A Thesis for the Degree of Master of Science

Automatic Data Plane provisioning in a Sliced Mobile Network Scenario

Javier Jose Diaz Rivera

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

June 2019

Automatic Data Plane provisioning in a Sliced Mobile Network Scenario.

Javier Jose Diaz Rivera
(Supervised by Professor Wang-Cheol Song)

Submitted to the Department of Computer Engineering and the Faculty of Graduate School of Jeju National University in partial fulfillment of the requirements for the degree of Master of Computer Engineering

2019.06.27

This thesis has been examined and approved.

안 기 중

Thesis Director, Khi-Jung Ahn, Professor, Jeju National University

변영철

Thesis Director, Yung-Cheol Byun, Professor, Jeju National University

송 왕 철

Thesis Supervisor, Wang-Cheol Song, Professor, Jeju National University

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

Dedicated to
My Family

Acknowledgments

This thesis represents the culmination of my graduate studies for a master’s degree. I would like to thank a number of people who have accompanied me in this exciting journey and also people that have been supporting me to achieve my goal.

First of all, I would like to thank my wife, kids, and parents for their love, patience, and support. Without them, I would not have the motivation and energy for finishing successfully this degree.

I want to express my sincere gratitude and appreciation to my supervisor Prof. Wang-Cheol Song for his patience, motivation, knowledge, and support, but most importantly for his trust in my skills and judgment. Without my supervisor, I would not have been able to enter the Network Convergence Laboratory where I met excellent colleagues and widened my professional experience.

I would also like to thank Prof. Elda Quiroga, Head of the Computer Science Major on my Alma Mater “Instituto Tecnologico de Estudios Superiores de Monterrey”. I do not have enough words to express how grateful I am to her, for the trust and recommendation letter that were fundamental for earning the Korean Government Scholarship that allowed me to do my master’s degree studies.

I also want to thank the National Institute for International Education (NIIED) in South Korea who gave me the opportunity to apply and earn the Korean Government Scholarship. Also, special thanks to Miss Ji Hyojung for her invaluable help and guidance on applying to Jeju National University.

Finally, over the past 2 years, I have met wonderful people. I want to especially thank my colleagues and friends Talha Ahmed and Asif Mehmood who I respect and appreciate tenfold. I also want to thank Adeel Rafiq, Ali Jibran, Tahir Abbas, Khizar Abbas, Jahanzeb Ahmed, Muhammad Afaq, Mudassar Liaq, Alexander Tsarev, Wei-Yu Chen for his important work in ONF and M-CORD related developments, and many others that I met and were part of my learning process during these 2 years. Thanks for the friendship and memories.

Table of Contents

List of Figures.

List of Tables

Glossary

Data Plane Slice:	Combination of SPGW-C and SPGW-U
Data Plane:	SPGW-U
DPPM:	Data Plane Provisioning Module
EPC:	Evolved Packet Core
HSS:	Home Subscriber Service
IBN:	Intent-Based Networking
M-CORD:	Mobile Central Office Rearchitected as a Datacenter
MME:	Mobility Management Entity
NFV:	Network Function Virtualization
NMA:	Network Monitoring Agent
NSSF:	Network Slice Selection Function
ONOS:	Open Network Operating System
QoS:	Quality of Service
SDN:	Software Defined Networking
SPGW-C:	Serving Packet Data Network Gateway - Control
SPGW-U:	Serving Packet Data Network Gateway - User
TOSCA:	Topology and Orchestration Specification for Cloud Applications
UDPR:	Unused Data Plane Repository
UE:	User Equipment
VNF:	Virtual Network Functions

Abstract

One of the main concerns that motivate the innovation in the development of new technologies for 5G networks is the automation of Network Services catered to the Users. The focus on research and manufacture of specialized hardware is shifted to the use of virtualization technologies such as Network Function Virtualization (NFV) and Software Defined Networking (SDN) which drastically reduces time to market deployment. The benefits of these technologies bring more flexibility on how and when to deploy network services in order to adapt to the market needs. This flexibility allows network operators to increase or decrease network performance on the basis of the user's consumption and also provide specific resources that cater to the need of a specific Tenant. This thesis proposal centers its idea in the automatic provision of Data Plane network functions based on Video Streaming Network Usage. It achieves this by adding a Network Monitoring Agent (NMA) in order to collect Data Plane related information from the network and use it to extend the multiple slice selection functionality provided by a Network Slice Selection Function (NSSF). The automatic data plane provisioning is handled from a Top to Bottom approach, by taking advantage of a previously developed Intent Based Networking Tool alongside the open-source Mobile Central Office Rearchitected as a Data-Center (M-CORD) Platform by introducing a Data Plane Provisioning Module (DPPM) that receives the monitoring information from the NMA and defines the directives for creation of new Data Plane network functions. The results of this proposal aim to achieve a further step into network automation by taking advantage of the capabilities of a Network Controller and consolidating emerging solutions related to Network Slicing and QoS policy definition based on Network Contracts.

1 Introduction

1.1 Background

The evolution of mobile networking has brought with it a shift in the paradigm for the use of telecommunication technologies. Starting with the invention of the cellular phone and consequently the inception of the 1G Mobile Networks, the focus on the technology was to mirror the functionality of land networks i.e The Telephone and provide a communication experience that was not tied to the bounds of cables and physical location. The 2G brought with it a digitalization of signals, bigger coverage and the inclusion of Short Message Service (SMS) which became a disruptive technology on its own, this generation of networks took advantage of the shift of digital signals to also provide a limited form of internet connection for mobile devices. Network data provision was a big catalyst for the move to 3G which gave a "broadband" network connection experience to capable devices. It was during this generation that more powerful and complex user equipment (UE) know as Smartphones came to be. These UE have the capability to provide a wide variety of services to the users, which in turn started to use their devices for consuming more Data Services instead of Voice Services (Cellular communication). The smartphones pushed the evolution of mobile networks to more Data Transfer oriented services, which led the 4G networks to focus on the provision of high bandwidth technologies that could cater to these services. [1] Nowadays, the mobile user equipment is used more for Internet Data consuming services than actual voice calls, which paired with the variety of services provided by the smartphones have started to make the current mobile infrastructure insufficient.

5G mobile networks are becoming a reality, this step is a natural evolution to 4G that many see as an increase in the Data Network bandwidth, however, the main motivation of the Fifth Generations of mobile networks is different. 5G aims to be an integrative ecosystem that not only focuses on Smartphones, but encompass and support a wide range of applications, namely, IoT (e.g. smart-home, drones, 4th

Industrial Revolution, health, IoV) mission-critical applications, 4k-8k Video Streaming, Augmented Reality, Virtual Reality, Entertainment, etc. [2] These scenarios range from similar to drastically different performance requirements which having them under the same umbrella of mobile infrastructure can pose a serious challenge if we just consider previous mobile network generation technologies approaches.

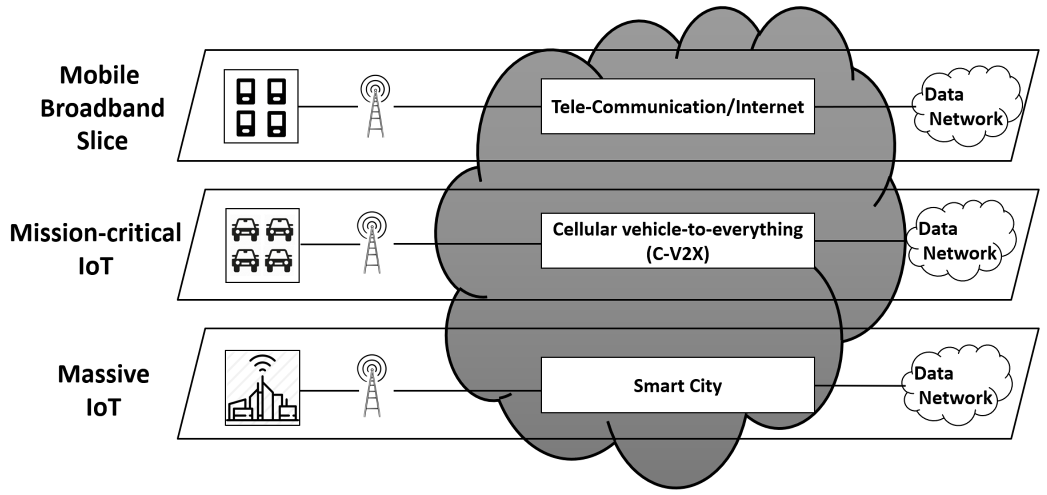


Figure 1. Network Slicing in 5G Mobile Networks

Network Slicing is a functionality embedded in 5G mobile networks and it's the enabler for achieving the necessary conditions for a plethora of services to optimally perform under the same mobile network infrastructure [3] This is done by creating multiple isolated virtual network channels that can be configured to satisfy the specific needs and performance requirements of multiple services over a single physical infrastructure.

Emerging technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) have paved the road for Network Slicing to materialize as a viable solution for 5G mobile networks. SDN refers to the separation of the Control Plane and the Data Plane of a network. The Control Plane is the part of a network that carries signaling traffic and is responsible for routing, on the other hand, the Data Plane is the part of the network that carries the data traffic, in other words, the physical link of the network. These concepts go hand to hand with the current generation of Mobile Networks, as with the Advent of 4G LTE, the concept of Evolved Packet Core (EPC) and the multiple Network Functions that focus on Registration and Control e.g. Mobility Management Entity (MME) and Home Subscriber Server (HSS), etc. and user/data Plane focused functions e.g. Packet Data Network Gateway (PGW) and Serving Gateway (SGW), etc. [4] accommodate the principles of Control/User plane separation of SDN, likewise as 5G is a direct evolution of LTE-A, the network functions integrated in the Fifth Generations Mobile Network Architecture also approach this Control-User Plane Separation (CUPS). [5]

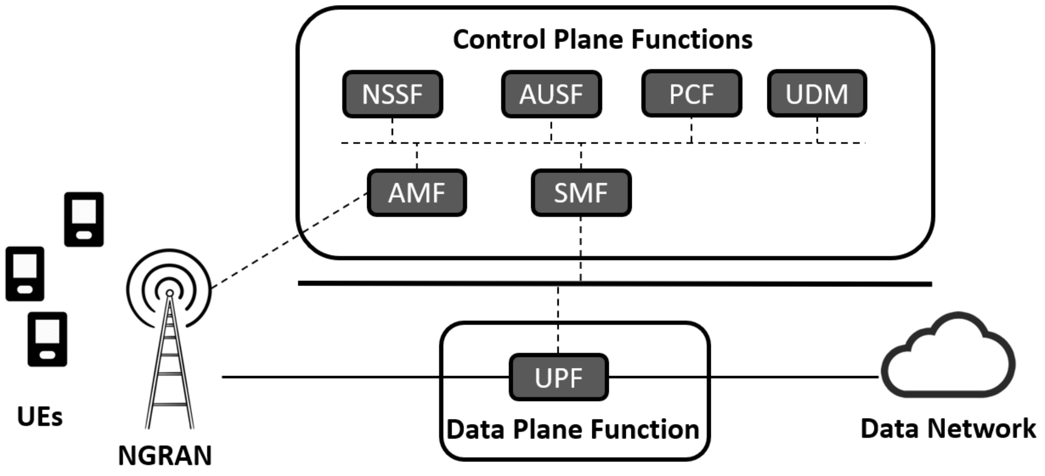


Figure 2. CUPS 5G Network Functions

NFV is commonly known as the process of decoupling the network functions from proprietary hardware equipment in order for them to run in software on commodity hardware. It is a concept that can be applied to any kind of network, including mobile networks. By virtualizing the network functions, Service Providers and Telco. can improve on the Control, Reliability, Scalability and Cost Efficiency of their network resources, e.g. (Real-time and dynamic provisioning, Reduction in complexity of physical device technology, Acceleration of implementation, Power consumption reduction). [6] As mentioned above, one of the key benefits of applying NFV to mobile networking is the Scalability of network resources by providing new functionality or replicating existing network functions on the fly. This capability, coupled with the specific UE requirements for consumption of diverse services on a mobile network, is what allows Network Slicing to be implemented in this current generation.

