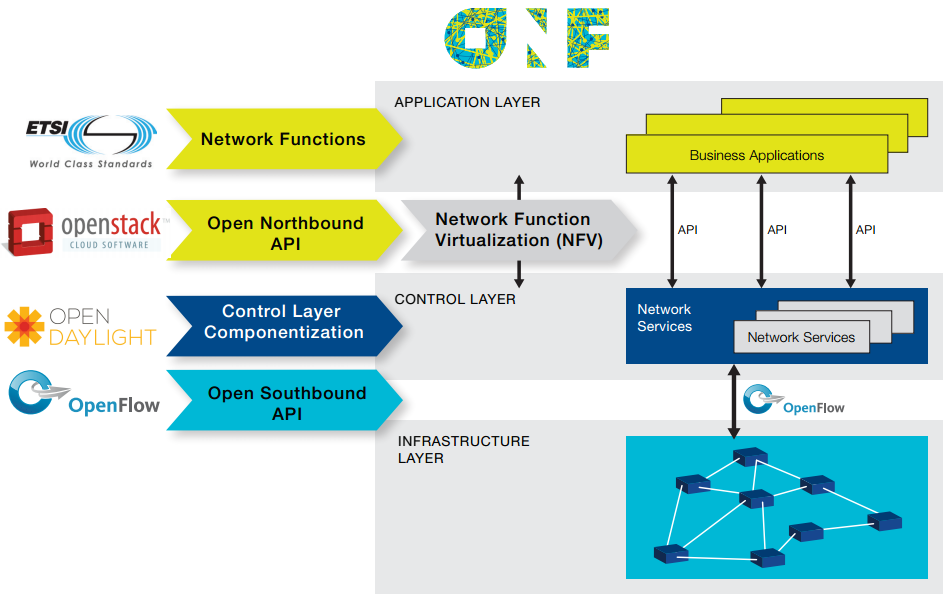



**Figure 2-7:** The Open SDN architecture, source [7]

1. **Application layer and Northbound Application Programming Interface (API):** The *Northbound Open API* is defined as a software interface between the controller platform and the VNF applications running on the top. These interfaces are open to customers, partners, and the open source community for development. One of these open source providers is OpenStack, a development committee which aims to bring NFV API into the SDN picture.

2. **Control layer:** Figure 2-7 shows an open controller (i.e "OpenDaylight") in the core of the Open SDN architecture, which will be further presented in Section 4-11. Nevertheless, any of the existing controllers can be used as a controller platform. These controllers consists of programmable modularized structure.

3. **Infrastructure layer:** The *Southbound Protocols* define the control communications between the controller platform and data plane devices, including physical and virtual switches. Support for a wide range of physical and virtual switches, ensure that customers have the maximum choices and flexibility in designing and deploying their software-defined network. One of the protocols defined for the *Southbound interface* is OpenFlow, but the flexible Open SDN architecture also supports pluging configuration for other standardized protocols (e.g. Border Gateway Protocol (BGP), SNMP).

Figure 2-7 shows SDN and NFV as complementary technologies: OpenFlow-based SDN focuses to optimize the underlying infrastructure, whereas NFV concentrates on the deployment of network functions (e.g. firewalls, DNS, etc.) in the application layer [7].

#### 2-4-1-1 OpenFlow Protocol

The OpenFlow protocol constitutes a mean for the SDN controller to add, update, and delete new entries in flow tables, both reactively (in response to packets) and proactively.

The ***Reactive flow instantiation*** happens when a new flow comes into the switch, the OpenFlow agent does a lookup in the flow tables and if no match for the flow is found, the switch creates an OpenFlow (OF) PACKET_IN message and forwards it to the controller for instructions. Reactive mode reacts to traffic, consults the OpenFlow controller and creates a rule in the flow table based on the instruction.

In contrast to that, ***proactive flow instantiation*** implies there is no lookup into the flow table and the flows are defined in advance, eliminating the latency introduced by interrogating the controller.

The ***hybrid flow instantiation*** is a combination of both reactive and proactive flow instantiation. Employing this flow instantiation mode will allow the flexibility of reactive mode for a granular traffic control, while preserving a low-latency forwarding for the rest of the traffic [46].

The most recent OpenFlow Switch version (1.4.0) has been defined by ONF in paper [8] and it is composed of three main modules (Figure 2-8):

1. *OpenFlow Channel* connects the switch to a controller and allows commands and packets to be sent between the controller and the switch. The controller manages the switch via the OpenFlow protocol running over Secure Sockets Layer (SSL). That is the reason why *OpenFlow Channel* is also referred as *Secure Channel*;

2. *A Flow Table* with an action associated to each flow entry which dictates the switch how to process the flow. Each flow table contains a set of flow entries (i.e. match fields, counters and a set of instructions);

3. *Group Table* contains group entries and each entry has a list of *actions buckets*.These actions are applied to packets which are sent to the group entries [8].

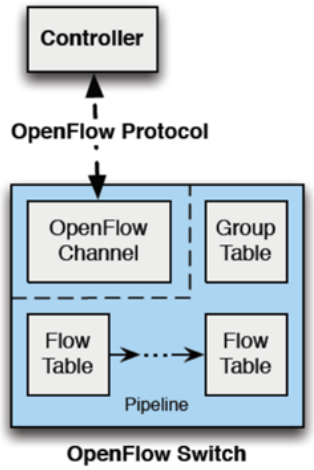



**Figure 2-8:** OpenFlow switch, source [8]

### 2-4-1-2 OpenFlow Switch

An OpenFlow switch is modeled as a group table and a collection of flow tables containing three columns: *rules, actions, and counters* (Figure 2-9). The *rules* column specifies the header fields that define the flow. Rules are matched against the headers of incoming packets. If a *rule* matches, the actions from the *action* column are applied to the packet and the counters in the *counter* column are updated. If a packet matches multiple rules, the rule with the highest priority is applied.

Each rule specifies an exact match from header fields or a wild card i.e. *ANY*. The set of possible actions are: forward the packet to an output port, modify the packet in some fashion, or send the packet to the next table or to the group table. An OpenFlow switch supports three types of OpenFlow ports: *physical ports, logical ports* and *reserved ports* [8].

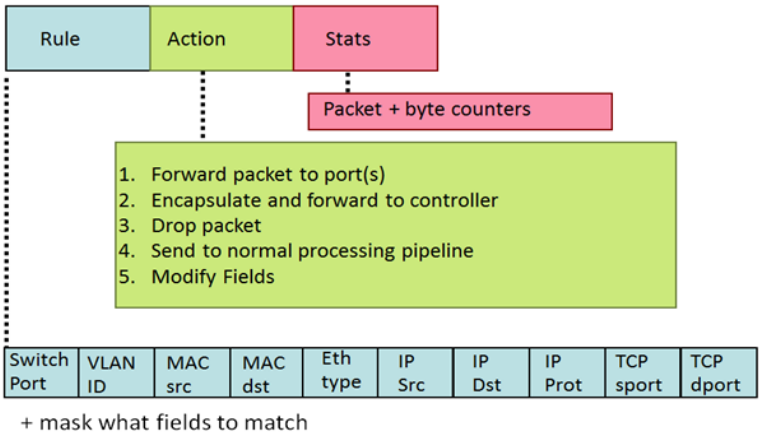



**Figure 2-9:** Flow Table, source [9]

The OpenFlow *physical* ports correspond to a hardware interface of the switch (i.e. on an Ethernet switch, physical ports map one-to-one to the Ethernet interfaces). The OpenFlow

*logical* ports are switch defined ports that do not correspond directly to a hardware interface of the switch (i.e. packet encapsulation). The only difference between physical ports and logical ports is that a packet associated with a logical port may have an extra metadata field called *Tunnel-ID*. The *reserved* ports are defined in OpenFlow switch specification [8]. A switch is not required to support all reserved ports, just those marked as "required".

### 2-4-2 Open vSwitch

Open vSwitch is an open source software switch designed to be used as a virtual switch in virtualized server environment. A vSwitch forwards traffic between different Virtual Machine (VM)s on the same physical host and also forwards traffic between VMs and the physical network. Open vSwitch can currently run on any Linux-based virtualization platform (kernel 2.6.32 and newer), including: VMware, Kernel-based Virtual Machine (KVM), VirtualBox, Proxmox, etc.

Like a physical OpenFlow switch it supports standard management interfaces and protocols (e.g. NetFlow, sFlow, Switch Port Analyzer (SPAN), Remote SPAN (RSPAN), Command Line Interface (CLI), Local Control and Accountability Plan (LACP),Virtual LAN (VLAN)(802.1ag)) and multiple tunneling protocols (Generic Routing Encapsulation (GRE), Virtual Extensible LAN (VXLAN), IPsec, GRE and VXLAN over IPsec). In addition to that, the Open vSwitch is designed to support distribution across multiple physical servers. Open vSwitch configurations consists of bridges and ports. Ports represent connections to physical interfaces and patch cables [47].

Packets from any given port on a bridge are shared with all the other ports on that bridge. Bridges can be connected through Open vSwitch virtual patch cables or using Linux virtual Ethernet cables (*veth*). Additionally, bridges appear as network interfaces to Linux, thus IP addresses can be assigned to them (Figure 2-10).

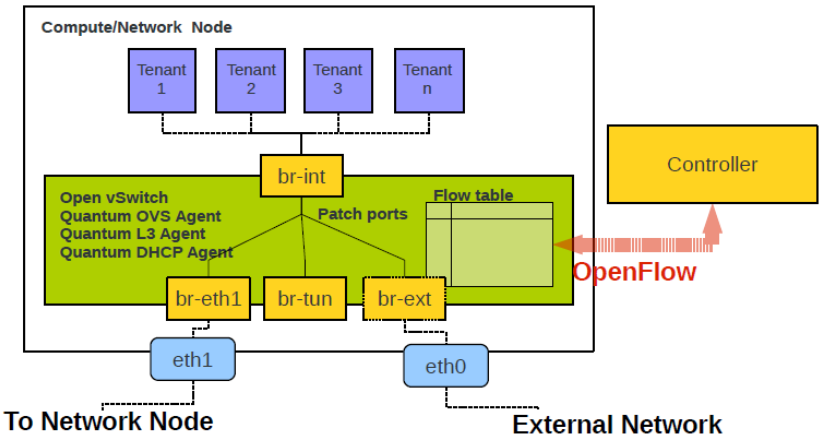



**Figure 2-10:** openvSwitch architecture, source [10]

# Chapter 3

# Proposed System Design

## 3-1 Evolved Packet Core (EPC) Software Defined Networking (SDN)-based architecture

The SDN integration in the current EPC is a complex task which involves careful assessment of the 3rd Generation Partnership Project (3GPP) standardized protocols. The most important challenge is how to preserve the Long-Term Evolution (LTE) functionality within a new flexible centralized EPC SDN-based architecture. This Chapter proposes a novel approach to re-design the main procedures for both control and data planes relying on new techniques such as SDN and Network Function Virtualisation (NFV). Firstly, the challenges of this new architecture are addressed and the proposed solutions are presented. Secondly, possible enhancements are investigated in terms of flexibility, complexity as well as the performances based on related technologies that could eventually optimize the proposed system design.

### 3-1-1 Architectural Functionality

The proposed architecture aims to slightly change the existing 3GPP architecture in order to integrate the legacy core components into the OpenFlow network. The main core interfaces: *S1-MME, S1-U, S6a* and *Gx* functionality are maintained, as well as the User Equipment (UE) authentication, authorization and intra-3GPP mobility management performed by Mobility Management Entity (MME). The current Serving Gateway (SGW) / Packet Data Network Gateway (PGW) selection mechanism based on Domain Name System (DNS) weights has been changed. The MME will query the OpenFlow controller via the *NorthBound interface* (Representational State Transfer (REST) Application Programming Interface (API)) that is able to install forwarding rules in the OpenFlow switches (see Figure 3-1). The functionality of the main components is presented as follows:

- The ***SDN controller*** is the central entity in charge of the user plane management and routing decisions between eNodeB User Plane (eNB-U) and S/PGW User Plane

(S/PGW-U). One of the SDN controller responsibilities is to maintain updated port statistics, balance the traffic and perform flow scheduling. The controller queries the transmitted bytes on each switch periodically and based on the reported information, it will equally distribute the load. For a default bearer, the flow scheduling mechanism calculates a path from source to destination based on the bandwidth limitation, UE's profile demands and installs flow entries in eNB-Us and S/PGW-U switches. The controller can install rules in the OpenFlow (OF) switches in both reactive and proactive modes: during the default bearer establishment, the controller functions in proactive mode (it populates the flow entries in advance) whereas, for dedicated bearers it takes decisions based on the information contained in the packet header (reactive mode).

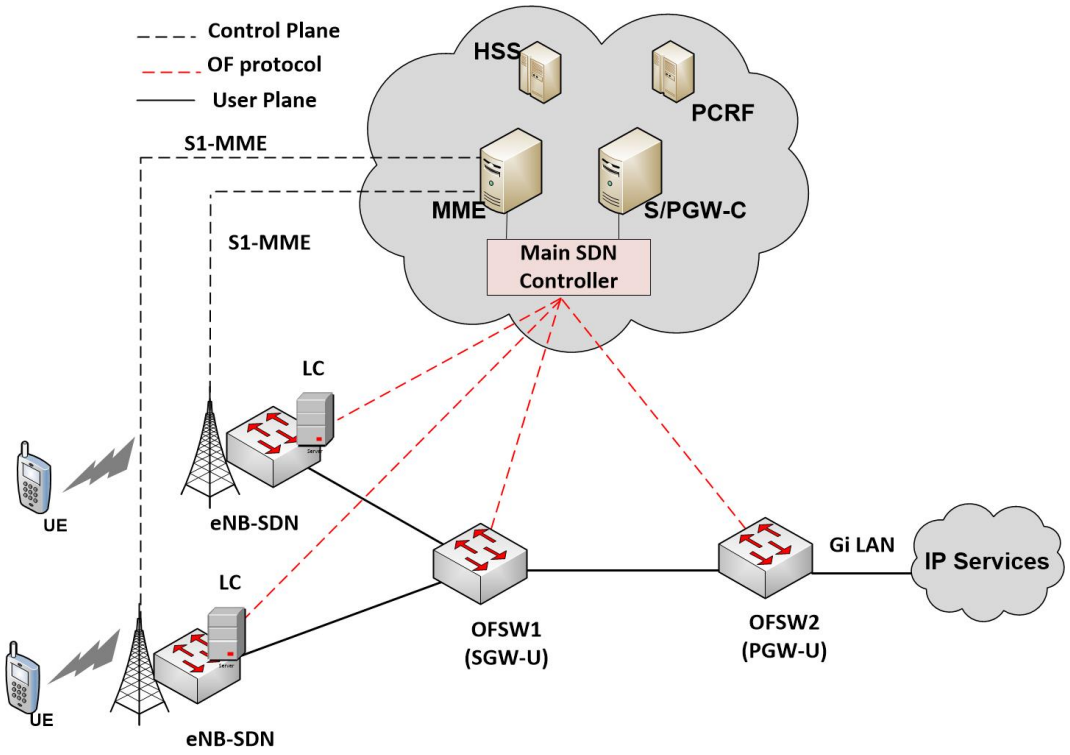



**Figure 3-1:** EPC SDN-based proposed architecture

- The ***MME*** is no longer responsible of S/PGW-U selection and it queries the controller via NorthBound interface using the Open API. One of the main advantages of this solution is the high elasticity, which allows mobile operators to keep standalone MME or integrate it into the cloud. For instance, MME, S/PGW Control Plane (S/PGW-C) application and Policy and Charging Rules Function (PCRF) can run as Virtual Machine (VM)s (i.e. OpenStack Neutron) on the top of the controller and in this manner the standard interfaces such as *S1-MME* (between Evolved Node B (eNB) and MME), *S6a* (between MME and Home Subscriber Server (HSS)) and *Gx* (between PGW and PCRF) could be maintained.

- ***S/PGW-C***: the control functions of SGW and PGW such as Tunnel Endpoint Identifier (TEID) allocation are lifted on the top of the controller. A database is required with the goal to store the UE Internet Protocol (IP) address after the MME query and

keep a mapping between bearer Quality of Service (QoS) parameters (transmitted by PCRF to **SWG!** (**SWG!**)-C and the user's profile). In this architecture the control plane and GPRS Tunneling Protocol Control Plane (GTP-C) is maintained in the sense that it adds transparent functionality to the 3GPP standard control interfaces.

- ***eNB-SDN/ S/PGW-U***: are OF switches which are able to process the flow entries received from the controller in their flow tables. These switches are responsible for user traffic forwarding between Evolved Node B (eNodeB) and IP services (i.e. Internet).
- ***Local Controller (LC)*** is responsible for packet classification at the access switches based on Location IP (LocIP) (consisting of UEs IP address and Base Station (BS) prefix). It communicates to the central controller through an Open API.
- ***eNodeB*** preserves the radio functions specified in 3GPP standards for the radio interface.

#### 3-1-1-1 Control Plane

**3-1-1-1-1 Local Controller (LC):** The LC is responsible to classify the packets based on several attributes, apply wildcard rules and form the location dependent address (LocIP) (see Figure 3-1). This is mainly done to identify the users connected to the same base station and to handle mobility when the users attach to another eNB. The LocIP consists of the UE ID and the base station prefix.

Even though the access switches can maintain per-flow state, it is not scalable for the controller to manage the network at the flow level, thus the main goal of the flow aggregation is to minimize the processing resources in the controller. Flows coming from the same user (source IP) and destination port can be aggregated in a single bundle and routed in the same manner. After the aggregation, the local controller sends the classified packets to the central controller. A major function of the LC is to distribute the flows equally and avoid congestion in the network at the edge switches. The LC communicates with the main SDN controller through OpenFlow protocol.

**3-1-1-1-2 Main SDN controller (MC):** Packets entering a Differentiated Services (DiffServ) domain can be classified upon several parameters: source and destination IP addresses, Layer 4 protocol and port numbers, Type of Service (ToS) byte value or Layer 2 information. As soon as these packets are classified at least on the basis of one of the parameters mentioned above, they can be processed, conditioned and marked [11]. The central controller role is to categorize traffic into different classes and apply QoS parameters to those classes.

The DiffServ architecture defines the DiffServ field, which replaces the ToS field in IPv4 to make Per-Hop Behavior (PHB) decisions about packet classification and traffic conditioning functions, such as metering, marking, shaping, and policing. To accomplish this, packets are first divided into classes by marking the ToS byte in the IP header. The original IPv4 ToS as defined in [16] is composed of a four bit field as presented in Figure 3-2. In [48], the ToS consists of the Differentiated Services Code Point (DSCP) (i.e. first 6 bits) which adds 2 zero bits at the end in order to form the entire ToS byte.

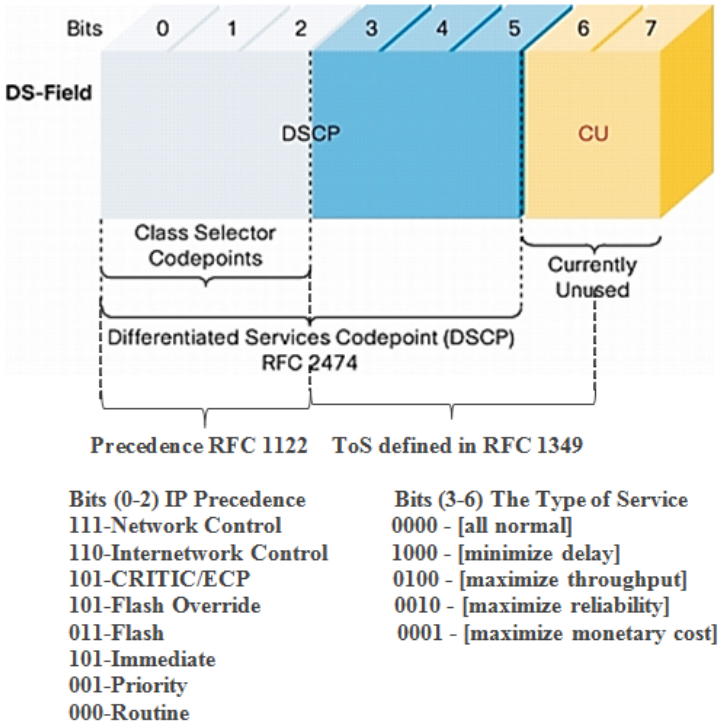



**Figure 3-2:** DiffServ Codepoint Field, source [11]

| Class | DSCP | Standard | QCI | ARP | QoS Level | Traffic Class | Applications |
|---|---|---|---|---|---|---|---|
| **BE** | 0 | RFC 2474 | 8; 9 | 1-15 | Bronze | Background | Email, SMS, MMS Alert Notification |
| **AF21** | 10 | RFC 2597 | 6 | 1-15 | Silver | Interactive | Web, Database, GIS, Records |
| **AF22** | 12 | RFC 2597 | 6 | 1-15 | Silver | Interactive | Web, Database, GIS, Records |
| **AF23** | 13 | RFC 2597 | 6 | 1-15 | Silver | Interactive | Web, Database, GIS, Records |
| **AF41** | 34 | RFC 2597 | 2 | 1-15 | Gold | Streaming | Surveillance video, In-Car Streaming |
| **AF42** | 36 | RFC 2597 | 2 | 1-15 | Gold | Streaming | Surveillance video, In-Car Streaming |
| **AF43** | 38 | RFC 2597 | 2 | 1-15 | Gold | Streaming | Surveillance video, In-Car Streaming |
| **EF** | 46 | RFC 3246 | 1 | 1-15 | Platinum | Conversational | Voice, Telephony |

**Table 3-1:** DSCP Mapping to QCI, source [16]

**3-1-1-1-3 Packet Classification: Class of Services** The basic classes defined by DiffServ are ”default” or Best Effort (BE), Expedited Forwarding (EF) and Assured Forwarding (AF). Among these, EF is used for ”strict” priority (e.g. video and voice), and AF is used for business differentiation (e.g. weighted-fair priority). Packets can be marked with an arbitrary DSCP value, corresponding to the appropriate AF, EF or BE class. In 3GPP, the QoS Class Indicator (QCI) presented in Section 2-1-2 (Chapter 2) maps directly to DSCP defined in [48] (see Table 3-1).

1. *Best Effort (The Default PHB (BE)* (defined in [48]): The default PHB specifies that a packet marked with a DSCP value (recommended) of ‘000000’ gets the traditional best effort service from a DS-compliant node (a network node that complies to all core DiffServ requirements). A BE service can be associated with a Hypertext Transfer Protocol (HTTP) session (Transmission Control Protocol (TCP) port 80) which is sent, for instance, to a firewall that blocks all the other traffic except from port ”80” (see

Figure 3-3). Because the BE traffic with the same destination port "80" can come from different eNBs with different tags, we need to optimize the resource utilization of the gateways. Thus, a best-effort system should give all the users a reasonable chance of getting a share of the available bandwidth.

2. *Expedited Forwarding PHB (EF)* (defined in [49]): The EF is the main key in DiffServ for providing a low-loss, low-latency, low-jitter, and assured bandwidth service. Applications such as Voice over IP (VoIP), video, and online trading programs require a robust network-treatment. Expedited forwarding can be implemented using priority queuing, along with rate limiting on the class. Although EF, when implemented in a DiffServ network, provides a premium (i.e platinum) service, it should be specifically targeted towards the most critical applications, because when congestion appears, it is not possible to treat all or most traffic as high priority. Expedited Forwarding is especially suitable for applications (like VoIP) that require very low packet loss, guaranteed bandwidth, low delay and low jitter. The recommended DSCP value for EF is '101110'=46 (Table 3-1). The ToS byte uses all eight bits, while the DSCP is only using the leading six digits. The EF pattern discussed above will become '10111000'=184 when considering the entire octet.

3. *Assured Forwarding PHB (AF)* (defined in [50]): The AF divides traffic into four classes, where each AF class is guaranteed a minimum number of resources (i.e. capacity and buffering). Within each class, packets are further partitioned into one of three drop preference categories. In case of a congestion in a DiffServ-node on a specific link, and packets of a particular AFx class (i.e AF1) need to be dropped, the packets in AFxy will be dropped such that the dP(AFx1) <= dP(AFx2) <= dp(AFx3), where dP(AFxy) is the probability that packets of the AFxy class will be dropped. The AFx class can be denoted by the DSCP 'xyzab0,' where 'xyz' is 001/010/011/100 and 'ab' represent the drop precedence bits. As an example, packets in AF13 will get dropped before packets in AF12 and before packets in AF11. The level of forwarding assurance of an IP packet depends on the amount of resources reserved for its AF class, the current load of its AF class, and the drop precedence of the packet [51].

#### 3-1-1-2 User Plane

**3-1-1-2-1 PDN Gateway User (PGW-U) (Gateway switch):** The PGW-U switch caches the state of the packets sent to the Internet and performs Network Address Translation (NAT) without querying the central controller to translate the UE address. In order to cache the associated user state, the PGW-U switch will need a larger table which does not necessarily imply more expensive Ternary Content-Addressable Memory (TCAM) with optimized processing capabilities. Due to the vulnerability towards malicious attacks coming from the Internet, these threats can be avoided by configuring access lists or firewalls (Figure 3-3).

#### 3-1-1-3 Service Chaining

**3-1-1-3-1 Midleboxes:** Middleboxes are required between PGW-U switch and Internet in order to add extra rules to different traffic (Figure 3-3). These midleboxes can be either

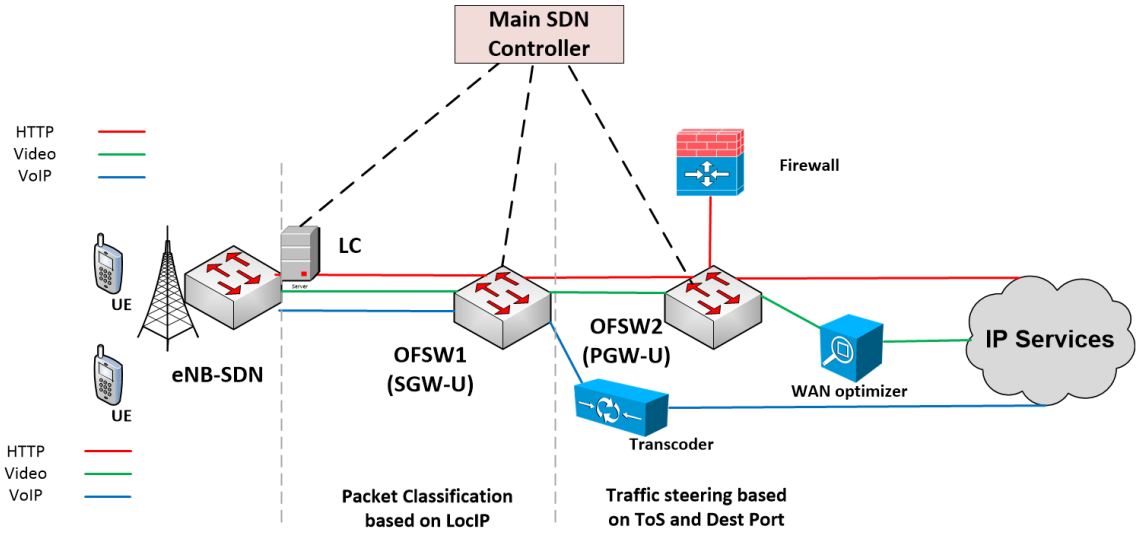



**Figure 3-3:** Packet classification and traffic steering

deployed in standalone hardware in multiple locations (i.e. firewalls, transcoders, Wide Area Network (WAN) optimizers, web proxy caches, intrusion detection systems, etc.) or run on virtual machines from one location.

## 3-2 Challenges

The proposed EPC SDN-based architecture brings extra challenges such as protocol stack, dynamic flow aggregation, policy paths and the routing centralized solution. The following subsections present these issues along with the proposed solutions to these problems.

### 3-2-1 Challenge 1: Protocol Stack

#### 3-2-1-1 Challenge 1-a): GPRS Tunneling Protocol (GTP) encapsulation

The GTP is the key protocol within EPC: in the user plane GTP tunnel is uniquely identified by a pair of TEIDs (corresponding to source and destination nodes) together with IP source and destination addresses and User Datagram Protocol (UDP) port numbers. In the traditional EPC architecture, every bearer is identified by a tunnel and one UE can have multiple sessions corresponding to multiple bearers. Since the employed architecture is OpenFlow based, an OF switch is the platform used for processing all the traffic. However, the OpenFlow protocol most recent version (OpenFlow 1.4.0 [8]) does not support GTP matching, neither TEID in the GTP header.

#### 3-2-1-2 Solution 1-a):

There are two possible solutions to this problem. One option is to extend the OpenFlow protocol in order to support GTP tunnel termination and TEIDs, respectively. Another

possibility would be to keep OpenFlow untouched and add a separate entity (i.e. a line card) responsible for GTP tunneling termination. This should function as a ”decapsulator” and it can be placed between eNodeB and the eNB-U switch. This entity main function is to remove the TEIDs. In this manner, the network between eNBs and the Internet is an IP core that can be easily managed by the controller (Figure 3-4).

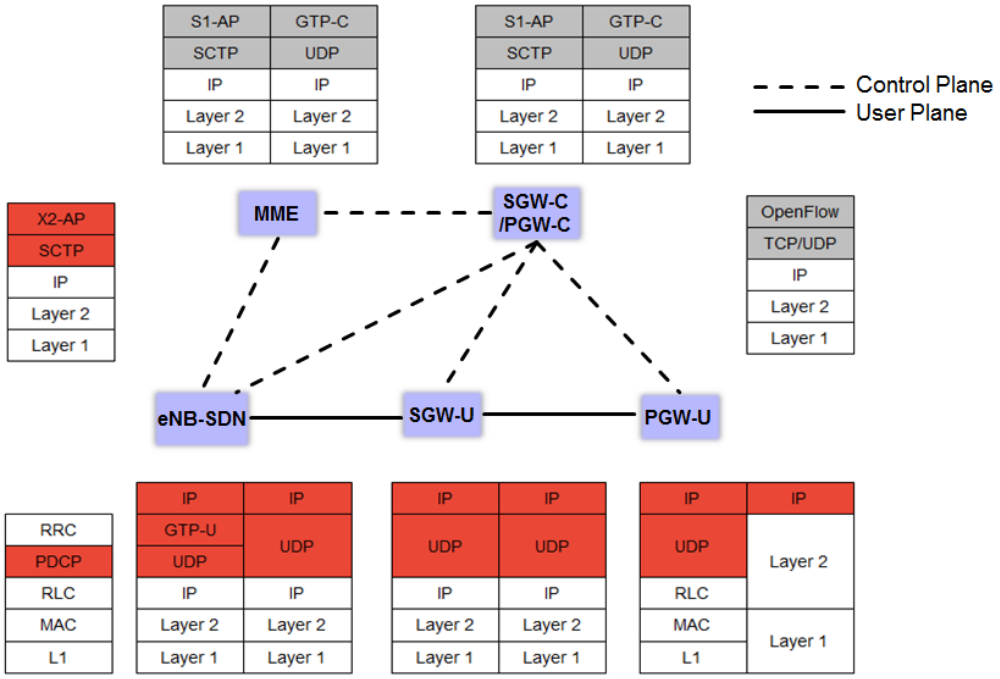



**Figure 3-4:** EPC-SDN based protocol stack

### 3-2-1-3 Challenge 1-b): GTP decapsulation

Nevertheless, there is no Ethernet header after the UDP header in the GTP stack. After GTP decapsulation, the packet will not have any Layer 2 information, thus, in order to implement the GTP protocol in the Open vSwitch, Layer 2 information is required. Hence, in the case when we want to extend Open vSwitch to support GTP protocol, the Layer 2 information has to be added.

**3-2-1-3-1 Solution 1-b):** A possible solution to this problem is to create a simple Ethernet header and the packets will be forwarded based on Layer 3 information, since the Layer 2 information does not exist. Another option is to simply route the packets based on the available information. In this way, we take full advantage of the Open vSwitch capabilities by only specifying the input and output ports. The performance of this solution is expected to be better in terms of throughput than the solution in which we can enable Layer 3 mapping and modify the Ethernet header. The Layer 3 mapping and Ethernet header change will increase the overhead of the GPRS Tunneling Protocol User Plane (GTP-U) tunnel. These two solutions has been throughly compared in [24]. Based on the obtained performance, this

paper adopts the first solution without the implementation of the GTP in the Open vSwitch (Figure 3-4).

## 3-2-2 Challenge 2: Routing centralized decision making

### 3-2-2-1 Default and Dedicated bearers

In the standardized LTE network, the QoS is implemented between UE and PGW on a set of bearers which provide special treatment to a set of traffic. These bearers can be either *default bearer* (which is "best-effort" service) when the UE attaches to the network for the first time and the connection is maintained as long as UE is attached, or *dedicated bearer* which acts as an extra bearer on the top of default bearer. The dedicated bearer provides differentiated treatment for different sessions (e.g. VoIP, video, etc.) [39]. In the EPC SDN-based architecture, the concepts of *default* and *dedicated* bearers have to be also defined.

### 3-2-2-2 Solution:

***Default Bearer:*** The central controller will return the TEIDs for control and IP addresses of the eNB-U and S/PGW-U, respectively. For instance, based on a load balancing mechanism and receive of periodical load statistics, the controller will get the information about the less overloaded switches and it will install flow entries throughout the data path. The default bearer establishment does not require the main controller intervention (reactive instantiation), since the switches have already a flow entry installed.

***Dedicated Bearers:*** The LC aggregates the flows in bundles based on the LocIP, which includes the UE and eNB ID. By employing wildcard matching, we reduce the number of flow entries in the SGW User Plane (SGW-U) and the processing time. At this stage, the number of bundles is proportional to the number of users since all the user sessions are grouped in one flow with an unique ID (bundle ID or policy tag). In the following step, the SGW-U switch will send a PACKET_IN message to the main controller which will look in its database table at the QCI value for that session (corresponding to the Destination port) and the ToS bit and it will establish a policy path. We can either employ one of the class of services defined in Section 3-1-1-1-3 and limit the transmission rate or redirect the traffic to above specified midleboxes.

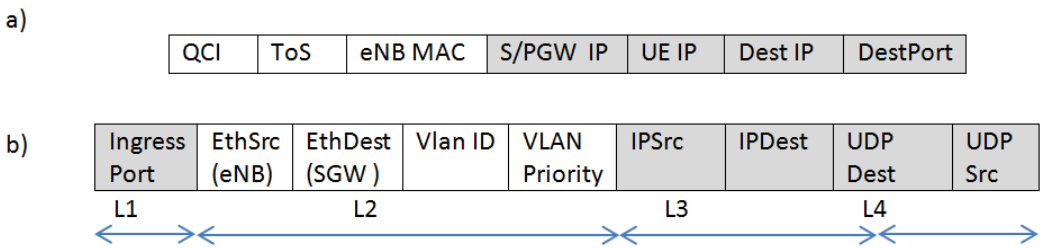



**Figure 3-5:** Controller Table (a) and Flow table in eNB-SDN (b)

The main SDN controller is responsible for choosing the shortest path, install policies on the path and configure the flow entries in the switches. As soon as the UE is authenticated, the

MME sends the *Attach Request* message to the controller. The database table is filled up with UE IP, the QCI values corresponding to the user sessions and the type of subscription. The SDN controller maintains an updated record of the port statistics by interrogating the local controller and the S/PGW-U within a period of milliseconds (Figure 3-5).

### 3-2-3 Challenge 3: Dynamic Flow Aggregation

#### 3-2-3-1 Asymmetric edge

Typically, in a mobile network scenario, millions of users are attached to thousands of eNBs, which are connected to several gateways. Due to the fact that a cellular network has to serve millions of customers with different requests, this implies fine-grained and sophisticated policies on the service chain. In comparison to a data center where the traffic stays inside, in a cellular network the mobile traffic goes back and forth from the user to the Internet. Another characteristic of mobile network is the fact that it has asymmetric edge (i.e. the access edge has lower bandwidth than the gateway edge). Most traffic is initiated from mobile devices entering the access edge.