SDN, NFV and the implementation of Network Slicing bring many benefits in the fields of mobile networks, but also posed a set of challenges on the management and orchestration of the diverse virtual resources. Positioning, Deploying, Proper-Interconnection, etc. of Virtual Network Functions (VNF) requires a proper entity capable of overseeing in a Vertical and Horizontal perspective. [7] Due to this, diverse organizations have engaged in the task to define and create a proper Framework to guarantee the required orchestration of network functions. A prominent head in NFV activities, the European Communication Standards Institute, have defined a reference architecture for NFV Management and Orchestration by the name of MANO [8].

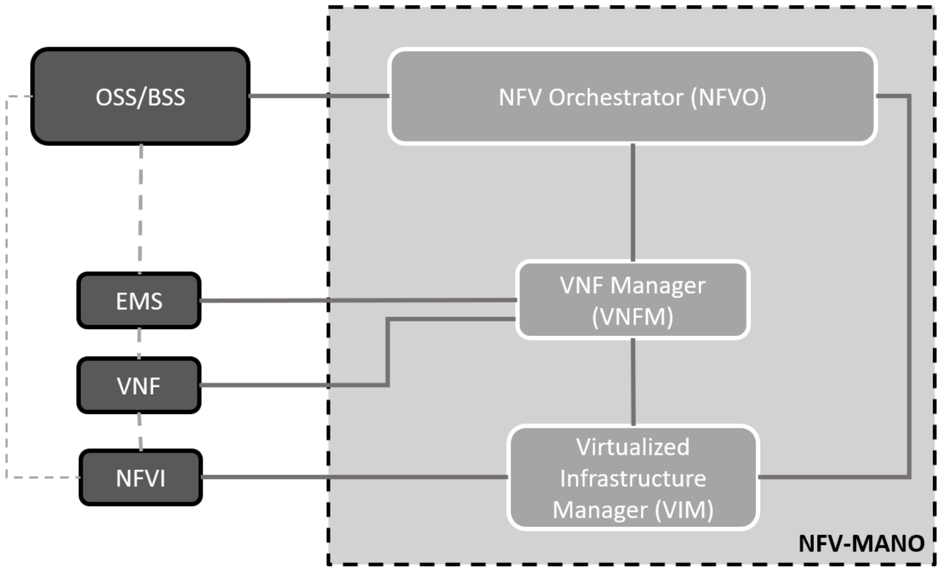


Figure 3. NFV-MANO Architecture

MANO is composed of 3 functional blocks, a Virtual Infrastructure Manager (VIM), NFV Orchestrator (NFVO) and a VNF Manager (VNFM). These 3 blocks together aim to guarantee the proper operation of a VNF ecosystem. From the lower level, VIM manages the physical and virtual resources of VNF instances, The VNFM manages and keeps track of the status of each instance and, the NFVO keeps control of the relation between VNF with the aim of having end to end services.

MANO framework has become a reference for NFV orchestration. For actual implementation and use cases, open-source projects like the Open Network Foundation's (ONF) Central Office Re-architected as a Datacenter (CORD) [9] follows on the principle of management and orchestration to deploy a network environment. CORD proposes its own architecture that differs from MANO in implementation but considers the essence of the three main blocks that comprise the ETSI proposed Architecture. One advantage of CORD is that it gives a workable solution that can be implemented in relative any environment (As long as it meets the system requirements) and more importantly it comes with a Mobile Network oriented profile by the name of M-CORD that is prepackaged with a Service Orchestrator, a Network Controller, and a Cloud Management Platform for deployment of Mobile Network Functions.

The M-CORD Platform provides the right environment for research on network function development, interconnection and orchestration, for this reason, the M-CORD platform has been selected in order to investigate on Network Slicing by adding a Network Slice Selection Function (NSSF) [10] to the network functions that represent the EPC. Furthermore, with the purpose to automate the procedure of creation of policies for the definition of Quality of Service (QoS) for each slice, an Intent-Based Networking (IBN) Tool has been added. It serves as an Application Layer where a User/Network Operator can define through a GUI a series of QoS parameters that are processed into policies and pushed to the physical layer [11]. The purpose of this tool, as its name implies, is to simplify the configuration of network components by telling the Underlying System what to do, instead of how to do it, minimizing greatly the steps of the configuration of a complex network system thus giving a degree of automation in the network service configuration.

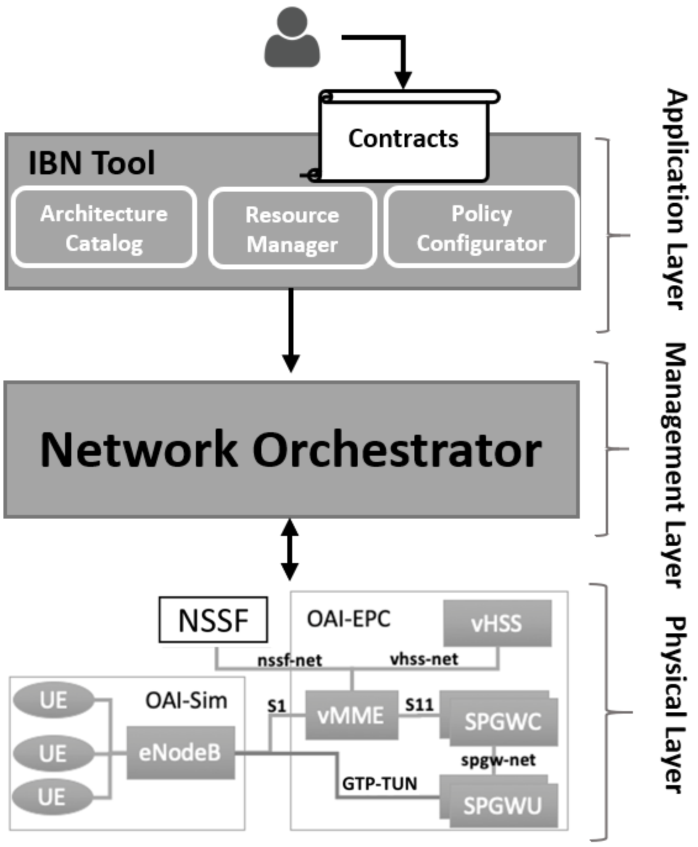


Figure 4. Virtual Mobile Network System of Network Convergence Laboratory

1.2 Research Contribution

The research of this Thesis focuses on extending the grade of automation that the IBN Tool is providing to a Network Platform in order to meet the traffic demands of different user equipment that are assigned to specific network slices, specifically for the consumption of Video Services. Video streaming is a very popular service and one that can quickly fill the network traffic of any infrastructure. According to [12], Video streaming accounted for 60% of the total network data traffic in 2016. 5G faces bigger challenges related to high bandwidth data streaming, and even though Network Slicing can solve part of this issue, there is still room for improvement.

The objective of this investigation is to provide automatic Data Plane (SPGW-U) deployment on the basis of network traffic that is flowing on assigned network slices, e.g. video streaming that has already taken the full bandwidth of a designated slice thus triggering the additional provision of network slices without manual input. To achieve this, enhancements to both the IBN Application and the M-CORD platform are proposed and developed for the testbed and showcase of results. Taking into consideration the overall picture of the IBN Tool, a Data Plane Provisioning Module is added into the Application Layer, as well as a Network Monitoring Agent is built as part of the Physical Layer. Furthermore, enhancements are done to the Physical Layer by modifying the functionality of the NSSF to accommodate this new automatic Data Plane Slices. The proposed system can be overviewed in *Figure 5*

1.3 Structure of the Thesis

This document is organized as follows. Chapter 2 contains the relevant Literature Review details, chapter 3 focuses on the Network Sliced Mobile Network System Description, chapter 4 presents the Data Plane Provisioning Module and Network Monitoring Agent logic and implementation, chapter 5 showcases the evaluation and results, and finally, chapter 6 concludes the thesis.

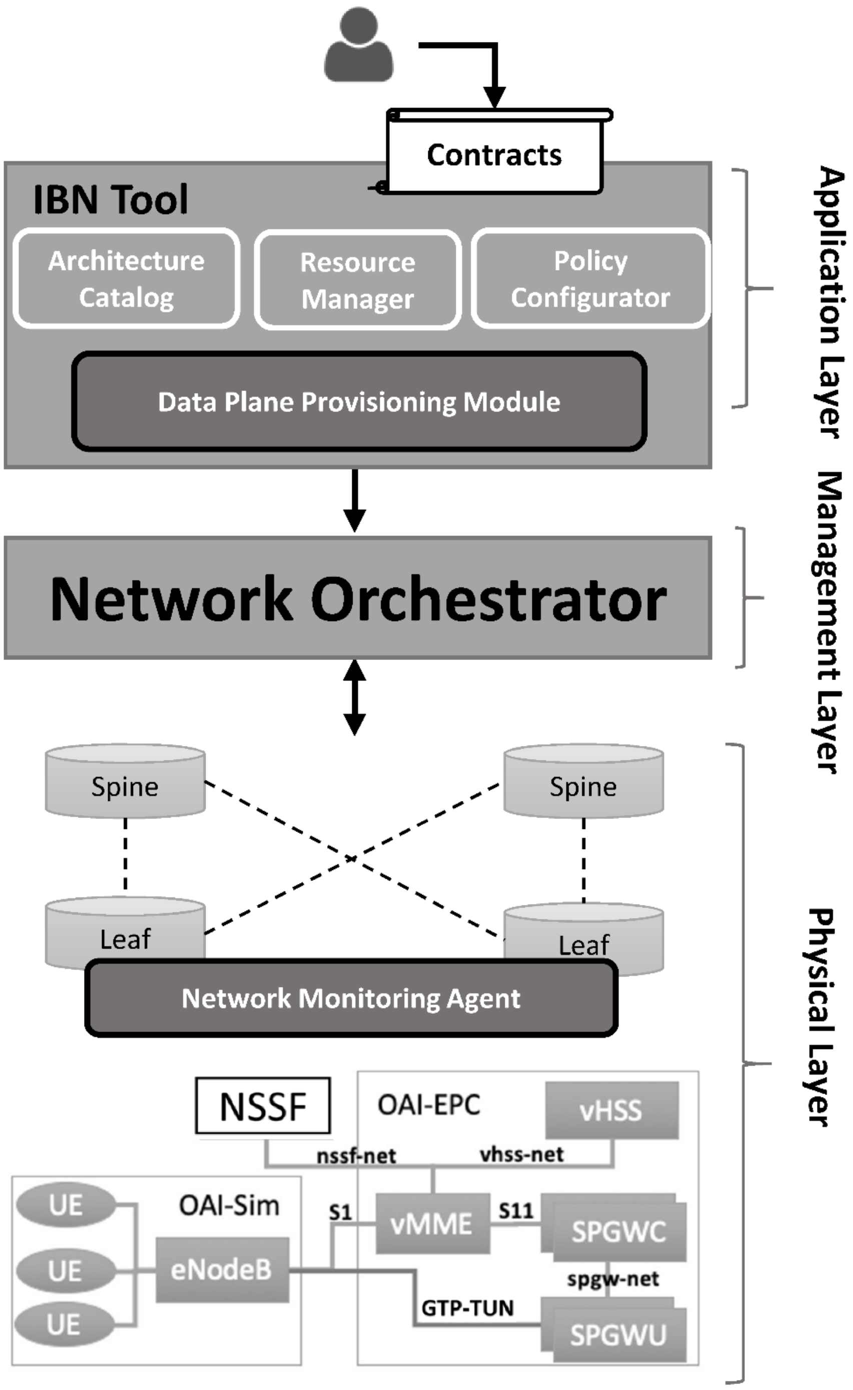


Figure 5. Thesis proposed system

2. Literature Review

2.1 Data Plane Provisioning.

Various techniques, architectures, and proposals related to Data Plane Provisioning in Network Slicing by using SDN and NFV concepts have been presented in literature over the last years. In [13], Vinod Kumar Choyi et al. have defined various QoS requirements for the creation of network slices. On the basis of SDN and NFV solutions, they also proposed a framework and mechanism for defining high-level policies that will translate into the underlying system. To elaborate, application-specific slice selection and User Equipment association is well described in the literature. Definition of VNF and connection between them using Service Function Chaining (SFC) is also presented in this research, as well as routing in what is called a Software Function Path (SFP) selection that refers to the multiple deployments of slices that represent the User/Data Plane within a 5G network. Still, there is no system implementation and evaluation results provided in this paper.

In [14], Jose Ordonez-Lucena et al. presented relevant concepts pertaining to virtualization, orchestration, and isolation for designing network slices in the forwarding plane (Data Plane). By using the Open Networking Foundation (ONF) SDN based architectures alongside NFV design references, they managed to present a complete solution that spans across different layers. Scaling of resources of the forwarding plane (Data Plane) is done with the objective of delivering tailored services to users located in the Application Layer. Although the solution and proposal of this research paper are attractive, it relies on a complex architecture that poses difficulties on the actual implementation, thus it remains as purely conceptual design.

In [15], Peter Roost et al. presented various architectural principles for achieving Network slicing in the Core Network part of a mobile network. In this research, the authors propose the existence of a Software Defined Mobile Network Controller (SDM-C) which can oversee the management of different slices. In line with NFV orchestration, they also introduce a coordinator (SDM-X) that is in charge of managing and controlling the process of sharing Network Functions (NFs) and resources between different network slices, It also ensures maintaining high resource efficiency while guaranteeing individual Service Level Agreements (SLAs) set by the service provider. In other words, this entity is in charge of modifying the QoS of the Data Plane according to the needs of the User Equipment. This work mainly describes a conceptual design for Network Slice and Data Plane Provisioning and Orchestration of VNF in 5G networks, but the actual implementation is left for further work.

In [16], the authors present a testbed with results of an SDN and NFV based LTE EPC implementation. Conclusions of their research have conveyed that an SDN-Based LTE EPC is ideal for managing big amount of data traffic, as forwarding packets are managed from a centralized SDN controller compared to an NFV-Based EPC where forwarding decisions are made in the Data Plane. Implementation details and results are showcased in this literature, also, available open-source development is presented for the EPC components of the mobile network.

In [17], the authors propose a Bit-rate Aware Autoscaling (BAAS) mechanism for autoscaling the Data Plane of a Mobile Network. They used the NFV-Based LTE EPC development from [16] to perform actual experiments. The authors mention the use of a tool for timely monitoring the throughput of a Data Plane and reporting the status to the MME which in turn has the logic of the BAAS that auto-scales the bit rate of the Data Plane when necessary. Another objective of the BAAS is to minimize resource utilization of network slices without affecting the throughput that is served to the UE, thus it can scale down the DP as needed. Although the solution presented in this paper goes in hand with the benefits of Network Slicing in 5G, the system proposal and testbed is just centered in the physical layer of the network, without considering a Service Orchestrator, Application Layer and Automatic management of policies. There is also no mention of a Network Slice Selection Function to handle a selection of slices because all the slice selection logic is put in the MME, thus the proof of concept only refers to this particular NFV-Based scenario.

2.2 Network Monitoring.

Network Monitoring is an essential mechanism to achieve self-management and self-optimization of network resources. There are multiple solutions and tools that range from commercial grade to open source. In this sense, OpenNetMon [18] is a solution that continuously monitors all flows between predefined link and destination pairs on throughput, packet loss, and packet delay that is developed as a module for the OpenFlow controller POX. In order to determine throughput, OpenNetMon regularly queries switches about the number of bytes that are sent, for it to retrieve Flow Statistics and the duration of each flow. Equally, OpenNetMon uses these flow statistics to compute the packet loss by measuring the packet counters from the first and the last switch of each path between the link and destination. Although it is a powerful tool for network monitoring, it is a specific development for the POX controller which is python based and not compatible with the Open Network Foundation ONOS Network Controller that is used in the NCL research.

Another monitoring solution, PayLess [19] was proposed as a low-cost active monitoring framework based on the active monitoring approach. PayLess is built on top of an OpenFlow controller thru communication via a flexible RESTful API interface and is capable of monitoring information from switches in the data plane. It also provides statistics collection that delivers highly accurate information in real-time without incurring significant network overhead. Although theoretically, it could be compatible with the ONOS controller by using REST APIs, test and development were focused on the RYU controller.

Finally, in [20] the authors propose a flow monitoring approach for traffic engineering by using sFlow metrics. sFlow enabled switches to exist in the Data Plane and provide periodical samples of flows that pass between devices. These samples are collected in traffic analysis tools, e.g. sFlow-RT, that sits on the control plane of the network which in turn provides real-time flow summary statistics to control application or network controllers through northbound REST APIs. sFlow is now considered an industry standard, and diverse switch manufactures include this metric as part of the protocol stack inside of the switches. For the case of virtual environments, OpenvSwitch has also included sFlow traffic monitoring which enables extended visibility into virtual servers.

3. Network Sliced Mobile Network System Description

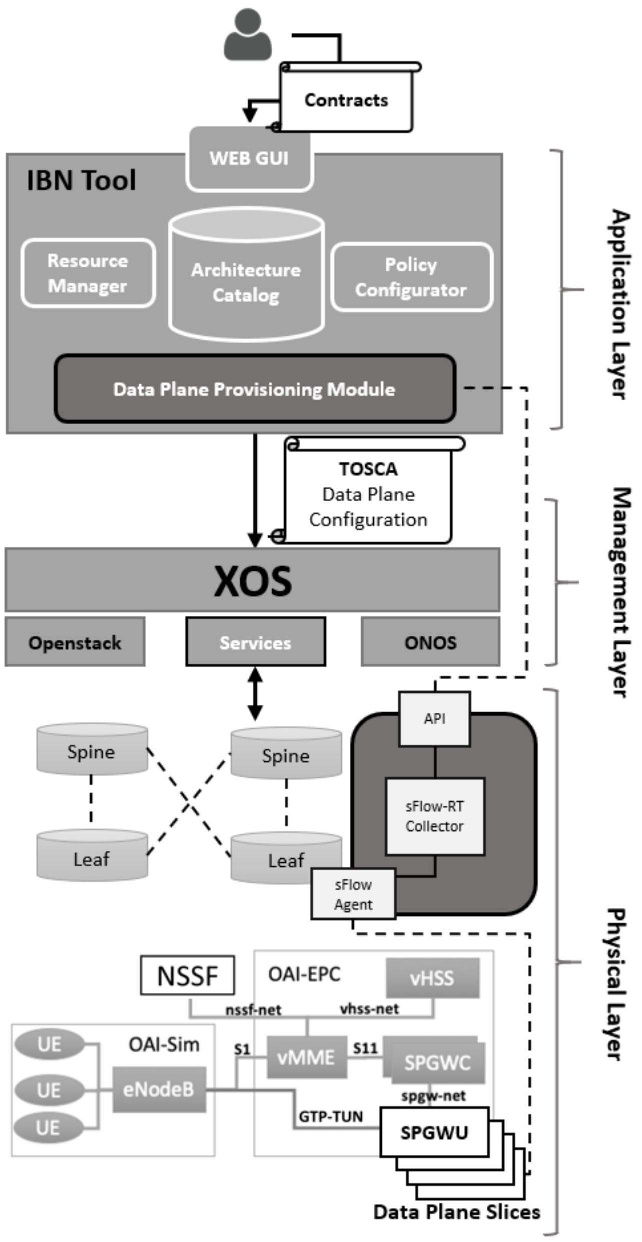


Figure 6. Mobile Network System with Data Plane Provisioning Module.

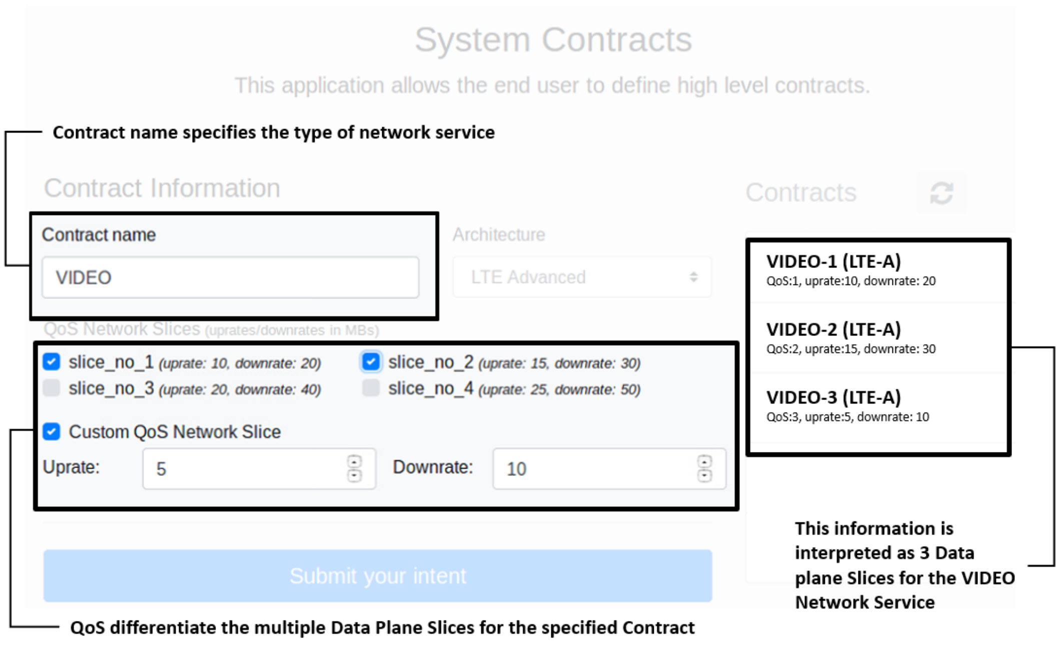
The Automatic Provision of Data Planes requires a working mobile network system for its operation, due to this, the solution is integrated inside a three-layered architecture as shown in *Figure 6* that aims to integrate a complete network sliced virtual mobile network system. The main objective of this proposal is to provide a high grade of automation for the deployment of multiple data planes in order to guarantee an optimal flow of traffic for mobile network services. For achieving this, specific functionality is provided on each level of the proposed architecture which provides complete integration between Business and Operation rules definition, Implementation of Policies, Service Orchestration, Virtual Network Function instantiation, and operation. The high-level description of the operation is as follows: The user defines a set of QoS policies from the Application Layer which are translated into TOSCA definitions that are pushed into the Management Layer. These QoS policies represent the multiple network slices that are going to be created depending on the services that the user is consuming in the mobile network. The Management Layer takes this policies and proceeds to generate the VNFs that are part of each network slice and uses the configuration of the QoS that was set up on the Application Layer to differentiate the transmission rate of each Data Plane. Inside of the Physical Layer, a Network Slice Selection Function gets the complete information from the Management Layer of all the network slices that were created and when mobile network operation starts redirects the requests from the UE to an appropriate network slice depending on the requirements of the network service. During this operation, a Network Monitoring Agent collects data from the OVS that are part of the network fabric, as can be seen in *Figure 6* this is achieved through a sFlow Agent statistics that are periodically sent to a sFlow-RT collector that focuses on network traffic information. Whenever traffic does not meet the QoS created in the Application Layer, the Network Monitoring Agent notifies a Data Plane Provisioning Module that resides on the top Layer of the architecture, this module automatically defines new rules that create a new Data Plane Slice for the service that is being consumed e.g. Continuous video streaming. Following this, new Data Plane rules are fed into the Management Layer which proceeds to create the new VNFs for the new Data Plane Network Slice. Once the VNFs are provisioned in the Physical Layer, the old Data Plane operation shifts to the newly created Data Plane Slice. In order to provide more details for each step of the operation, the following sections describe the functionality of each of the levels of the three-layered Architecture.