**3-2-3-1-1 Solution:** If we employ the proposed EPC SDN-based architecture, in order to save resources and Central Processing Unit (CPU) power, flow granularity can be used as a way to aggregate flows. Instead of having millions of flows with ten OpenFlow entries, we can bundle them rather than route each flow individually. Therefore, one of the options is to do flow aggregation at the access edge (eNB) and then apply rules at the gateways in order to implement policy paths towards the Internet. The packet classification is performed when traffic enters the network at the access edge and encodes classification results in packet header. Then, when traffic returns from the Internet, the gateway edge only needs to do forwarding, as classification results are implicitly piggybacked in the packet header.

### 3-2-4 Challenge 4: Policy Paths

Due to the increase in real-time applications, the QoS in mobile networks has become an important requirement. An end-to-end QoS mechanism should ensure a special traffic treatment for different user profiles and different sessions. This can be achieved by installing policies on the path along with traffic shaping.

**3-2-4-0-2 Solution:** These policies can be applied not only to individual flows, but also to application / service specific bundles, in this way tailoring the circuit to have characteristics beneficial for the application or service (i.e. the path over which the bundle is routed). For instance, VoIP traffic must benefit from low latency paths. In the case of a VoIP bundle, a circuit can be dynamically created between source-and-destination packet switches, where the circuit path is the one with the smallest propagation-delay (Figure 3-3). In the same manner, all HTTP traffic can be redirected through a firewall on the path to the Internet. Another example is video traffic, where the low-latency for video is not as important as low-jitter. Moreover, we the video session can be routed over the non-shortest-propagation path in the employed topology.

The QoS has the responsibilities of storing classes of services along with their associated DSCP values, applying policies that either take advantage of the service class or apply a type queuing technique on the queues attached to ports on a switch. The approach starts by providing a "module" inside the SDN controller that will perform the matching, classification, flow insertion, flow deletion, and policy handling for QoS. The module is capable of two actions within the OpenFlow: "Enqueue" and modify the "Network ToS" bits. Furthermore, the module itself allows the definition of two types of policies: a "Queuing Policy" and a "ToS/ DSCP Policy". A queuing policy utilizes the "enqueue" action in OF switch to enqueue different types of flows in the network that match on certain criteria. The QoS module will send those flows to a queue on a switch port that has a predefined action of its own, such as rate limits, minimum bandwidth and/or bandwidth ceilings [35].

The actual configuration of the queues within a given switch is under the duties of the network administrator and it constitutes a Command Line Interface (CLI)-based configuration process. Fortunately, there is a new protocol currently under development that tries to eliminate this process by providing a control and data-plane separation for the configuration of queues, ports, and other QoS based arrangements. This protocol is called OF-Config and this protocol allows one or more OpenFlow data planes to be instantiated and assign queues and ports to the OpenFlow data planes [52].

## 3-3 User Mobility

In order for the proposed system design to be fully compliant with the mobility context, several aspects should be taken into consideration. First of all, the *Idle / Connected* state transitions have to be properly discussed. The call flows for two important procedures: *UE-triggered request* and *Network Triggered Request* are presented.

### 3-3-1 Idle / Connected States

One of the main characteristics of Evolved Packet System (EPS) is the "always-on" feature: there is permanently a default bearer set up as long as the user has his device turn on. The mobile core network is part of the Default EPS Bearer and the IP address assigned to the UE, stays with the UE as long as the UE remains powered on. The Default EPS Bearer is the first bearer that is established and the last bearer that is released. This allows the UE (specifically the applications using the IP connection) to transmit information to the server and the server to transmit information to the UE at any time. This state UE is called the UE *REGISTERED* state.

In contrast to the core network, which keeps the Default EPS Bearer resources and the IP address "persistently" allocated to a specific UE (as long as the UE is in the *REGISTERED* state), the Radio Network adopts a different approach. It dynamically allocates and releases radio network resources to support the Default EPS Bearer with transitions between the two modes of the *REGISTERED* state, *CONNECTED* and *IDLE* modes, respectively. These transitions last about 100 milliseconds (Figure 3-6).

In *ECM-CONNECTED* state, there exists a signalling connection between UE and MME. This signalling connection consist of two parts: an Radio Resource Control (RRC) connection

between UE and eNodeB, and an *S1-MME* connection between eNodeB and MME. When the UE has new traffic to send, or learns about the network's intention to send new traffic, it sends the MME a *Service Request* message, transmitting to *EPS Connection Management (ECM)/ RRC-CONNECTED* state. When a UE is still registered at a network, but its *S1* connection is released due to inactivity, the UE has no radio resources available. Therefore, the UE is in *EPS Mobility Management (EMM)-REGISTERED*, while being in *ECM-IDLE* state.


**Figure 3-6:** Idle/Connected Transition, source [12]

Service requests can be triggered either by the UE or by the network and can be categorized as follows, depending on where the new traffic is generated:

1. In *UE-triggered Service Request* there is uplink data to be sent from UE to the network;
2. In *Network-triggered Service Request* there is downlink data to be sent from the network to UE;

## 3-3-2 Call Flows

### 3-3-2-1 User Triggered Request

A *Service Triggered Request* is a procedure that the UE executes when is in *Idle* mode and it needs to establish the bearers to send data or it has to send signaling traffic to the MME. The *default bearer establishment* has been analyzed in Figure 3-8 (steps 1)-10))as follows:

1. The Initial Attach Procedure in the proposed design is compliant to 3GPP architecture preserving the main messages exchange on *S1-MME* and *S11* interfaces. The UE initiates the *Attachment Procedure* by sending to eNodeB the *Attach Request* message (International mobile Subscriber Identity (IMSI), last Tracking Area Identify (TAI), and Attach Type). The *Attach Request* message is sent in the initial UE message to the MME over the *S1AP* interface. The *Attach Request* is embedded in the *Initial UE Message*. The message also includes the *Packet Data Networks (PDN) Connectivity Request* message. The TAI and E-UTRAN Cell Global Identifier (ECGI) are included as well. The eNodeB uses the eNB-UE-S1AP ID to uniquely identify the UE.
2. MME checks authentication in HSS. The MME retrieves the useful information to authenticate and interrogate the UE's profile from HSS. This which stores the subscription data for the UEs, such as: IMSI (the unique identifier of a Subscriber Identity

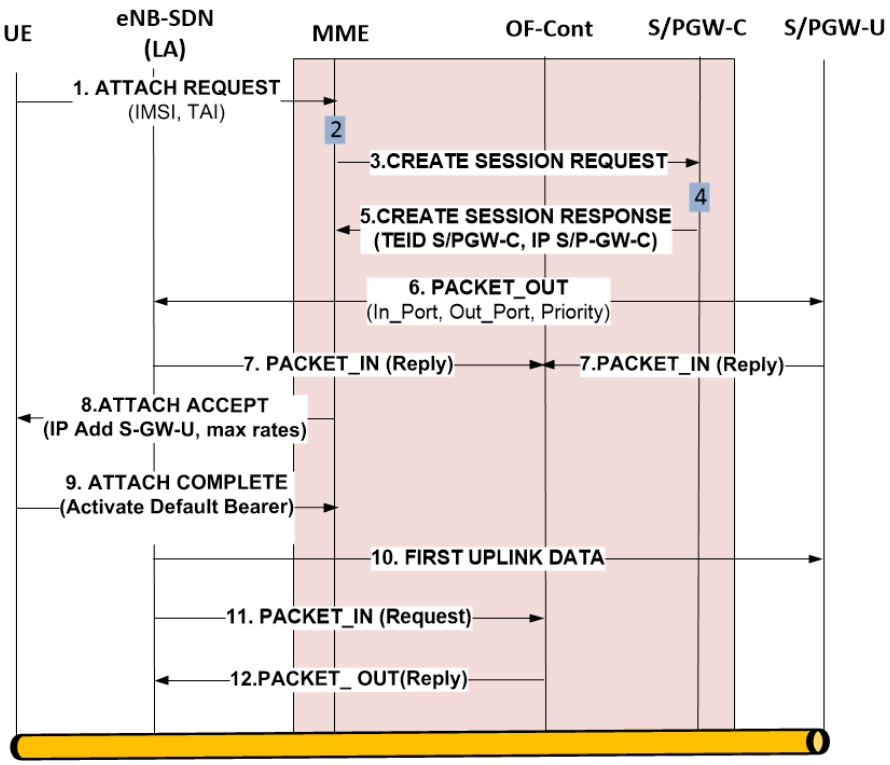



**Figure 3-7:** Initial Attachment Procedure

Module (SIM) card), subscribed QoS and Access Point Name (APN)s for that subscriber. MME performs UE authentication based on the security vector parameters. If the MME has changed since the last detach, the MME will send an Update Location Request (ULR) to the HSS which forwards the*Cancel Location* to the old MME. The HSS acknowledges the Update Location message by sending an Update Location Answer (ULA) message to the new MME [27].

3. Upon MME UE authentication in HSS (see steps 1) and 2) in Figure 3-8) in a standard 3GPP deployment, the MME queries DNS to perform selection of SGW and PGW. In the proposed architecture MME sends *Create Session Request* to the SDN controller via Northbound interface, an Open API.

4. In this step, the S/PGW-C application queries the PCRF in order to obtain the QoS for the initiated session, as well as the maximum download rates for uplink and downlink allowed at the APN level.

5. The S/PGW-C application returns to the SDN controller the TEID for control and IP addresses of the less overloaded OpenFlow switches together with the downlink and uplink maximum rates. A *Create Session Response* message is established between the SDN controller and MME.

6. Furthermore, the central controller communicates with the local controller to configure flow entries in the eNB-SDN switch and establishes policy rules in the S/PGW-U switches.

7. An OpenFlow response is sent from the switches to the local controller which further communicates with the central controller. The *PACKET_IN* (reply) message includes the port statistics: i.e. received and transmitted packets as well as the number of dropped packages. By employing steps 6) and 7) in Figure 3-8 with predefined rules, the switch processing time needed in packet forwarding is decreased.

8. The *Attach Accept* message sent by MME to eNB includes the IP address of S/PGW-U, as well as the downlink and uplink maximum rates. Nevertheless, the Non Access Stratum (NAS) messages are transparent to eNB which forwards them to UE. The RRC *Connection Reconfiguration* message is sent to activate the default radio bearer.

9. The message also carries the *Attach Complete* message since NAS Payload and UE replies back to MME via eNB. The *RRC Connection Reconfiguration* message is sent to activate the default radio bearer.

10. After the MME has received the *Attach Complete* message, the eNB-SDN sends the first UE's data packet on the uplink direction (i.e toward Internet).

11. When the packet arrives, the LC checks the source and destination IPs of the user packet, as well as the eNB Media Access Control Address (MAC) address and it modifies the packet header with the LocIP address. For a *default bearer establishment*, there is no need for aggregation, neither for policy rules on the path since in LTE the *default bearer* provides Non-Guaranteed Bit Rate (GBR). When the user has multiple sessions, a *dedicated bearer* has to be established. The LC will aggregate flows based on LocIP and it will further send the bundles to the S/PGW-U switches in a *PACKET_IN Request* message.

12. The central SDN controller analyzes the Destination Port of the user packet and maps the correspondence between the QCI of that particular session, user profile (i.e. premium, gold subscription) and ToS. Depending on the session type, it will install a policy path and configure flow entries in the S/PGW-U switches.

#### 3-3-2-2 Network Triggered Request

A *Network Triggered Service Request* is a procedure that the UE executes when it is in *Idle* mode and the network needs to establish the bearers to send data or it needs to send signaling to the UE (Figure 3-8).

(A) When the SGW-U receives a downlink data packet for an UE which is in *ECM-Idle*, it will perform a header matching in its flow table. The SGW-U sends a *Downlink Data Notification* message to the main SDN controller which has the UE context contained in *PACKET_IN* message. The controller queries MME, which replies with a *Downlink Data Notification Acknowledgment*. This acknowledgment will be further sent to the S/PGW-U switch in a *PACKET_OUT* message.

(B) If the UE is registered in the MME, the MME will send a *Paging* message to each eNodeB belonging to the Tracking Area (TA) in which the UE is registered. When the eNodeB receives the paging message from the MME, the UE is paged by the eNodeB. If the UE

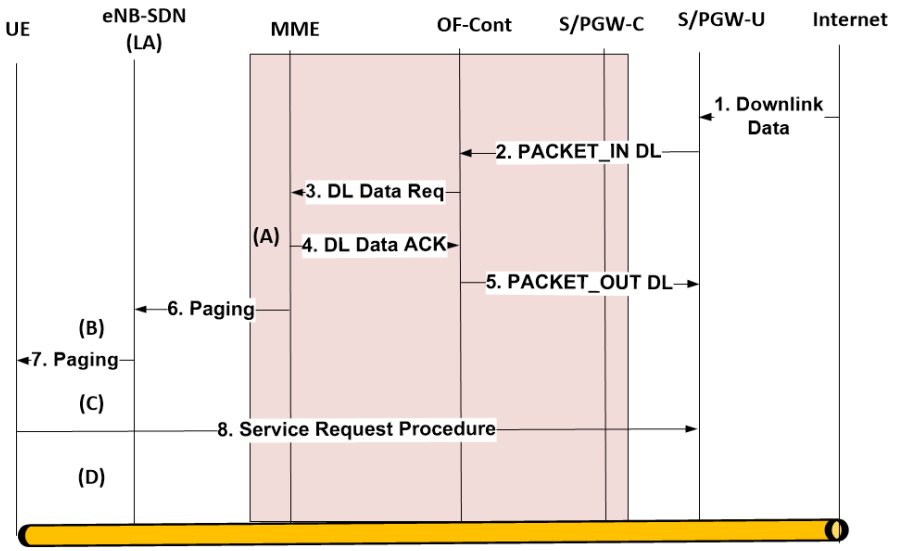



**Figure 3-8:** Initial Attachment Procedure

is in *Idle* mode, the MME will know the UE's last know location. Thus, if MME wants to find UE that is in idle mode, it will simply send *Paging* message to the eNB and eNB will page for the UE. But if UE is in new TA and that is not in the list received during initial attach, it will initiate a Tracking Area Update (TAU).

(C) Upon receipt of a paging indication, the UE initiates the UE triggered *Service Request* procedure which was previously discussed. To initiate a TAU, the UE should have a RRC connection. Firstly, it attaches to the nearest eNB by performing cell re-selection process. As soon as the RRC connection is established, the UE sends a TAU to network. Note that ECM state in network is still *Idle* for that UE, but TAU may move the ECM state in network from *Idle to Connected*.

(D) Once network receives TAU, the LC will store the UE's location info and it will install flow entries in the eNB-SDN switch. The SGW-U transmits downlink data towards the UE [12].

## 3-4 Analysis of the proposed system design and possible enhancements: scalability, complexity and performance

After a detailed presentation of the EPC SDN-based architecture along with the foreseen challenges, this section aims to explore possible enhancements to the proposed design in terms of:

1. *Flexibility*: new services, software upgrades and traffic management;
2. *Deployment Complexity*: implementation, scalability and costs;
3. *Performances*: routing, latency and traffic monitoring.

### 3-4-1 Flexibility

- *Control Plane:*

  Even though it preserves the standard 3GPP control interfaces, the flexibility introduced by the virtualized functions eliminates the overprovisioning in the traditional EPC, is meant to bring extra automation and intelligence in a standardized environment. The orchestration management layer allows dynamic capacity allocation for the VMs based on the requirements (both horizontal and vertical scalability). This is enhanced by traffic management and optimized routing in the S/PGW-C application. For instance, software upgrades can be maintained by the operator with no need for vendor implication. Another advantage that virtualisation might bring to operators, is the deployment of both production and test environments on the same platform.

- *User Plane:*

  The user plane elasticity is mostly dictated by the Open vSwitch capabilities. Therefore, by sending periodic updates to the controller of the received / transmitted and dropped packets, the overall redundancy in the network increases, so the controller will be aware of the congested links. This characteristic is strongly related to the load balancing mechanism which is meant to equally distribute flows to less overloaded switches. The flow priority mechanism and aggregation can establish the QoS for any particular type of service (VoIP, video, etc.).

  One of the major drawbacks in standardized LTE architecture is the tunnel establishment between eNB and PGW, even when there is no data traffic to be sent. This anomaly can be avoided if Open vSwitch triggers tunnel erase after a predetermined period of several seconds, or the connection period expires. Moreover, on the service chaining path, the traffic steering issue can be easily solved by the controller traffic management.

### 3-4-2 Deployment Complexity

- *Control Plane:*

  The complexity of the employed solution can be analyzed in terms of implementation tools, scalability and costs. Even if the proposed solution preserves the control plane functionality, from the implementation point of view, it requires both hardware and software changes. For instance, the MME, S/PGW-C and PCRF functions can run as VMs in OpenStack Neutron (in data centers) and can be communicate with the central controller through the REST API. Since the controller and OpenStack can be implemented on virtualized server platform, it drastically decreases the complexity of exceeded hardware found in an operator network. For instance, one controller can attach to hundreds of switches and process millions of requests per second. The performance of testbed consisting of a 100 switches allocating 16 cores for process with 400-500 effective flows per second was analyzed in [53]. In this manner, the scalability grows proportional to the complexity decrease. The cost in Capital Expenditure (CAPEX) and Operating Expenditure (OPEX) is one of the major advantages of deploying SDN in telecom network. According to [54], the EPC virtualization will increase CAPEX up to 30M Euro in the third year of deployment, whereas the decrease in OPEX is almost

100M Euro in the operations sector and up to 25M Euro lower budget allocated for maintenance purposes.

- *User Plane:* The innovative user plane consists of multiple OpenFlow hardware switches with integrated Open vSwitch (OVS). Open vSwitch [47] is a virtual switch which supports flows, Virtual LAN (VLAN)s, trunking, QoS, port aggregation and Layer 3. The employed solution simplifies the user plane GTP encapsulation in the traditional EPC by implying Layer 2 forwarding and flow aggregation based on ToS. The LocIP address aggregates flows coming from the same UE and eNB in a single tag and differentiates between user sessions based on the destination ports. Nevertheless, in a large production environment it might guarantee the network scalability. For instance, one of the possible deployments can be a fat-tree topology with edge (access), aggregation and core switches, which is very common topology in data center. This topology is very similar to the case of a SGW in pool when multiple SGWs can serve the same TA and multiple TAs can be served by the same SGW. The proposed topology in this paper is presented in Section 4-2-1-1, Chapter 4. One of the advantages of this solution relies on the low cost implementation, since the price of a single hardware OpenSwitch is thousands of Euro in comparison to millions that operators pay for the deployment and configuration of each gateway.

### 3-4-3 Performance

- *Control Plane:*

  The control plane performance in the EPC SDN-based architecture is expected to increase in terms of routing reliability in comparison to the standardized EPC deployment. For instance, the current load balancing mechanism is proactive and does not take into account the instantaneous load or capacity due to lack of load communication between GTP-C elements, which might lead to overload in one or more nodes of a SGW or PGW. While maintaining the GTP-C functionality, a new load balancing mechanism based on periodic updates of flow statistics can be employed in the S/PGW-C application.

  In paper [55] a dynamic flow scheduling system has been proposed in order to collect statistics from the switches, compute non-conflicting paths for the flows and re-route the traffic accordingly. A network-wide scheduler should measure the utilization of all links in the network and move flows from highly-utilized links to less utilized links. Based on the flow size detected at the edge switches, the scheduler measures the bandwidth consumed by each large flow, as well as the required capacity and uses placement algorithms to compute good paths which are further installed on the switches.

  Due to the fact that in the proposed architecture the main SDN-controller can manage thousands of switches and LC, it implicitly introduces a single point of failure into the network. New techniques like DevoFlow have been proposed in order to mitigate the flow scheduling overheads while maintaining the same network performance [56]. The network redundancy can be increased by deploying two central controllers which can communicate to each other via HyperFlow application [57]. The second controller serves as a backup in case of failure of the primary controller.

- *User Plane:*

The OpenFlow protocol allows controller software running on a server to configure the hardware forwarding tables in a network of switches. Whenever the switch receives a packet, it will first look up in its flow table for L2-L4 header matching and then perform one of the actions: forward the packet to a specific port, encapsulate the packet or drop it. If no match is found, the packet will be sent to the controller. In the proposed deployment the latency of the controller is decreased by installing predefined rules in the switch, thus for a default bearer establishment the switch will be able to take an action based on the information contained in the flow table. In every PACKET_IN message sent to the controller, the switch includes port statistics with the number of received/transmitted and dropped packets. Hence, in case of congested links, the traffic anomalies can be easily determined by the SDN controller.

Paper [32] addresses the challenges of implementing monitoring rules in the network in order to determine the load on each switch. In this paper, an adaptive algorithm based on linear prediction has been proposed with the goal to mitigate the amount of processed data by the controller and the overhead in the network produced by the load information. Recent paper [58] proposes to use sFlow standard in combination with SDN architecture in order to detect large flows in real-time and send this information to the Load Balancing Application on the top of the controller. In this manner, the extra load information in the packet header can be eliminated as mentioned in [32].

Paper [58] presents the use case of the load balancing where the sFlow-RT collector communicates with the Load Balancing Application via Open API. At a first glance, one of the approaches used to isolate the users and eNBs network is to employ VLAN tagging, but a more advanced solution could be FlowVisor, which attributes the packets coming from slice or cluster of switches to be processed by a particular controller. This solution might prevent a malfunctioning controller to saturate the switch CPU [59]. In case of the multiple LCs, this can be considered an extra identification procedure, apart from LocIP to separate different groups of eNBs.

# Chapter 4

# Technologies and Simulation Tools

This chapter presents the Network Function Virtualisation (NFV) and Software Defined Networking (SDN) tools and technologies used for the implementation and configuration of two test environments under different topologies: one emulated in a network simulation tool and another configured using a physical test bed. In addition to that, two open-source SDN controllers are throughly analyzed and compared. Furthermore, the details of Quality of Service (QoS) algorithms and implementation of a new load balancing module are described.

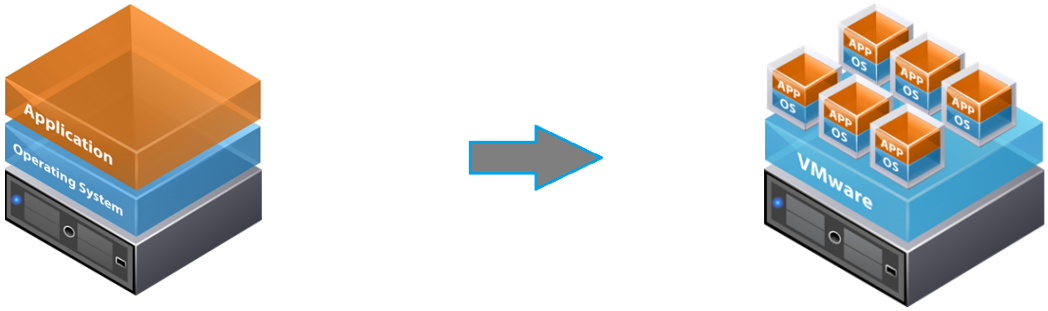
## 4-1 NFV-Virtualisation Environments

Today's x86 computer hardware is designed to run a single operating system and limited number of applications, leaving most machines mostly underutilized. Virtualisation technologies allow to run multiple virtual machines on a single hardware platform, with each virtual machine sharing the resources of that one physical computer across multiple environments. Different operating systems and multiple applications can run in different Virtual Machine (VM)s on the same physical computer. The system resources can also be dived between multiple VMs (Figure 4-1). A hypervisor is the software which allows to run multiple VMs instances on a single physical machine.

Nowadays there are many existing virtualisation tools such as: VMware, VirtualBox, Kernel-based Virtual Machine (KVM), Proxmox, Virtual Machine Manager (VMM), etc. In this project, two of them are presented: VMware which was used to set up the simulation test environment presented in Section 4-2 and KVM for the physical testbed and the Long-Term Evolution (LTE) simulator described in Section 4-3.

### 4-1-1 VMware Workstation 10 Features

VMware Workstation 10 is a Virtualised Environment provided by VMware. This is a hypervisor which allows having several operating systems installed on single server or computer




(a) Traditional Architecture (b) Virtual Architecture

**Figure 4-1:** VMware virtualisation environment, source [13]

and run multiple applications in progress. The resource allocation is based on the operational needs.

This following Subsection aims to present the benefits of the current VMware software version necessary for the implemented testbed. More details about the supported features and configuration are provided in product documentation [60].

#### 4-1-1-1 Configuring network connections

The VMware Workstation 10 allows three network connections modes: *Bridged Networking, Network Address Translation* and *Host-Only*. The configuration mode used in the simulation presented in this thesis is *Bridged Networking*.

1. *Bridged Networking Configuration* - Bridged networking connects a VM to a network by using the network adapter on the host system. The host network adapter enables the VM to connect to the Local Area Network (LAN) that the host system uses. Bridged networking works with both wired and wireless host network adapters. By default, *VMnet0* is set to use auto-bridging mode and it is configured to bridge to all active network adapters on the host system. A VM must have its own identity on a bridged network. For example, on a Transmission Control Protocol (TCP) / Internet Protocol (IP) network, the virtual machine needs its own IP address.

2. *Network Address Translation (NAT) Configuration* - With NAT, a virtual machine does not have its own IP address on the external network. Instead, a separate private network is set up on the host system. In the default configuration, VMs are dynamically assign an address on the private network from the virtual Dynamic Host Configuration Protocol (DHCP) server. The host system has a virtual network adapter on the NAT network. The NAT device forwards network data between one or more VMs and the external network, identifies incoming data packets intended for each VM and sends them to the correct destination.

3. *Host-Only Networking Configuration* - In a host-only network, the VM and the host virtual network adapter are connected to a private Ethernet network. The network connection between the VM and the host system is provided by a virtual network

adapter that is visible on the host operating system. The virtual DHCP server provides the IP addresses on the host-only network [60].

#### 4-1-1-2 Snapshot possibility

A VMware snapshot is a copy of the Machine Disk File (VMDK) at a given point in time. Snapshots provide a change log for the virtual disk and are used to restore a VM to a particular point in time when a failure or system error occurs. This is a very important feature of the VMware, as it allows to go back to a previous machine state (Figure 4-2).

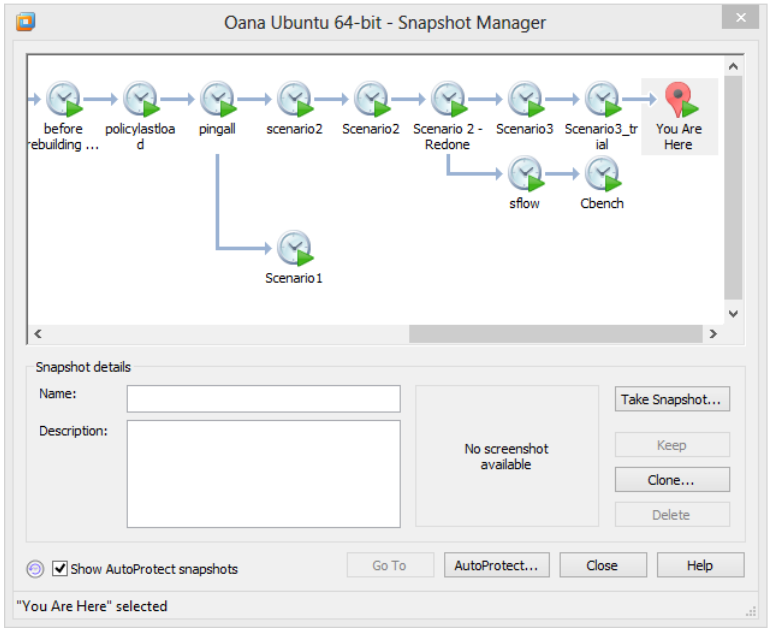



**Figure 4-2:** VMware snapshot

Any data that was writable on a VM becomes read-only when the snapshot is taken. VMware administrators can take multiple snapshots of a VM to create multiple possible point-in-time restore points. When a VM reverts to a snapshot, current disk and memory states are deleted and the snapshot becomes the new parent snapshot for that VM. The snapshot file cannot exceed the size of the original disk file, but it requires overhead disk space.

#### 4-1-1-3 VMs migration

The VMware Workstation 10 includes VMware hardware Version 10 and it is compatible with vSphere 5.5. The VMware vSphere feature is called enables the live migration of VMs from one host to another with continuous service availability. It enables the high availability of VMs by restarting them on a different host in the event of a failure.

Firstly a resource pool of servers is defined, then when a failure occurs, the VM state is transferred to another server from the pool, even though there are several applications in progress (Figure 4-3).

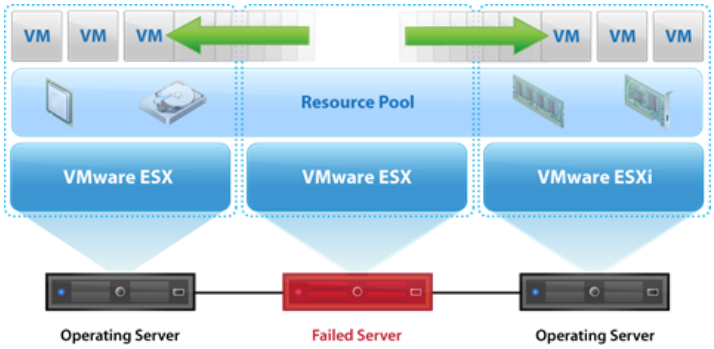



**Figure 4-3:** VMware live migration, source [13]

This feature minimizes the downtime and IT service disruption as it reduces the cost and complexity and it provides backup solution in case of failure or software errors in one or more VMs.

## 4-1-2 KVM

The KVM is a similar virtualisation tool to VMware. The difference between the two is that VMware can run on hosts operating systems like Windows 8, 7, Windows Vista, Ubuntu, Fedora, whereas KVM is a full virtualization solution developed only for Linux.

Using KVM, one can deploy multiple VMs running unmodified Linux or Windows images. Each VM has private virtualized hardware: a network card, disk, graphics adapter, etc. The kernel component of KVM is included in mainline Linux, version 2.6.20 [61].

A software tool which has been configured to run on KVM is OpenEPC, a full LTE network simulator. Figure 4-4 illustrates some of the KVM capabilities to monitor Central Processing Unit (CPU) usage of each VM. The Mobility Management Entity (MME) and Serving Gateway (SGW) are running in the sameVM, whereas the Home Subscriber Server (HSS), IP Multimedia System (IMS) and Policy and Charging Rules Function (PCRF) run in separate VMs (Figure 4-4).

### 4-1-2-1 OpenEPC (virtualized Evolved Packet Core (EPC))

OpenEPC is a project implementation of 3rd Generation Partnership Project (3GPP)'s EPC developed by the Fraunhofer FOKUS Competence Center for Next Generation Network Infrastructure (NGNI) and Technical University of Berlin (TU Berlin) [62].

OpenEPC consists of several software components offering advanced IP mobility schemes, policy based QoS control, and integration with different application platforms in converging network environments. This platform toolkit, in addition to fostering research and development, enables academic and industry researchers to realize state of the art Next Generation Mobile Network (NGMN) infrastructure and application testbeds.

The IP configuration of the testbed is illustrated in Figure 4-5. OpenEPC is composed of 4 switches: *Net_a_switch*, *Net_b_switch*, *Mgmt_switch* and *Net_d_switch*. Each of the 5

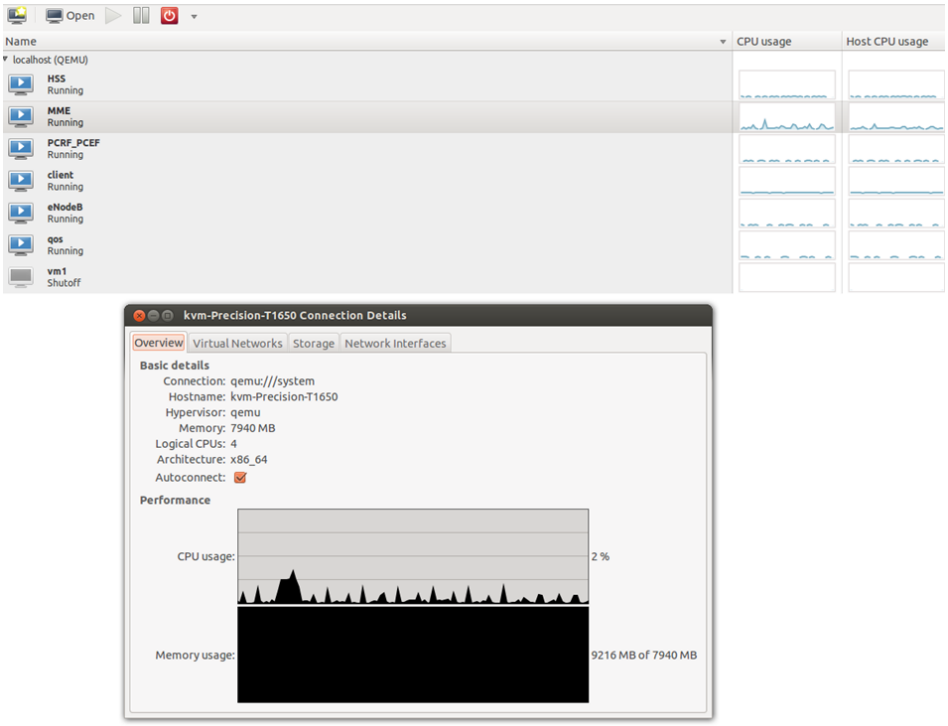



**Figure 4-4:** OpenEPC running in KVM

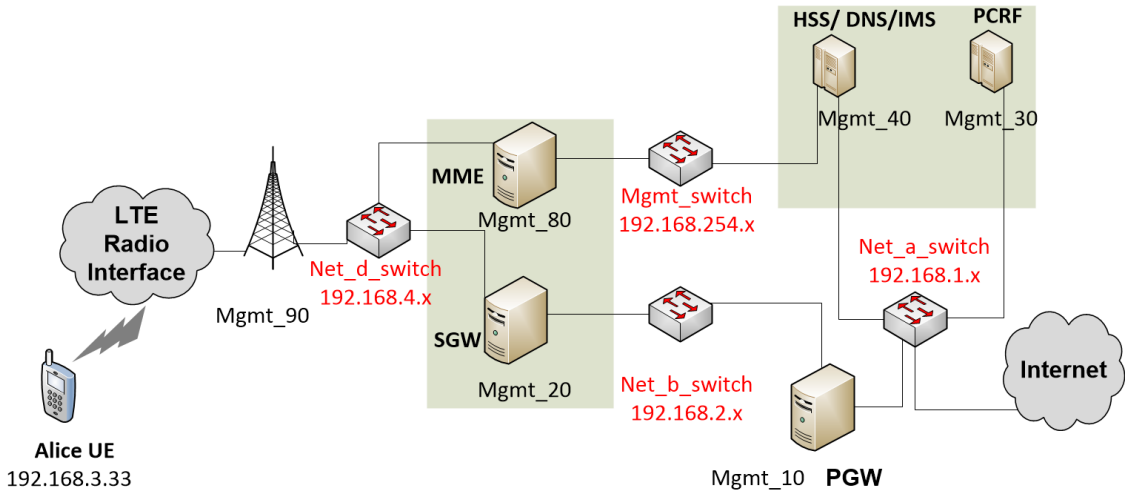



**Figure 4-5:** OpenEPC Architecture

VMs have IP configured on different interfaces corresponding to the 4 enabled switches (Table 4-1).

| VM | NIC Name | IP address |
|---|---|---|
| User Equipment (UE) | Mgmt | 192.168.3.33 |
| Evolved Node B (eNB) | Net_d_switch | 192.168.4.90 |
| MME | Net_d_switch; Mgmt_switch | 192.168.4.80; 192.168.254.80 |
| SGW | Net_d_switch; Net_b_switch | 192.168.4.20; 192.168.2.20 |
| Packet Data Network Gateway (PGW) | Net_b_switch; Net_a_switch | 192.168.2.10; 192.168.1.10 |
| HSS/Domain Name System (DNS)/IMS | Net_a_switch; Mgm_switch | 192.168.1.40; 192.168.254.40 |

**Table 4-1:** OpenEPC IP configuration

## 4-2 Network Emulation Test Bed: Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run on standard Linux network software and can be used as a flexible network testbed for developing OpenFlow applications. Mininet tool allows complex topology testing without any physical connections, only using Python Application Programming Interface (API) and it also includes a Command Line Interface (CLI) which is topology-aware.