3.1 Application Layer.

The upper layer main functionality is to provide an ease of configuration for the creation of network slices and definition of QoS for each of the slices that are created. This is achieved by applying the concept of Intent-Based Networking (IBN) which focuses on defining “what to do” without specifying “how to. This concept has as its priority, to deliver the business goals in the most abstracted form as possible independently of the complexity of the network system that sits in the bottom layer. The IBN Tool is composed of a User Interface (GUI), Architecture Catalog, Resource Manager, Policy Configurator and a Data Plane Provisioning Module.

The GUI provides a clean interface where a user can define network slices as “Contracts”. Each contract has an associated slice with specific QoS that are defined based on the type of service that the user is consuming from the mobile network. The contracts also have information about the type of architecture for what the slices are going to be defined, this information exists on a database called Architecture Catalog, which includes specific deployment information of VNF, e.g. LTE-A has slice QoS definitions for SPGWU and 5G may have definitions specific to a UPF (User Plane Function). This information is an important part of the contract as it lets the Policy Configurator know how to properly translate the contract policies into TOSCA configuration files which may differ from one network architecture to another. These TOSCA files have the information of which data plane slices are going to be instantiated as VNF with a proper definition of QoS. Before passing the TOSCA configuration to the Management Layer, the Resource Manager has the task of verifying that appropriate RAM and CPU resources exist on the Physical Layer for the instantiation of the VNFs. Finally, the Data Plane Provisioning Module only acts after the physical layer Data Plane Slices are operational, its functionality relies on direct communication with the Network Monitoring Agent that exists in the Physical

Layer which will feed the current state of each Data Plane in terms of link utilization, the Data Plane Provisioning Module triggers a redeployment of new Data Plane Network Slices only if the link utilization of current Data Planes do not meet the requirements of the QoS that were previously configured. The Data Plane Provisioning Module design and implementation details are presented in chapter 4.


System Contracts

This application allows the end user to define high level contracts.

Contract name specifies the type of network service

Contract Information

Contract name: VIDEO

Architecture: LTE Advanced

QoS Network Slices (uprates/downrates in MBs)

- [✓] slice_no_1 *(uprate: 10, downrate: 20)*
- [✓] slice_no_2 *(uprate: 15, downrate: 30)*
- [ ] slice_no_3 *(uprate: 20, downrate: 40)*
- [ ] slice_no_4 *(uprate: 25, downrate: 50)*
- [✓] Custom QoS Network Slice

Uprate: 5 Downrate: 10

Submit your intent

Contracts

VIDEO-1 (LTE-A)
QoS:1, uprate:10, downrate: 20

VIDEO-2 (LTE-A)
QoS:2, uprate:15, downrate: 30

VIDEO-3 (LTE-A)
QoS:3, uprate:5, downrate: 10

This information is interpreted as 3 Data plane Slices for the VIDEO Network Service

QoS differentiate the multiple Data Plane Slices for the specified Contract

Figure 7. Relevant Information for definition of Contracts

3.2 Management Layer.

This layer takes the TOSCA configuration files defined by the Application Layer and uses it to configure each VNF that is defined as a Data Plane Network Slice by the contracts. The heart of this layer is the M-CORD Platform which contains an extensible service control plane known as XOS that serves as the orchestrator, a network controller (ONOS) and a cloud manager (OpenStack).

Without the management layer, the instantiation and configuration of multiple VNFs in the Physical Layer is no trivial task, due to this, the orchestrator is an essential component of the M-CORD platform. It coordinates the creation of VNF by managing the Networks, Ports, Image and Resource allocation in coordination with OpenStack and ONOS. To guarantee the operation of the Physical Layer, XOS uses an abstracted representation of the VNF in the form of a Service. Each service defines the data model properties of underlying VNF but does not contain any logic of its functionality, its main purpose is to ease the management and configuration of the instantiation of network functions. The relationship between a service and its VNF is one-to-many, i.e. for each service, there could be many instances sharing the same properties through the data model. To differentiate each instance from one another, the concept of tenancy is used, and each tenant can have specific properties that will affect the operation of the VNF. Also, the services are essential for defining the relations between VNF, as a bidirectional mapping of services creates what is known as a Service Graph which will be used in back-end process to define the required Service Function Chaining (SFC) between VNF, minimizing the complexity of managing the connections between multiple instances of network functions.[21]

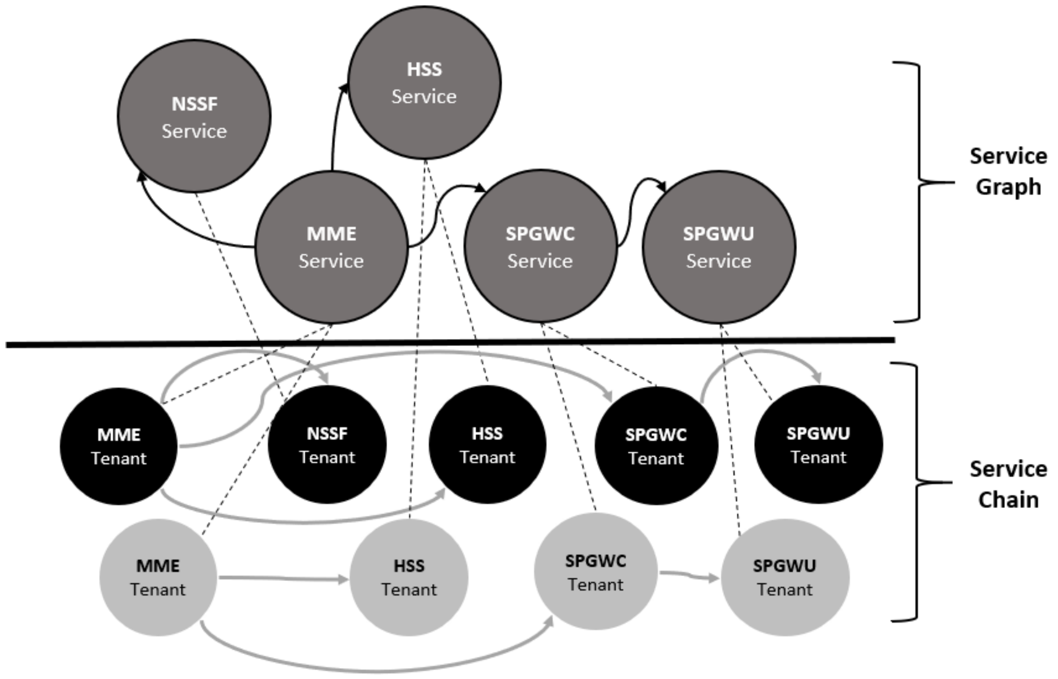


Figure 8. Service Graph and corresponding Service Chain for a Sliced Scenario (Black) and regular E2E mobile scenario (Gray)

The XOS services also include a component that monitors the operation of the VNFs. The synchronizer's role is to keep the management layer up to date with the status of the physical layer. Each service has a synchronizer that monitors basic VNF status e.g. Number of instances, deployment errors, network interface, and IP-Address assignment, etc. It also can be configured in such a way that one service can collect the status of all the VNFs that are managed by other services. Through the synchronizer, the Management Layer can get bidirectional communication with all the running VNF and keep up to date with the operation of the underlying system.

3.3 Physical Layer

The multiple VNFs that represent a mobile network sits on this layer. For the purpose of the research presented in this Thesis, LTE-A based open source components from Open Air Interface were used for both the EPC and the eNodeB Emulation. [22] The configuration of the mobile network is done by the Management Layer for each VNF, it will automatically provision proper IP-Addresses, ports, network configuration (Dependent on the relationship defined in the Service Graph) and virtual machine image allocation. This is achieved by feeding proper TOSCA configuration files that include the Service Graph definitions, network, and service instance creation. As the aim of the research is to provide multiple data plane network slices to this layer, this implementation includes a Network Slice Selection Function which sole functionality is to select the slice that will serve a UE on the basis of the service that is been consumed. The physical layer is in constant change due to the policies that are pushed from the upper layer and the NSSF will always contain the up-to-date status of the system, thanks to the synchronizer that is running on the management layer.

The networking fabric of switches in the physical layer is organized on a leaf-spine fashion. sFlow statistics are collected from the Leaf switch that serves as the ToR (Top of the Rack) switch for the compute node that houses the Data Plane VNFs. This information is collected through a sFlow-RT collector, as part of the Network Monitoring Agent that will calculate the link utilization of each data plane and forward it to the Data Plane Provisioning Module in the Application Layer.

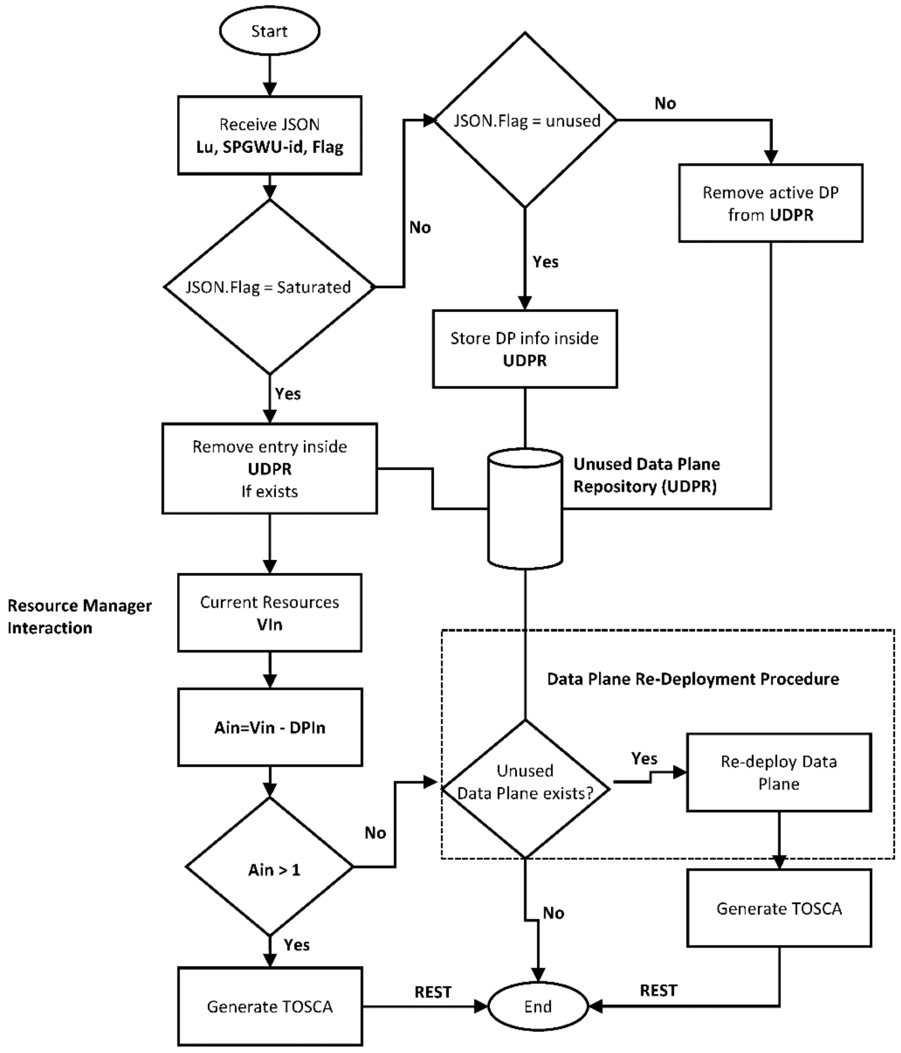
4. Automatic Data Plane provisioning in a Sliced Mobile Network

The implementation details for the main idea of this thesis are presented in this chapter. As previously mentioned, to achieve the automatic deployment of data plane network slices, inter-operation within a virtual mobile network system is essential. Inside this system, there are three actors that play a key role to achieve the objectives of this research, namely, the Data Plane Provisioning Module, the Network Monitoring Agent and the Network Slice Selection Function plus minor modifications inside the physical layer VNFs required for the handover of Data Planes.