Mininet uses process-based virtualisation to run many (up to 4096) hosts and switches on a single Operating System (OS) kernel. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using *virtual ethernet (veth)* pairs [63]. It can be downloaded as a VM which comes along with an Ubuntu 13.04 image. The major Mininet advantage is that it has developed its own controller, but it also allows the emulated network to connect to any other controllers by specifying the *IP* on which the controller is running.

### 4-2-1 Topology

Since there are no "correct" or "wrong" topologies, the majority of operators deploy their networks based on geographical and business requirements. Very similar to the data center model, a mobile topology is composed of three layers: access, aggregation and core. In paper [64] several topologies and strategies for mobile backhaul were studied in terms of scalability, capacity and costs. Paper [65] presents different topologies ranging from hub-spoke, ring, mesh or a combination of them. A mobile backhaul topology such as ring provides better resiliency and capacity than a tree topology, while a ring topology is more scalable and achieves lower latency. Therefore, a ring topology is recommended to connect the eNBs, whereas the mobile core network should be deployed in a full-mesh configuration for maximum resiliency and geo-redundancy purposes.

#### 4-2-1-1 Proposed Topology

The proposed topology (presented in Figure 4-6) consists of a wide fat-tree topology with 2 core switches, 4 aggregation switches and 8 edge switches attached to 5 hosts each. This topology is also known as "Clos" where every lower-tier switch is connected to each of the top-tier switches in a full-mesh topology. The core switches have the same number of links such as the number of aggregation switches, whereas the aggregation switches have the same number of links such as edge switches (Figure 4-6). If $k$ is the number of core switches, $k^2$ is the number of switches deployed in the aggregation layer, whereas $k^3$ is the number of edge

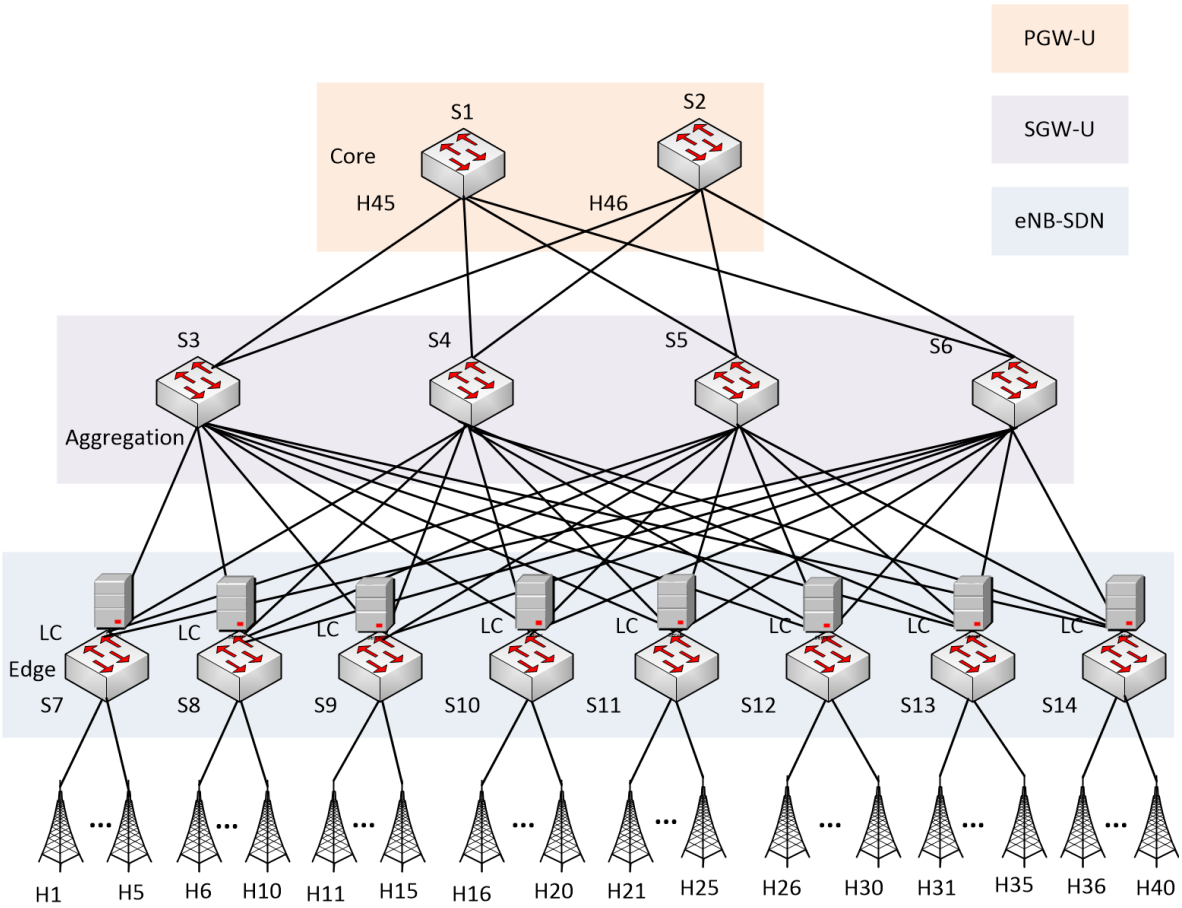



**Figure 4-6:** Fat-tree mobile topology

(Top of Rack) switches. This is a common topology encountered in data centers due to its high resiliency and possibility of load distribution.

This topology aims to simulate a mobile core network of 40 eNBs connected to 8 edge switches. The number of edge (Top of Rack) switches equals the number of Local Controller (LC)s which form the Location IP (LocIP), unique for every user. In the proposed topology each eNB has only one LocIP (i.e. UE IP and Base Station (BS) prefix) assigned. The 4 aggregation switches replace the SGW user plane functions with virtual OpenFlow switches. In contrast to the common EPC deployment, where the number of SGWs entities equals the number of PGWs, in the proposed topology, the number of aggregation switches corresponding to the SGW nodes is twice the number of core switches.

In the Mininet code (see Appendix 4-6), the links between the switches has been configured with different bandwidth capacity: between hosts and edge switches the link is limited to 50 Mbps, whereas links which connect the edge switches with the aggregation switches are configured for 250 Mbps and the links between the aggregation and core switches are limited to 2 Gbps.

```
1 sudo mn --custom ~/mininet/custom/FatTree.py --topo FatTree --switch ovsk
    --controller=remote, ip=127.0.0.1 --ipbase=10.0.0.0/8 --link tc
```

In the above command, the data rate of each link is enforced by Linux Traffic Control (*tc*), which has a number of packet schedulers to shape traffic to a configured rate. Each emulated host has its own virtual Ethernet interface(s) (created and installed with ip link add/set). The topology implementation can be found in Appendix A-1-1. The controller IP is the localhost ( *127.0.0.1* ) and the hosts are defined in the *10.0.0.0/8* network.

## 4-3 Physical Test Bed: Pica8 physical switches

The physical configuration consisted of 2 servers (*Helium* and *Lithium*) and 2 OpenFlow switches Pica8-P3290 both running in Open vSwitch (OVS) Mode. The configuration is illustrated in Figure 4-7.

Open vSwicth was installed on *Helium* server which acts as a hypervisor to support VM configuration. The VM runs on a KVM instance and has the IP address *145.97.20.116* assigned which is visible to the outside Internet.

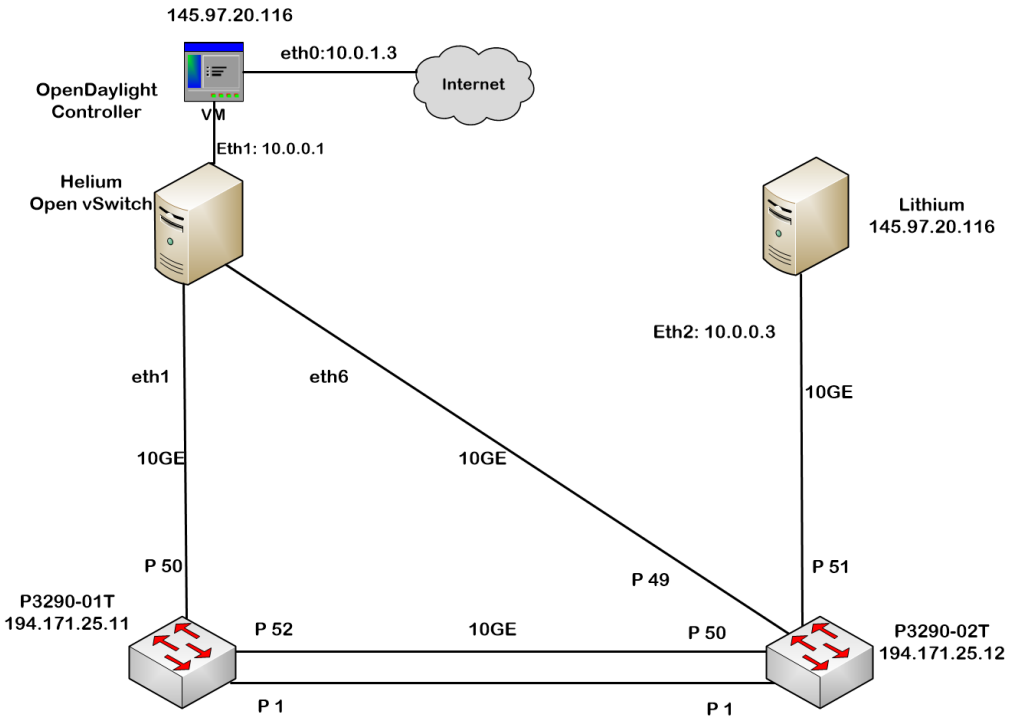
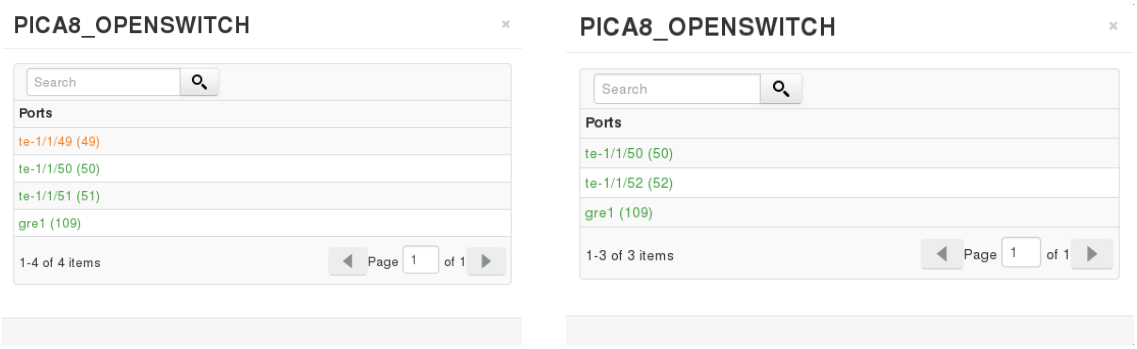



**Figure 4-7:** SURFnet Test Bed

In PicOS OVS operating system the ports can be configured in trunk mode. Both P3290−01T and P3290 − 02T switches have the same Virtual LAN (VLAN) assigned.

```
1 ovs-vsctl add-port br0 ge-1/1/50 vlan_mode=trunk tag=1 -- set Interface
    ge-1/1/50 type=pica8
2 ovs-vsctl add-port br0 ge-1/1/52 vlan_mode=trunk tag=1 -- set Interface
    ge-1/1/52 type=pica8
3
```




(a) P3290-01 switch ports (b) P3290-02 switch ports

**Figure 4-8:** Switch ports

```
4 ovs-vsctl add-port br0 ge-1/1/50 vlan_mode=trunk tag=1 -- set Interface
    ge-1/1/50 type=pica8
5 ovs-vsctl add-port br0 ge-1/1/51 vlan_mode=trunk tag=1 -- set Interface
    ge-1/1/51 type=pica8
6 ovs-vsctl add-port br0 ge-1/1/49 vlan_mode=trunk tag=1 -- set Interface
    ge-1/1/49 type=pica8
```

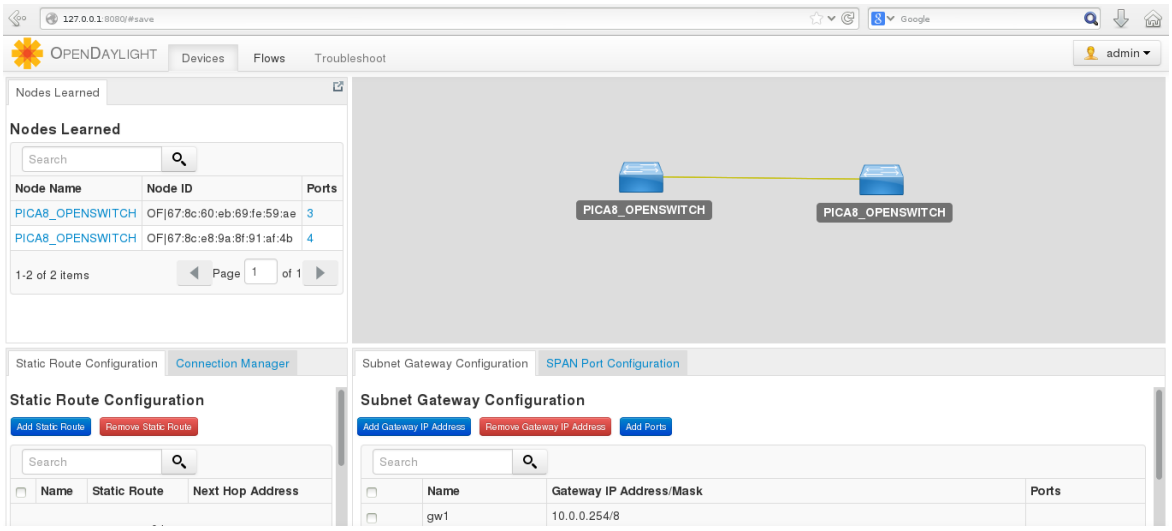



**Figure 4-9:** PICA 8 switches connected to OpenDaylight

The configured Flows can be seen in the "Flows" Section where flow characteristics are also visible (see Figure 4-9).

### 4-3-0-2 Generic Routing Encapsulation (GRE) Tunnel Configuration

To create GRE tunneling, two flows had to be configured: one that sends traffic to GRE and the other one that sends traffic out of the tunnel. The configuration example has been presented for P3290 – 01T switch. The Media Access Control Address (MAC) address of

the traffic which is sent throughout the tunnel should also be configured. For instance, ”60:eb:69:fe:59:ae” is the MAC address of switch P3290-01T, whereas the second switch, P3290-02T, has a MAC address equal to ”e8:9a:8f:91:af:4c”.

```
ovs-vsctl add-port br0 gre1 -- set Interface gre1 type=pica8_gre options:
    remote_ip=194.171.25.12 options:local_ip=194.171.25.11 options:vlan=1
    options:src_mac=60:eb:69:fe:59:ae options:dst_mac= e8:9a:8f:91:af:4c
    options:egress_port=ge1/1/52
```

The GRE port number is 109 which is the port number of ”gre1”. All the traffic coming from port 52 will be sent through GRE tunnel whose port number is 109. The second command indicates that all the traffic coming from the GRE tunnel will be forwarded to port 50. The MAC address of the VM is ”fa:16:3e:30:e1:bd”.

```
ovs-ofctl add-flow br0 in_port=50,actions=output:109
ovs-ofctl add-flow br0 in_port=52,
actions=mod_dl_src: 60:eb:69:fe:59:ae,mod_dl_dst:fa:16:3e:30:e1:bd,output
    :50
```

Figure 4-10 captures the configuration of ports on bridge ”br0” for $P3290-02T$ switch. The “gre1” port configuration is illustrated in green frame. The field ”true” indicates that the switch is connected to the controller that has the VM IP address *145.97.20.116*. This can be observed under the red frame in Figure 4-10.

## 4-4 Software Defined Networking Controllers

The SDN technology brings the development of multiple open-source controllers. A list of these projects along with a brief description of each has been summarized in [66]. Among them, the most popular and recent controllers are OpenDaylight and Floodlight. This Section presents a comparison of the two controllers in terms of their architecture, Graphical User Interface (GUI) and their existing modules which were used in the two test beds. Moreover, an extension of these modules and the algorithms behind them are throughly explained in the following Subsections.

### 4-4-1 OpenDaylight Controller

The OpenDaylight Project is a recent open-source project founded by some of the big vendors such as: Big Switch Networks, Brocade, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Red Hat and VMware. OpenDaylight is developed as a modular, pluggable, and flexible controller platform. This controller is completely programmed and it is integrated within its own Java Virtual Machine (JVM). Hereby, it can be deployed on any hardware and operating system platform that has Java environment installed. From a high level approach, SDN is commonly described in layers:

1. *Network Apps & Orchestration:* The top layer consists of business and network logic applications that primarily control and monitor network behavior. In addition, it contains other complex orchestration applications needed for cloud and NFV services in accordance with the requirements of those environments.

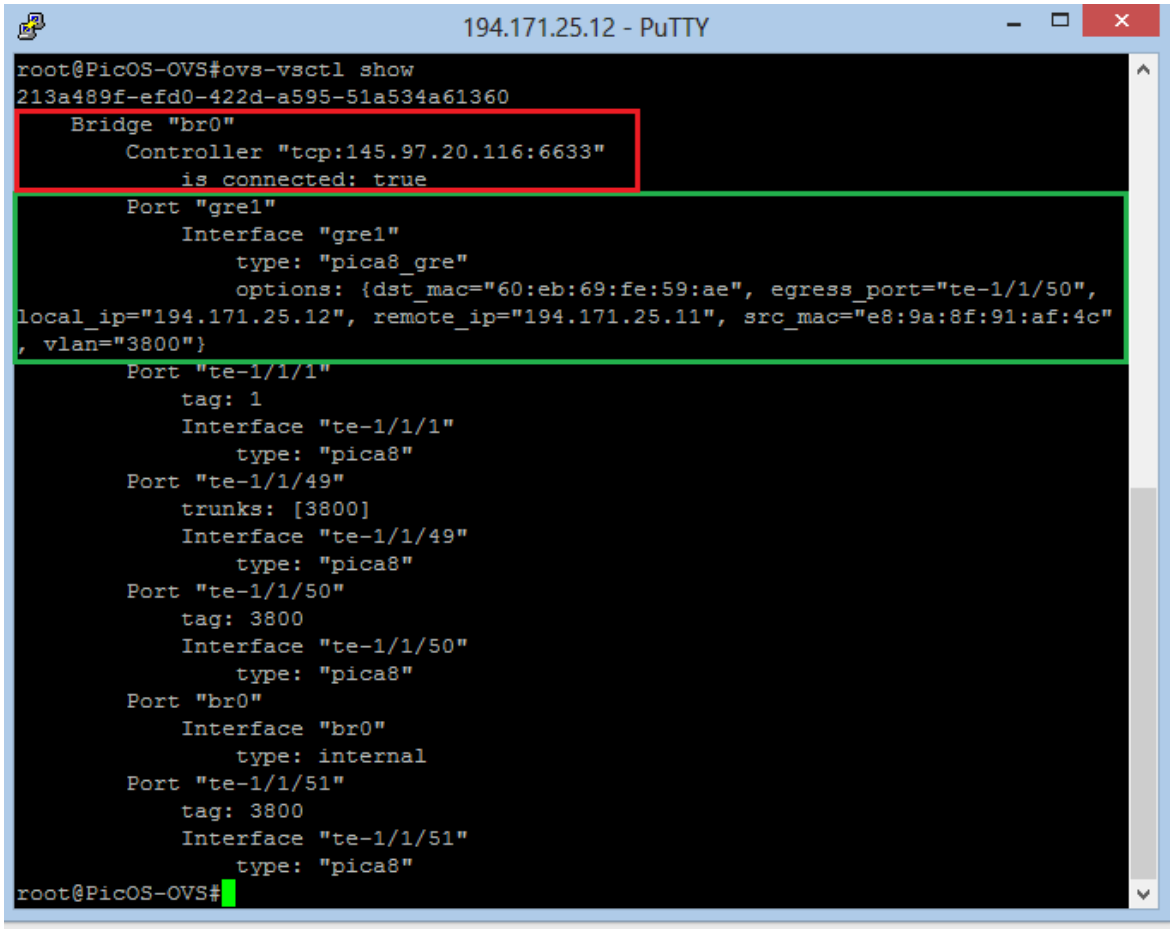



**Figure 4-10:** P3290 – 02T Port Configuration

2. *Controller Platform:* The middle layer provides the framework in which the SDN abstractions can run and consists of a set of common APIs to the application layer (i.e. the Northbound interface). Moreover, other protocols for command and control of the physical hardware such as OpenFlow, Netconf, Open vSwitch Database Management Protocol (OVSDB), Locator / ID Separation Protocol (LISP), Border Gateway Protocol (BGP), etc. are implemented within the network (on the Southbound interface).

3. *Physical & Virtual Network Devices:* The lowest layer consists of forwarding elements, which are basically physical and virtual devices, switches, routers, etc., that connect all endpoints within the network by forwarding the messages.

The top layer applications communicate with the controller via the Open API interface. OpenDaylight supports the Open Service Gateway initiative (OSGi) framework and bidirectional Representational State Transfer (REST) for the Northbound API. The OSGi framework is used for applications that are in the same network pool with the controller, while the REST (web based) API is used for applications that do not run in the same address space (they can even run on different VMs than the controller). The business logic and algorithms are part of the application layer. The applications use the controller to manage and monitor the network, run algorithms to perform analytics, and implement new rules in the network.

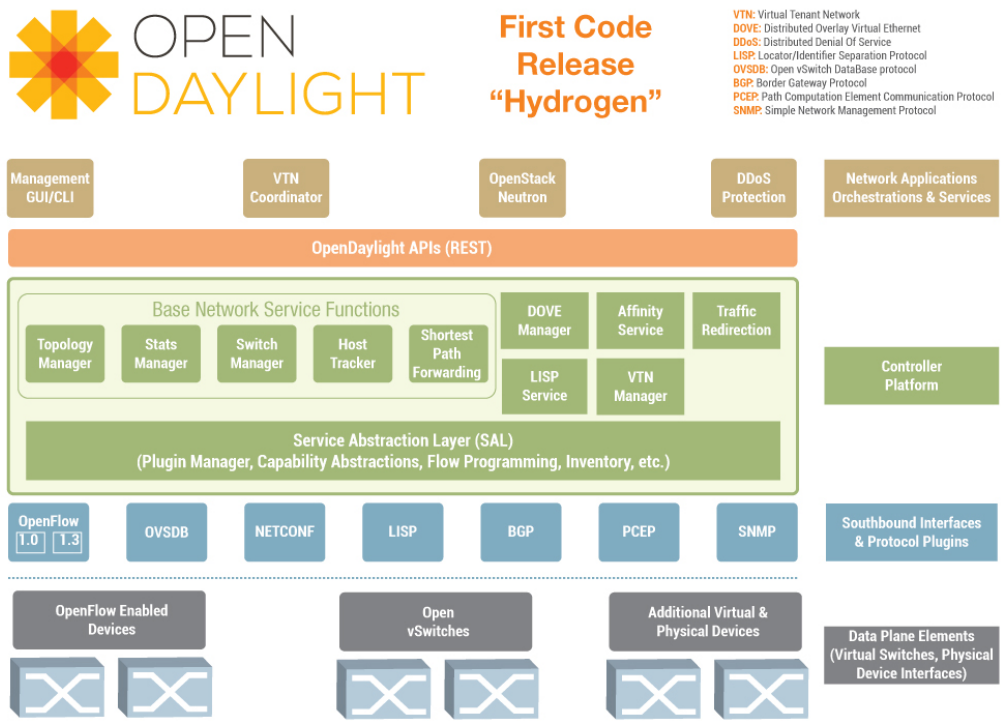



**Figure 4-11:** OpenDaylight controller, source [14]

The Southbound interface is capable of supporting multiple protocols as separate plugins, i.e. OpenFlow 1.0, OpenFlow 1.3, BGP, etc. These modules are connected to a Service Abstraction Layer (SAL) which is the core of the modular design of the Controller and it allows to support multiple protocols on the Southbound interface. The SAL determines how to map the requested service independent of the underlying protocol used between the controller and the network devices.

**4-4-1-0-1 OpenDaylight Web Interface** The OpenDaylight Controller includes an application called *Simple Forwarding* which allows to use the basic services for making forwarding decisions and install flows across all devices on the OpenFlow network. This application discovers the presence of a host via the Address Resolution Protocol (ARP) message and installs destination-only / 32 entries across all switches in the network, with the corresponding output ports towards the host.

This can be seen when logging into the web interface with OpenDaylight Controller and with Mininet running. A Fat-Tree topology of 20 switches and 16 hosts is illustrated in Figure 4-12. The port details along with the statistics can be seen in the "Flows" window. One of OpenDaylight main advantages in comparison to other existing controllers is that it provides the capability to install flow entries (L1-L4) in the switches using the Web Interface.

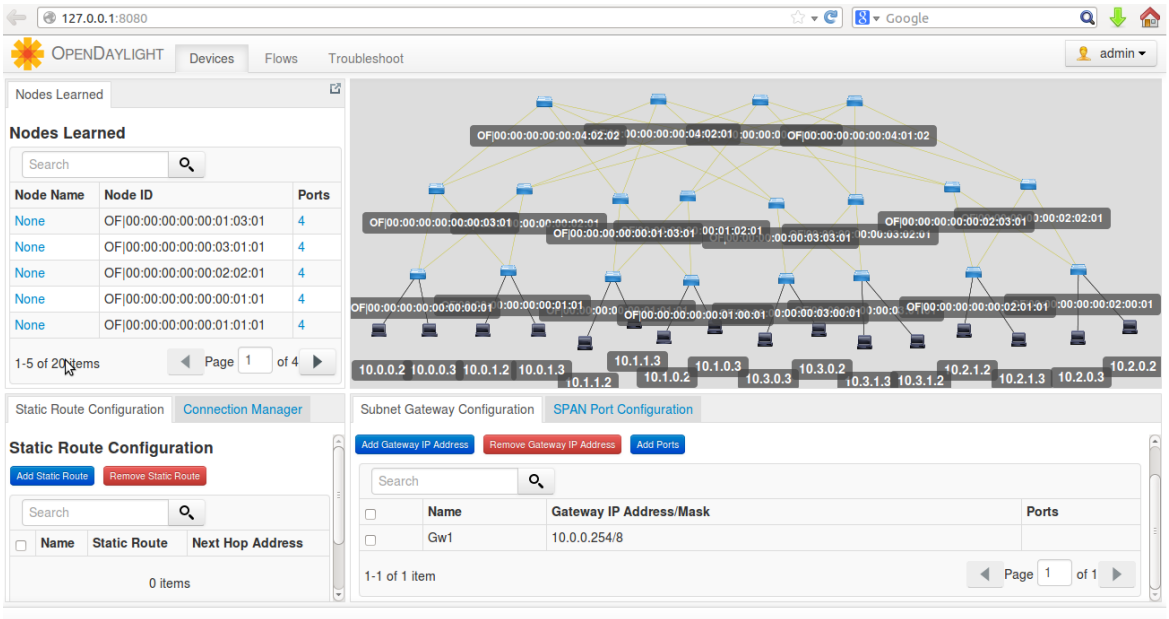



**Figure 4-12:** OpenDaylight controller GUI

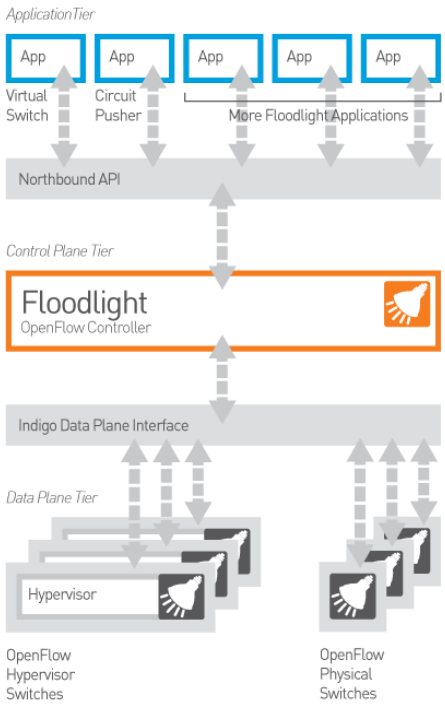



**Figure 4-13:** Floodlight Architecture, [67]

## 4-4-2 Floodlight Controller

Floodlight SDN Controller is another open-source SDN project supported by Big Switch Networks and the SDN community. Other companies and organizations that have contributed to the Floodlight controller APIs are: Arista, Brocade, Citrix, Dell, Extreme Networks,

Fujitsu, Google, HP, IBM, Intel, Juniper Networks and Microsoft, among others. Floodlight is an Apache-licensed OpenFlow controller, more mature than OpenDaylight, also developed in Java. It supports a broad range of hypervisor-based virtual switches like Open vSwitch, as well as physical OpenFlow switches which provide connection between OpenFlow and non-OpenFlow networks [67].

Figure 4-13 shows the relationships inside the Floodlight Controller, Java applications modules and the applications built over the Floodlight. Similar to the OpenDaylight architecture, these applications communicate with the controller through REST API. In Figure 4-14 the modules illustrated in blue color represent the existing modules in the Floodlight current version. The green modules are already contained in the QoS application and the yellow boxes represent the extra modules developed in this thesis and added to the current implementation. The load balancing module has been also added as a separate module application.

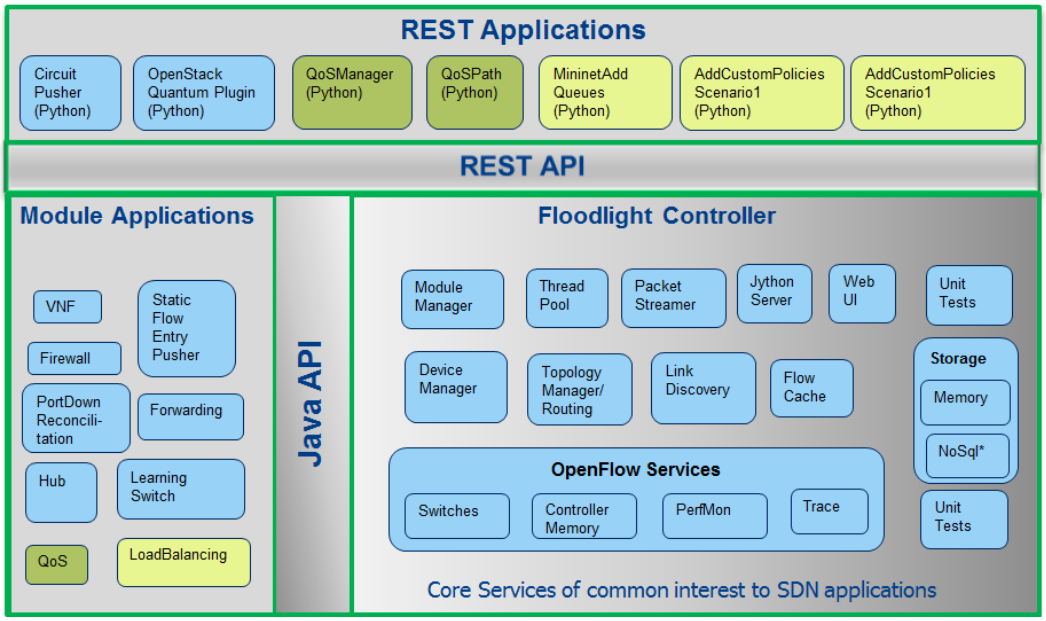



**Figure 4-14:** Qos and Load Balancing Modules in Floodlight Architecture

**4-4-2-0-2 Floodlight Web Interface** The topology mentioned in Figure 4-6 was implemented in Mininet and it was connected to the Floodlight controller on localhost (127.0.0.1). Floodlight also allows visualizing the topology in a GUI Web Interface (Figure 4-15). Apart from the "Topology" Section, the "Tools" Section display the output of the "QoS" application (4-15). In contrast to OpenDaylight, the current Floodlight GUI presents no possibility for configuring flow entries. This can be done either in the application layer or on the OpenFlow switches.

### 4-4-2-1 QoS Module

The QoS Module allows "QoS" state to be pushed into the network through REST API. This module allows configuration of maximum / minimum rate limit and DiffServ Type of Service (ToS) module, defines a specific QoS rule and installs a policy path.

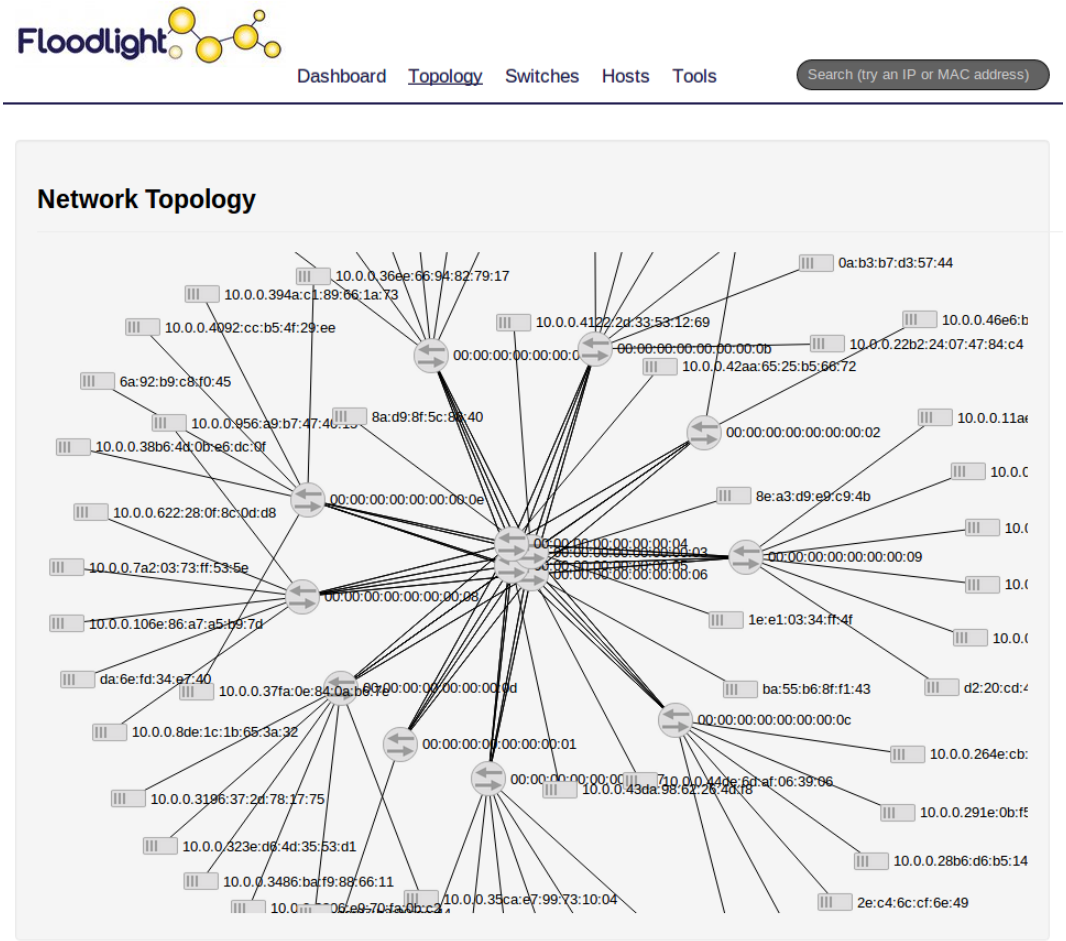



**Figure 4-15:** Topology in Floodlight

The QoS Module consists of three main application scripts: “CircuitPusher.py”,“QosManager.py” and “QoSPath”. These scripts have been modified from the original version found at [68] and apart from these, other scripts have been configured (i.e. ”Mininet_Add_Queues.py ,“Add_Custom_Policies_Scenario1.py”, “Add_Custom_Policies_Scenario2.py”, etc.) (Figure 4-14).

Although, the OpenFlow 1.0 specification and OpenFlow 1.3 specification support to push QoS states to the switches, the current application uses OpenFlow 1.0 specification. The OpenFlow 1.0 version specifies a way how to set the network ToS on a flow and enqueue the packets matching the flow to a specific queue on a specific port. The OpenFlow version 1.3 adoption should allow an extension of the QoS module and other features such as “set-queue” and meter tables.

The QoS module is implemented in Java and comes with a set of Python scripts that allow different configurations. The policies are added on the path based on the ToS byte corresponding to each session.

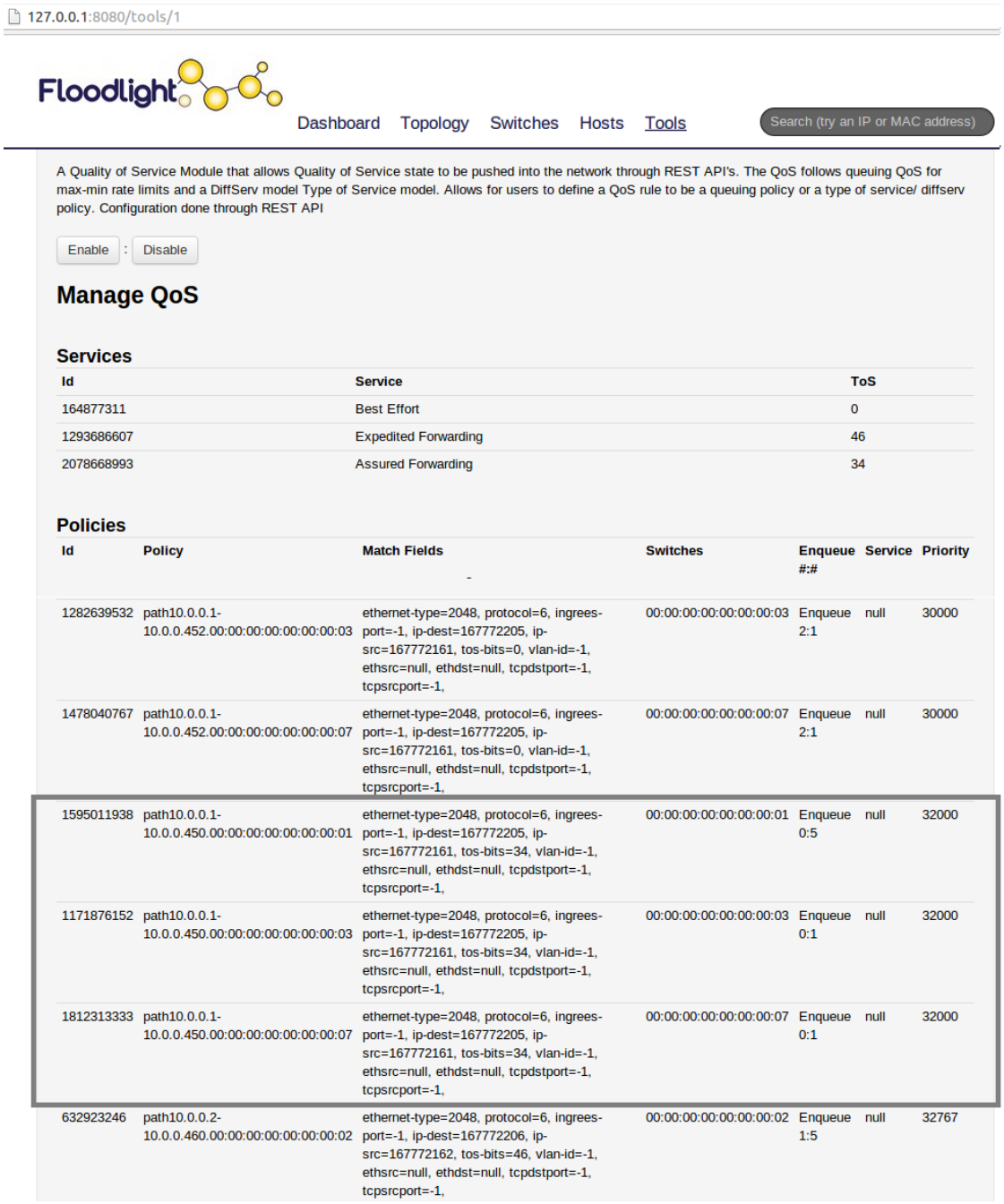



**Figure 4-16:** QoS module

**4-4-2-1-1 Class of Services and Queuing** The "QoS" application consists of the three *Classes of Services* mentioned in Chapter 3 (Best Effort (BE), Assured Forwarding (AF) and Expedited Forwarding (EF)) along with the corresponding ToS values. In the "Policies" section, the policy path is composed of the switches calculated by the shortest path. For instance, in Figure 4-16 a policy path has been established between $H_1$ and $H_{45}$ which traverses switches $S_1$, $S_3$ and $S_7$. The video session has a ToS = 34 assigned and priority "32000".

This script configures every switch port on every bridge with three QoS queues (a bridge has multiple ports) (see Appendix A-1-2). These queues are assigned as follows: $Q_0$ - 20 Mbps, $Q_1$ - 8 Mbps and $Q_2$ - 6 Mbps. The output of this algorithm consists of choosing a path between the client and server and applies a predefined policy along that path. The QoS routing policy returns the shortest path between source and destination.

Open vSwitch uses the Linux "traffic control" capability for rate-limiting. In the configuration presented below, a maximum rate of 1 Gbps ($10^9$ bits per second) has been defined and three QoS queues called $Q_0$, $Q_1$ and $Q_2$. The first queue, $Q_0$ has assigned the maximum bandwidth, whereas $Q_1$ has a maximum and minimum bandwidth rate of 20 Mbps ($2\times10^7$ bits per second) and $Q_2$ 2Mbps ($2 \times 10^6$ bits per second) as configured below:

```
queuecmd = "sudo ovs-vsctl \%s -- --id=@defaultqos create qos type=linux-
    htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=
    @q0 create queue other-config:min-rate=20000000 other-config:max-rate
    =20000000 -- --id=@q1 create queue other-config:max-rate=8000000 other
    -config:min-rate=8000000 -- --id=@q2 create queue other-config:max-
    rate= 6000000 other-config:min-rate=6000000"
```

A voice session is characterized by low-latency and a strict priority treatment with "Expedited Forwarding" service. We impose a traffic limit of 20 Mbps ($Q_1$) for video sessions, whereas Voice over IP (VoIP) sessions will be shaped to 8 Mpbs and Hypertext Transfer Protocol (HTTP) will receive 6 Mbps.

For instance, in Scenario 1 presented in Chapter 5, Section 5-2-1, the voice sessions are considered coming simultaneously from the users attached to the eNBs: $H_2$, $H_7$, $H_{12}$, $H_{17}$, $H_{22}$, $H_{27}$, $H_{32}$ and $H_{37}$. This would be mapped to $Q_1$ which limits the bandwidth to 8 Mbps. The video session is "better than BE" service and can be categorized as EF class. Each user would receive the maximum of bandwidth divided by the number of users (20 Mbps). The BE data service is limited to $Q_2$ which shapes the bandwidth to 6 Mbps per session. All the users attached to the rest of edge switches with data services have together a limited bandwidth of 6 Mbps. A policy path has been defined between each client and core server. If host $H_1$ has two policies defined with the same bandwidth limitations for both server $H_{45}$ and $H_{46}$ attached to switch $S_1$ and $S_2$, then by applying load balancing, the traffic is balanced between the two servers (see Scenario 2 in Chapter 5, Section 5-2-2).

After assigning three queues for each port, then the policies can be installed on the path. An example of policy rules has been presented in the following configuration:

```
./qospath.py add --qos-path 1-45 10.0.0.1 10.0.0.45 '{"eth-type":"0
    x0800","protocol":"17","tos":"34","queue":"0","priority":"32000"}'
    127.0.0.1 8080
```

The above command example establishes a path between $H_1$ and $H_{45}$ with the following flow characteristics: EtherType: '0x0800', protocol type: User Datagram Protocol (UDP), Differentiated Services Code Point (DSCP) '34', which corresponds to a video session, bandwidth limit of 20 Mbps and a priority of 32000.

The mechanism of installing policies on the path has been recursively implemented in Appendix A-2-1 and A-2-2-1 for the two proposed scenarios.

#### 4-4-2-2 Routing

The QoS application module consists of several scripts and modules which are interdependent. For example, the "QosPath.py" script recalls the "CircuitPusher.py" which will interrogate "Topology Manager / Routing" module in Floodlight. This further recalls the "TopologyInstance.java" module to apply Tarjan algorithm and calculate the shortest path. Figure 4-17 illustrates the interdependencies between scripts and modules.


**Figure 4-17:** Modules Interdependencies

The following command in "CircuitPusher.py" retrieves the route from source to destination using the routing REST API to communicate with the Floodlight "Topology Manager / Routing" module:

```
1  command = "curl -s http://%s/wm/topology/route/%s/%s/%s/%s/json" % (
       controllerRestIp, sourceSwitch, sourcePort, destSwitch, destPort)
```

The routing algorithm from "TopologyInstance.java" uses the Tarjan's algorithm to form clusters which is based on Depth First Search (DFS). The Tarjan algorithm is a procedure for finding strongly connected components of a directed graph. A strongly connected component is a maximum set of nodes, in which exists at least one oriented path between every two nodes. The entire Floodlight routing mechanism can be divided in several steps:

1. Compute clusters ignoring broadcast domain links. Every cluster is a strongly connected component. The network may contain unidirectional links. The computation of strongly connected components is based on Tarjan algorithm.
2. Compute shortest path trees in each cluster for unicast routing. The trees are rooted at the destination. Cost for tunnel links and direct links are the same. The shortest path algorithm is computed using Dijkstra in each cluster.
3. Compute broadcast tree in each cluster. Cost for tunnel links are high to discard use of tunnel links. The cost is set to the number of nodes in the cluster + 1, to use as minimum number of clusters as possible.
4. The path is returned and the topology is displayed.

Since the routing algorithm is topology dependent, there are no isolated clusters in a two-level mesh topology presented in Figure 4-6. As a result, only steps 1) ,2) and 4) are performed. The proposed topology consists of a single cluster, hence step 3) does not apply to the proposed topology.

**4-4-2-2-1 Depth First Search (DFS):** For the presented topology in Figure 4-6, the Tarjan algorithm will be applied once for one cluster and it will return the list of nodes in a tree walk through.

```
input  : graph G = (V, E)
output: set of strongly connected components (sets of nodes)
index := 0;
S := empty;
foreach v in V do
    if v.index is undefined then
        strongconnect (v);
    end
end
function strongconnect(v):
    /* Set the depth index for v to the smallest unused index                    */
    v.index := index;
    v.lowlink := index;
    index := index + 1;
    S.push(v);
    /* Consider successors of v                                                  */
    foreach (v,w) in E do
        if w.index is undefined then
            /* Successor w has not yet been visited; recurse on it               */
            strongconnect (w);
            v.lowlink := min(v.lowlink, w.lowlink);
        else
            if w is in S then
                /* Successor w is in the stack S and hence in the current SCC    */
                v.lowlink := min(v.lowlink,w.index);
            end
        end
    end
    /* If v is a root node, pop the stack and generate an SCC                    */
    if v.lowlink = v.index then
        /* start a new strongly connected component                              */
        repeat
            w := S.pop();
            /* add w to current strongly connected component                     */
        until w=v;
        /* output the current strongly connected component                       */
    end
```

**Algorithm 1:** Tarjan algorithm, source [69]

The Tarjan's algorithm is based on DFS. The DFS algorithm is a search method for finding the strongest connected components of a graph. The algorithm takes as input a directed graph and the output is to group the graph's strongly connected vertices into clusters. All nodes of the graph are part of one of these clusters and if a node has degree 0, it can also form itself a strongly connected component.

The depth-first search begins from an arbitrary node and it is only applied to all the nodes that have not been defined yet, which means that the search visits every node exactly once. The first discovered component discovered is called the root. Any node of a strongly connected component might serve as the root, if it happens to be the first node of the component that is discovered by the search. An index value is attributed to each sets of undefined nodes $v$ in the set of vertices $V$. The index value is defined as a counter for the depth-first search node number. $S$ is defined as the node stack, which stores the history of discovered nodes before being linked to a strongly connected component.

The *lowlink* represents the smallest index of any node known to be reachable from $v$. It is calculated as the minimum between its successor *lowlink* index and its own *lowlink* index. Otherwise, in the case the successor has been visited, the current node *lowlink index* will be calculated as the minimum between its current value and the successor's *index*. When a node's *lowlink index* becomes equal to its *index*, then the node is popped out from the stack. Following the same procedure, all the nodes will be extracted until the first root. This search will return a list of nodes from root to the last successor.

**4-4-2-2-2 Shortest Path (Dijkstra)** The Shortest Path is a routing mechanism based on Dijkstra algorithm. Dijkstra is a greedy algorithm which makes at each step in the computation the best choice at that moment with the target that the global optimum is reached at the end.

The current version of Dijkstra algorithm in *TopologyInstance.java* module implemented in the Floodlight Controller is computed based on equal weights (i.e. equal costs on the links). For an accurate approach, the algorithm has been modified and it was considered that the weight on the links correspond to random loads on each aggregation switch. Therefore, each weight has been randomly assigned a positive value.

The initialization consists of assigning to all links an infinite value for all nodes and all predecessors. At this point all the nodes in the network are unreachable. In the next step, all nodes are inserted in the queue $Q$. Furthermore, while there are still nodes in the queue, the node $u$ with the shortest weight is extracted from the queue. The $u :=$ vertex in Q with min $dist[u]$ searches for the vertex $u$ in the vertex set $Q$ that has the least $dist[u]$ value. In the last part of the algorithm, the relaxation concept causes the shortest path $dist[v]$ to descend monotonously towards the actual shortest-path weight. The Dijkstra algorithm relaxes each link no more than once. For each node $v$ adjacent to the discovering node $u$, it is tested whether the shortest weight can be decreased going further to the discovering node. In this manner, the path along node $u$ is the shortest found and the predecessor of $v$ is updated to $u$ [71].

```
function Dijkstra(Graph, source):
    dist[source] :=0 ;                                          /* Distance from the source to source */
    /* Initializations                                                                                  */
    foreach vertex v in Graph do
        if v != source then
            dist[v] := infinity ;                               /* Unknown distance function from source to v */
            previous[v] := undefined ;                          /* Previous node in optimal path from source */
        end
        add v to Q ;                                            /* All nodes initially in Q */
    end
    /* The main loop                                                                                    */
    while Q is not empty do
        u := vertex in Q with min dist[u] ;                     /* Source node in first case */
        remove u from Q;
        foreach neighbor v of u do /* where v has not yet been removed from Q                           */
            alt := dist[u] + length(u,v);
            if alt < dist[v] then /* A shorter path to v has been found                                  */
                dist[v] := alt;
                previous[v] := u;
            end
        end
    end
    return dist[],previous[]
```

**Algorithm 2:** Dijkstra algorithm, source [70]

#### 4-4-2-3 Load Balancing

**4-4-2-3-1 Round Robin** The load balancing mechanism distributes the clients requests across multiple servers. The round robin algorithm maintains a list of servers and forwards a new connection to the next server in the members list. The output of this algorithm is to balance traffic between the servers which share the same policy with the clients.

To use this service, a Virtual IP (VIP) should be exposed to the clients of this service and used as the destination address. A VIP is an entity that consists of a virtual IP, port and protocol (TCP or UDP). The idea of Virtual Server is also described in paper [72], which includes the application port and the IP address and its main function is to provide a bi-directional NAT to the outside world.

Nevertheless, several assumptions have been made in the deployed scenario:

1. One or more VIPs can be mapped to the same server pool. All VIPs that share the same pool, must also share the same load balancing policy (random or round robin).
2. Each server pool corresponds to only one VIP.
3. All flow rules are installed with an idle timeout of 5 seconds.
4. Packets to a VIP must leave the OpenFlow cluster from the same switch from where it entered.
5. When VIP or a server pool or a server from a pool has been deleted, the service does not delete the flow rules it has already installed. The flow rules should automatically be timed out after the idle timeout of 5 seconds.
6. Since VIP does not exist in the network, Opendaylight controller is not able to resolve ARP for IP of the VIP. Load balancer application assumes that if VIP is configured

and exposed by user to internal / external network, packets that are destined to the VIP's IP address will be routed to it throughout external mechanisms. The ARP can be locally resolved for the VIP's, adding static entries to the ARP table of the client/source host. In contrast to that, the Floodlight is able to automatically resolve ARP.

The VIP pool configuration for Floodlight controller can be found in Appendix A-3-1, whereas for OpenDaylight in Appendix A-3-2.

### 4-4-3 CBench

CBench (controller benchmarker) is a program for testing OpenFlow controllers by generating packet-in events for new flows. Cbench emulates a bunch of switches which connect to an SDN controller. Each emulated switch sends a number of new flow (OpeFlow PACKET_IN) messages to the OpenFlow controller, waits for the appropriate reply (OpenFlow PACKET_OUT messages) and records the difference between request and response [73].

The controller performances can be measured in terms of throughput, latency and minimum / maximum controller response time.

```
1 cbench -c localhost -p 6633 -m 10000 -l 10 -s 16 -M 1000 -t
```

CBench commands consists of a number of arguments:

**M** - number of MACs / hosts to emulate per switch
**s** - number of switches to emulate
**t** - throughput (vs. latency mode)

The above example emulates 16 switches with 1000 hosts per switch.

CBench tool allows to specify the number of emulated hosts per switch, the number of switches and the running mode. There are two configuration modes that can be chosen: *latency* and *throughput*.

The *Latency* mode measures the OpenFlow controller's request processing time under low-load conditions, whereas in *throughput* mode each switch sends as many request as the buffer supports.

# Chapter 5

# Testing and Evaluation

This chapter presents different scenarios configured using the testbeds described in Chapter 4 and interprets the obtained results. Additionally, a brief insight of the expected results is given in order to estimate the possible outcome. The chapter is divided in three main parts corresponding to the three different testbed deployments: a simulated network topology, a physical testbed remotely controlled and an Long-Term Evolution (LTE) simulator running on a virtualized environment.

## 5-1 Evolved Packet Core (EPC) Software Defined Networking (SDN)-performances and expectations

The EPC SDN-based architecture envisions the flexibility of decoupling the user from the control plane as it was discussed in Section 3-4. The deployment complexity of the proposed topology was presented in Section 4-2-1-1. Furthermore, the implementation of Quality of Service (QoS) module was described in Section 4-4-2-1 and the routing algorithms were presented in Section 4-4-2-2. The QoS and load balancing modules are integrated in the Floodlight controller.

The performances of the network emulated testbed were analyzed based on the obtained results in terms of throughput, loss and packet delay variation (jitter) for different session types: Hypertext Transfer Protocol (HTTP), Voice over IP (VoIP) and video. It is expected to achieve a throughput close to the bandwidth limitation for each of the sessions according to the rate limit queues installed on the switch ports (see Section 4-4-2-1-1). For VoIP and interactive video (Internet Protocol (IP) Video conferencing), the requirements according to [51] are the following: an end-to-end latency of no more than 150 ms, an average jitter less than 30 ms and a loss smaller than 1 %. For streaming video, the loss must not exceed 5 % and latency should be no greater than 4-5 seconds (depending on the video application buffering capabilities).

The load balancing mechanism, which is based on Round-Robin, is expected to perform for both OpenDaylight (Section 4-11) and Floodlight (Section 4-15) controllers according to the

algorithms presented in Section 4-4-2-3. Therefore, the two controllers performances with the load balancing module installed, should be similar in terms of throughput, jitter and packet loss.

## 5-2 Experimental Simulated Set-up: Mininet

The emulated network set-up consists of five scenarios. All the scenarios follow the topology presented in Figure 4-6. The first two scenarios focus on the policy path installation: in the first scenario each host has one policy installed, whereas in the second scenario each host has multiple policies installed. In both scenarios the hosts send traffic to the servers simultaneously. The main goal of the first two scenarios is to simulate a mobile network of 40 Evolved Node B (eNB)s with multiple users connected. The User Equipment (UE)s devices send different sessions at the same time: video, VoIP and HTTP. The returned path is meant to give a SGW User Plane (SGW-U) selection and set a load balancing Round-Robin mechanism at the PDN Gateway User (PGW-U) switches.

In the third scenario, the Round-Robin mechanism is tested at the eNB-SDN level in order to avoid congestion in the eNB-SDN clusters and manage the handover between base stations. The servers are connected in a pool and serve the clients in a Round-Robin fashion. The load balancing mechanism was tested for both Floodlight and OpenDaylight controllers.

In the fourth scenario, the Round-Robin performances are compared with a random load balancing module implemented in OpenDaylight and Floodlight controllers. In the last scenario, the two controllers performances were analyzed. A benchmark tool was used to test the processing limit and over-stress the two controllers.

### 5-2-1 Scenario 1

In the first scenario all five hosts $H_1$-$H_5$ corresponding to the eNBs attached to switch $S_7$ are sending traffic to the server $H_{45}$ (Internet) through $S_1$ (PGW-U). Several policies have been installed between the clients and servers. The configuration script of this scenario can be found in Appendix A-2-1.

For instance, $H_1$ sends video traffic with Differentiated Services Code Point (DSCP) 34, which corresponds to Type of Service (ToS) equal to 136, in class Assured Forwarding (AF) 41 and has a bandwidth limitation of 20 Mbps. When transmitting an interactive video session (i.e. IP Videoconferencing), the traffic should be marked to DSCP AF 41, whereas excess interactive video traffic can be mapped to AF 42 or AF 43.

Due to its strict service-level requirements, VoIP is well suited to the Expedited Forwarding (EF) Per-Hop Behavior (PHB), as defined in Section 3-1-1-1-3 in Chapter 3. Therefore, it should be marked to EF class with DSCP equal to 46 and assigned a strict priority servicing at each node.

Table 5-1 below summarizes all the policies installed between clients $H_1$-$H_5$ and server $H_{45}$. The highest flow priority was attributed to voice session and the lowest to HTTP connection.

| Client | Server | Protocol | Session | DSCP | ToS | Service | Queue | Bandwidth | Priority |
|---|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | UDP | 1 Video | 34 | 136 | Assured Forwarding | 0 | 20 Mbps | 32000 |
| $H_2$ | $H_{45}$ | UDP | 1 Voice | 46 | 184 | Expedited Forwarding | 1 | 8 Mpbs | 32767 |
| $H_3$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_4$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_5$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |

**Table 5-1:** Scenario 1

### 5-2-1-1 Throughput

As it was mentioned above, all clients send traffic simultaneously to the server with a the bandwidth limitation set up according to the installed policy. Figure 5-1 shows that video session ($H_1$) receives 19 Mbps, voice session ($H_2$) receives 7.5 Mbps and each HTTP session receives almost 2 Mbps (all three sessions together 6 Mbps). For the 3 HTTP connections, the throughput decreases due to the simultaneous requests sent to the same server, sharing the bandwidth of 6 Mbps.

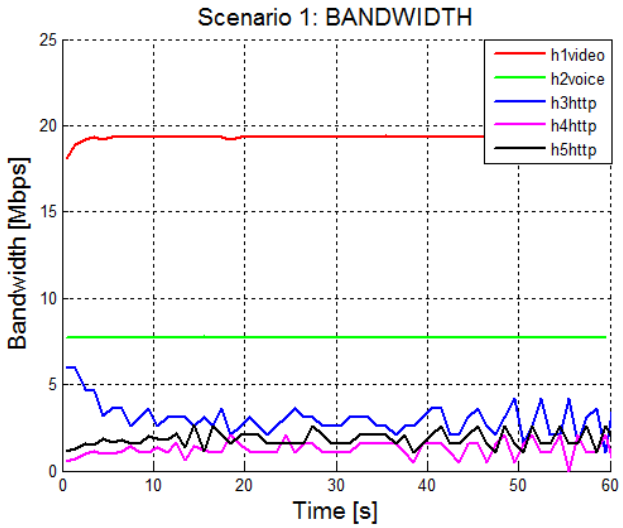


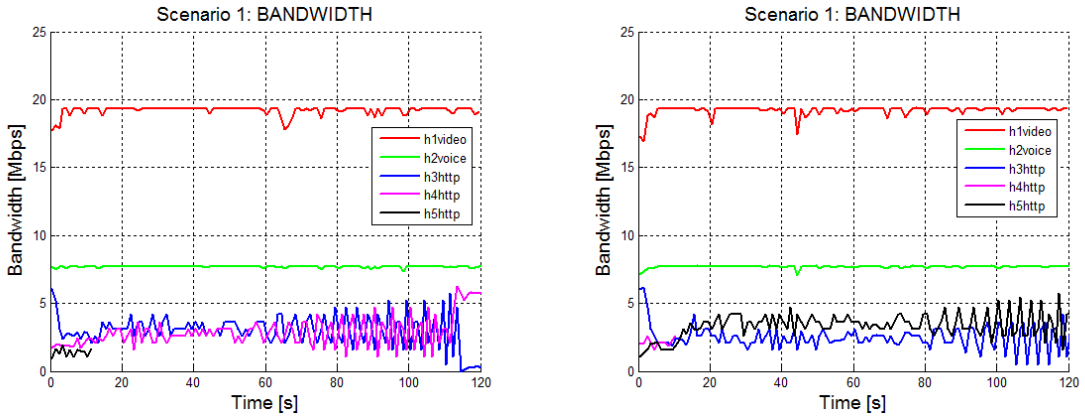

**Figure 5-1:** Scenario 1: Throughput

In Figure 5-1, the throughput for all simultaneous connections is illustrated for a time interval of 60 s (i.e. the test interval lasts 60 s). This has been tested using *Iperf* tool for both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) traffic, whereas the results were processed in Matlab tool (see Appendix A-4-1). The following commands show the *Iperf* configurations:

```
1 H1: iperf -c 10.0.0.45 -S 136 -i 1 -t 120 -u -b 20m -w 256k
2 H2: iperf -c 10.0.0.45 -S 184 -i 1 -t 120 -u -b 8m -w 256k
3 H3, H4, H5: iperf -c 10.0.0.45 -S 0 -i 1 -t 120 -w 128k
```

Where $S$ represents ToS value, *1-t* 120 gives the connection test duration of 120 s and $W$ equal to 256 K represents the window size.

For TCP, $W$ sets the TCP window size, whereas for UDP, it represents the buffer the datagrams are received in, thus it limits the largest receivable datagram size. The simulation results are illustrated in Figure 5-2-a).




(a) Throughput window size 256 K, without fragmentation

(b) Throughput window size 1024 K, after fragmentation

**Figure 5-2:** Scenario 1: Throughput

The following commands show an increase in the window size and a fragmentation of the packets sent:

```
1 H1: iperf -c 10.0.0.45 -S 136 -i 1 -t 120 -u -b 20m -w 1024k -l 250
2 H2: iperf -c 10.0.0.45 -S 184 -i 1 -t 120 -u -b 8m -w 1024k -l 250
3 H3,H4,H5: iperf -c 10.0.0.45 -S 0 -i 1 -t 120 -b 6m -w 256k -l 250
```

Where $l$ equal to 250 bytes represents the buffer length to read or write. The *Iperf* tool works by writing an array of *len* bytes a number of times. Default is 8 $KB$ for TCP and 1470 bytes for UDP. For UDP, this is the datagram size and it needs to be lowered in order to avoid fragmentation.

Figure 5-2 shows a comparison of the throughput obtained for a window size of $W = 256$K, before fragmentation and $W = 1024$ K after fragmentation. The packet size was lowered to 250 bytes. Due to the fact that the bandwidth is shared between the three HTTP connections, one of the data streams is blocked after 10 seconds. The 6 Mbps bandwidth is only divided between two data connections.

#### 5-2-1-2 Datagram Loss

Figure 5-3 illustrates that the datagram loss is maintained below 5% for both video and voice during a test interval of 60 s. The datagram is the IP Packet (Application layer) corresponding to the Transport layer. However, when the datagram or packet is processed by the Network layer, it adds the IP Header to the data and it becomes an IP packet.

Figure 5-4 represent the datagram loss before and after fragmentation for a test that lasts for 120 s. In Figure 5-4-a) the loss is maintained below 10 % for both video and voice sessions, whereas in Figure 5-4-b), the video session loss is higher than before re-fragmentation took place, around 14% with frequent peak variation up to 40%. In the case of voice session, the average loss value after fragmentation remains below 5% with one steep peak variation. The packet loss is less significant for the VoIP than video which is in accordance to the

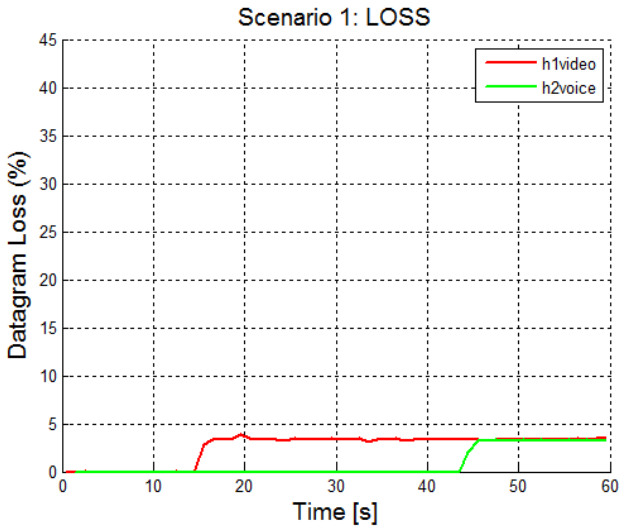
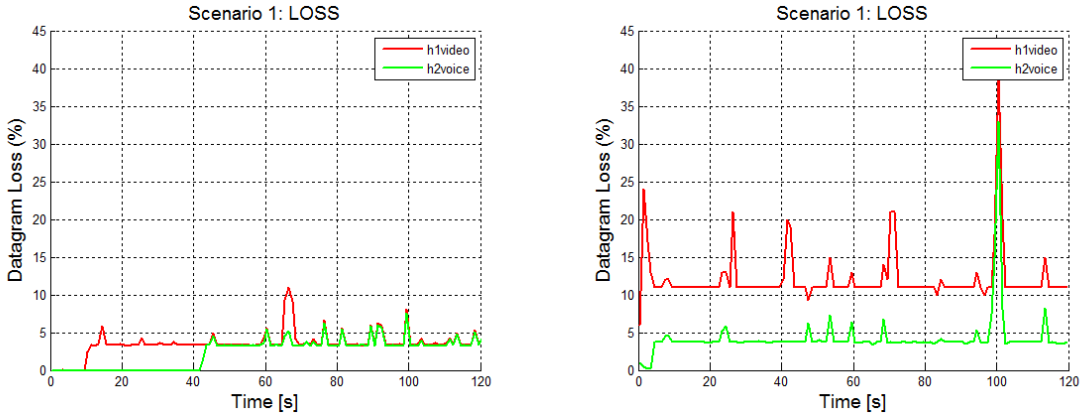



**Figure 5-3:** Scenario 1: Datagram Loss




(a) Loss window size 256 K, without fragmentattion (b) Loss window size 1024 K, after fragmentation

**Figure 5-4:** Scenario 1: Datagram Loss

recommended values presented in Table 2-1, Chapter 2. The VoIP networks are typically designed for very small packet loss percentage, with the only actual packet loss caused by L2 bit errors or network failures.

Severe packet loss also occurs due to TCP synchronization. When TCP connections share a bottleneck node, the evolutions of their congestion windows are likely synchronized. This can be explained as follows: the TCP synchronization can fill in the node buffer for a long duration repeatedly. Even if the node is not completely full of packets, the UDP datagrams are still transmitted constantly and must be dropped successively. Therefore, the UDP stream suffers harmful effects caused by the TCP synchronization.

The packet-loss rate is not reduced despite of the lower UDP transmission rates (Figure 5-4-b)). The TCP connections are capable of sharing all the available bandwidth among them by using their flow control mechanism. Therefore, even if the UDP stream reduces

its transmission rate, the resulting available bandwidth will be rapidly consumed by TCP connections. This is why reducing transmission rate of UDP packets will not contribute to the decrease of the UDP packet-loss.

#### 5-2-1-3 Packet Delay Variation

Jitter can be regarded as a variation in the delay of received packets and it refers to how variable latency is in a network. Jitter usually occurs because of network congestion, improper queuing or route changes. Nevertheless, a jitter buffer can be used to handle jitter. A more accurate definition jitter is given by the mean deviation (smoothed absolute value) of the difference $D$ in packet spacing at the receiver compared to the sender for a pair of packets. This is equivalent to the difference between packet's timestamp and the receiver's clock at the time of interval.

The obtained jitter was measured using the Iperf tool which calculates jitter based on the formula defined in RFC 3550 for the Real-time Transport Protocol (RTP) protocol [74].

$S_i$ represents the timestamp from packet $i$ and $R_i$ is the time of arrival in timestamp units for packet $i$. Therefore, the delay, $D$, for the two packets $i$ and $j$ can be expressed follows:

$$D(i,j) = (R_i - R_j) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i) \tag{5-1}$$

The inter-arrival jitter can be calculated for each data packet $i$ received using the difference $D$ (previously defined in Eq. 5-1) between the packet itself and the previous packet $i$-1 according to the following equation:

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16} \tag{5-2}$$

In Eq. 5-2, the parameter expressed as $\frac{1}{16}$ gives a good noise reduction ratio, while maintaining a reasonable rate of convergence.

The packet delay variation is maintained below 0.4 ms on average for both voice and HTTP connections. In Figure 5-5 the jitter for voice session varies between 0.15 ms and 0.55 ms, whereas for video session it reaches the maximum peak of 0.65 ms and a minimum value of 0.18 ms. In the case of packet delay variation, after the fragmentation, the jitter is lower for the video session. In Figure 5-6 both sessions have a jitter less than 1.7 ms, which is defined within the standard parameters (less than 30 ms).

Figure 5-6 shows that higher transmission rates (before packet fragmentation) cause the jitter to increase. The reason for this is the fact that the high transmission rates cause queues, producing irregularities in the packet inter-spacing. On the other hand, the small packet size leads to a decrease in the average jitter.