4.1 Data Plane Provisioning Module.

Situated inside the Application Layer, the module is in charge of high-level decision making for creating and configuring the data planes. It receives the Link Utilization (*Lu*) data alongside the **Data Plane Id (SPGWU Ip-Address)** of all the Data Planes that are currently active and operational i.e. A UE has established PDU session into the mobile network. The logic of the module can be seen in *Figure 9*. The ***Lu*** arrives flagged by the Network Monitoring Agent, the flag has three possible values, *Unused*, *Normal* and *Saturated*. If the ***Lu*** comes **flagged** as *Saturated*, it means that Data Plane link utilization is not optimal and the QoS that was configured for that specific Data Plane Network Slice are not being successfully met. In the case that the ***Lu*** arrives **flagged** as *Unused*, the information will be saved inside an "**unused data plane repository**" **(UDPR)** for later processing, in case the of a *Normal* flag the Data Plane Information will be used to update the **UDPR** by removing active entries. A ***Lu*** flagged as *Saturated* triggers the Data Plane Provisioning procedure which requires firstly to remove the Data Plane entry inside the **UDPR (If it exists),** after this, interaction with the Resource Manager is required in order to calculate the current physical resources available before deploying the new VNFs.

The Resource Manager calculates the available resources for the compute node that houses the Data Plane VNFs. It follows a standard formula that calculates the available number of instances (*VIn*) by considering the available CPU and RAM of the compute node, ***VIn=(OR*CPU)/VCPU*** or ***VIn=(OR*RAM)/VRAM.***

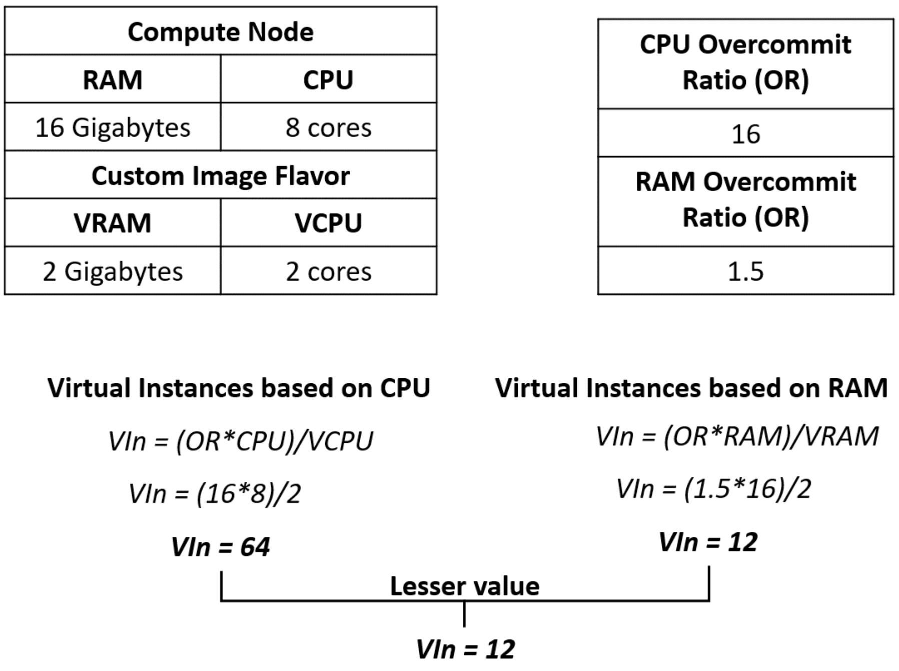


Start
Receive JSON
Lu, SPGWU-id, Flag
JSON.Flag = Saturated
No
Yes
JSON.Flag = unused
No
Yes
Remove active DP from UDPR
Store DP info inside UDPR
Remove entry inside UDPR If exists
Unused Data Plane Repository (UDPR)
Resource Manager Interaction
Current Resources VIn
Ain=Vin - DPIn
Ain > 1
No
Yes
Data Plane Re-Deployment Procedure
Unused Data Plane exists?
Yes
No
Re-deploy Data Plane
Generate TOSCA
Generate TOSCA
REST
REST
End

Figure 9. Data Plane Provisioning Module Flow Chart

VIn is the maximum number of instances that can be allocated to a compute node, *CPU* is the number of physical cores, *VCPU* is the number of virtual cores required per instance, *RAM* is the total physical RAM of the compute node and *VRAM* is the Virtual RAM required per instance. *OR* is the overcommit ratio which is different between RAM and CPU. For CPU the OR=16 and for RAM OR=1.5 (OpenStack

Standard [36]), due to this, in a typical scenario, a compute node capability for Virtual Instantiation is constrained by the total RAM. *Figure 10* shows the VNF Instance allocation for a compute node with 8 cores for CPU and 16 gigabytes of RAM while using a customized deployment flavor for the VNF that allocates 2 virtual cores and 2 gigabytes of RAM per instance.



Compute Node	
RAM	**CPU**
16 Gigabytes	8 cores
Custom Image Flavor	
VRAM	**VCPU**
2 Gigabytes	2 cores

CPU Overcommit Ratio (OR)
16
RAM Overcommit Ratio (OR)
1.5

Virtual Instances based on CPU

*VIn = (OR*CPU)/VCPU*

*VIn = (16*8)/2*

VIn = 64

Virtual Instances based on RAM

*VIn = (OR*RAM)/VRAM*

*VIn = (1.5*16)/2*

VIn = 12

Lesser value

VIn = 12

Figure 10. Possible Virtual Instances based on lesser value of RAM and CPU calculations

The Data Plane Provisioning Module will take the ***VIn*** and subtracts the amount of Data Plane Slices that are currently instantiated on the compute node, ***AIn=VIn-DPIn,*** where ***AIn*** refers to the Available space for Instances in the current compute node and ***DPIn*** to the number of Data Plane Slices that are currently instantiated, this value can be obtained from the Application Layer itself, as active contracts are mapped to Data Plane Slices and this record exists inside the Contracts Table. The value of ***AIn*** is pivotal for continuing with the rest of the process, by default it needs to be ***AIn>1*** *as we need to leave the room of operation for the compute node to avoid overloading of resources*. In case the value of **AIn is greater than 1** the Data Plane

Provisioning Module generates TOSCA configuration for a new Data Plane Slice Instance by taking the **Data Plane Id** that corresponds to the **flagged Lu** and copying its QoS policies, effectively creating a new Data Plane Slice for an already defined contract. The TOSCA configuration is pushed to the management layer by means of REST communication.

Algorithm - Unused Data Plane Repository

```
1: function FINDUDP(sDP)Finds an unused data plane inside the repository,
   receives a Saturated Data Plane (sDP)
2:     dbUDP ← Unused Data Plane Repository
3:     if dbUDP.records != NULL then
4:         for all record ∈ dbUDP.records do
5:             contract ← findContract(record.spgwuid)
6:             if contract.dataplaneslices greater than 1 then
7:                 found ← deployDP(sDP, contract.dataplaneslice[record.spgwuid])
8:             end if
9:         end for
10:        if found then
11:            return ”Re-deployed”
12:        else
13:            return ”No suitable dataplane found”
14:        end if
15:    end if
16: end function
17: function FINDCONTRACT(spgwuId)Finds a contract on the basis of a Data
    Plane Id
18:    jSONArray ← httpconn.request(”GET Data Plane Slices”) REST CALL
19:    contracts ← Contracts table contents
20:    if jSONArray != NULL and contracts != NULL then
21:        for all item ∈ jSONArray do
22:            if item.dataplaneId equals spgwuId then
23:                contracName ← item.servicename
24:            end if
25:        end for
26:        for all contract ∈ contracts do
27:            if contract.name equals contractName then
28:                return contract
29:            end if
30:        end for
31:    end if
32: end function
33: function DEPLOYDP(saturatedDP,contract.dataPlane)Assigns the Data
    Plane to a Saturated Slice
34:    contract.dataPlane.servicaname ← saturatedDP.servicename
35: end function
```

Algorithm 1. Find a proper Data Plane from an Unused Data Plane repository

On the other hand, if the value of **AIn is lesser or equals to 1** the Data Plane Provisioning Module will carry on with the procedure illustrated on *Algorithm 1*. The **UDPR** contains all the Data Plane Slices that have been instantiated but are never used. This information is taken as the basis for the three procedures that are presented in the Algorithm. The DPPM has the capability of selecting a slice from this repository by triggering the **findUPD (Find Unused Data Plane)** function which receives the saturated Data Plane Slice (**sDP**) Information as its argument. The function iterates the records of the **UDPR** and finds the contract information of the oldest (being stored for more than 60s) **unused data plane slice** by calling the **findContract()** function, Before re-deploying a Data Plane Slice, the contract information is used to verify if **more than 1** Data Plane Slice is assigned. This constraint exists due to the nature of the contracts defined by the Application Layer, which mandates that a contract defined for a specific service must have at least one data plane. In other words, even if there are available unused Data Plane Slices inside the **UDPR,** they cannot be used for **re-deploying** when they are the only Data Plane Slice assigned to a contract. Finally, the re-deployment of slices is realized by the **deployDP()** which uses the saturated Data Plane service name (Which is associated with a contract) and assigns it to the unused Data Plane Slice. Once the procedure finishes, a new TOSCA file with this configuration is generated and pushed into the Management Layer. By following the logic of the Algorithm, it is evident that, if the data contained inside the **UDPR** does not meet this requirement, re-deployment cannot be achieved.

To elaborate more on the **findContract()** functionality, it fetches from the Management Layer the total information of Data Planes as a **jSONArray.** This array is iterated according to a **spgwuId (Data Plane Id)** that was received as a parameter for this function when a proper record of a Data Plane is found inside the **JSONArray** the **service name** attribute is extracted and used for finding the whole contract information directly from the **contracts** repository inside the Application Layer. The function returns this contract to the **findUPD()** function for constraint verification.

The value of the flags plays an important role in the selection of Data Plane Slices that the NSSF will perform in the Physical Layer. The DPPM is able to change the selection status of configured Data Plane Slices according to the type of flag that the module receives. When a Data Plane arrives flagged as *saturated,* the DPPM changes the **selectable property** of the slice as *disabled,* on the other hand, when the data planes arrive with any other type of flag e.g. (normal, unused) the DPPM *enables* the selectable property of the data plane. This is reflected in the TOSCA configuration file that is sent to the management layer. A reference file can be seen in *APPENDIX A*. Further details of how this **selectable property** affects the NSSF operation is detailed in *section 4.3*.