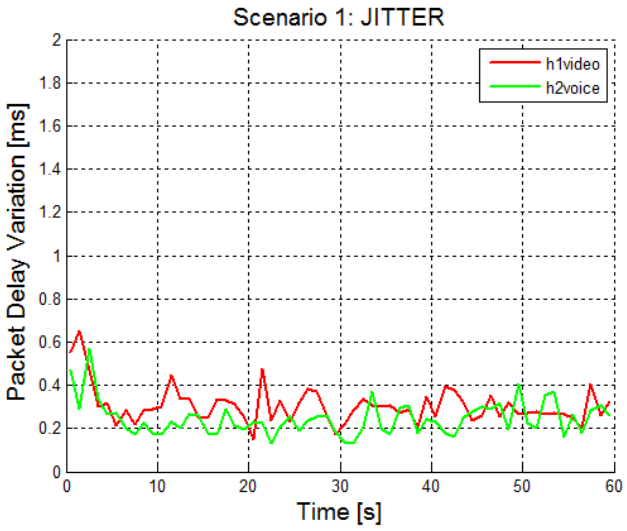
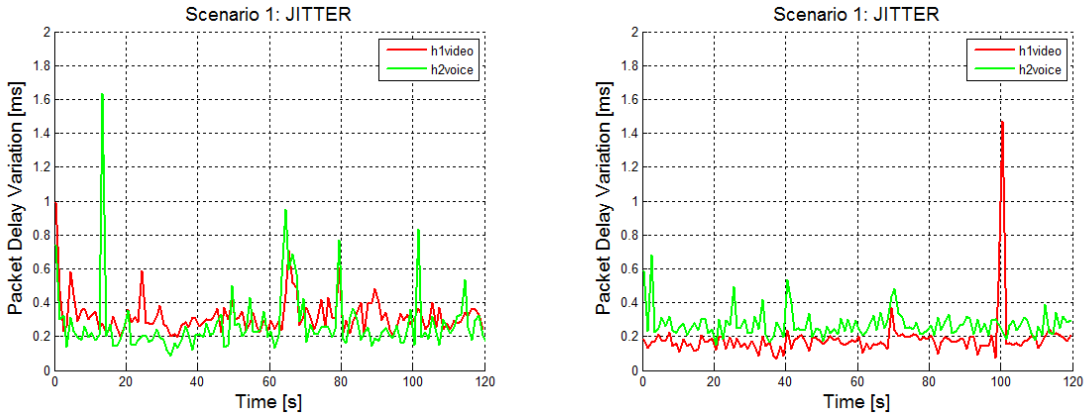



**Figure 5-5:** Scenario 1: Jitter




(a) Jitter window size 256 K, without fragmentation (b) Jitter window size 1024 K, after fragmenatation

**Figure 5-6:** Scenario1: Jitter

#### 5-2-1-4 Round-Trip Time

The minimum time between sending of a data segment and the arrival of its corresponding acknowledgment is exactly one round-trip time. Without the TCP's window mechanism, the TCP would only be able to send one segment per round-trip time, since it would have to wait for the acknowledgment before sending the next data segment.

The following Eq. 5-3 gives the relation between the peak throughput and the round-trip delay in case of TCP traffic ([71, 75]):

$$max(Throughput) = \frac{W \cdot S}{R} \tag{5-3}$$

In Eq. 5-4, $R$ is the round-trip delay, $W$ is the window size and $S$ is the maximum segment size.

Therefore, the round-trip time is:

$$R = \frac{W \cdot S}{max(Throughput)} \tag{5-4}$$

In Eq. 5-5, $D$ is the one-way delay to the receiver:

$$D = \frac{R}{2} \tag{5-5}$$

#### 5-2-1-5 Loss probability

In [71], a relation between the loss probability $p$ and windows size $W$ is established:

$$W = \sqrt{\frac{8}{3 \cdot p}} \tag{5-6}$$

The average throughput of a TCP connection in [packets/s] can be expressed as follows:

$$T = \frac{3 \cdot W \cdot S}{4 \cdot R} \tag{5-7}$$

Therefore, if the window size parameter, $W$ from Eq. 5-6 is introduced in Eq. 5-7, the average throughput becomes:

$$T = \frac{K \cdot S}{R\sqrt{p}} = \frac{\sqrt{\frac{3}{2}}S}{R\sqrt{p}} \tag{5-8}$$

where $S$ is the maximum segment size, and $K$ is a constant equal to $\sqrt{\frac{3}{2}}$, $R$ is the round-trip time and $p$ is the loss rate.

The average throughput does not depend on the maximum window size, $W$, as in the case of peak throughput (Eq. 5-3).

From Eq. 5-8, the loss probability $p$ can be calculated as follows:

$$p = \frac{\frac{3}{2}S^2}{R^2T^2} \tag{5-9}$$

The values for round-trip time and loss before and after fragmentation are presented in Table 5-2.

*Note*: The considered window size is $W$=128K before fragmentation and $W$=256K after fragmentation. Maximum segment size is $S$=8$KB$ before fragmentation and $S$=250 bytes after fragmentation.

| Hosts | Round-trip time before frag.[ms] | Loss before frag.[%] | Round-trip time after frag.[s] | Loss after frag.[%] |
|---|---|---|---|---|
| $H_3$ | 170.3827 | $0.3613 \times 10^{-9}$ | 2.6622 | $0.1445 \times 10^{-14}$ |
| $H_4$ | 489.9522 | $0.0437 \times 10^{-9}$ | 7.6555 | $0.0175 \times 10^{-14}$ |
| $H_5$ | 379.2593 | $0.0729 \times 10^{-9}$ | 5.9259 | $0.0292 \times 10^{-14}$ |

**Table 5-2:** Round-trip time and Loss before and after fragmentation

#### 5-2-1-6 Path Calculation

The Dijkstra algorithm presented in Section 4-4-2-2-2 in Chapter 4 returns the shortest path. Table 5-3 presents the chosen paths between the edge and core switches. In the EPC network, the algorithm performs the SGW-U selection.

#### 5-2-1-7 Analysis

The obtained values for throughput in Scenario 1 validate the imposed values for rate limit in the case of each session (i.e. 20 Mbps in the case of video, 8 Mbps for voice and 6 Mbps bandwidth shared between the 3 HTTP connections).

The TCP connections share all the available bandwidth among them through their flow control mechanism. Therefore, even if the UDP stream reduces its transmission rate or it is fragmented, the resulting available bandwidth will be rapidly consumed by TCP connections. Moreover, when the TCP connections share a bottleneck node, the evolutions of their congestion windows are likely synchronized. As a result, the UDP streams become strongly affected by TCP synchronization. This is the reason why the UDP packet-loss performance can not improved, even though the transmission rate of UDP packets is reduced after fragmentation. The UDP loss determines end-user experience in audio and video streaming applications, such as VoIP and video sessions. High loss percentage might also cause high jitter and delays in UDP traffic. The VoIP applications cannot be re-transmitted, otherwise the information would no longer be in updated in real-time. In contrast to RTP, UDP does not use sequence numbers or timestamps for resynchronization at receiver, neither multicasting. The UDP protocol does not guarantee delivery of the datagrams. The UDP traffic is not acknowledged. This means that the client can send traffic with any bandwidth without caring about how many packets will be lost. The difference between UDP and TCP is that TCP will re-transmit lost packets and UDP does not.

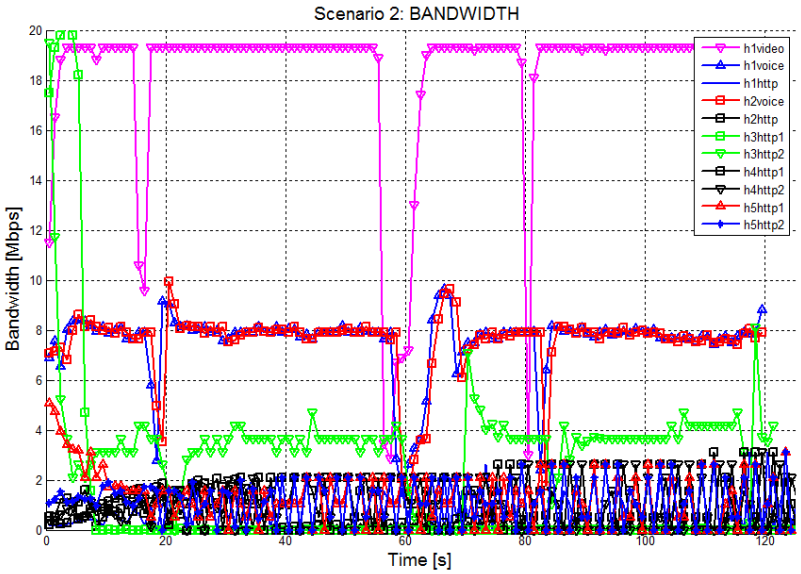
Despite of the bandwidth sharing, the packet delay variation and the loss percentage remain within the specified parameters as mentioned in Section 5-1.

### 5-2-2 Scenario 2

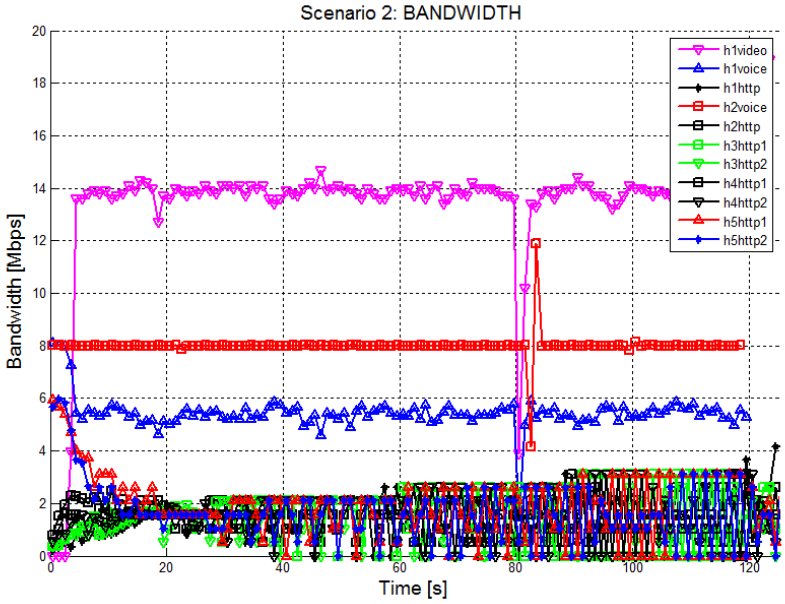
In the second scenario each host corresponding to the eNB is sending more than one policy simultaneous to the servers (PGW-U). Client 1 ($H_1$) has three sessions: 1 video, 1 voice and 1 HTTP session. Client 2 ($H_2$) has 1 video and 1 HTTP, whereas $H_3$, $H_4$ and $H_5$ have two HTTP sessions. The way in which the policies are installed can be done in a Round-Robin fashion based either on host or session type. Both Round-Robin methods use a window size $W$ of 1024 $K$ for the TCP connections, whereas the UDP buffer size is kept as default (1470 bytes).

| Client | Server | Edge Switch | Aggregation Switch | Core Switch | Session | ToS | Service | Bandwidth |
|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | $S_7$ | $S_4$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_2$ | $H_{45}$ | $S_7$ | $S_4$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_3$ | $H_{45}$ | $S_7$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_4$ | $H_{45}$ | $S_7$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_5$ | $H_{45}$ | $S_7$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_6$ | $H_{45}$ | $S_8$ | $S_5$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_7$ | $H_{45}$ | $S_8$ | $S_5$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_8$ | $H_{45}$ | $S_8$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_9$ | $H_{45}$ | $S_8$ | $S_5$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_10$ | $H_{45}$ | $S_8$ | $S_5$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{11}$ | $H_{45}$ | $S_9$ | $S_3$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_{12}$ | $H_{45}$ | $S_9$ | $S_3$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{13}$ | $H_{45}$ | $S_9$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{14}$ | $H_{45}$ | $S_9$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{15}$ | $H_{45}$ | $S_9$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{16}$ | $H_{45}$ | $S_{10}$ | $S_6$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_{17}$ | $H_{45}$ | $S_{10}$ | $S_6$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{18}$ | $H_{45}$ | $S_{10}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{19}$ | $H_{45}$ | $S_{10}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{20}$ | $H_{45}$ | $S_{10}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{21}$ | $H_{45}$ | $S_{11}$ | $S_6$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_{22}$ | $H_{45}$ | $S_{11}$ | $S_5$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{23}$ | $H_{45}$ | $S_{11}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{24}$ | $H_{45}$ | $S_{11}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{25}$ | $H_{45}$ | $S_{11}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{26}$ | $H_{45}$ | $S_{12}$ | $S_4$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_{27}$ | $H_{45}$ | $S_{12}$ | $S_5$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{28}$ | $H_{45}$ | $S_{12}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{29}$ | $H_{45}$ | $S_{12}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{30}$ | $H_{45}$ | $S_{12}$ | $S_4$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{31}$ | $H_{45}$ | $S_{13}$ | $S_6$ | $S_1$ | Video | 136 | Expedited Forwarding | 20 Mbps |
| $H_{32}$ | $H_{45}$ | $S_{13}$ | $S_5$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{33}$ | $H_{45}$ | $S_{13}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{35}$ | $H_{45}$ | $S_{13}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{36}$ | $H_{45}$ | $S_{13}$ | $S_6$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{36}$ | $H_{45}$ | $S_{14}$ | $S_5$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_{37}$ | $H_{45}$ | $S_{14}$ | $S_3$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_{38}$ | $H_{45}$ | $S_{14}$ | $S_5$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{39}$ | $H_{45}$ | $S_{14}$ | $S_5$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_{40}$ | $H_{45}$ | $S_{14}$ | $S_5$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |

**Table 5-3:** Returned path, Scenario 1




a) Throughput, Round-Robin Host Type




b) Throughput, Round-Robin Session Type

**Figure 5-7:** Scenario 2: Throughput

| Client | Server | Protocol | Session | DSCP | ToS | Service | Queue | Bandwidth | Priority |
|---|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | UDP | 1 Video | 34 | 136 | Assured Forwarding | 0 | 20 Mbps | 32000 |
| $H_1$ | $H_{46}$ | UDP | 1 Voice | 46 | 136 | Express Forwarding | 1 | 8 Mbps | 32767 |
| $H_1$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_2$ | $H_{46}$ | UDP | 1 Voice | 46 | 184 | Express Forwarding | 1 | 8 Mpbs | 32767 |
| $H_2$ | $H_{45}$ | UDP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mpbs | 30000 |
| $H_3$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_3$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_4$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_4$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_5$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_5$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |

**Table 5-4:** Scenario 2, Round-Robin Host Type

| Client | Server | Protocol | Session | DSCP | ToS | Service | Queue | Bandwidth | Priority |
|---|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | UDP | 1 Video | 34 | 136 | Assured Forwarding | 0 | 20 Mbps | 32000 |
| $H_1$ | $H_{45}$ | UDP | 1 Voice | 46 | 184 | Express Forwarding | 1 | 8 Mbps | 32767 |
| $H_1$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_2$ | $H_{46}$ | UDP | 1 Voice | 46 | 184 | Express Forwarding | 1 | 8 Mpbs | 32767 |
| $H_2$ | $H_{45}$ | UDP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mpbs | 30000 |
| $H_3$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_3$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_4$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_4$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_5$ | $H_{45}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |
| $H_5$ | $H_{46}$ | TCP | 1 HTTP | 0 | 0 | Best Effort | 2 | 6 Mbps | 30000 |

**Table 5-5:** Scenario 2, Round-Robin Session Type

The configuration script for the Round-Robin Host Type can be found in Appendix A-2-2-1, whereas the configuration for the Round-Robin Session Type is given in Appendix A-2-2-2.

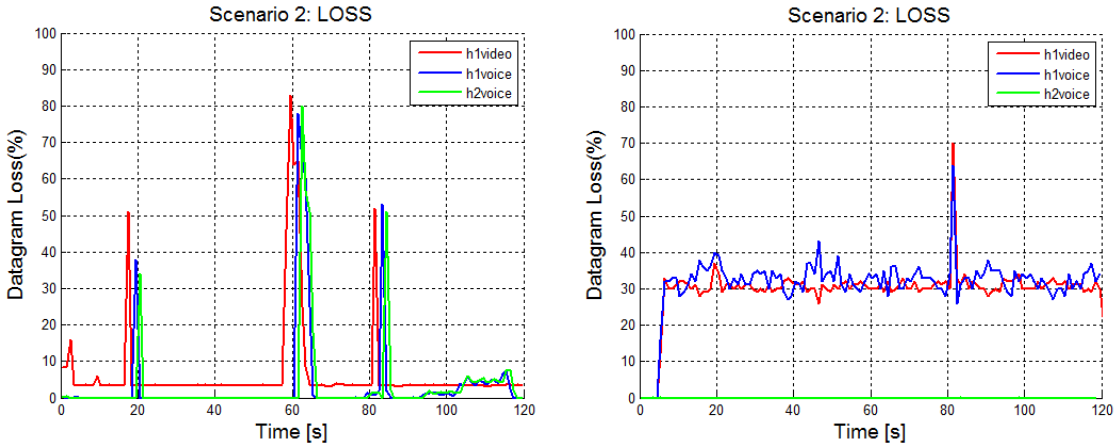
The characteristics of each flow in Round-Robin Host Type is illustrated in Table 5-4. The traffic is equally balanced between server $H_{45}$ and $H_{46}$.

The characteristics of each flow in Round-Robin Session Type are described in Table 5-5. The traffic is distributed based on session type. The first time a session is initiated, it goes to server $H_{45}$, whereas the second traffic of the same session goes to $H_{46}$.

In Figure 5-7-a), there are 8 HTTP connections and each of the connections takes approximately 2 Mbps from the server. The video bandwidth is limited to 19 Mbps, whereas the two voice sessions arise at 8 Mbps. All the TCP sessions are defined for a window size $W$ equal to 1024 $K$. In contrast to the first scenario, in the second scenario the UDP sessions do not have buffer size configured. In Figure 5-7-b) the video session has to share the bandwidth of 20 Mbps with the voice traffic. As a result, the video session takes 14 Mbps, while the voice stream is limited to 6 Mbps.

The average loss for the two voice sessions in the Round-Robin Host Type connections is lower than in the case of Round-Robin Session Type, but presents some peak variations (Figure 5-8-a)). These variations correspond to the bandwidth fluctuations at the same moment (i.e at the time interval of 60 s the video bandwidth drastically decreases). The video session experiences a higher loss percentage than the rest of the sessions.

In the case of the Round-Robin Session Type (Figure 5-8-b)), the loss for the video and voice traffic coming from $H_1$ is higher than for the voice connection sent by $H_2$ which has no loss. This can be caused as in the first scenario by the presence of TCP streams and




(a) Loss, Round-Robin Host Type

(b) Loss, Round-Robin Session Type

**Figure 5-8:** Scenario 2: Datagram Loss

its flow mechanism, or due to the congestion windows synchronization. Another reason for the high loss is the high bandwidth (20 Mbps), required for the video session and the actual achieved throughput of 14 Mbps. From the loss point of view, the Round-Robin Host Type load distribution algorithm shows a better performance than in the Round-Robin Session Type case.

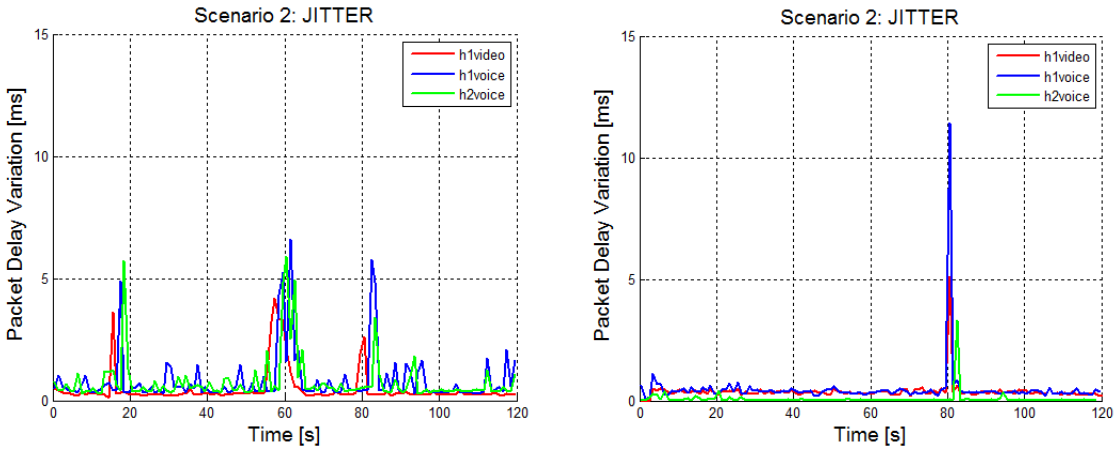
Figure 5-9-a) shows that the packet delay variation (jitter) value is very small for both VoIP and video sessions and it is maintained below 5 ms with small variations. This is a normal value, less than 30 ms as it is recommended in [51]. For the Round-Robin Session Type, the achieved jitter is even smaller than in the case of Round-Robin Host Type. Figure 5-9-b) shows a high spike in the jitter values around 10 ms which is produced after 80 s. This jitter variation corresponds to the same throughput variation for the video stream at a time interval equal to 80 ms (Figure 5-7-b)).

#### 5-2-2-1 Path Calculation

In the first scenario presented in Section 5-2-1-6, where only one policy was installed for all the 40 hosts, all the aggregation switches are traversed on the path to servers $H_{45}$ and $H_{46}$. In contrast to the path calculation in Scenario 1, Table 5-6 and Table 5-7 summarize the traversed paths for the multiple policies installed on the first five hosts. The returned path in for both Host and Session load balancing varies at the aggregation switch level between switches $S_3$ and $S_5$.

Table 5-6 confirms that the traffic is distributed to both servers, $H_{45}$ and $H_{46}$ independent on the session type. The returned path alternates at the aggregation level between switches $S_3$ and $S_5$.

In the case of Round-Robin Session Type, the returned path for all sessions sent by $H_1$ follows the same path (i.e. $S_7$, $S_3$ and $S_1$). However, the sessions coming from $H_2$ are sent throughout $S_2$, $S_5$ and $S_7$ (Table 5-7). The path is dependent on the chosen server, thus all the traffic sent to server $H_{45}$ takes the route via $S_3$, whereas server $H_{46}$ has the path containing $S_5$ assigned.




(a) Jitter, Round-Robin Host Type

(b) Jitter, Round-Robin Session Type

**Figure 5-9:** Scenario 2: Packet Variation Delay

| Client | Server | Edge Switch | Aggregation Switch | Core Switch | Session | ToS | Service | Bandwidth |
|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_1$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_1$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_2$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | Voice | 184 | Expedited Forwarding | 6 Mbps |
| $H_2$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_3$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_3$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_4$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_4$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_5$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_5$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |

**Table 5-6:** Returned path Round-Robin Host Type

| Client | Server | Edge Switch | Aggregation Switch | Core Switch | Session | ToS | Service | Bandwidth |
|---|---|---|---|---|---|---|---|---|
| $H_1$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | Video | 136 | Assured Forwarding | 20 Mbps |
| $H_1$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | Voice | 184 | Expedited Forwarding | 8 Mbps |
| $H_1$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_2$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | Voice | 184 | Expedited Forwarding | 6 Mbps |
| $H_2$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_3$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_3$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_4$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_4$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_5$ | $H_{45}$ | $S_7$ | $S_3$ | $S_1$ | HTTP | 0 | Best Effort | 6 Mbps |
| $H_5$ | $H_{46}$ | $S_7$ | $S_5$ | $S_2$ | HTTP | 0 | Best Effort | 6 Mbps |

**Table 5-7:** Returned path Round-Robin Session Type

The difference between Session and Host Type load balancing consists in the distribution of the same traffic type and the chosen server.

#### 5-2-2-2 Analysis

In this scenario all the policies were added based either on Round-Robin host or session instantiation. Following the Host Type, $H_1$ sends one video session to server $H_{45}$ and one voice session to server $H_{46}$, whereas in the "session" type both UDP streams connect to server $H_{45}$. For instance, the first voice session coming from $H_1$ is sent to server $H_{45}$, whereas the second voice session transmitted by $H_2$ is sent to $H_{46}$. Consequently, even though it is more stable, the throughput in the second case is divided between video and voice sessions.

The first configuration also performs better in terms of the obtained loss. Both Round-Robin Host and Session Type experienced loss because of the TCP flow control mechanism. The UDP is a simple protocol without connection setup delays, flow control, and retransmission, providing applications with a more raw interface to the network. The window size of 1024 $K$ causes TCP congestion synchronization.

In Figure 5-8 -b) (Round-Robin Session Type) the voice stream connected to server $H_{46}$ achieved a very low loss value, close to the recommended parameters. On the other hand, the other voice session has to share the bandwidth with the video session on server $H_{45}$, thus the number of dropped packets is very high, resulting in high percentage of packet loss. Nevertheless, the obtained jitter is smaller in the case of Round-Robin Session Type and it is due to the throughput stability. The variations of throughput also causes the packet delay variation.

In terms of returned path, the traffic distribution alternates between switches $S_3$ and $S_5$ at the aggregation level. The Round-Robin Host Type algorithm sends one session to server $H_{45}$ followed by the next stream to server $H_{46}$, whereas the Round-Robin Session Type never sends the same type of traffic to the same server consequently.

### 5-2-3 Scenario 3: Round Robin Balancer

In this scenario, the Round Robin balancing mechanism was tested using two different SDN controllers: Floodlight and OpenDaylight presented in Chapter 4. The configuration scripts for this scenario are provided in Appendices A-3-1 and A-3-2.

In Figure 5-10 the bandwidth of 1 Mbps was set up for both of the controllers. Client 1 ($H_1$) sends all the traffic to IP address *10.0.0.100*, which is configured as the Virtual IP (VIP) for UDP traffic. For TCP traffic, the assigned VIP is *10.0.0.200*.

The VIP contains a pool of servers: $H_2$, $H_6$, $H_7$, $H_{11}$, $H_{12}$, $H_{16}$, $H_{17}$ which also includes $H_1$ (the client). This cannot run in both client / server at the same time and it will always skip connecting to itself. The round robin mechanism was validated for both Floodlight and OpenDaylight controllers: the client connects to each of the servers in a chain manner during 1 second.

The Opendaylight controller shows better performance in terms of throughput which is more constant, around 1 Mbps, whereas the Floodlight controller gives frequent variations at the beginning of each round (Figure 5-10).

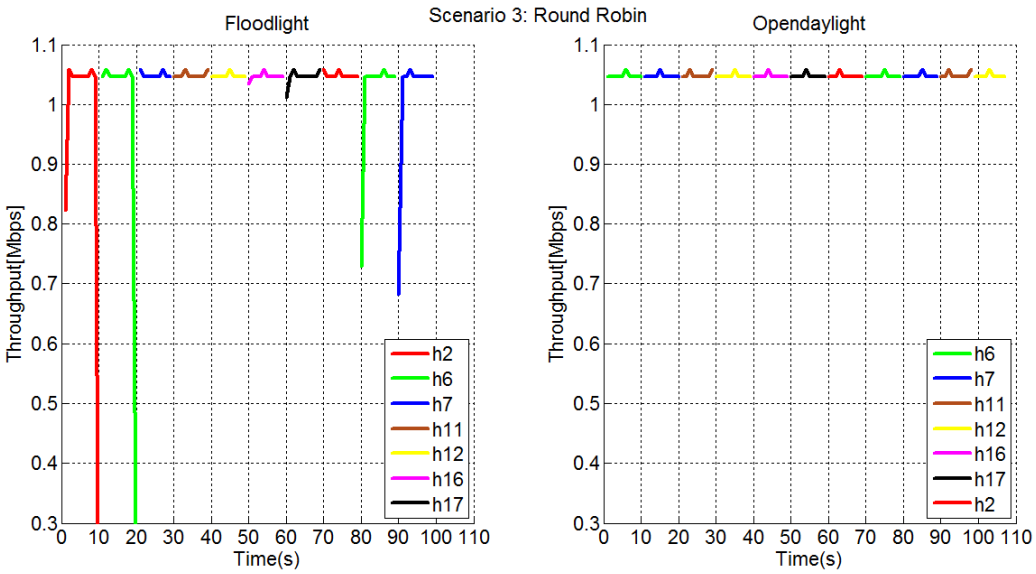



**Figure 5-10:** Scenario 3: Throughput

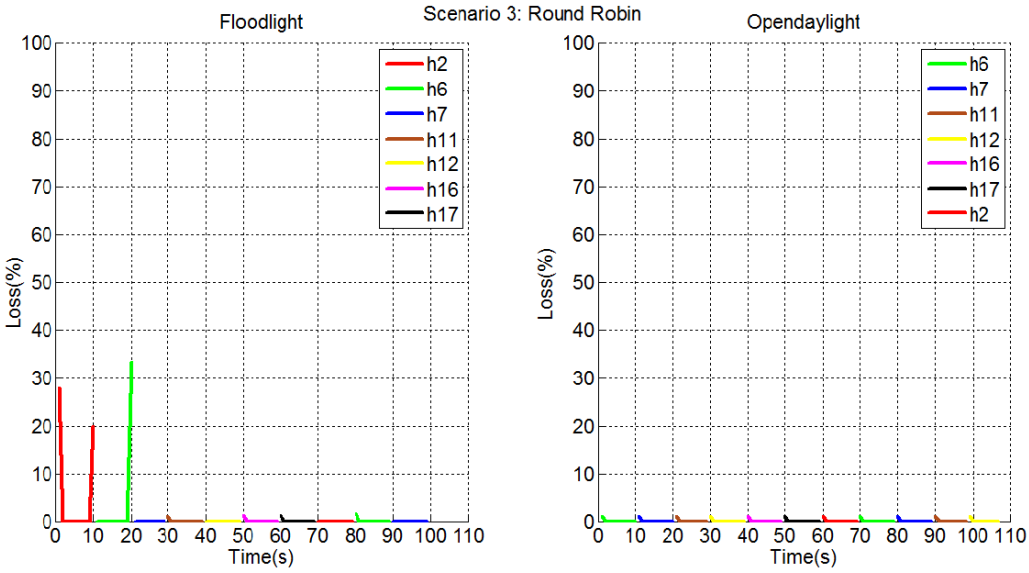



**Figure 5-11:** Scenario 3: Loss

In Figure 5-11, the packet loss is illustrated for both of the controllers (only the UDP traffic for voice and video sessions). At the beginning of each server connection, the losses obtained for the Floodlight load balancing are higher, whereas for Opendaylight the losses do not exceed 2%.

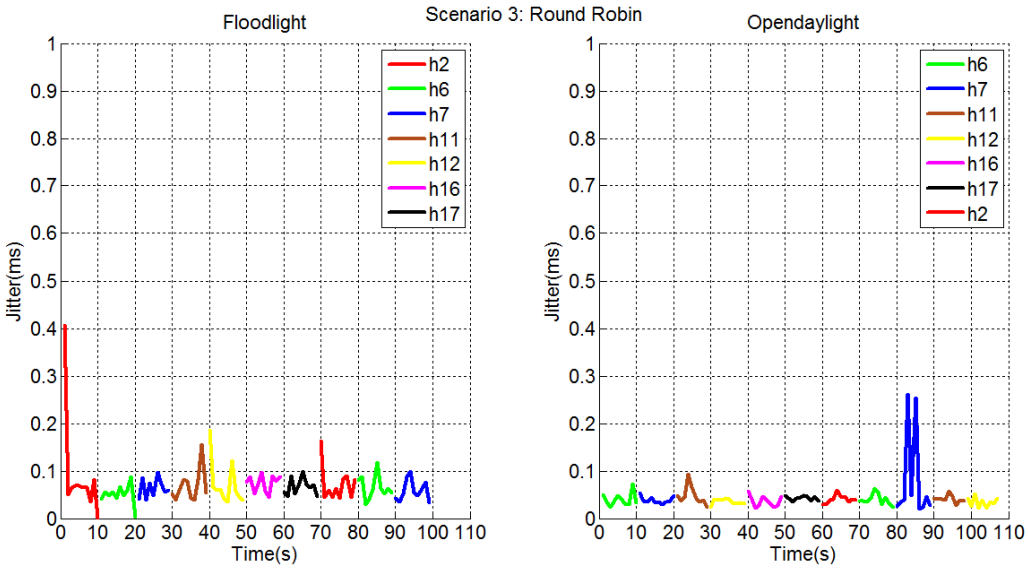



**Figure 5-12:** Scenario 3: Jitter

Also in the case of jitter (Figure 5-12), the values for packet delay variation are below 0.3 ms for the Opendaylight controller, while for the Floodlight, the jitter is even greater than 0.3 ms. The server connection follows the round robin chain, in which each server connection lasts for 10 s interval and the order is the one defined in the VIP pool sequence.

#### 5-2-3-1 Analysis

The results obtained for the two controllers are comparable in terms of throughput, loss and jitter. In the Floodlight controller, the Round-Robin load balancing module cannot be combined with the QoS module due to the fact that the load destination address is changed each time. No policy on path can be established between source and destination (the VIP). When the QoS is activated and we install policies on the paths, all the traffic is limited to the maximum queue value.

For both OpenDaylight and Floodlight controllers, 10 tests were run, with the observation that for the OpenDaylight the whole test lasted 10 s more, when server 1 passed through the chain. Floodlight automatically excludes the first server from its pool, whereas for OpenDaylight, one host can connect to itself only in the case it has been defined in the VIP pool, but it does not receive any traffic back from the server. In its current version, OpenDaylight does not support topologies containing loops. Therefore, only classic tree topology has been tested.

### 5-2-4 Scenario 4: OpenDaylight Load Balancer

In this scenario, the round robin load balancing mechanism was compared to a random balancing which run as a separate module on the OpenDaylight controller. In contrast to Floodlight, in the OpenDaylight controller the Round-Robin mechanism can be tested for a bandwidth larger than 1 Mbps.

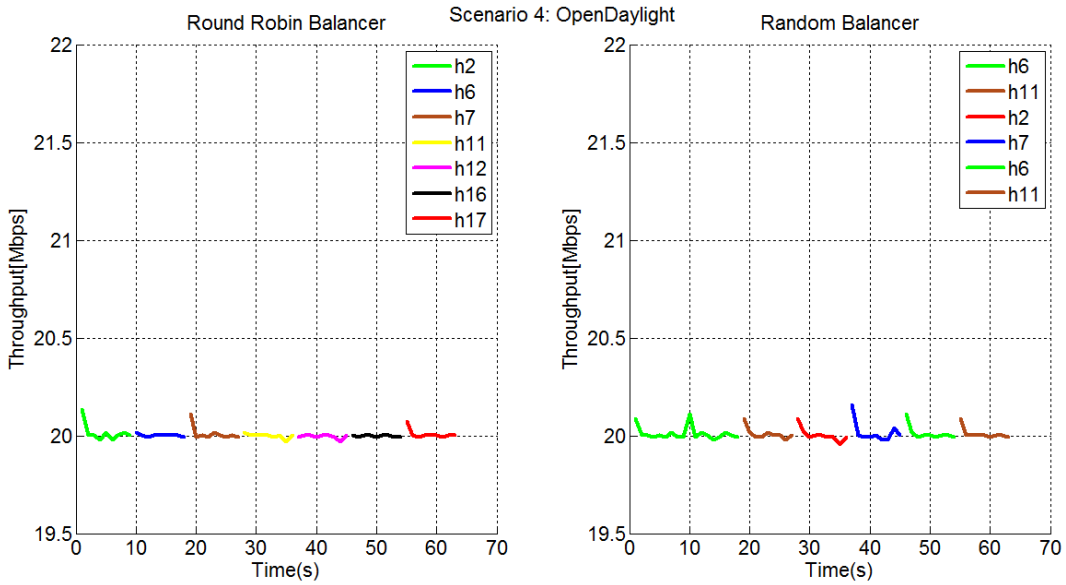



**Figure 5-13:** Scenario 4: Throughput

Therefore, in Scenario 4 the bandwidth has been tested for 20 Mbps, which is the rate limit established for the video connection in the first two scenarios.

Figure 5-13 validates the achieved throughput around 20 Mbps for both random and Round Robin load balancing.