Lastly, the **DPPM** takes into consideration the event when no available physical resources exist for creating new data plane slices, e.g. (***AIn*** **is lesser or equals to 1** and the **UDPR** does not contain any candidate for re-deployment). Due to this, DDPM operation cannot supply new data plane slices which means that the maximum mobile network slice capability has been reached. This limitation is only evident in the implementation done in this research as the physical resources available in the laboratory are constrained.

4.2 Network Monitoring Agent.

The NMA spans across three components that work together for monitoring the status of the Data Plane Slices, namely, sFlow Agent, sFlow-RT, and an API module for Link Utilization Calculation.

4.2.1 sFlow Agent

sFlow is a standard sampling technology for a network traffic monitoring solution. sFlow stats are measured on the switch level of a network infrastructure. It applies a scalable (allows a virtually infinite number of interfaces to be monitored from a single location) technique for measuring network traffic, collecting, storing, and analyzing traffic data. sFlow has minimal impact on the performance of core network devices, without adding significant network load [23].

The sFlow Agent runs as a process inside the management software of a switch as seen in *Figure 11*. Its main purpose is to create **sFlow datagrams,** which are a combination of interface counters and flow samples, these datagrams are sent to a sFLow Collector (Which does the monitoring and representation of flows) that exists as another entity inside the network. Due to this, the sFlow Agent does very little processing. It only packages data into sFlow Datagrams that are immediately sent to the collector. This Immediate forwarding of data minimizes memory and CPU requirements associated with the sFlow Agent.

sFlow is available for a wide variety of devices, most importantly for this research, it is also available in the software implementation of OpenFlow switch known as OpenvSwitch (OVS), which is an opensource implementation of a virtual switch. OVS is a standard for virtual network implementation and supports multiple protocols used in computer networks.

Our research implementation uses four OVS deployed in a Leaf-Spine Fabric, our Leaf2 connects two compute nodes that contain the Data Plane Slices and a sFLow Collector (sFlow-RT) respectively.

sFlow Datagram Contents
The IP address of the sFlow agent
The number of samples
The interface through which the packets entered the agent
The interface through which the packets exited the agent
The source and destination interface for the packets

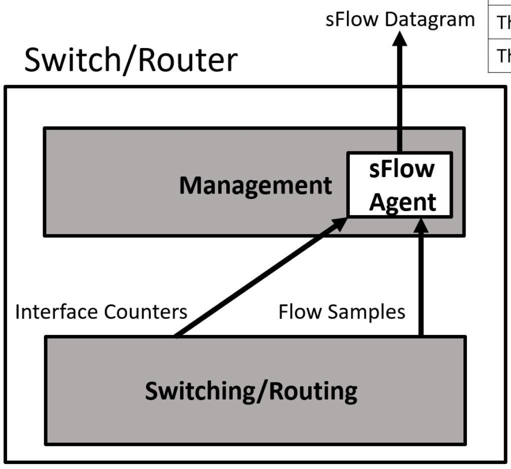


Figure 11. sFlow Agent inside a Switch/Router

For sFlow to operate, identification of the interfaces that connect the Host machine that contains the VNFs to be monitored and the Host that contain the sFlow Collector is necessary. *Figure 12* shows the OVS configuration that is set up in the physical server. As part of the M-CORD deployment, multiple virtual interfaces are created (from **vnet1** to **vnet11**) these are reserved for interconnecting compute nodes and switches. As this thesis implementation includes three compute nodes, Leaf1 has been assigned a **vnet11** interface between the switch and node1. Leaf 2 has two interfaces, **vnet8** for node2 and **vnet3** for node3 (interface assignment to switches is a random process of the M-CORD deployment). For this scenario, node2 contains the Data Plane Slices that are going to be monitored and node3 houses the sFlow Collector for **sFLow datagram** analysis. It is important to take notice that this Spine-Leaf Fabric is not enabled by default during the M-CORD deployment, specific configuration must be modified in order to created four switches and properly set up the connection to the ONOS controller, see *APPENDIX B*. Also, each compute node has an OVS called **br-int** that is also managed by ONOS, which the main purpose is to provide interconnection to the underlying VNF.

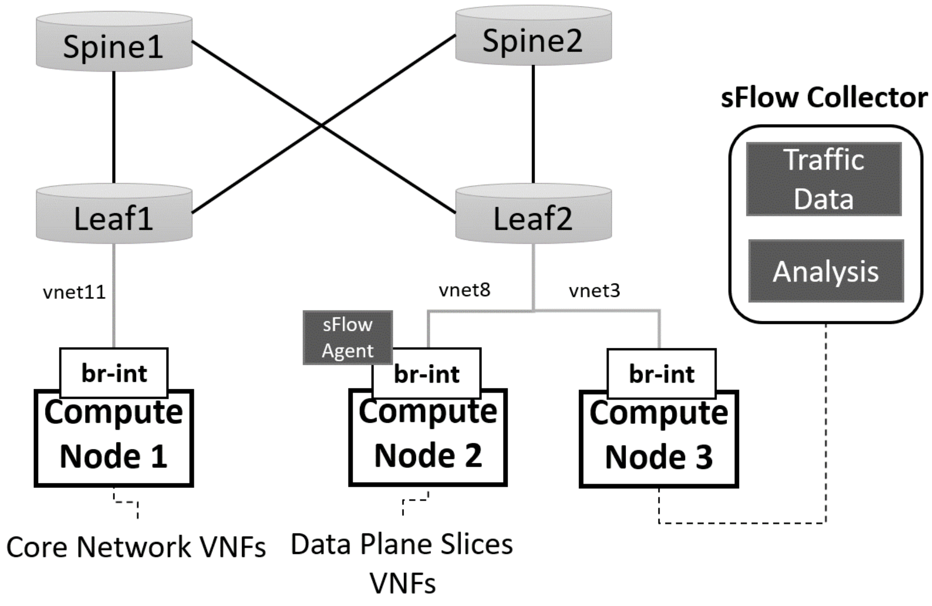


Figure 12. LEAF-SPINE with interfaces and SFLOW

The configuration required for enabling sFlow statistics is configured in the compute node where the OVS configuration for the VNF resides. It follows some definitions of Environmental Variables that assign the values for COLLECTOR_IP, COLLECTOR_PORT, AGENT_IP, HEADER_BYTES, SAMPLING_N, POLLING_SECS. The *COLLECTOR_IP* address refers to the management IP that is assigned to the compute node where the sFlow Collector resides, the *COLLECTOR_PORT* is standard and uses the value 6343, the *AGENT_IP* refers to the name of the interface that connects the OVS and the compute node that has the Data Plane Slices that are going to be monitored. The rest of the variables use a default value that is enough for monitoring all the flows that an OVS can handle. *Figure 13* has the values used for this scenario.

```
COLLECTOR_IP=10.0.10.16
COLLECTOR_PORT=6343
AGENT_IP=fabric
HEADER_BYTES=128
SAMPLING_N=64
POLLING_SECS=10
```

Figure 13. sFLOW Environmental Variables configuration example

For activating the sFlow agent the following command is issued in the Compute Node that needs to be monitred.

```
$ ovs-vsctl -- --id=@sflow create sflow agent=${AGENT_IP}
target="${COLLECTOR_IP}\:${COLLECTOR_PORT}" header=${HEADER_BYTES} sampling=${SAMPLING_N}
polling=${POLLING_SECS} -- set bridge br-int sflow=@sflow
```

Which takes the values that were set as Environmental Variables and applies them to bridge **br-int.** A **bridge** is the actual representation of an OVS. For this deployment, seven bridges exist, **spine1, spine2, leaf1, leaf2,** and three **br-int** for the compute nodes. We only required sFLow statistics on the compute node that houses the Data Plane Slices. By setting up the sFlow Agent inside **br-int** it will periodically send the data to the sFlow Collector.

4.2.2 sFlow-RT

The sFlow Collector chosen for this scenario is **sFlow-RT.** There exist multiple opensource sFlow Collector available with different capabilities. The main reasoning for using sFlow-RT is the strong collection of REST API that comes packed with the solution, which is essential for feeding the monitoring data to the rest of the Mobile Network System. The multiple benefits of sFlow-RT can be seen in *Figure 14*[24]

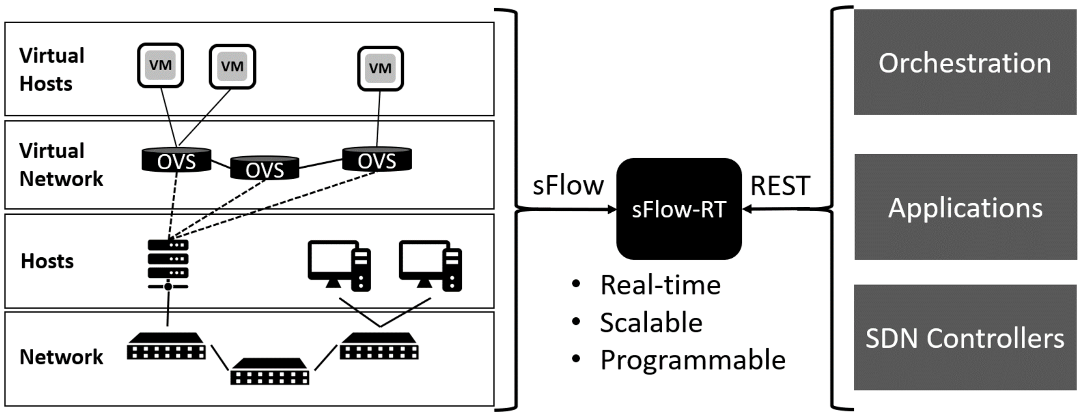


Figure 14. sFlow-RT Integration inside a Network

As evident from the above picture, besides the native benefits of sFlow (Real-Time and scalability), sFlow-RT also include a level of programmability for definitions of how the **sFlow datagrams** are going to be represented and organized. With this, it is possible to define flows that match packets or transactions that share common attributes and do real-time computation. For this scenario, we require the definition of a flow that captures the source and destination IP addresses of requests done to the Data Plane Slices connected to **br-int** inside the Compute Node 2 and then calculate bytes per second for each flow. This can be achieved with the following instruction.

```
setFlow('Bps',
 { keys:'ipsource,ipdestination,tcpsourceport,tcpdestinationport',
  value:'bytes', log:true }
);
```

The above statement creates a Flow definition name **Bps** (Bytes per second) which include the following Flow Keys: **ipsource** that refers to the IP-address of the host that initiates the flow, **ipdestination** refers to the IP-address of the destination host, **tcpsourceport** refers to port that is going to be used for accepting the reply of the request that is being made by the host, and the **tcpdestinationport** refers to the type of request that is being generated e.g. an HTTP request will have **tcpdestinationport** as 80. It is important to notice that this information already exists in the sFlow datagram that is being sent regularly to the sFlow-RT collector by the sFlow Agent inside the OVS, but it is necessary to give the proper representation for further calculations. The statement will return all the flows of the Data Plane Slices, formatted according to the Flow Keys that were used and also the Transfer Rate associated with each flow.