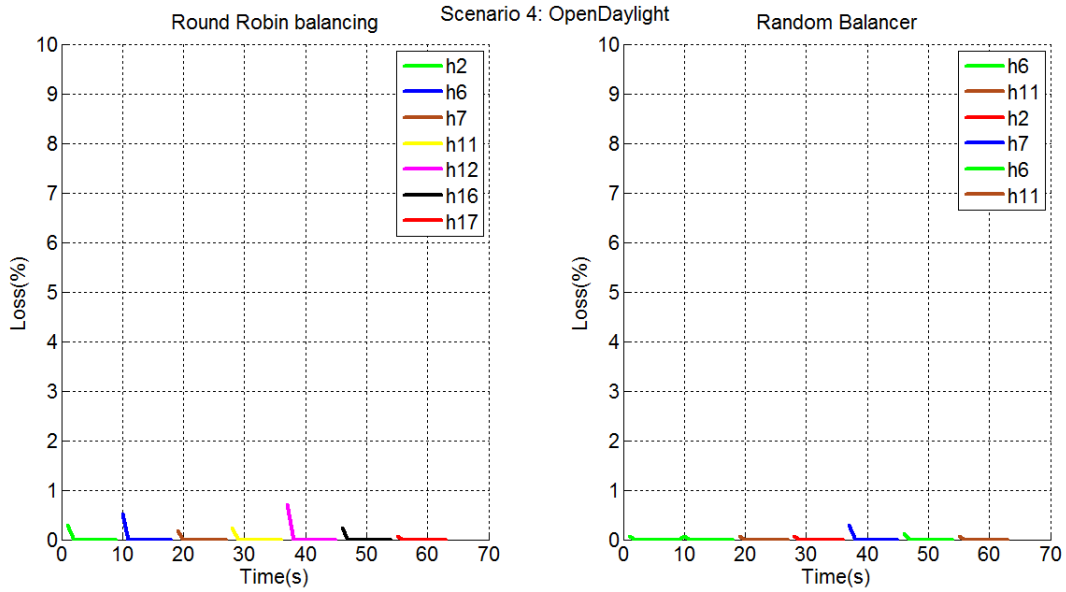



**Figure 5-14:** Scenario 4: Datagram Loss

The datagram loss is comparable in both cases of round robin and random load balancing mechanism and it has a very small value (Figure 5-14). This is close to the ideal recommendation for a video connection with no loss.

In Figure 5-15, the values for packet variation delay are on average below 0.3 ms for both random and load balancing mechanisms. This is in accordance with the specified values for video and voice loss. In case of the random balancer, the jitter fluctuates very often at the beginning of each connection.

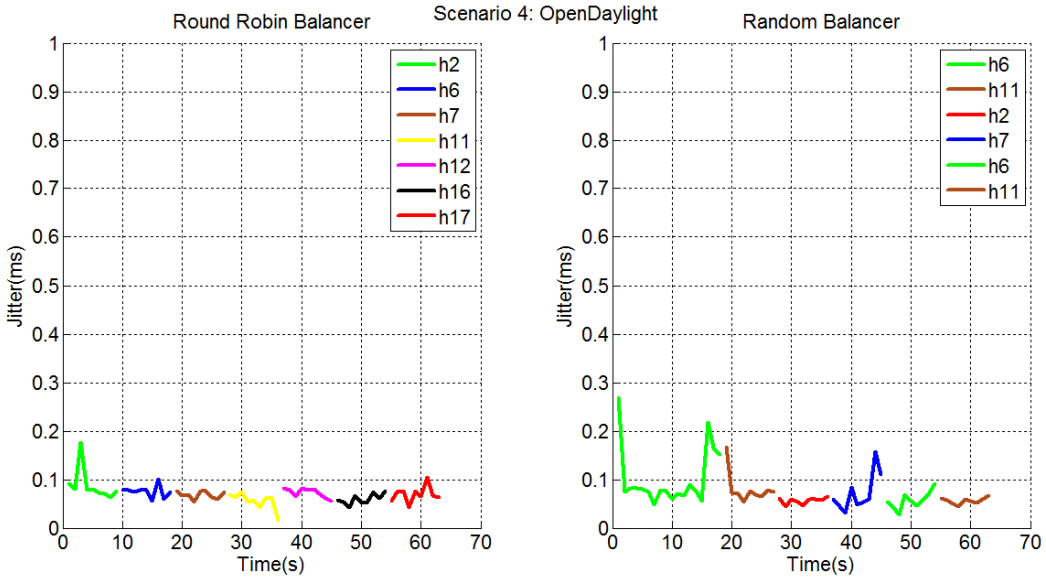



**Figure 5-15:** Scenario 4: Jitter

#### 5-2-4-1 Analysis

The difference between the two load balancing mechanisms is the order in which the client connects to the server. In the random balancing case, the client only connects to the four servers ($H_6$, $H_{11}$, $H_2$ and $H_{11}$), whereas the Round-Robin load balancer goes through the whole chain of seven servers ($H_2$, $H_6$, $H_7$, $H_{11}$, $H_{12}$, $H_{16}$ and $H_{17}$). The compulsory rule for Round Robin is to connect only once to all of the servers. The random balancer connects twice to servers $H_6$ and $H_{11}$. This might cause congestion in the network when the client connects to the same server more than once. That is the reason why a random load balancing is not indicated as a solution to congestion issues. Moreover, the jitter introduced by the random balancer varies at the beginning of each connection.

### 5-2-5 Scenario 5: Comparison between Controllers

In this scenario, a control plane comparison between Floodlight and OpenDaylight controller was performed using the *CBench* tool presented in Section 4-4-3.

The results were processed in *Python* and the plot was realized using *gnuplot* tool. This tool allows to benchmark how many flows per second the controller can handle, as well as the Central Processing Unit (CPU) and Random-Access Memory (RAM) usage of each iteration.

In Figure 5-16, the performances of the two controllers were tested for Scenario 1 (consisting of 8 edge switches, each of them having 5 hosts connected with one policy per host) presented

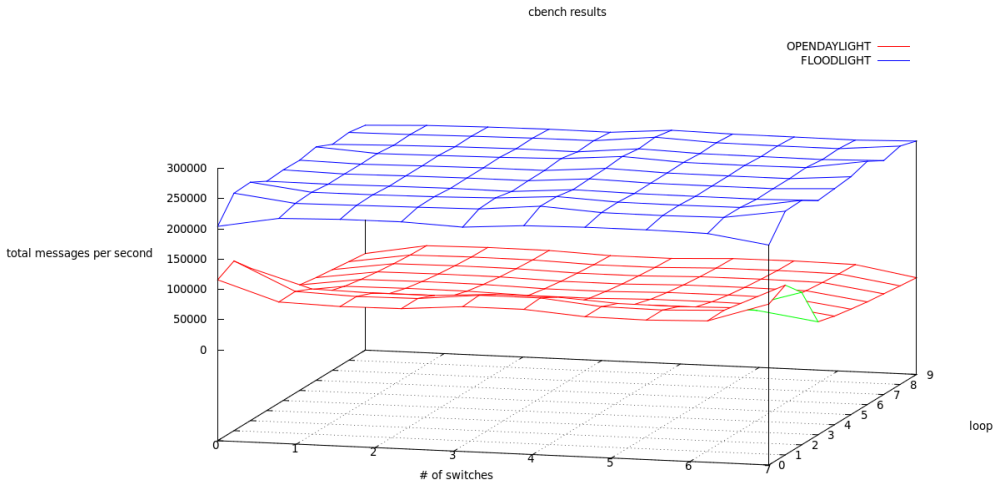



**Figure 5-16:** Controller comparison: Scenario 1

in Section 5-2-1. The Floodlight performance in terms of number of processed messages per second is better with almost $10^5$ processed messages per second than the OpenDaylight controller processing time.

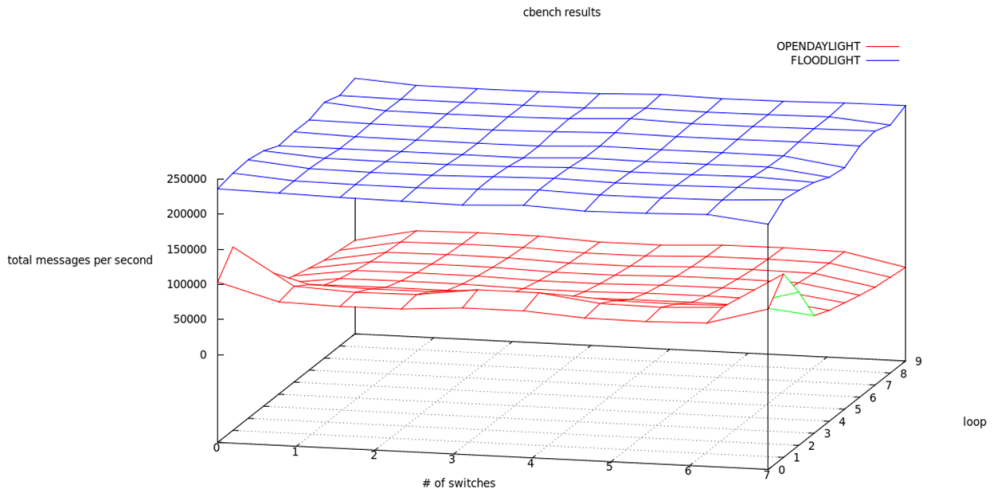



**Figure 5-17:** Controller comparison: Scenario 2

In Figure 5-17, the two controllers were compared for Scenario 2 (consisting of 8 edge switches, each of them having more than one policy per host installed) previously presented in Section 5-2-2. The number of processed messages per second arises in the case of the Floodlight controller up to $250 \times 10^3$, whereas in the case of OpenDaylight controller, the number of processed messages does not change in comparison to the previous configuration (Figure 5-17).

For a configuration of 1000 hosts per switch, the OpenDaylight controller collapses, whereas Floodlight processing capacity increases to 1 million of processed flows per second (see Figure 5-18).

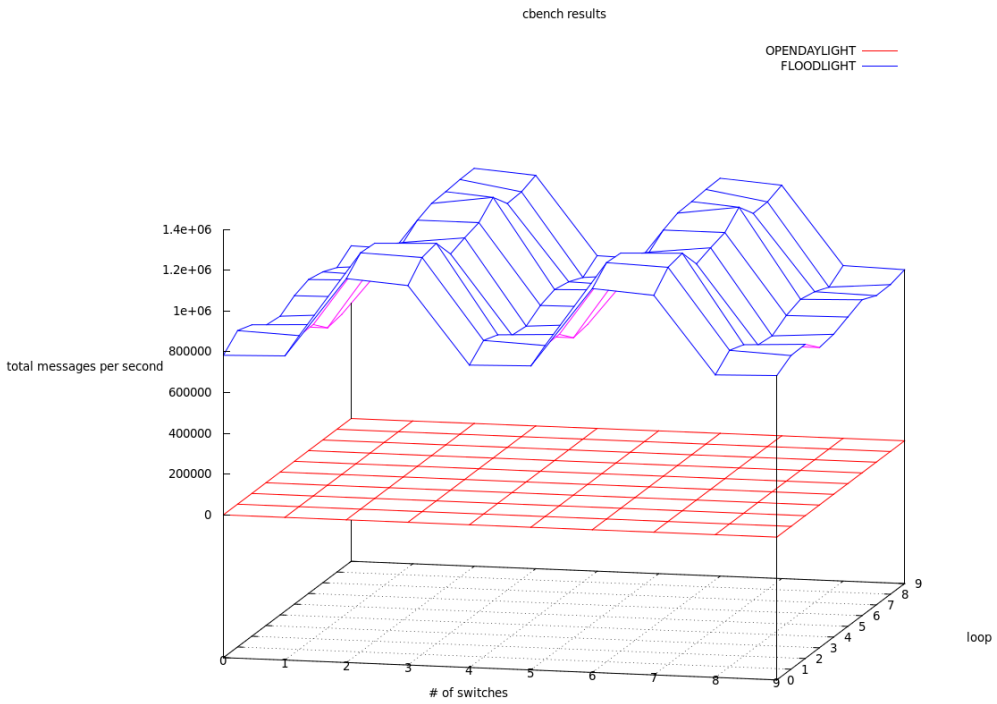



**Figure 5-18:** Controller comparison: 1000 hosts

### 5-2-5-1 Analysis

The OpenDaylight can only run in *latency* mode, whereas Floodlight can run in both *throughput* and *latency* modes. In the *latency* mode, OpenDaylight performs accordingly, whereas in *throughput* mode the CPU goes very high and cannot process incoming packet-in messages in an stable way.

The *Latency* mode measures the OpenFlow controller's request processing time under low-load conditions, while in *throughput* mode, the switches send as many request as the buffer supports. The OpenDaylight controller cannot process millions of flows per second as it functions only up to some point, after that it is overwhelmed and goes unpredictably down. In contrast to that, the Floodlight's achieved performances are far more better in terms of the processed number of messages per second.

### 5-2-5-2 Comparison of Controller Performances

The performances of the two controllers for the presented scenarios were summarized in Table 5-8. Both controllers (Floodlight and OpenDaylight) present advantages as well as disadvantages, thus depending on the use case and application their performances are different. Although it does not have any QoS module implemented in the current version, the load balancer performances are better than in the case of Floodlight which only performs load balancing for a maximum bandwidth of 1 MB. Nevertheless, when it comes to process millions of flows per second, the OpenDaylight controller gets blocked. Therefore, in deployments of tens of switches with hundred of hosts, OpenDaylight can be used for load balancing purposes,

| **Floodlight** | **OpenDaylight** |
|---|---|
| QoS (based on policy paths and classes of Service) | No QoS module |
| Supports Load Balancing (Round Robin VIP Pool) | Supports Load Balancing (Round Robin VIP Pool) |
| No Random Load Balancer module | Random Load Balancer module (it can pass multiple times through the chain) |
| Load Balancing Bandwidth performance limited to 1 MB | No Bandwidth limitation for Load Balancing |
| Throughput variations and higher loss | Better throughput and lower loss |
| Shorter test duration excluding the host itself in the pool | Longer test duration including the host itself in the pool |
| Does support topologies with loop configuration | Does not support topologies with loop configuration |
| Can run both in "latency" and "throughput" modes | Can only run in "latency" mode |
| Better performance for the first two scenarios in terms of processed messages per second | Lower performance with almost 1 $K$ processed messages per second in the first two scenarios |
| Can process up to 1 million of flows per second | Can process up to 100000 flows per second (unable to process millions of flows) |

**Table 5-8:** Comparison of the two Controllers performances

whereas Floodlight is more suitable for larger deployments as it can process up to millions of messages per second.

### 5-2-6 Experimental Simulated Set-up: Mininet Final remarks

The obtained results in the first four scenarios can be also attributed to the simulation environment. One of the Mininet-based network drawback is that it cannot (currently) exceed the CPU or bandwidth available on a single server. For instance, the measurements were run on a Virtual Machine (VM) with 2.27 GHz on two cores and 2 GB RAM. Due to the fact that the measurements were run on a single VM, the resources had to be balanced and shared among the virtual hosts and switches. That is the reason why this testbed uses slower links (50 Mbps or 250 Mbps) with lower performance, rather than 10 Gbps on dedicated switching hardware.

Apart from this, in the experiment the CPU bandwidth of the Mininet hosts was limited to a maximum bandwidth of 20 Mbps (for video session). This choice was determined by the fact that the packets were forwarded by a collection of software switches (e.g. Open vSwitch) that share CPU and memory resources. This happened to have lower performance than dedicated switching hardware.

# 5-3 Experimental Physical Set-up: SurfCloud, Amsterdam

## 5-3-1 Deployment

This test bed was implemented using OpenFlow switches and a remote connection via Secure Shell Protocol (SSH) authentication using *Putty* terminal. The location was Amsterdam, on a physical testbed held by the SURFnet company [76].

The measurements were conducted using *Jperf* tool that is the graphical interface of *Iperf* bandwidth tool. The configuration details were presented in Section 4-3. The *Jperf* tool uses the *Iperf* traffic generator in a client-to-server connection. In the topology presented in Figure 4-7, 'Lithium" is configured as server (IP 10.0.0.3), whereas the client is represented by the VM (IP 10.0.1.3). The reverse configuration could not be achieved due to the lack of privileged rights on the "Lithium" server.

### 5-3-1-1 User Plane

**5-3-1-1-1 TCP** In this paragraph, the TCP throughput was tested for a window size of 100 KBytes and a buffer length of 500 MB and 1 GB, respectively.

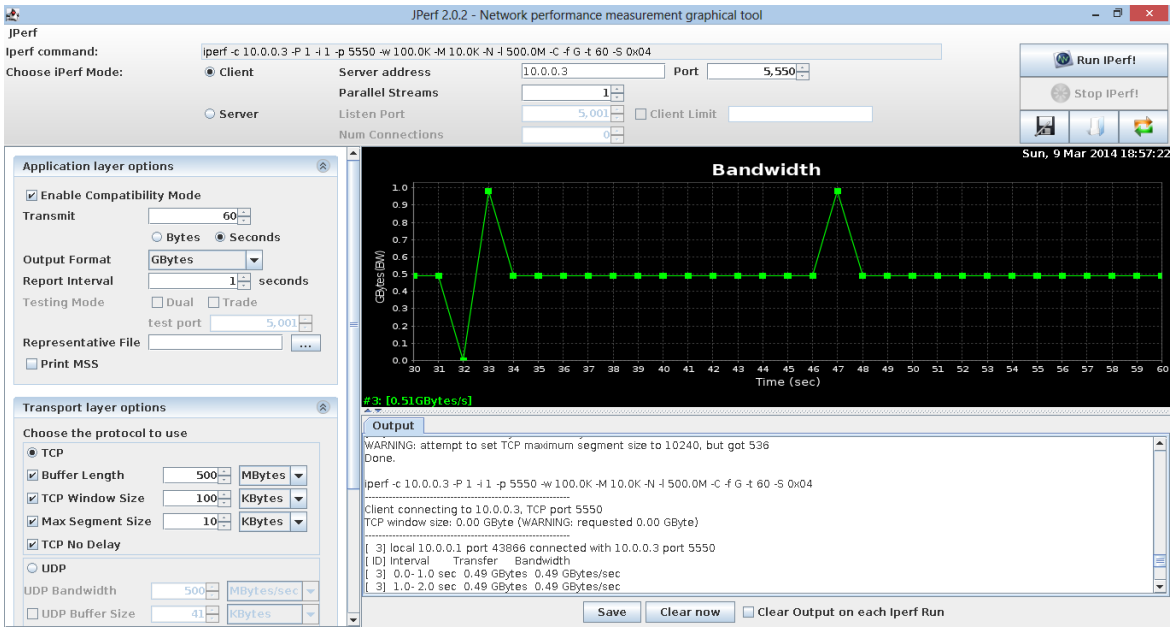



**Figure 5-19:** TCP throughput, window size 500 MB

From Figure 5-19 it can be seen that for a TCP buffer of 500 MB, the throughput is constant (around 490 Mbps) with small variations. The throughput test duration lasted for 60 s.

When the buffer size is increased to 1 GB, the obtained throughput fluctuates very rapidly (Figure 5-20). The ToS field is set according to Table 5-9 which was defined in [16] and the "throughput" corresponds to a value of "0x08". The ToS values were presented in Figure 3-2.

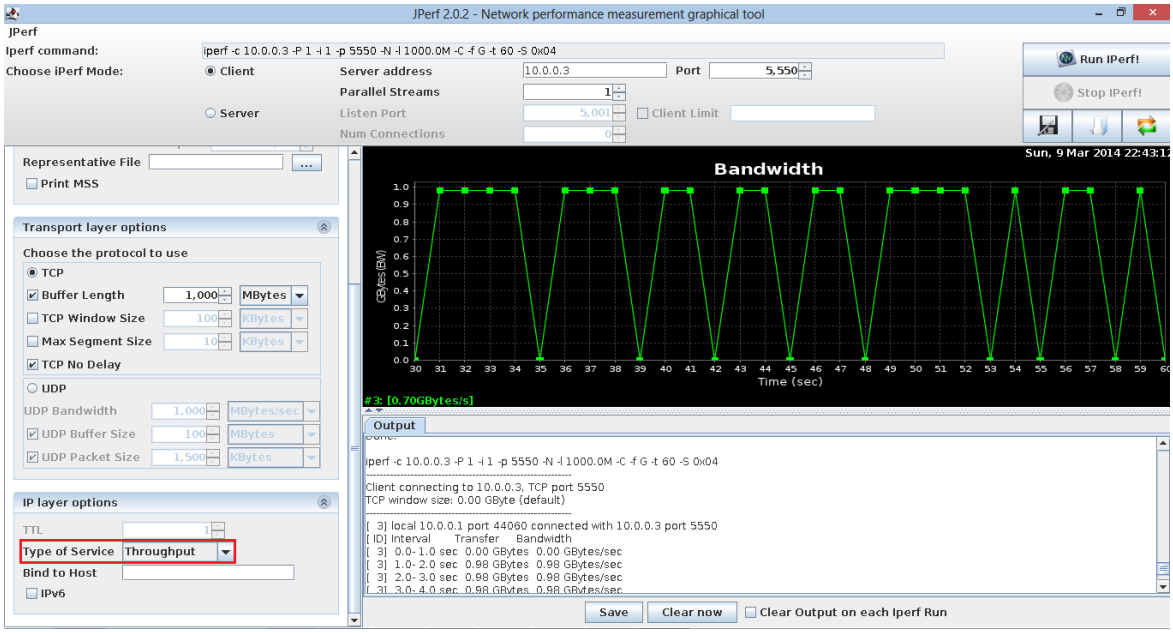



**Figure 5-20:** TCP throughput, window size 1 GB

| IPTOS | Action | Values |
|---|---|---|
| IPTOS_LOWDELAY | minimize delay | 0x10 |
| IPTOS_THROUGHPUT | maximize throughput | 0x08 |
| IPTOS_RELIABILITY | maximize reliability | 0x04 |
| IPTOS_LOWCOST | minimize cost | 0x02 |

**Table 5-9:** Class Selector, source [16]

#### 5-3-1-1-2 UDP

As in the previous case of TCP session, the UDP throughput was tested for a bandwidth of 500 Mbps and 1 Gbps.

Figure 5-21 shows that the obtained throughput is constant and does not exceed the value of 600 Mbps. In Figure 5-22 the achieved throughput for a buffer size of 100 MB is constant in comparison to TCP tested throughput and does not exceed the value of 1 Gbps.

### 5-3-1-2 Analysis

In the case of TCP traffic, the throughput drastically varies for a bandwidth of 1 GB, whereas in the case of UDP it is constant. This might occur due to TCP window synchronization. Due to TCP window, the flow control mechanism determines a packet retransmission. Simultaneously, the TCP reduces its congestion window size and implicitly the output rate is decreased to avoid congestion. As soon as the congestion has been avoided, the TCP increases the congestion window size as well as the output rate. In the absence of flow control mechanism, the UDP transmission rate is constant.

It can be seen from Figure 5-20 and Figure 5-22 that the selected ToS is set up as “Throughput” mode, which corresponds to a value of “0x08” according to Table 5-9. Setting the ToS field is intended to give the better service when it is available, in this case trying to maximize the throughput.

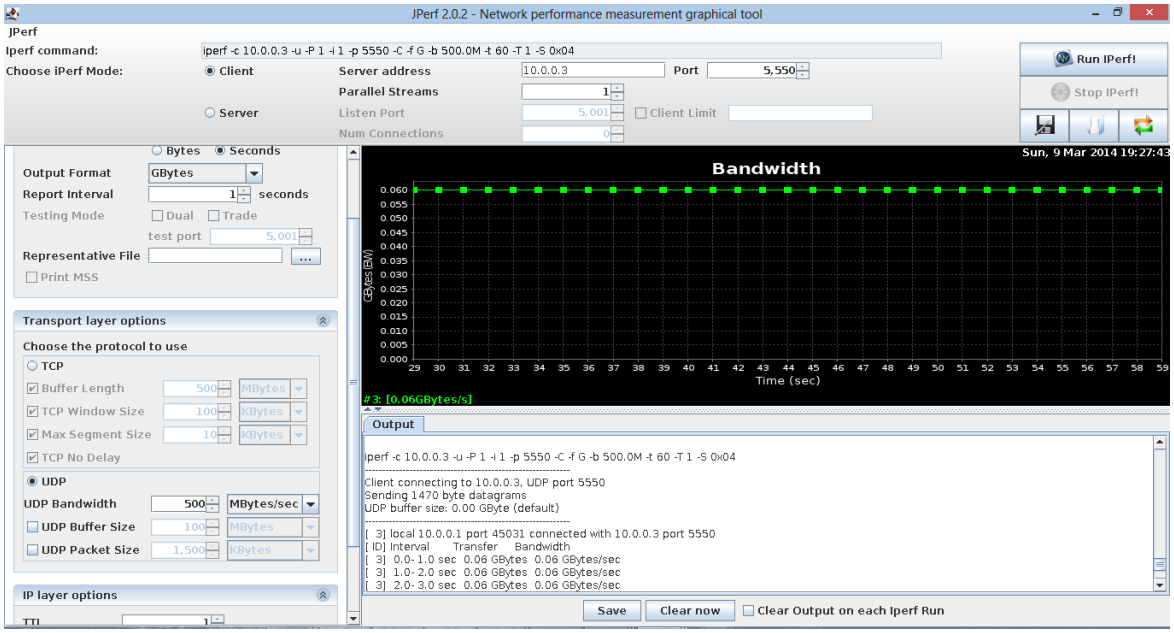



**Figure 5-21:** UDP throughput, buffer size 500 MB

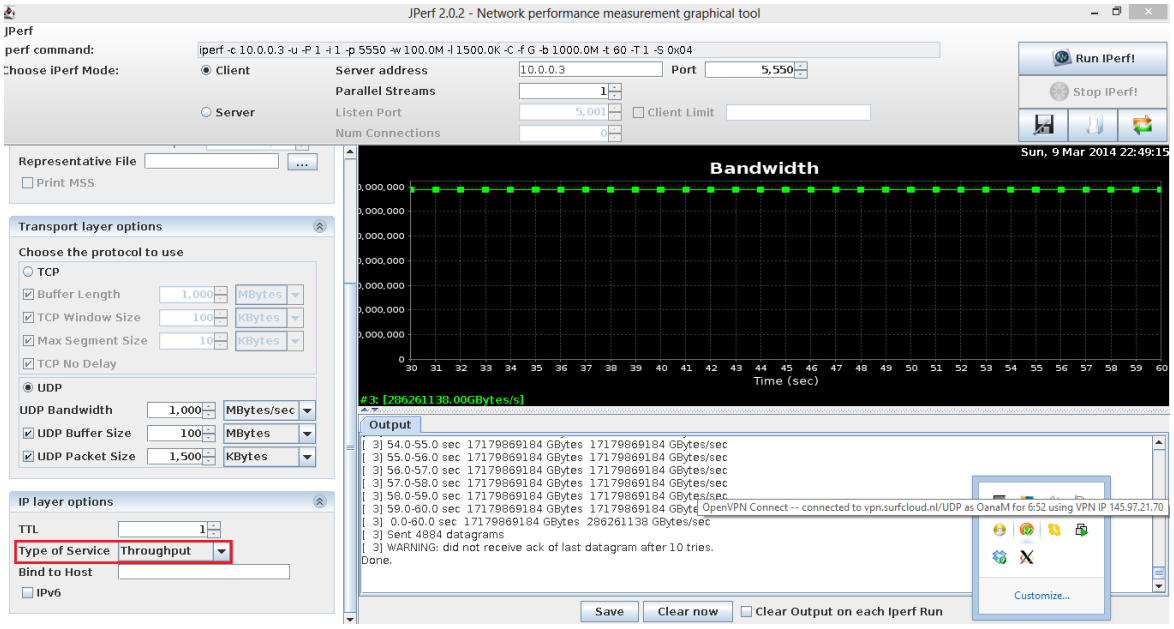



**Figure 5-22:** UDP throughput, buffer size 1 GB

### 5-3-1-3 Control Plane

One of the drawbacks in the current *Wireshark* version is the fact that it does not support OpenFlow captures. Nevertheless, there is a possibility to extend it in order to visualize OpenFlow packets. Therefore, the *OpenFlow Wireshark Dissector* has been installed. This is a tool which provides an additional feature to *Wireshark* [77] to be able to capture OpenFlow packets.

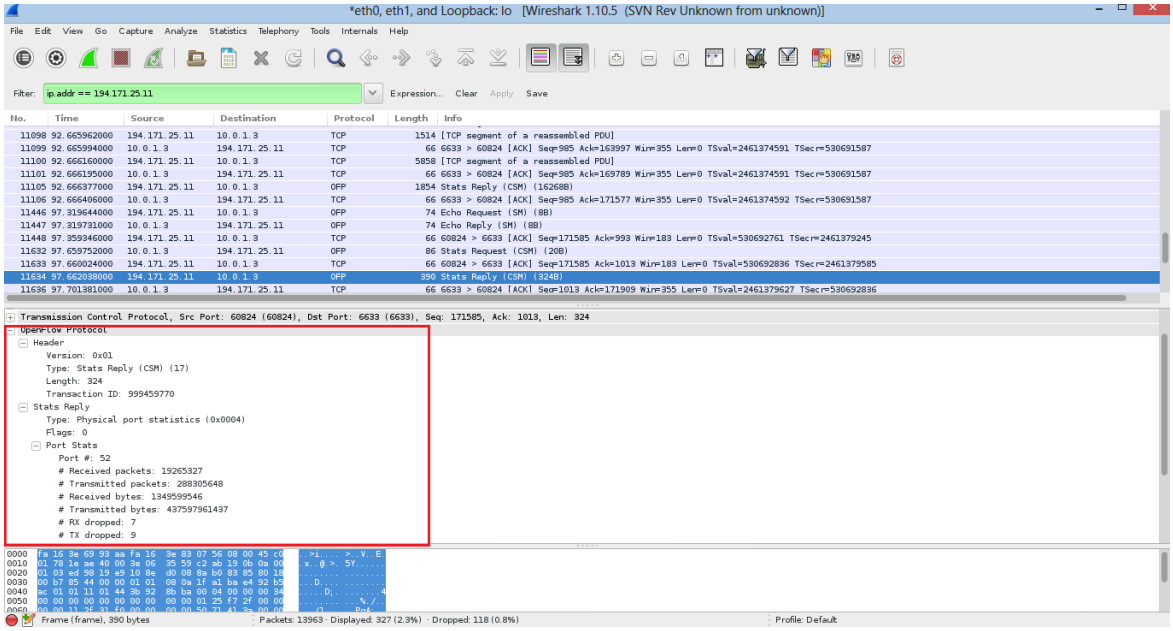



**Figure 5-23:** OpenFlow capture

Figure 5-23 illustrates the Southbound interface on which the SDN controller communicates with the switches using OpenFlow protocol. In this scenario, a reactive mode instantiation can be observed, in which the controller (*10.0.1.3*) queries the OpenFlow switch (*194.171.25.11*). Several *Request Reply* messages are exchanged between the control and the switch.


**Figure 5-24:** Control Plane throughput comparison

Figure 5-24 shows the control traffic composed of OpenFlow messages sent by the controller and OpenFlow traffic coming from the switches. The OpenFlow throughput measured on interface 10.0.1.3 exceeds 0.5 Gbps. On the other hand, the overall TCP throughput (summed for all interfaces) is very high (it can reach up to 1 Gbps), since it provides the transport layer upon which OpenFlow runs. All the exchange messages PACKET_IN PACKET_OUT between the OpenFlow switches and the SDN controller use TCP as transport layer.

#### 5-3-1-4 Analysis

The OpenFlow Switch respond to the controller and it provides the Port Statistics: the number of transmitted/received packets as well as the dropped packets. This information is very useful to the controller in order to get an overview of the current load in the network.

## 5-4 OpenEPC measurements

#### 5-4-0-5 S1-MME interface

In the proposed system design presented in Chapter 3, the *S1-Mobility Management Entity (MME)* interface between eNB and MME has been preserved. The control plane interface, *S1-MME* was simulated using the *OpenEPC* tool described in Section 4-1-2-1.

In this deployment, 192.168.4.90 is the IP address of the eNB, whereas 192.168.4.80 is the IP address of the MME. The eNB sends an *Attach Request (PDN connectivity request)* to the MME. The MME will answer with a *SACK* message. Then an *Initial Context Setup* is sent from eNodeB to MME. After the MME reply, the *Attachment Procedure* is completed and *Activate default EPC Bearer Context Accept* is confirmed.

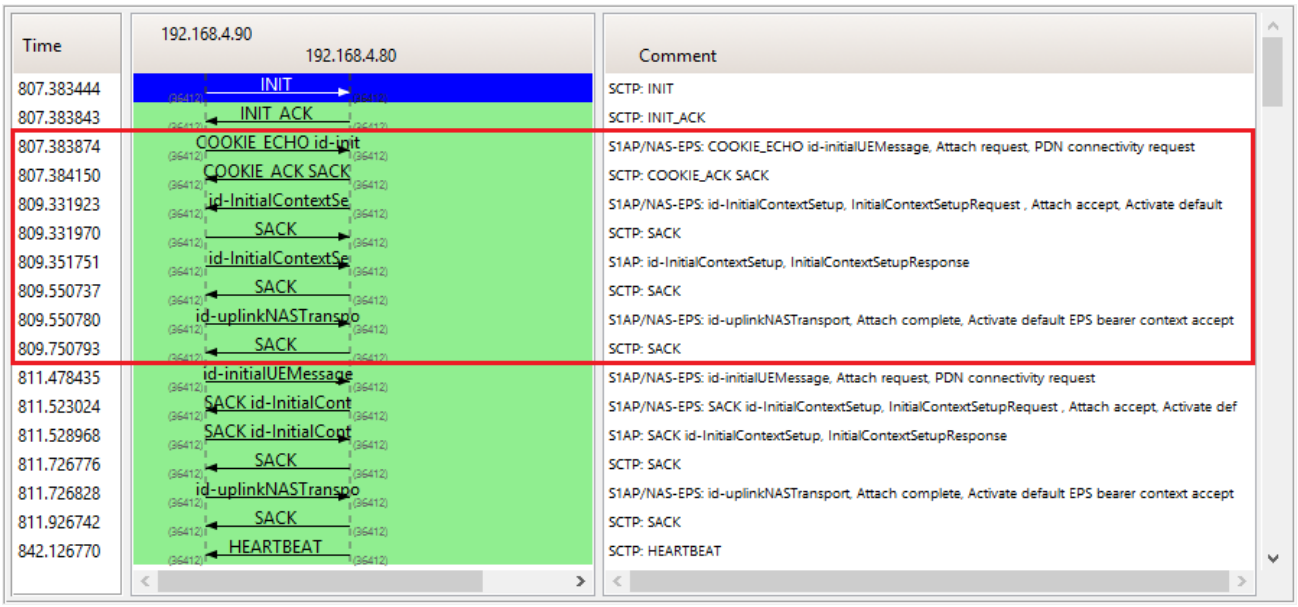



**Figure 5-25:** S1-MME interface

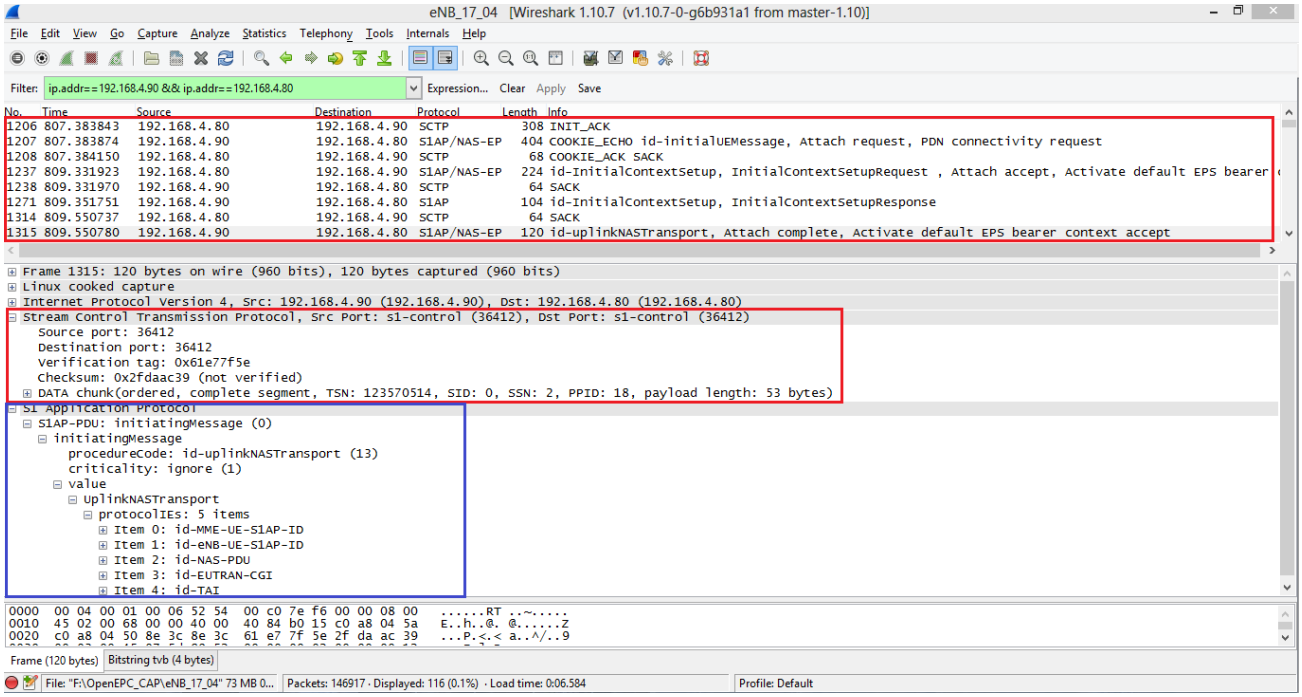



**Figure 5-26:** S1-MME interface

In Figure 5-25 there are presented two *Initial Context Setup* procedures. These are in accordance with the 3GPP specification [78]. The Stream Control Transmission Protocol (SCTP) Destination Port Number assigned by Internet Assigned Numbers Authority (IANA) for *S1-AP* is 36412.