4.2.3 API module for Link Utilization Calculation

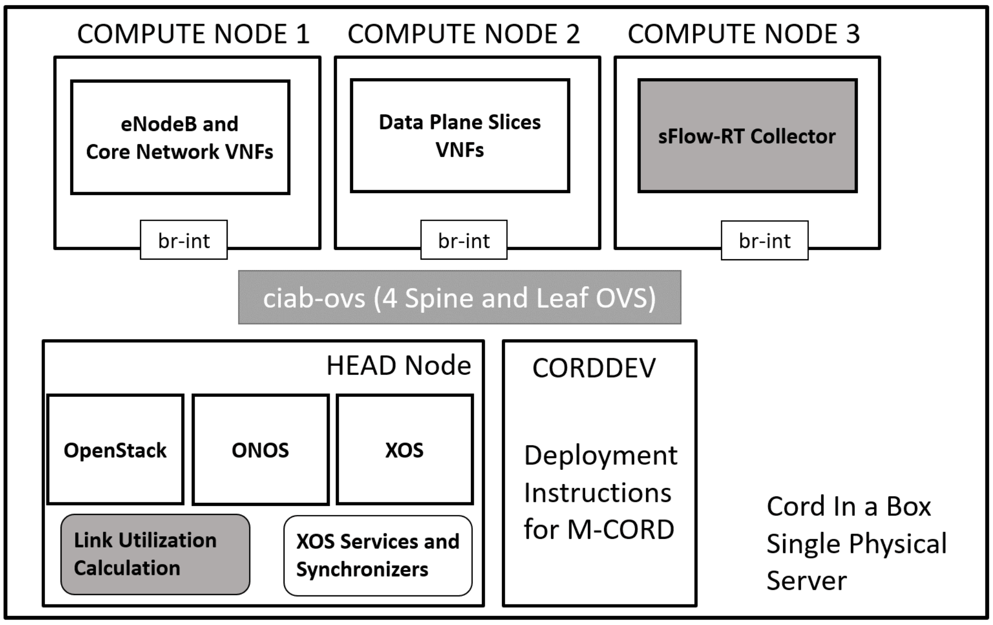

COMPUTE NODE 1
COMPUTE NODE 2
COMPUTE NODE 3
eNodeB and Core Network VNFs
Data Plane Slices VNFs
sFlow-RT Collector
br-int
br-int
br-int
ciab-ovs (4 Spine and Leaf OVS)
HEAD Node
OpenStack
ONOS
XOS
Link Utilization Calculation
XOS Services and Synchronizers
CORDDEV
Deployment Instructions for M-CORD
Cord In a Box Single Physical Server

Figure 15. CiaB deployment figure

The module for Link Utilization Calculation resides as a standalone process running on the Head Node of the M-CORD deployment. *Figure 15* shows the block diagram of a deployment of M-CORD 4.1 in a single physical server also known as Cord in a Box (CiaB). In this representation, the Head Node is the embodiment of the Management Layer that houses XOS, OpenStack and ONOS controller. For this research, the Link Utilization Module is implemented as a process that constantly proves the sFlow-RT collector for getting the current Transfer Rate of flows that exist in the Data Plane Slices.

Algorithm - Link Utilization Calculation

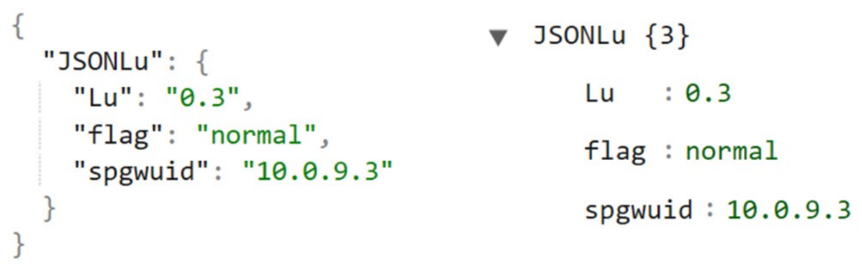
```
1: jSONDP ← httpconn.request(”GET Data Plane Slices”) Obtains Current
   Data Planes with QoS
2: arq ← Initialize Object Dict with value,ipsource,FLu,linkrate attributes.
3: jSONLu ← Initialize JSON Object with Lu,flag,spgwuid attributes.
4: rq1 ← requests.get(’activeflows/ALL/Bps/json’) Requests Data Plane
   Flows from sFlow-RT
5: wait(10s)
6: rq2 ← requests.get(’activeflows/ALL/Bps/json’) Second request for calcu-
   lating avarage values
7: for all $flow \in rq1$ do
8:     for all $flow \in rq2$ do
9:         if $flow$ equals $flow2$ then
10:            arq.value ← (flow2.Rate + flow.Rate)/2
11:            arq.ipsource ← flow.ipsource
12:        end if
13:    end for
14: end for
15: for all $item \in arq$ do
16:    for all $data \in$ jSONDP do
17:        if $item.ipsource$ equals data.spgwuip then
18:            item.linkrate ← data.linkrate
19:        end if
20:    end for
21:    item.FLu ← (item.value * 8)/item.linkrate
22: end for
23: jSONLu ← sum(arq) Adds all the values of Flu for flows with the same
    ipsource and assigns Lu and spgwuip values to jSONLu object.
24: if jSONLu.Lu is less than 0.8 then
25:    jSONLu.flag ← normal
26: else
27:    jSONLu.flag ← saturated
28: end if
29: sendtoDPPM(jSONLu)
```

Algorithm 2. API Link Utilization

The logic of this module is as follows. Firstly, the module will send a REST-API request to the sFlow-RT collector to get the current flows and transfer rates (***rq1***), then a pause occurs for 10 seconds (***t=10s***), after this a second request for transfer rates is done (***rq2***). The module will then calculate the transfer rate average for flows that have the same Flow Key values **ipsource, ipdestination, tcpsourceport, tcpdestinationport** between ***rq1*** and ***rq2*** (***arq_n= ($rq2_n$ + $rq1_n$)/2***). As the average transfer rate(***arq_n***) for each flow is expressed in **Bytes/sec** it needs to be converted to **bits/sec** multiplying it by **8** and then divided by the **LinkRate** in bits/sec of the **Data Plane Slice** that generates the flow (***$FLu_n = (arq_n * 8)/LinkRate$***). The Flow Link Utilization ***FLu*** is a decimal number from 0 to 1. As many Flows can occur for the same Data Plane Slice, the ***FLu*** for flows with the same **ipsource** are summed up to get the **Data Plane Link Utilization (*Lu*).** If the ***Lu*** is equals or greater than **0.8** the module will flag the Data Plane as *saturated* else, it flags the Data Plane as *normal*. This value is heuristically applied by taking into consideration the "*Best Practices in Core Network Capacity Planning*" from CISCO which states that average link utilization should not be higher than **approximately** 70 percent [37]. The whole process repeats every 10 seconds

A parallel process is done for detecting **unused** data plane slices. The module sends a REST-API request to the sFlow-RT collector to get the current flows and transfer rates (***rq1***), then a pause occurs for 60 seconds (***t=60s***), after this a second request for transfer rates is done (***rq2***). The module will request the List of Data Plane Slices to XOS by means of REST-API and compare the Flow Key **ipsource** of the ***rq1*** and ***rq2*** with the IP-Addresses of the Data Plane Slices requested to XOS. If a Data Plane IP-Address does not exist in both the ***rq1*** and ***rq2*** flow information, it is flagged as *unused*. The process repeats every 60 seconds.

The flagged ***Lu*** data is sent to the Data Plane Provisioning Module for processing and deployment of new Data Plane Slices (If required). The data is sent in a JSON Format that includes the ***Lu***, the **Data Plane Id (SPGWU Ip-Address)** and the **Flag** as seen below.



```
{
  "JSONLu": {
    "Lu": "0.3",
    "flag": "normal",
    "spgwuid": "10.0.9.3"
  }
}
```

```
▼ JSONLu {3}
    Lu     : 0.3
    flag   : normal
    spgwuid : 10.0.9.3
```

Figure 16. JSON Link utilization example.

An important value that is highly dependent on the QoS policies for the Data Plane Slices is the **LinkRate.** This is obtained by fetching the QoS properties of the slice where the current calculation of the ***Lu*** is taking place. For example, if calculation for ***Lu*** with an **ipsource** of 10.0.9.3 corresponds to a Data Plane Slice with QoS set for a **3Mbps** of down-rate, then, the **LinkRate** is 2 **(Multiplier)** times the specified QoS, in this case, **6Mbps.** This specification for the **LinkRate** is particular for the experiments done on this research and does not follow any industry standard. The reasoning behind it depends on the limitations of the physical resources for the testbed. As the operation of the network only allows a maximum of three UE to connect simultaneously, their number represents the highest throughput that can be achieved for being able to saturate a Data Plane Slice. In the case that more UE could be supported, the **Multiplier** for the **LinkRate** will change accordingly.

4.3 Network Slice Selection Function.

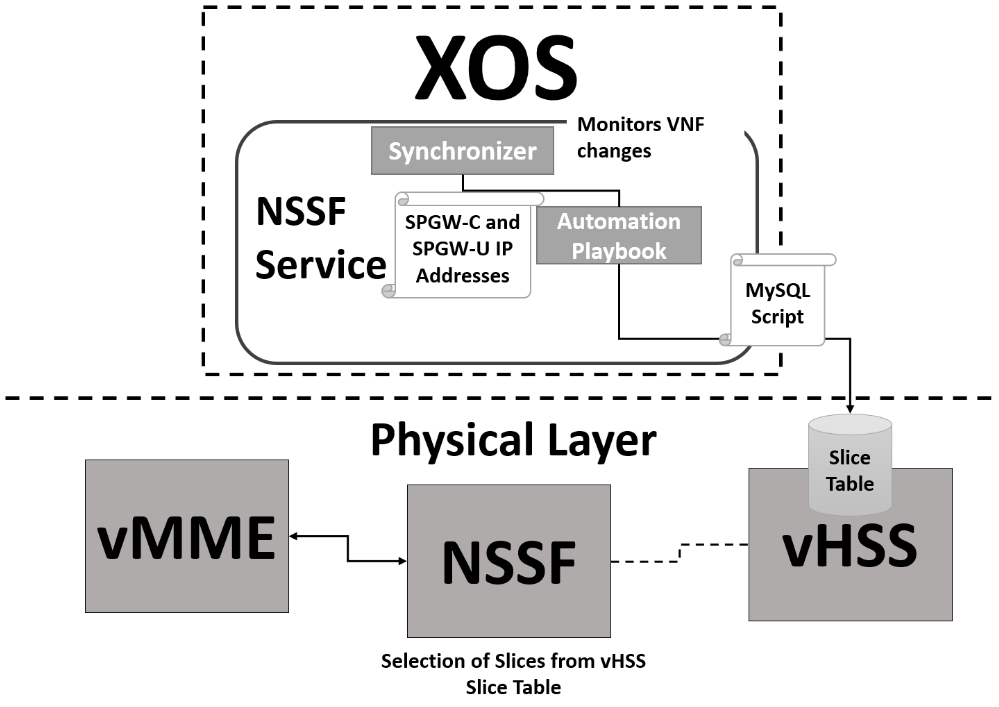


Figure 17. NSSF XOS Service

The NSSF is based on 3GPP implementation for the 5G Mobile Network Architecture [5]. Due to the complexity and unavailability of open source 5G components, the research of this thesis use LTE-A network functions for creating the Mobile Network inside the physical layer, for this, the absence of an AMF prompted the selection of network slices to be exclusively on the NSSF which requires to keep the current status of the VNF that represent the Data Plane Network Slices, namely, SPGW-C and SPGW-U. *Figure 17* Shows how the XOS Service for the Network Slice Selection Function plays a key role in collecting the information of current deployed VNFs and keeps it on record in a Data Base that the NSSF VNF will access for selecting slices.