In Figure 5-26 a Wireshark capture was analyzed, in which the SCTP protocol is used as Layer 4 and S1-AP as Layer 5. The *S1-AP* provides provides the signalling service between Evolved UMTS Terrestrial Radio Access Network (E-UTRAN) and the EPC.

### 5-4-0-6 Analysis

In contrast to TCP, the SCTP is a multi-streaming protocol which allows data to be delivered in multiple, independent streams, so that if there is data loss in one stream, delivery will not be affected by other streams. The sequence preservation is essential for messages which affect the same resource.

Other *Wireshark* captures with *OpenEPC* tool for both control (GPRS tunneling protocol version 2 (GTPv2)) and user plane (GPRS Tunneling Protocol (GTP)) are presented in Appendix B.

# Chapter 6

# Conclusions and Future Work

Software Defined Networking (SDN) and Network Function Virtualisation (NFV) are foreseen as revolutionary technologies in the standardized mobile network architecture. The mobile operators' concerns regarding the 3rd Generation Partnership Project (3GPP) short-term approaches are driven by issues related to cost, network overprovisioning, complexity and limited performances. On the other hand, SDN and NFV could bring flexibility in controlling architecture components, smart usage of the network resources (e.g. load balancing) and intelligent traffic steering (e.g. Quality of Service (QoS)), while decreasing the Capital Expenditure (CAPEX) and Operating Expenditure (OPEX) costs.

One of the main questions addressed by this work was to investigate which parts of the mobile network can be virtualized and how can SDN technology be smoothly integrated into the legacy network. Due to the high increase in mobile traffic that might lead to inefficient load distribution, and because of its flat Internet Protocol (IP) architecture, the Evolved Packet Core (EPC) emerged as a suitable candidate for the deployment of NFV and SDN technologies.

This thesis presented a new approach that might help mobile operators to adopt the optimal strategy in integrating SDN and NFV in their network deployments. In this new NFV / SDN context, the mobile core network can be transformed by employing a more flexible separation between control and user plane. This can be achieved throughout network virtualisation of the control functions in the current EPC such as: Mobility Management Entity (MME) and S/PGW Control Plane (S/PGW-C), while eliminating the GPRS Tunneling Protocol User Plane (GTP-U) tunneling. As a result, the user plane can be replaced by OpenFlow switches which are managed by a central entity, called the SDN controller. Therefore, the SDN might provide the infrastructure upon which NFV can run.

The main SDN controller could be responsible to proactively install flows in the switches and take routing decisions based on session type, as well as to equally distribute the load in the network. On the other hand, another Local Controller (LC) can be placed at the access level, connected to Evolved Node B (eNB) that is in charge of aggregating flows and to form the Location IP (LocIP). This is especially required when the user changes its current position

from one Base Station (BS) to another. The mobility handling is a critical characteristic of mobile networks which has been carefully addressed under two procedures: *UE Triggered Request* and *Network Triggered Request*.

The proposed architecture is different from the other current solutions, as it preservers the control plane functionality, while it decreases the complexity introduced by Tunnel Endpoint Identifier (TEID)s in the user plane. Nevertheless, there are several proposals to implement GTP-U in the Open vSwitch, as the current OpenFlow switches only support Generic Routing Encapsulation (GRE), Virtual Extensible LAN (VXLAN) or Multiprotocol Label Switching (MPLS). The cumbersome and complex implementation of such solutions is not compensated by better performances. On the contrary, eliminating GPRS Tunneling Protocol (GTP) in the user plane will reduce the overload in the packet header while decreasing the processing time in the switches. Another reason for not using encapsulation / decapsulation in the user plane refers to the forwarding continuity in the transport network that currently uses MPLS. Managing MPLS tunnels by a central controller has been already considered an early use case for the SDN deployment.

In the existing literature, other papers tend to focus on the GTP encapsulation or completely change both user and control plane. However, this new architecture preserves the most tedious messages exchange on *S1-MME* between eNB and MME, mostly serving to the user authentication which is very sensitive procedure in mobile networks. The proposed EPC SDN-based architecture targets to simplify the current EPC network, while introducing better traffic management and QoS policies.

Apart from the cost reduction, the most attractive advantages of NFV and SDN technologies are the flexibility in traffic management and congestion mechanisms, which current 3GPP architecture is lacking of, or require very long standardization process. In the proposed architecture, the traffic can be handled at two levels in the network: at the level of the LC for groups of Evolved Node B (eNodeB)s, or it can be performed by the central SDN controller in a round-robin fashion. Moreover, the traffic steering on the service chaining might be easier done by employing policies dependent on the session type (e.g. Hypertext Transfer Protocol (HTTP), Voice over IP (VoIP) and video) with different bandwidth requirements.

The presented work also involved the validation of the load balancing and QoS mechanisms. The load balancing and QoS modules were implemented on the top of the main SDN controller as an application layer that communicates with the controller on the *NorthBound* interface. The two use cases were validated and presented in Chapter 5. The obtained results revealed that parameters such as loss, packet delay variation and throughput are within the requirements corresponding to the employed session types. Nevertheless, the latency in the controller message processing introduced by the simulation tool (Mininet) might affect the network performance. This can be eliminated by repeating the experiment on physical switches according to the proposed topology.

Additionally, two SDN controllers features were investigated (i.e. Floodlight and OpenDaylight presented in Chapter 4) by comparing their performances on the load balancing use case in the proposed topology. The results were similar for a relative small number of flows, with a slightly better improvement for the OpenDaylight controller. Nevertheless, when the two controllers are overstressed with millions of flows, Floodlight behaves better, whereas OpenDaylight is not able to process all the coming messages.

Still in its early stages, open-source SDN controllers can be far more better optimized, but this would require lots of effort on the programming side. However, any enhancement is possible, especially because of the software elasticity, which allows to meet any particular requirement.

Due to their high deployment flexibility, it can be concluded that NFV and SDN are very attractive technologies for new mobile operators on the market. From a major traditional operator perspective, adopting such a proposal would involve removing legacy hardware and adopt a very well-defined strategy. By carefully following several transition phases in their deployment, operators should be able to fully take advantage of the aforementioned benefits.

## 6-1 Future Work

This project aimed to focus, apart from the design of a new EPC SDN-based architecture, on the validation of the load balancing and QoS use cases. Nevertheless, these algorithms can be improved / optimized, for example, by configuring for each defined queue on the switch ports a different weight. This can be translated in a weighted-fair-queuing scheduler which dynamically adjusts the weights based on the flow size. Moreover, the shortest path module in the controller should be changed to dynamically assign weights on the links based on the load information that it constantly receives from the switches. This would lead to a predefined path with proper bandwidth adjustment for each flow according to its size. A similar approach to this proposal is the Global First Fit scheduler presented in paper [44].

Another important requirement for this architecture is to deploy the full EPC control functionality into the cloud (i.e. OpenStack integration) or in the case of standalone elements to build the Open Application Programming Interface (API)s on *Northbound* interface. For instance, running standalone MME would involve to actually build the *Representational State Transfer (REST) API* interface between MME and the central controller. Since the proposed architecture implies to preserve the *S1-MME* interface between MME and the central SDN controller, the possibility of replacing it with the OpenFlow protocol should also be analyzed. Therefore, the benefits of a new control plane with all the EPC elements running on the top of the controller could be investigated in a further work. A first step in this direction has already been done by comparing the functionality of the two architectures for both control and user plane in a joined work with Telecom Bretagne University during the three months internship in France.

A more careful assessment of how to integrate NFV in the SDN architecture could also be investigated. One of the exploring directions might be the OpenStack integration with both OpenDaylight and Floodlight controllers. This can be either deployed in an external cloud, or running on standalone platforms in operators' data centers. Other upcoming technologies such as Internet Protocol version 6 (IPv6), which will definitively challenge mobile operators' deploying strategies can be better supported and more easily configured in the SDN framework. In this manner, each flow has a certain *flow ID* assigned that simplifies the forwarding process at the switch level. In this manner, we could identify that particular flow in the network, or aggregate multiple flows in a single bundle. This can constitute as well the premises of a future work in this domain.

In order to gather a better insight of the marketing aspects that come along with these new technologies, a product survey must be conducted. This should imply careful assessment of

different vendors products (OpenFlow switches) or software solutions. A useful conclusion to be reached should be, whether it is more cost-effective for operators to buy full solutions, accommodate only some features to their needs, or choose to invest in the OpenFlow hardware. This would imply to implement their own SDN controllers on specific servers and adopt the cloud virtualisation. This survey is meant to be carried out in a different project.

In order to have a complete solution which might actually be adopted by mobile operators, it is very important to consider all the aforementioned aspects. Nevertheless, this thesis primary goal was to envision a first approach on the way to NFV / SDN upcoming technologies.

# Appendix A

# Appendix A

## A-1 Mininet

### A-1-1 Topology

#### A-1-1-1 Topology.py

```
#!/usr/bin/python

# usage: mn --custom <path to mesh.py> --topo mesh ...

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class FatTreeTopo(Topo):
        def __init__(self,hosts_per = 5, **opts):
            Topo.__init__(self, **opts)
            hnum = 0
            snum = 0
            tor = 8
            agg = 4
            core = 2
            c_sw = []
            a_sw = []
            t_sw = []
       link1 = dict(bw=2000)
       link2 = dict(bw=250)
       link3 = dict(bw=50)
            # add core switches
            for i in range(core):
```

```
27                  snum += 1
28                  core_sw = self.addSwitch('s%s' %(snum))
29                  c_sw.append(core_sw)
30
31              for i in range(agg):
32                  snum += 1
33                  agg_sw = self.addSwitch('s%s' %(snum))
34                  a_sw.append(agg_sw)
35                  for rhs in c_sw:
36                      self.addLink(agg_sw, rhs)
37
38              for i in range(tor):
39                  snum += 1
40                  tor_sw = self.addSwitch('s%s' %(snum))
41                  t_sw.append(tor_sw)
42                  for rhs in a_sw:
43                      self.addLink(tor_sw, rhs)
44                  for _ in range(hosts_per):
45                      hnum += 1
46                      host = self.addHost('h%s' % hnum)
47                      self.addLink(tor_sw, host)
48
49              for sw in a_sw:
50                  hnum += 1
51                  host = self.addHost('h%s' % hnum)
52                  self.addLink(host,sw)
53
54              for sw in c_sw:
55                  hnum += 1
56                  host = self.addHost('h%s' % hnum)
57                  self.addLink(host,sw)
58
59  topos = { 'FatTree': FatTreeTopo }
```

## A-1-2 Add queues on switch ports

### A-1-2-1 Mininet_Add_Queues.py

```
1  #! /usr/bin/python
2  #  coding: utf-8
3
4  '''
5  Add queues to Mininet using ovs-vsctl and ovs-ofctl
6  @Author Ryan Wallner
7  '''
8
9  import os
10 import sys
11 import time
12 import subprocess
13
14 def find_all(a_str, sub_str):
```

```
15      start = 0
16      b_starts = []
17      while True:
18          start = a_str.find(sub_str, start)
19          if start == -1: return b_starts
20          #print start
21          b_starts.append(start)
22          start += 1
23
24
25  if os.getuid() != 0:
26          print "Root permissions required"
27          exit()
28
29
30  cmd = "ovs-vsctl show"
31  p = os.popen(cmd).read()
32  #print p
33
34  brdgs = find_all(p, "Bridge")
35
36
37  switches = []
38  count = 0
39  for bn in brdgs:
40      endsw = p.find("\"",bn+8)
41      #sw =  p[(bn+8):(bn+10)]
42      sw =  p[(bn+8):endsw]
43      switches.append(sw)
44  print switches
45
46  ports = find_all(p,"Port")
47
48  prts = []
49  for prt in ports:
50          endpt = p.find("\"",prt+6)
51          prt = p[(prt+6):endpt]
52          if '"' not in prt:
53                  print prt
54                  prts.append(prt)
55  config_strings = {}
56  for i in range(len(switches)):
57          str = ""
58          sw = switches[i]
59          for n in range(len(prts)):
60                  #verify correct order
61                  if switches[i] in prts[n]:
62                          #print switches[i]
63                          #print prts[n]
64                          port_name = prts[n]
65                          str = str+" -- set port %s qos=@defaultqos" %
                                port_name
66          config_strings[sw] = str
```

```
67
68 #build queues per sw
69 print config_strings
70 for sw in switches:
71         queuecmd = "sudo ovs-vsctl %s -- --id=@defaultqos create qos type
               =linux-htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1
               ,2=@q2 -- --id=@q0 create queue other-config:min-rate=20000000
                other-config:max-rate=20000000 -- --id=@q1 create queue other
               -config:max-rate=8000000 other-config:min-rate=8000000 -- --id
               =@q2 create queue other-config:max-rate=6000000 other-config:
               min-rate=6000000" % config_strings[sw]
72         q_res = os.popen(queuecmd).read()
73         print q_res
```

## A-2 QoS module

### A-2-1 Scenario 1

#### A-2-1-1 Add_Custom_Policies_Scenario1.py

```
1  #! /usr/bin/python
2  #  coding: utf-8
3
4  import os
5  import sys
6  import time
7  import subprocess
8
9  # 31/05/2014 - Scenario1 script with all policies
10
11 if os.getuid() != 0:
12         print "Root permissions required"
13         exit()
14
15 ip_server = "10.0.0.45"
16
17 sleep_time = 15
18
19 hosts1 = range(1,40,5)
20 hosts2 = range(2,40,5)
21 rest =  list(set(range(1,41))-set(hosts1)-set(hosts2))
22 print hosts1
23 print hosts2
24 print rest
25
26 ip_hosts1=[]
27 for h in range(len(hosts1)):
28     ip_hosts1.append("10.0.0."+str(hosts1[h]))
29 print ip_hosts1
30
31 ip_hosts2=[]
```

```
32 for h in range(len(hosts2)):
33     ip_hosts2.append("10.0.0."+str(hosts2[h]))
34 print ip_hosts2
35
36 ip_rest=[]
37 for h in range(len(rest)):
38     ip_rest.append("10.0.0."+str(rest[h]))
39 print ip_rest
40 print len(ip_rest)
41
42 dscp = "34"
43 queue = "0"
44 for h in range(len(ip_hosts1)):
45     cmd = "./qospath.py add --qos-path path_"+ip_hosts1[h]+" "+ip_hosts1[
          h]+" "+ip_server+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800
          \",\"protocol\":\"6\",\"queue\":\""+queue+"\",\"priority
          \":\"32000\"}\' 127.0.0.1 8080"
46     print "Adding: ",cmd
47     subprocess.Popen(cmd,shell=True).wait()
48     time.sleep(sleep_time)
49
50 dscp = "46"
51 queue = "1"
52 for h in range(len(ip_hosts2)):
53     cmd = "./qospath.py add --qos-path path"+ip_hosts2[h]+" "+ip_hosts2[h
          ]+" "+ip_server+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800
          \",\"protocol\":\"6\",\"queue\":\""+queue+"\",\"priority
          \":\"32767\"}\' 127.0.0.1 8080"
54     print "Adding: ",cmd
55     subprocess.Popen(cmd,shell=True).wait()
56     time.sleep(sleep_time)
57
58 dscp = "0"
59 queue = "2"
60 for h in range(len(ip_rest)):
61     cmd = "./qospath.py add --qos-path path"+ip_rest[h]+" "+ip_rest[h]+"
          "+ip_server+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800\",\"
          protocol\":\"6\",\"queue\":\""+queue+"\",\"priority\":\"30000\"}\'
           127.0.0.1 8080"
62     print "Adding: ",cmd
63     subprocess.Popen(cmd,shell=True).wait()
64     time.sleep(sleep_time)
```

## A-2-2 Scenario 2

### A-2-2-1 Round Robin Host Type – Scenario2_Host_Type.py

```
1 #! /usr/bin/python
2 #  coding: utf-8
3
4 import os
5 import sys
```

```
6  import time
7  import subprocess
8
9
10 if os.getuid() != 0:
11         print "Root permissions required"
12         exit()
13
14 ip_server = ["10.0.0.45","10.0.0.46"]
15 policy_counter = 0
16 sleep_time = 15
17
18 host1_ip = "10.0.0.1"
19 host2_ip = "10.0.0.2"
20 host3_ip = "10.0.0.3"
21 host4_ip = "10.0.0.4"
22 host5_ip = "10.0.0.5"
23
24 # traffic type : 0 - video; 1 - voice; 2 - http
25 host1_traffic = [0,1,2];
26 host2_traffic = [1,2];
27 host3_traffic = [2,2];
28 host4_traffic = [2,2];
29 host5_traffic = [2,2];
30
31 traffic=[];
32 traffic.append({'dscp':'34','queue':'0','priority':'32000'})
33 traffic.append({'dscp':'46','queue':'1','priority':'32767'})
34 traffic.append({'dscp':'0','queue':'2','priority':'30000'})
35
36 for h in range(len(host1_traffic)):
37     cmd = "./qospath.py add --qos-path path"+host1_ip+"-"+ip_server[
          policy_counter % 2]+str(h)+" "+host1_ip+" "+ip_server[
          policy_counter % 2]+" \'{\"tos\":\""+traffic[host1_traffic[h]]['
          dscp']+"\",\"eth-type\":\"0x0800\",\"protocol\":\"6\",\"queue\":\"
          "+traffic[host1_traffic[h]]['queue']+"\",\"priority\":\""+traffic[
          host1_traffic[h]]['priority']+"\"}\' 127.0.0.1 8080"
38     print "Adding policy ",policy_counter," : ",cmd
39     policy_counter = policy_counter + 1
40     subprocess.Popen(cmd,shell=True).wait()
41     time.sleep(sleep_time)
42
43 for h in range(len(host2_traffic)):
44     cmd = "./qospath.py add --qos-path path"+host2_ip+"-"+ip_server[
          policy_counter % 2]+str(h)+" "+host2_ip+" "+ip_server[
          policy_counter % 2]+" \'{\"tos\":\""+traffic[host2_traffic[h]]['
          dscp']+"\",\"eth-type\":\"0x0800\",\"protocol\":\"6\",\"queue\":\"
          "+traffic[host2_traffic[h]]['queue']+"\",\"priority\":\""+traffic[
          host2_traffic[h]]['priority']+"\"}\' 127.0.0.1 8080"
45     print "Adding policy ",policy_counter," : ",cmd
46     policy_counter = policy_counter + 1
47     subprocess.Popen(cmd,shell=True).wait()
48     time.sleep(sleep_time)
```

```
49
50 for h in range(len(host3_traffic)):
51     cmd = "./qospath.py add --qos-path path"+host3_ip+"-"+ip_server[
           policy_counter % 2]+str(h)+" "+host1_ip+" "+ip_server[
           policy_counter % 2]+" \'{\"tos\":\""+traffic[host3_traffic[h]]['
           dscp']+"\",\"eth-type\":\"0x0800\",\"protocol\":\"6\",\"queue\":\"
           "+traffic[host3_traffic[h]]['queue']+"\",\"priority\":\""+traffic[
           host3_traffic[h]]['priority']+"\"}\' 127.0.0.1 8080"
52     print "Adding policy ",policy_counter," : ",cmd
53     policy_counter = policy_counter + 1
54     subprocess.Popen(cmd,shell=True).wait()
55     time.sleep(sleep_time)
56
57 for h in range(len(host4_traffic)):
58     cmd = "./qospath.py add --qos-path path"+host4_ip+"-"+ip_server[
           policy_counter % 2]+str(h)+" "+host4_ip+" "+ip_server[
           policy_counter % 2]+" \'{\"tos\":\""+traffic[host4_traffic[h]]['
           dscp']+"\",\"eth-type\":\"0x0800\",\"protocol\":\"6\",\"queue\":\"
           "+traffic[host4_traffic[h]]['queue']+"\",\"priority\":\""+traffic[
           host4_traffic[h]]['priority']+"\"}\' 127.0.0.1 8080"
59     print "Adding policy ",policy_counter," : ",cmd
60     policy_counter = policy_counter + 1
61     subprocess.Popen(cmd,shell=True).wait()
62     time.sleep(sleep_time)
63
64 for h in range(len(host5_traffic)):
65     cmd = "./qospath.py add --qos-path path"+host5_ip+"-"+ip_server[
           policy_counter % 2]+str(h)+" "+host5_ip+" "+ip_server[
           policy_counter % 2]+" \'{\"tos\":\""+traffic[host5_traffic[h]]['
           dscp']+"\",\"eth-type\":\"0x0800\",\"protocol\":\"6\",\"queue\":\"
           "+traffic[host5_traffic[h]]['queue']+"\",\"priority\":\""+traffic[
           host5_traffic[h]]['priority']+"\"}\' 127.0.0.1 8080"
66     print "Adding policy ",policy_counter," : ",cmd
67     policy_counter = policy_counter + 1
68     subprocess.Popen(cmd,shell=True).wait()
69     time.sleep(sleep_time)
```

### A-2-2-2 Round Robin Session Type – Scenario2_Session_Type.py

```
1  #! /usr/bin/python
2  #  coding: utf-8
3
4  import os
5  import sys
6  import time
7  import subprocess
8
9
10 if os.getuid() != 0:
11         print "Root permissions required"
12         exit()
13
14 ip_server = ["10.0.0.45","10.0.0.46"]
```

```
15 policy_counter = 0
16 sleep_time = 15;
17 hosts_video = range(1,2,1)
18 hosts_voice = range(1,3,1)
19 hosts_http  =  list(set(range(1,6)))
20 hosts_http.extend(list(set(range(3,6))))
21 print "hosts_video ",hosts_video
22 print "host_voice ",hosts_voice
23 print "hots_http ",hosts_http
24
25 ip_hosts_video=[]
26 for h in range(len(hosts_video)):
27     ip_hosts_video.append("10.0.0."+str(hosts_video[h]))
28 print ip_hosts_video
29
30 ip_hosts_voice=[]
31 for h in range(len(hosts_voice)):
32     ip_hosts_voice.append("10.0.0."+str(hosts_voice[h]))
33 print ip_hosts_voice
34
35 ip_hosts_http=[]
36 for h in range(len(hosts_http)):
37     ip_hosts_http.append("10.0.0."+str(hosts_http[h]))
38 print ip_hosts_http
39 print len(ip_hosts_http)
40
41 dscp = "34"
42 queue = "0"
43 for h in range(len(ip_hosts_video)):
44     cmd = "./qospath.py add --qos-path path"+ip_hosts_video[h]+ip_server[
          h % 2]+str(policy_counter)+" "+ip_hosts_video[h]+" "+ip_server[h %
           2]+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800\",\"protocol
          \":\"6\",\"queue\":\""+queue+"\",\"priority\":\"32000\"}\'
          127.0.0.1 8080"
45     print "Adding policy ",policy_counter," : ",cmd
46     policy_counter = policy_counter + 1
47     subprocess.Popen(cmd,shell=True).wait()
48     time.sleep(sleep_time)
49
50 dscp = "46"
51 queue = "1"
52 for h in range(len(ip_hosts_voice)):
53     cmd = "./qospath.py add --qos-path path_"+ip_hosts_voice[h]+ip_server
          [h % 2]+str(policy_counter)+" "+ip_hosts_voice[h]+" "+ip_server[h
          % 2]+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800\",\"protocol
          \":\"6\",\"queue\":\""+queue+"\",\"priority\":\"32767\"}\'
          127.0.0.1 8080"
54     print "Adding policy ",policy_counter," : ",cmd
55     policy_counter = policy_counter + 1
56     subprocess.Popen(cmd,shell=True).wait()
57     time.sleep(sleep_time)
58
59 dscp = "0"
```

```
queue = "2"
for h in range(len(ip_hosts_http)):
    cmd = "./qospath.py add --qos-path path"+ip_hosts_http[h]+ip_server[h
        % 2]+str(policy_counter)+" "+ip_hosts_http[h]+" "+ip_server[h %
        2]+" \'{\"tos\":\""+dscp+"\",\"eth-type\":\"0x0800\",\"protocol
        \":\"6\",\"queue\":\""+queue+"\",\"priority\":\"30000\"}\'
        127.0.0.1 8080"
    print "Adding policy ",policy_counter," : ",cmd
    policy_counter = policy_counter + 1
    subprocess.Popen(cmd,shell=True).wait()
    time.sleep(sleep_time)
```

## A-3 Load Balancing Module

### A-3-1 Floodlight Virtual IP (VIP) pool

```
#!/bin/sh

curl -X POST -d '{"id":"1","name":"vip1","protocol":"udp","address":"
    10.0.0.100","port":"15"}' http://localhost:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"1","name":"pool1","protocol":"udp","vip_id":"1"}'
     http://localhost:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"1","address":"10.0.0.1","port":"15","pool_id":"1"
    }' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"2","address":"10.0.0.2","port":"15","pool_id":"1"
    }' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"3","address":"10.0.0.6","port":"15","pool_id":"1"
    }' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"4","address":"10.0.0.7","port":"15","pool_id":"1"
    }' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"5","address":"10.0.0.11","port":"15","pool_id":"1
    "}' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"6","address":"10.0.0.12","port":"15","pool_id":"1
    "}' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"7","address":"10.0.0.16","port":"15","pool_id":"1
    "}' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"8","address":"10.0.0.17","port":"15","pool_id":"1
    "}' http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"2","name":"vip2","protocol":"tcp","address":"
    10.0.0.200","port":"20"}' http://localhost:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"2","name":"pool1","protocol":"tcp","vip_id":"2"}'
     http://localhost:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"9","address":"10.0.0.3","port":"20","pool_id":"2"
    }' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"10","address":"10.0.0.4","port":"20","pool_id":"2
    "}' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"11","address":"10.0.0.5","port":"20","pool_id":"2
    "}' http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"12","address":"10.0.0.8","port":"20","pool_id":"2
    "}' http://localhost:8080/quantum/v1.0/members/
```

```
20 curl -X POST -d '{"id":"13","address":"10.0.0.9","port":"20","pool_id":"2
       "}' http://localhost:8080/quantum/v1.0/members/
21 curl -X POST -d '{"id":"14","address":"10.0.0.10","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
22 curl -X POST -d '{"id":"15","address":"10.0.0.13","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
23 curl -X POST -d '{"id":"16","address":"10.0.0.14","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
24 curl -X POST -d '{"id":"17","address":"10.0.0.15","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
25 curl -X POST -d '{"id":"18","address":"10.0.0.18","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
26 curl -X POST -d '{"id":"19","address":"10.0.0.19","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
27 curl -X POST -d '{"id":"20","address":"10.0.0.20","port":"20","pool_id":"
       2"}' http://localhost:8080/quantum/v1.0/members/
```

### A-3-2 OpenDaylight VIP pool

```
1  #!/bin/sh
2
3  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/pool -d '{"name":"PoolRR1","lbmethod":"roundrobin"}'
4  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/pool -d '{"name":"PoolRR2","lbmethod":"roundrobin"}'
5
6  #pool 10.0.0.80 (udp)
7
8  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/vip -d '{"name":"VIP-RR1","ip":"10.0.0.80","protocol":"
       UDP","port":"5550","poolname":"PoolRR1"}'
9
10 curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/poolmember -d '{"name":"PM1","ip":"10.0.0.1","poolname"
       :"PoolRR1"}'
11
12 curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/poolmember -d '{"name":"PM2","ip":"10.0.0.2","poolname"
       :"PoolRR1"}'
13
14 curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
       default/create/poolmember -d '{"name":"PM3","ip":"10.0.0.6","poolname"
       :"PoolRR1"}'
15
16 curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
       type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
```

```
    default/create/poolmember -d '{"name":"PM4","ip":"10.0.0.7","poolname"
    :"PoolRR1"}'
17
18  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM5","ip":"10.0.0.11","poolname
    ":"PoolRR1"}'
19
20  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM6","ip":"10.0.0.12","poolname
    ":"PoolRR1"}'
21
22  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM7","ip":"10.0.0.16","poolname
    ":"PoolRR1"}'
23
24  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM8","ip":"10.0.0.17","poolname
    ":"PoolRR1"}'
25
26  #pool 10.0.0.90 (tcp)
27
28  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/vip -d '{"name":"VIP-RR2","ip":"10.0.0.90","protocol":"
    TCP","port":"5560","poolname":"PoolRR2"}'
29
30  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM9","ip":"10.0.0.3","poolname"
    :"PoolRR2"}'
31
32  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM10","ip":"10.0.0.4","poolname
    ":"PoolRR2"}'
33
34  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM11","ip":"10.0.0.5","poolname
    ":"PoolRR2"}'
35
36  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM12","ip":"10.0.0.8","poolname
    ":"PoolRR2"}'
37
38  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
```

```
    default/create/poolmember -d '{"name":"PM13","ip":"10.0.0.9","poolname
    ":"PoolRR2"}'
39
40  curl --user "admin":"admin" -H "Accept: application/json" -H "Content-
    type: application/json" -X POST http://127.0.0.1:8080/one/nb/v2/lb/
    default/create/poolmember -d '{"name":"PM14","ip":"10.0.0.10","
    poolname":"PoolRR2"}'
```

## A-4 Plot Results

### A-4-1 Scenario 1

```
1  clear all
2  clc
3  close all
4
5  path = 'C:\Users\Oana\Documents\MATLAB\OanaScripts';
6  fileudp = 'scenario12udp.iperf';
7  filetcp = 'scenario12tcp.iperf';
8
9  fudp = fopen([path '/' fileudp]);
10 ftcp = fopen([path '/' filetcp]);
11
12 h1 = [];
13 h2 = [];
14 h3 = [];
15 h4 = [];
16 h5 = [];
17 % set here ID H1
18 h1id = 5;
19 % set here ID H2
20 h2id = 4;
21 % set here ID H3
22 h3id = 5;
23 % set here ID H4
24 h4id = 7;
25 % set here ID H5
26 h5id = 6;
27
28
29 counterh1 = 1;
30 counterh2 = 1;
31 counterh3 = 1;
32 counterh4 = 1;
33 counterh5 = 1;
34
35 %% PARSE UDP FILE
36 tline = fgets(fudp);
37 while ischar(tline)
38     if tline(1)=='[' & tline(end-1)==')'
39         %    find id
40         st1 = findstr('[',tline);
```

```
41          en1 = findstr(']',tline);
42          id = strtrim(tline(st1+1:en1-1));
43          %       disp(id)
44          %       find time interval
45          st1 = findstr(']',tline);
46          en1 = findstr('sec',tline);
47          interv = strsplit(strtrim(tline(st1+1:en1-1)),'-');
48          interv(1) = (strtrim(interv(1)));
49          interv(2) = (strtrim(interv(2)));
50          %       disp(interv)
51          %       find transfer
52          st1 = findstr('sec',tline);
53          en1 = findstr('Bytes',tline);
54          transfer = strtrim(tline(st1+3:en1-1));
55          if transfer(end)=='M'
56              transfer = str2num(strtrim(transfer(1:end-1)))*1024;
57          else
58              transfer = str2num(strtrim(transfer(1:end-1)));
59          end
60          %       disp(transfer)
61          %      find bandwidth
62          st1 = findstr('Bytes',tline);
63          en1 = findstr('bits/',tline);
64          bandwidth = strtrim(tline(st1+6:en1-2));
65          if tline(en1-1)=='K'
66              bandwidth = str2num(bandwidth)/1024;
67          else
68              bandwidth = str2num(bandwidth);
69          end
70
71          %       find jitter
72          st1 = findstr('/sec',tline);
73          en1 = findstr('ms',tline);
74          jitter = strtrim(tline(st1+5:en1-1));
75          %       disp(jitter)
76          %       find loss
77          st1 = findstr('(',tline);
78          en1 = findstr(')',tline);
79          loss = strtrim(tline(st1+1:en1-2));
80          %       disp(loss)
81          if h1id == str2num(id)
82              disp('h1')
83              disp(tline)
84              h1.id(counterh1)=str2num(id);
85              h1.interval(counterh1,1)=str2num(interv{1});
86              h1.interval(counterh1,2)=str2num(interv{2});
87              h1.transfer(counterh1)=transfer;
88              h1.bandwidth(counterh1)=bandwidth;
89              h1.jitter(counterh1)=str2num(jitter);
90              h1.loss(counterh1)=str2num(loss);
91              counterh1 = counterh1+1;
92          end
93          if h2id == str2num(id)
```

```
94              disp('h2')
95              disp(tline)
96              h2.id(counterh2)=str2num(id);
97              h2.interval(counterh2,1)=str2num(interv{1});
98              h2.interval(counterh2,2)=str2num(interv{2});
99              h2.transfer(counterh2)=transfer;
100             h2.bandwidth(counterh2)=bandwidth;
101             h2.jitter(counterh2)=str2num(jitter);
102             h2.loss(counterh2)=str2num(loss);
103             counterh2 = counterh2+1;
104         end
105     else
106         disp(['$$$$$$ ' tline]);
107     end
108     tline = fgets(fudp);
109 end
110 %% PARSE TCP FILE
111 tline = fgets(ftcp);
112 while ischar(tline)
113     if tline(1)=='[' & tline(end-4:end-1)=='/sec'
114         %find id
115         st1 = findstr('[',tline);
116         en1 = findstr(']',tline);
117         id = strtrim(tline(st1+1:en1-1));
118         %         disp(id)
119         % find time interval
120         st1 = findstr(']',tline);
121         en1 = findstr('sec',tline);
122         interv = strsplit(strtrim(tline(st1+1:en1-1)),'-');
123         interv(1) = (strtrim(interv(1)));
124         interv(2) = (strtrim(interv(2)));
125         %         disp(interv)
126         % find transfer
127         st1 = findstr('sec',tline);
128         en1 = findstr('Bytes',tline);
129         transfer = strtrim(tline(st1+3:en1-1));
130         if transfer(end)=='M'
131             transfer = str2num(strtrim(transfer(1:end-1)))*1024;
132         else
133             transfer = str2num(strtrim(transfer(1:end-1)));
134         end
135         %         disp(transfer)
136         % find bandwidth
137         st1 = findstr('Bytes',tline);
138         en1 = findstr('bits/',tline);
139         bandwidth = strtrim(tline(st1+6:en1-2));
140         if tline(en1-1)=='K'
141             bandwidth = str2num(bandwidth)/1024;
142         else
143             bandwidth = str2num(bandwidth);
144         end
145         if h3id == str2num(id)
146             disp('h3')
```

```
147             disp(tline)
148             h3.id(counterh3)=str2num(id);
149             h3.interval(counterh3,1)=str2num(interv{1});
150             h3.interval(counterh3,2)=str2num(interv{2});
151             h3.transfer(counterh3)=transfer;
152             h3.bandwidth(counterh3)=bandwidth;
153
154             counterh3 = counterh3+1;
155         end
156         if h4id == str2num(id)
157             disp('h4')
158             disp(tline)
159             h4.id(counterh4)=str2num(id);
160             h4.interval(counterh4,1)=str2num(interv{1});
161             h4.interval(counterh4,2)=str2num(interv{2});
162             h4.transfer(counterh4)=transfer;
163             h4.bandwidth(counterh4)=bandwidth;
164             counterh4 = counterh4+1;
165         end
166         if h5id == str2num(id)
167             disp('h5')
168             disp(tline)
169             h5.id(counterh5)=str2num(id);
170             h5.interval(counterh5,1)=str2num(interv{1});
171             h5.interval(counterh5,2)=str2num(interv{2});
172             h5.transfer(counterh5)=transfer;
173             h5.bandwidth(counterh5)=bandwidth;
174             counterh5 = counterh5+1;
175         end
176     else
177         disp(['$$$$$$ ' tline]);
178     end
179     tline = fgets(ftcp);
180 end
181
182 fclose(fudp);
183 fclose(ftcp);
184
185 %% PLOT RESULTS
186 %plot bandwidth
187 f1 = figure;
188 hold on;
189 grid on;
190 p1 = semilogy((h1.interval(1:end-1,1)+h1.interval(1:end-1,2))/2,h1.
        bandwidth(1:end-1),'r','LineWidth',2);
191 p2 = semilogy((h2.interval(1:end-1,1)+h2.interval(1:end-1,2))/2,h2.
        bandwidth(1:end-1),'g','LineWidth',2);
192 p3 = semilogy((h3.interval(1:end-1,1)+h3.interval(1:end-1,2))/2,h3.
        bandwidth(1:end-1),'b','LineWidth',2);
193 p4 = semilogy((h4.interval(1:end-1,1)+h4.interval(1:end-1,2))/2,h4.
        bandwidth(1:end-1),'m','LineWidth',2);
194 p5 = semilogy((h5.interval(1:end-1,1)+h5.interval(1:end-1,2))/2,h5.
        bandwidth(1:end-1),'k','LineWidth',2);
```

```
195
196 %round-trip tcp before fragmentation
197 w1 = 128*10^3;
198 l1 = 8*10^3;
199 r1 = ((w1*l1)/max(h3.bandwidth(1:end-1)))/10^6;
200 r2 = ((w1*l1)/max(h4.bandwidth(1:end-1)))/10^6;
201 r3 = ((w1*l1)/max(h5.bandwidth(1:end-1)))/10^6;
202
203 r=[r1 r2 r3] % round-trip vector
204
205 p1=(1.5*l1^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r1^2));
206 p2=(1.5*l1^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r2^2));
207 p3=(1.5*l1^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r3^2));
208
209 p=[p1 p2 p3] %loss vector
210
211
212 %after fragmentation
213 w1f = 256*10^3;
214 l1f = 250;
215 r1f =(w1f*l1f^2)/max(h3.bandwidth(1:end-1))/10^6;
216 r2f = (w1f*l1f^2)/max(h4.bandwidth(1:end-1))/10^6;
217 r3f = (w1f*l1f^2)/max(h5.bandwidth(1:end-1))/10^6;
218
219 rf=[r1f r2f r3f]
220
221
222 p1f=(1.5*l1f^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r1f^2));
223 p2f=(1.5*l1f^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r2f^2));
224 p3f=(1.5*l1f^2)/(((mean(h3.bandwidth(1:end-1))*10^6)^2*r3f^2));
225
226 pf=[p1f p2f p3f]
227
228
229
230 hold off;
231 axis([0 60 0 25]);
232 title('Scenario 1: BANDWIDTH','FontSize',14);
233 legend('h1video','h2voice','h3http','h4http','h5http');
234 xlabel('Time [s]','FontSize', 14);
235 ylabel('Bandwidth [Mbps]','FontSize',14);
236
237 % plot loss
238 f2 = figure;
239 hold on;
240 grid on
241 p1 = plot((h1.interval(1:end-1,1)+h1.interval(1:end-1,2))/2,h1.loss(1:end
      -1),'r','LineWidth',2);
242 p2 = plot((h2.interval(1:end-1,1)+h2.interval(1:end-1,2))/2,h2.loss(1:end
      -1),'g','LineWidth',2);
243 hold off;
244 axis ([0 60 0 45]);
245 title('Scenario 1: LOSS','FontSize',14);
```

```
246 legend('h1video','h2voice');
247 xlabel('Time [s]','FontSize',14);
248 ylabel('Packet Loss','FontSize',14);
249
250 % plot jitter
251 f3 = figure;
252 hold on;
253 grid on
254 p1 = plot((h1.interval(1:end-1,1)+h1.interval(1:end-1,2))/2,h1.jitter(1:
    end-1),'r','LineWidth',2);
255 p2 = plot((h2.interval(1:end-1,1)+h2.interval(1:end-1,2))/2,h2.jitter(1:
    end-1),'g','LineWidth',2);
256 hold off;
257 axis ([0 60 0 2]);
258 title('Scenario 1: JITTER','FontSize',14);
259 legend('h1video','h2voice');
260 xlabel('Time [s]','FontSize',14);
261 ylabel('Packet Delay Variation [ms]','FontSize',14);
```

# Appendix B

# Appendix B

## B-1 Control Plane: S11 Interface

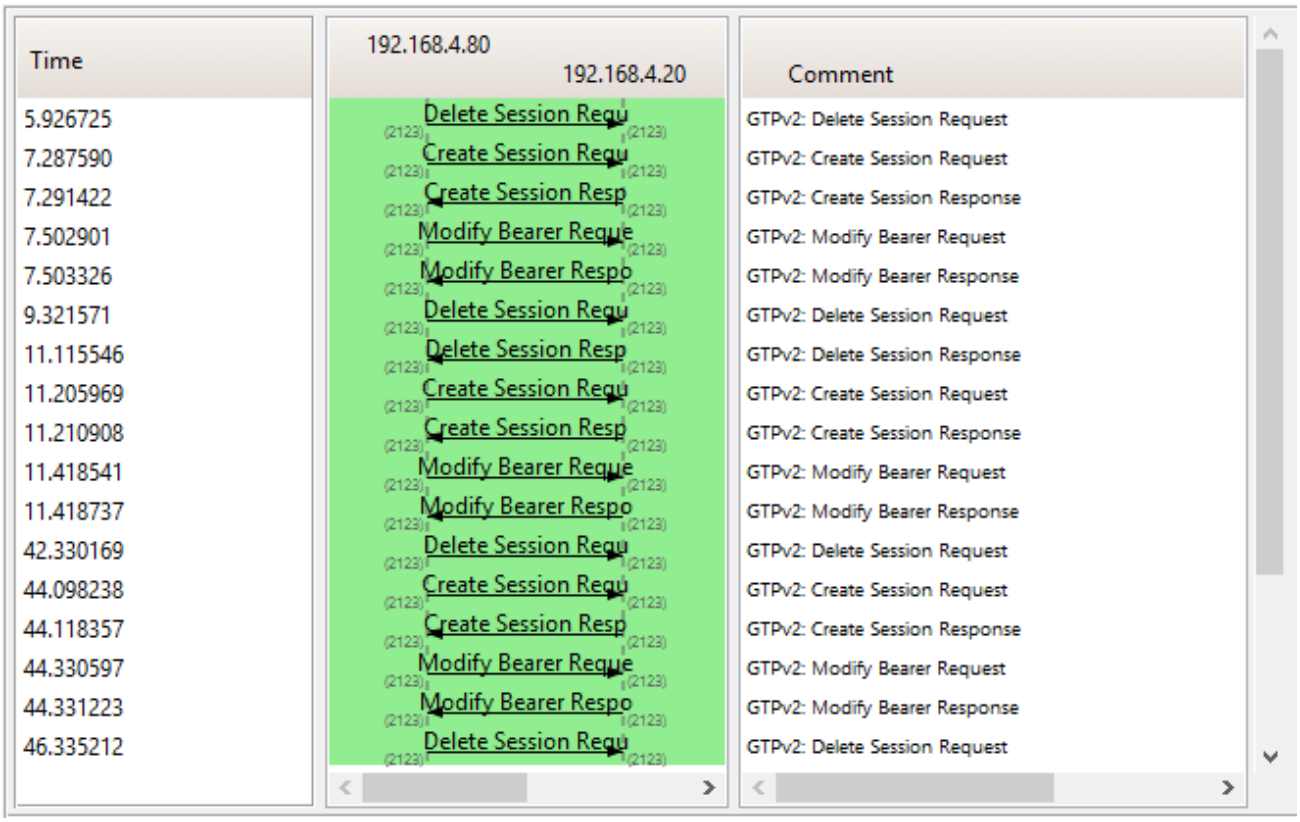



**Figure B-1:** S11 Interface (GTPv2)

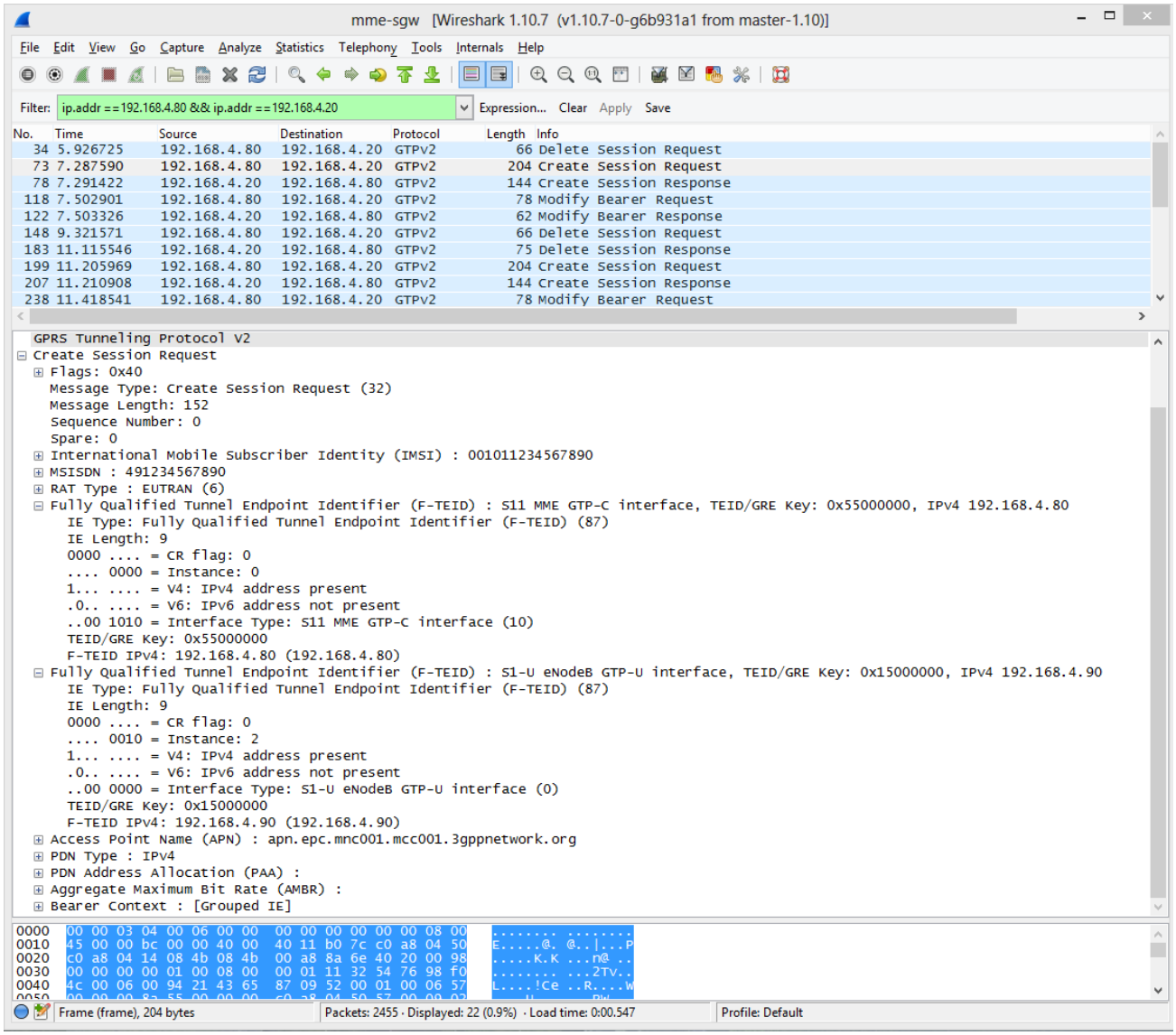



**Figure B-2:** S11 Interface (GTPv2)

## B-2 User Plane

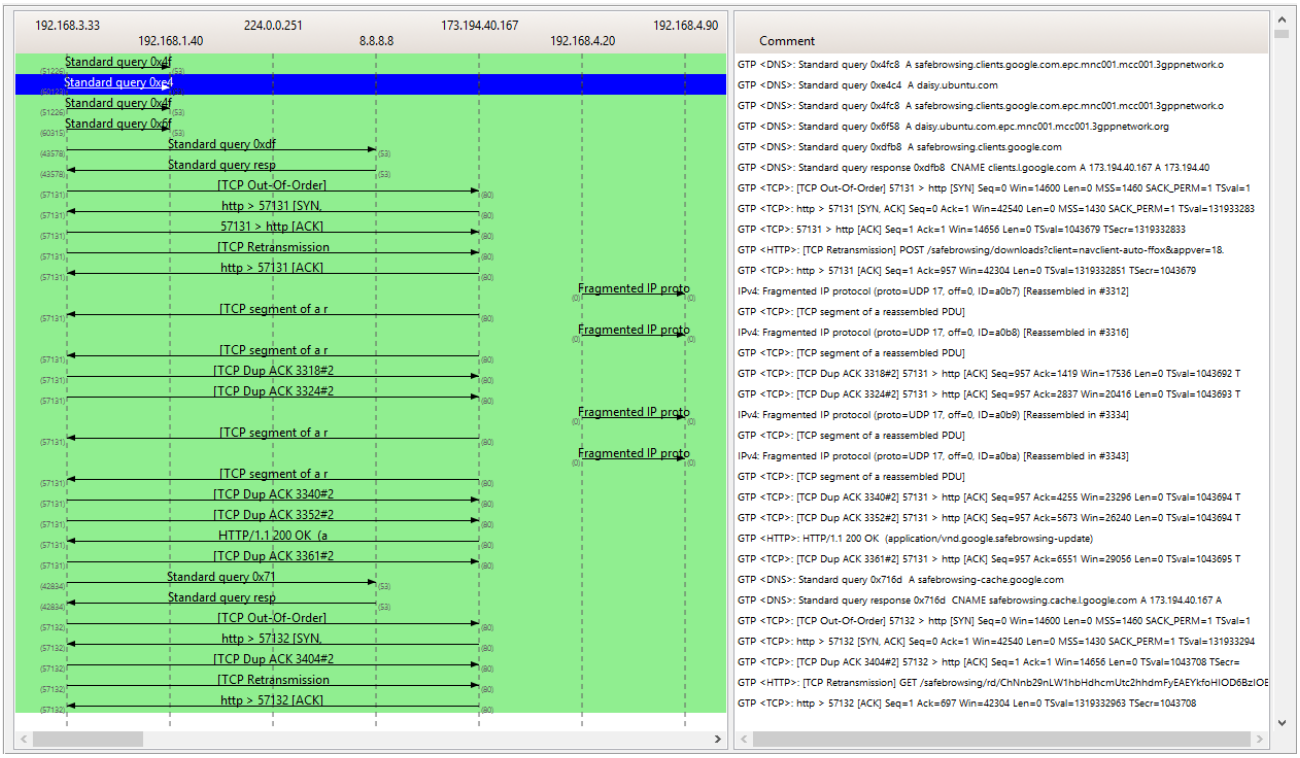



**Figure B-3:** GTP

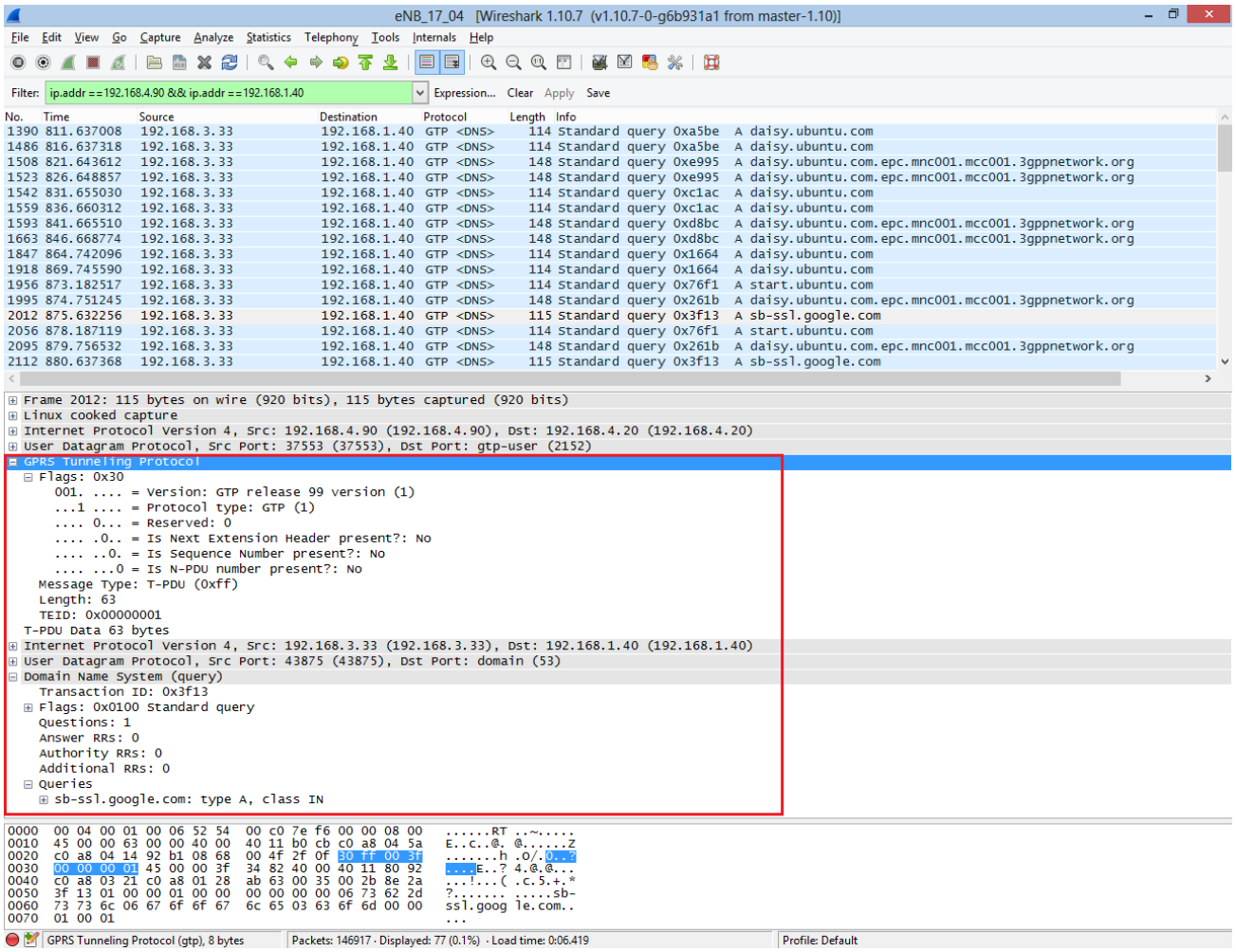



**Figure B-4:** GTP

# Bibliography


[1] 3GPP TS 23.401 V12.3.0 3rd Generation Partnership Project, "3rd Generation Partnership Project;*Technical Specification Group Services and System Aspects;General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (Release 12)*," December, 2013.

[2] "General Packet Radio System (GPRS) Tunneling Protocol User Plane (GTOv1-U)." 3GPP TS 29.281 version 11.6.0 Release 11, 2013.

[3] "3GPP Evolved Packet Packet System (EPC);Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control Plane (GTPv2-C)." 3GPP TS 29.274 version 10.9.0 Release 10, 2013.

[4] "TR 23.843 V12.0.0 3rd Generation Partnership Project;Technical Specification Group Services and System Aspects; Study on Core Network Overload (CNO) solutions (Release 12)," December 2013.

[5] ETSI GS NFV 001, "Network Functions Virtualisation (NFV); Use Cases," 2013.

[6] NFV White Paper, "Network Functions Virtualization: an Introduction, Benefits, Challenges and Call for Action," 2012.

[7] ONF, "OpenFlow-enabled Software Defined Networking (SDN) and Network Functions Virtualization," 2014.

[8] Open Networking Foundation (ONF), "OpenFlow Switch Specification, version v.1.4.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, 2013, [Online; accessed 24-May-2014].

[9] N. McKeown, "OpenFlow (Or: "Why can't I innovate in my wiring closet?")," http://cleanslate.stanford.edu, [Online; accessed 4-June-2014].

[10] T. Graf, "Underneath OpenStack Quantum: Software Defined Networking with Open vSwitch."

[11] Cisco, "Quality of Service - The Differentiated Services Model," [Online; accessed 6-June-2014].

[12] C. Mulligan, "EPC and 4G Packet Networks: Driving the Mobile Broadband Revolution." *Academic Press*, 2012.

[13] S. Chu, "vSphere Product Marketing," 2013.

[14] "OpenDaylight Project," http://www.opendaylight.org/project/technical-overview, [Online; accessed 24-May-2014].

[15] Ixia White Paper, "Quality of service (QoS) and policy management in mobile data networks – Validating Service Quality to ensure subscriber quality of experience (QoE)," 2013.

[16] IETF RFC 1349, "Type of Service in the Internet Protocol Suite," 1992.

[17] Cisco, "Cisco," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf, 2014, [Online; accessed 24-May-2014].

[18] NFV, "NFV," https://portal.etsi.org/tb.aspx?tbid=789&SubTB=789,795,796,801,800,798,799,797,802\, [Online; accessed 24-May-2014].

[19] "ETSI GS NFV 001 v1.1.1," October, 2013.

[20] White Paper, "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges and Call for Action."

[21] SearchSDN, "OpenFlow protocol guide: SDN controllers and apps," http://searchsdn.techtarget.com/guides/OpenFlow-protocol-tutorial-SDN-controllers-and-applications-emerge, [Online; accessed 24-May-2014].

[22] ONF, "Open Networking Foundation, Wireless and Mobile Working Group," https://www.opennetworking.org/images/stories/downloads/working-groups/charter-wireless-mobile.pdf, 2013, [Online; accessed 24-May-2014].

[23] J. Kempf, B. Johansson, S. Pettersson, H. Luning, and T. Nilsson, "Moving the mobile evolved packet core to the cloud," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on.* IEEE, 2012, pp. 784–791.

[24] B. Liu, "Software Defined Networking and Tunneling for Mobile Networks," 2013.

[25] M. S. G. Hampel and T.Bu, "Applying Software-Defined Networking to Telecom Domain." Computer Communications Workshops (INFOCOM WKSHPS), April, 2013.

[26] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, and E.-D. Schmidt, "A Virtual SDN-enabled LTE EPC Architecture: a case study for S-/P-Gateways functions," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for.* IEEE, 2013, pp. 1–7.

[27] P. Gurusanthosh, A. Rostami, and R. Manivasakan, "SDMA: A semi-distributed mobility anchoring in LTE networks," in *Mobile and Wireless Networking (MoWNeT), 2013 International Conference on Selected Topics in.* IEEE, 2013, pp. 133–139.

[28] S. B. H. Said, M. R. Sama, K. Guillouard, L. Suciu, G. Simon, X. Lagrange, and J.-M. Bonnin, "New control plane in 3GPP LTE/EPC architecture for on-demand connectivity service," in *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on.* IEEE, 2013, pp. 205–209.

[29] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: scalable and flexible cellular core network architecture," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* ACM, 2013, pp. 163–174.

[30] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *NSDI*, vol. 10, 2010, pp. 19–19.

[31] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila, "SDN Based Inter-Technology Load Balancing Leveraged by Flow Admission Control," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for.* IEEE, 2013, pp. 1–5.

[32] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* ACM, 2013, pp. 25–30.

[33] R. Krishnan, "Flow-aware Real-time SDN Analytics." Brocade Communications, 2014.

[34] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks," in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, 2012, pp. 1–8.

[35] R. Wallner and R. Cannistra, "An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller," *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, pp. 14–19, 2013.

[36] L. E. Li, Z. M. Mao, and J. Rexford, "CellSDN: Software-defined cellular networks," 2012.

[37] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Taking control of cellular core networks," 2013.

[38] ETSI TS 123 401, "LTE; Genaral Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) acess." 3GPP TS 23.401 version 11.9.0 Release 11, 2014.

[39] "GPRS Tunneling Protocol (GTP) in LTE," http://4g-lte-world.blogspot.nl/2013/03/gprs-tunneling-protocol-gtp-in-lte.html, [Online; accessed 24-May-2014].

[40] Sandvine, "Quality of Service in LTE," 2013.

[41] P. Mell and T. Grance, "The NIST Definition of Cloud Computing-Recommendations of the National Institute of Standards and Technology," 2011.

[42] R. P.-C. L. Badger, T.Grance and J.Voas, "DRAFT Cloud Computing Synopsis and Recommendations." NIST Special Publication 800-146, 2011.

[43] G. Ragoonanan, "Network Virtualisation opportunities for CSPS begin in the core of next-generation networks," http://www.analysysmason.com/About-Us/News/Insight/Network-virtualisation-CSPs-Oct2013-RMA16/, Oct. 2013, [Online; accessed 24-May-2014].

[44] ONF, "OpenFlow-enabled SDN and Network Function Virtualisation (NFV)," 2014.

[45] ONF White Paper, "Software-Defined Networking: The New Norm for Networks," 2012.

[46] B. Salisbury, "OpenFlow: Proactive vs Recative Flows," http://networkstatic.net/openflow-proactive-vs-reactive-flows/, 2013, [Online; accessed 24-May-2014].

[47] Open vSwitch, "Open Virtual Switch," http://openvswitch.org/, 2013, [Online; accessed 24-May-2014].

[48] RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," [Online; accessed 6-June-2014].

[49] IETF, RFC2598, "An Expedited Forwarding PHB," 1999.

[50] IETF, RFC2597, "Assured Forwarding PHB Group," 1999.

[51] T. Szigeti, C. Hattingh, R. Barton, and K. Briley, "End-To-End QoS Network Design: Quality of Service for Rich-Media and Cloud Networks." *Pearson Education*, 2013.

[52] ONF, "OpenFlow Configuration and Management Protocol," https://www.opennetworking.org/standards/of-config, 2013, [Online; accessed 24-May-2014].

[53] "OpenDaylight Integration Tests," https://wiki.opendaylight.org/index.php?title=GettingStarted:Performance_Integration_Tests&action=edit, 2013, [Online; accessed 24-May-2014].

[54] M. Weldon, "Alcatel-Lucent Cloud: NFV, the New Virtual Reality," http://fr.slideshare.net/Alcatel-Lucent/alcatel-lucentcloudcloud-bandnfvmarcusweldon, 2013, [Online; accessed 24-May-2014].

[55] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[56] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[57] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[58] "sFlow," http://blog.sflow.com/2013/01/load-balancing-lagecmp-groups.html/, [Online; accessed 24-May-2014].

[59] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," 2009.

[60] VMware, "Vmware workstation 10 documentation center," http://pubs.vmware.com/workstation-10/index.jsp#com.vmware.ws.using.doc/GUID-BAFA66C3-81F0-4FCA-84C4-D9F7D258A60A.html, 2013, [Online; accessed 24-May-2014].

[61] "Kernel Based Virtual Machine," http://www.linux-kvm.org/page/Main_Page, [Online; accessed 24-May-2014].

[62] "OpenEPC," http://www.openepc.net/index.html, 2012, [Online; accessed 24-May-2014].

[63] "Mininet," http://mininet.org/, [Online; accessed 24-May-2014].

[64] R. Nadiv and T. Naveh, "Wireless Backhaul Topologies: Analyzing Backhaul Topology Strategies, Ceragon Research," August 2010.

[65] M. Howard, "Using Carrier Ethernet to Backhaul LTE, Infonetics Research," February, 2011.

[66] "List of openflow software projects," http://yuba.stanford.edu/~casado/of-sw.html, [Online; accessed 29-May-2014].

[67] Project Floodlight, "Floodlight," http://www.projectfloodlight.org/floodlight/, [Online; accessed 24-May-2014].

[68] "How to implement quality of service using floodlight," http://www.openflowhub.org/display/floodlightcontroller/How+to+implement+Quality+Of+Service+using+Floodlight, [Online; accessed 3-June-2014].

[69] T. R.E, "Depth-first Search and Linear Graph Algorithms," 1972.

[70] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms. Section 24.3: Dijkstra's algorithm." MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7, 2001.

[71] P. van Mieghem, "Data Communications Networking." *Techne Press*, 2011.

[72] K. Salchow, "Load Balancing 101: Nuts and Bolts," http://www.f5.com/pdf/white-papers/load-balancing101-wp.pdf, [Online; accessed 26-May-2014].

[73] "Cbench," http://archive.openflow.org/wk/index.php/Oflops, [Online; accessed 26-May-2014].

[74] RFC 3550, " RTP: A Transport Protocol for Real-Time Applications," http://www.rfc-base.org/txt/rfc-3550.txt, [Online; accessed 30-June-2014].

[75] "Network-level Parameters," http://www.cs.unc.edu/~fhernand/diss-html/node25.html#SECTION00711100000000000000, [Online; accessed 5-June-2014].

[76] "Surfnet," http://www.surf.nl/en/about-surf/subsidiaries/surfnet, [Online; accessed 28-June-2014].

[77] ONF, "Openflow wireshark dissector," http://archive.openflow.org/wk/index.php/OpenFlow_Wireshark_Dissector, [Online; accessed 27-May-2014].

[78] 3GPP TS 36.413, "Evolved Universal Terrestrial Radio Acess Network (E-UTRAN); S1 Application Protocol (S1AP)," 2011.

# Glossary

## Acronyms

**AF** Assured Forwarding

**ALRP** Allocation and Retention Priority

**AMBR** Aggregate Maximum Bit Rate

**API** Application Programming Interface

**APN** Access Point Name

**ARP** Address Resolution Protocol

**BE** Best Effort

**BGP** Border Gateway Protocol

**BS** Base Station

**CAPEX** Capital Expenditure

**CDF** Charging Data Function

**CGF** Charging Gataway Function

**CLI** Command Line Interface

**CPU** Central Processing Unit

**CSP** Communications Service Providers

**DDoS** Distributed Denial of Service Attacks

**DFS** Depth First Search

**DHCP** Dynamic Host Configuration Protocol

**DiffServ** Differentiated Services

**DNS** Domain Name System

**DSCP** Differentiated Services Code Point

**ECGI** E-UTRAN Cell Global Identifier

**ECMP** Equal Cost Multipath

**ECM** EPS Connection Management

**EF** Expedited Forwarding

**EMM** EPS Mobility Management

**eNB** Evolved Node B

**eNodeB** Evolved Node B

**eNB-U** eNodeB User Plane

**EPC** Evolved Packet Core

**EPS** Evolved Packet System

**ETSI** European Telecommunications Standards Institute

**E-UTRAN** Evolved UMTS Terrestrial Radio Access Network

**FE** Forwarding Element

**4G** Fourth Generation

**GBR** Guaranteed Bit Rate

**3GPP** 3rd Generation Partnership Project

**GGSN** Gateway GPRS Support Node

**GLM** Gateway Load Manager

**GRE** Generic Routing Encapsulation

**GSM** Global System for Mobile Communications

**GPRS** General Packet Radio Service

**GTP** GPRS Tunneling Protocol

**GTP-C** GPRS Tunneling Protocol Control Plane

**GTP'** GPRS Tunneling Protocol Prime

**GTP-U** GPRS Tunneling Protocol User Plane

**GTPv2** GPRS tunneling protocol version 2

**2G** Second Generation

**3G** Third Generation

**GUI** Graphical User Interface

**GUTI** Globally Unique Temporary ID

**HSPA** High Speed Packet Access

**HSS** Home Subscriber Server

**HTTP** Hypertext Transfer Protocol

**IANA** Internet Assigned Numbers Authority

**IaaS** Infrastructure as a Service

**IMS** IP Multimedia System

**IMSI** International mobile Subscriber Identity

**IntServ** Integrated Services

**IP** Internet Protocol

**IPv6** Internet Protocol version 6

**ISG** Industry Specification Group

**JVM** Java Virtual Machine

**KVM** Kernel-based Virtual Machine

**LACP** Local Control and Accountability Plan

**LAN** Local Area Network

**LC** Local Controller

**LI** Lawful Interception

**LISP** Locator / ID Separation Protocol

**LocIP** Location IP

**LTE** Long-Term Evolution

**MAC** Media Access Control Address

**MBR** Maximum Bit Rate

**MME** Mobility Management Entity

**MPLS** Multiprotocol Label Switching

**NAS** Non Access Stratum

**NaaS** Network as a Service

**NAT** Network Address Translation

**NE** Networking Element

**NFV** Network Function Virtualisation

**NFV ISG** Network Functions Virtualisation Industry Specification Group

**NFVI** NFV infrastrucure

**NIST** National Institute of Standards and Technology

**NGMN** Next Generation Mobile Network

**NGNI** Next Generation Network Infrastructure

**OAM** Operations, Administration and Management

**OF** OpenFlow

**ONF** Open Networking Foundation

**OPEX** Operating Expenditure

**OS** Operating System

**OSGi** Open Service Gateway initiative

**OVSDB** Open vSwitch Database Management Protocol

**OVS** Open vSwitch

**PaaS** Platform as a Service

**PCEF** Policy Control Enforcement Function

**PCRF** Policy and Charging Rules Function

**PDN** Packet Data Networks

**PGW** Packet Data Network Gateway

**PGW-U** PDN Gateway User

**PLMN** Public Land Mobile Network

**PHB** Per-Hop Behavior

**P2P** Peer-to-Peer

**PoC** Proof of Concept

**PRB** Physical Resource Blocks

**QCI** QoS Class Indicator

**QoE** Quality of Experience

**QoS** Quality of Service

**RAM** Random-Access Memory

**RAT** Radio Access Transfer

**REST** Representational State Transfer

**RRC** Radio Resource Control

**RSPAN** Remote SPAN

**RTP** Real-time Transport Protocol

**SAE** System Architecture Evolution

**SAL** Service Abstraction Layer

**SaaS** Software as a Service

**SDN** Software Defined Networking

**SCTP** Stream Control Transmission Protocol

**SGSN** Serving GPRS Support Node

**SGW** Serving Gateway

**SGW-U** SGW User Plane

**SIM** Subscriber Identity Module

**SIP** Session Initiation Protocol

**SNMP** Simple Network Management Protocol

**S/PGW** Serving/Packet Data Network Gateway

**S/PGW-C** S/PGW Control Plane

**S/PGW-U** S/PGW User Plane

**SPAN** Switch Port Analyzer

**SPI** Security Parameter Index

**SSL** Secure Sockets Layer

**SSH** Secure Shell Protocol

**TA** Tracking Area

**TAI** Tracking Area Identify

**TAU** Tracking Area Update

**TCAM** Ternary Content-Addressable Memory

**TCP** Transmission Control Protocol

**TEID** Tunnel Endpoint Identifier

**TFT** Traffic Flow Template

**ToS** Type of Service

**UDP** User Datagram Protocol

**UE** User Equipment

**ULA** Update Location Answer

**ULR** Update Location Request

**UMTS** Universal Mobile Telecommunications System

**VIP** Virtual IP

**VLAN** Virtual LAN

**VXLAN** Virtual Extensible LAN

**VoLTE** Voice over LTE

**VoIP** Voice over IP

**VMDK** Machine Disk File

**VM** Virtual Machine

**VNF** Virtual Network Function

**WAN** Wide Area Network

**WCDMA** Wideband Code Division Multiple Access

**WLAN** Wireless Local Area Network