4.3.1 Network Slice Selection Function synchronizer

The synchronizer running inside the XOS service for the NSSF has the following functionality. *Algorithm 3* shows the logic behind the synchronizer. Firstly, it watches over the status of the mobile network by monitoring all available SPGW-C and SPGW-U VNFs that gets instantiated. For both cases, it will get a list of Tenants that are successfully provisioned. These tenants are represented as objects with specific attributes that were defined in the data model of their respective XOS service. Secondly, The NSSF synchronizer will verify for each SPGW-U tenant the value of the *selectable property* that was set by the DPPM according to the status of the Data Plane (Saturated, Normal, Unused). Thirdly, the NSSF extracts two attributes from both SPGW-C and SPGW-U tenant lists, the first attribute is the *network.port* and the second attribute is the network *service*. It will ignore any SPGW-U tenant that has a *selectable* property as ***disabled*.** As the synchronizer works on the Management Layer it can obtain all the tenant information directly from XOS, because all the Data Model definitions for the Services are kept in a database inside the Orchestrator*.* The *network.port* is represented by the IP-Address of the tenants, and the *service* is defined by the type of contract that was created in the Application Layer e.g. HD-Video, WEB, IoT. This attribute is the most important for association of network slices, as the SPGW-C and SPGW-U will be linked together based on the *service* that was assigned to them when the contracts were defined. The information pertaining to the total SPGW-C and SPGW-U tenants that currently exist in the network is inserted in a *Slice-Table* which is located inside the HSS database by the means of an Ansible Automation Playbook. This process executes whenever there is a change in the VNFs that the synchronizer monitors. The contents of the *Slice-Table* are indirectly affected by the DPPM, due to Provision of new Data Planes or Re-assignment of unused Data Planes to different network services. Also, when the DPPM sets the value of the *selectable* property of each Data Plane, it affects the NSSF synchronizer decision of which SPGW-U to include in the *Slice-Table.* This effectively prevents UE allocation to *saturated* Data Plane Slices, minimizing further network performance impact.

Algorithm - NSSF Synchronizer

```
vspgwcList ← VSPGWCTenant.object.all()
vspgwuList ← VSPGWUTenant.object.all()
for all $vspgwc \in vspgwcList$ do
    for all $vspgwu \in vspgwuList$ do
        if $vspgwu.selectable$ then
            if $vspgwc.service == vspgwu.service$ then
                sliceDict append $vspgwc.network.port, vspgwu.network.port$
            end if
        end if
    end for
end for
return sliceDict
```

Algorithm 3. NSSF Synchronizer Algorithm.

There is only one SPGW-C per *service_name* and multiple SPGW-U that can be linked to it. The reasoning behind this is to easily differentiate the multiple Data Plane Network Slices that are created in the Physical Layer, as the SPGW-C represents a stitching point for the SPGW-U. The SPGW-C role is to handle session establishment and SPGW-U association to the UE, due to this, network throughput is not considerably affected as only control messages are exchanged during its operation. On the other hand, the SPGW-U Tenants get the QoS configuration from the contracts and directly affect PDU session and further differentiate the Data Plane Network Slices.

4.3.2 Network Slice Selection Function VNF

Beside the synchronizer that runs on the Management Layer, the NSSF also has logic that runs on the Physical Layer as a VNF. It follows the UE Network Registration procedure and Session Establishment and interacts directly with the Mobility Management Entity (MME) for Data Plane Network Slice Selection and indirectly with the Home Subscriber Server (HSS) for accessing the information of available slices from the database that is housed inside this VNF. The procedure for slice selection can be appreciated in the sequence diagram presented in *Figure 18*.

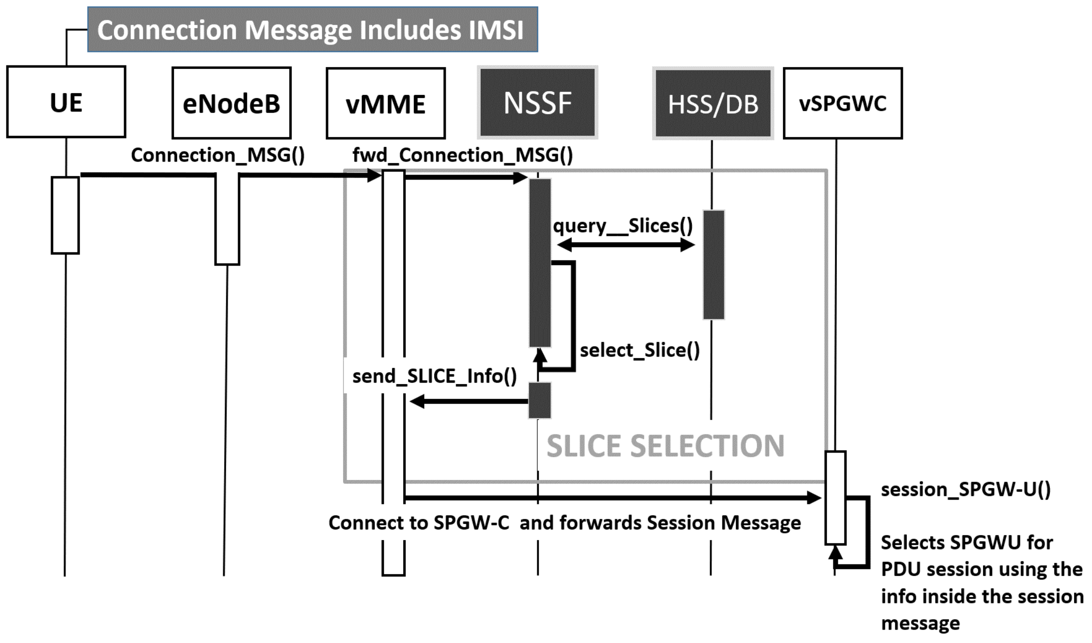
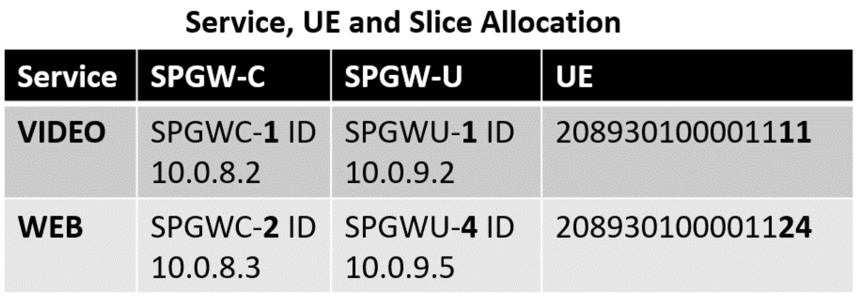

Connection Message Includes IMSI
UE
eNodeB
vMME
NSSF
HSS/DB
vSPGWC
Connection_MSG()
fwd_Connection_MSG()
query__Slices()
select_Slice()
send_SLICE_Info()
SLICE SELECTION
session_SPGW-U()
Selects SPGWU for PDU session using the info inside the session message
Connect to SPGW-C and forwards Session Message

Figure 18. NSSF VNF operation.

Firstly, the UE initiates network registration by sending the connection message from the eNodeB to the MME. The message includes the International Mobile Subscriber Identity (IMSI) which is a fifteen-digit combination that contains the Public Land Mobile Network (PLMN) for the first five digits and Mobile Subscription Identification Number (MSIN) for the rest of the enumeration. These numbers are used for both, UE identification and Data Plane Network Slice allocation. Secondly, once the MME finish the registration of the UE into the network, it forwards the IMSI to the NSSF which in turn connects to the database inside the HSS and queries the *Slice-Table* to get the current SPGW-C IP-Address that is enabled as a stitch point for the Data Plane Slices and also the SPGW-U IP-Address that is related to the network service requested by the UE, the NSSF keeps the record of each UE assigned to a Data Plane Slice. Thirdly, the IP-Addresses are forwarded to the MME which will proceed with the session establishment for the UE using the IP information for connecting with the SPGW-C and storing the SPGW-U IP-Address in the *bearer-context* message that is sent to the SPGW-C with session information. Finally, the SPGW-C uses the *bearer-context* information to select an SPGW-U

(From a list of registered Data Planes) using the IP-Address of the Data Plane Slice that was received from the MME and proceeds to create a PDU Session by enabling a GTP-Tunnel between the UE and the selected SPGW-U.

The IMSI plays a crucial part in the selection of a slice by the NSSF. This research has labeled the last two digits of the IMSI as Slice Differentiators (SD) by allocating the second last digit to identify the stitch point of a Data Plane Slice i.e. SPGW-C and the last digit to identify the SPGW-U that is going to serve the connection of the UE. *Figure 19* shows the case when two UE with specified IMSI try to access the network and are directed to their correspondent Data Plane Network Slices.



Service, UE and Slice Allocation

Service	SPGW-C	SPGW-U	UE
VIDEO	SPGWC-**1** ID 10.0.8.2	SPGWU-**1** ID 10.0.9.2	2089301000011**11**
WEB	SPGWC-**2** ID 10.0.8.3	SPGWU-**4** ID 10.0.9.5	2089301000011**24**

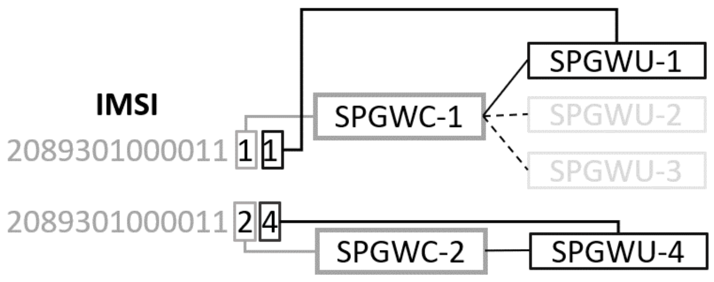


Figure 19. IMSI based slice selection.

By basing our research in the 5G Architecture from 3GPP the decision of adding the *Slice-Table* inside the HSS emulates the concept of a Unified Data Management (UDM) network function. Although in this implementation, the relationship between the NSSF and the HSS does not include any logic that makes these two network functions interact between each other apart from database manipulation from the NSSF side. As mentioned above, the table is filled by the NSSF synchronizer, due to this, it has the updated view of the status of the network. A representation of this table is shown in *Table 3* where six Data Plane Network Slices are configured for three services. The services are directly proportional to the number of SPGW-C. For a new service to exist, it needs to be defined in the Contract policies on the Application Layer, this will create a new SPGW-C tenant in the Physical Layer once the TOSCA configuration passes through the Management Layer. Following the same logic, any QoS defined for a Data Plane Slice will be associated with a Service and will create SPGW-U tenants that will register to their corresponding SPGW-C.

4.4 Physical Layer Modifications.

The physical layer is mostly composed by the Mobile Network VNFs from Open Air Interface The eNodeB and UE implementations are secured using the OpenAirInterface System Emulation (OAISIM). It includes a standard-compliant implementation of a subset of release 10 LTE for UE and eNodeB. The simulation of multiple UE that can act simultaneously in connecting to a single eNodeB can be tested with the use of OAISIM.

Like the eNodeB, the EPC is also part of OpenAirInterface Software Alliance solution for mobile open source VNFs. It houses the implementation of MME, HSS, SPGW-C, and SPGW-U that serve as core network components. Both projects are freely distributed by the OpenAirInterface Software Alliance (OSA) under the terms stipulated by a new open-source license catering to the intellectual property agreements used in 3GPP which allows contributions from members holding patents on key procedures used in the standard. The original implementation of the mobile core network from Open Air Interface does not consider a Network Slicing Scenario and does not have any communication with an NSSF. Fortunately, the code for each of the components is freely available which allowed modifications inside the MME, SPGW-C, and SPGW-U for achieving a proper interaction with the NSSF and effective selection of slices.

4.4.1 Mobility Management Entity (MME) Modifications.

The MME is the key control-node for the LTE access network. It works in conjunction with the eNodeB and SPGW-C in the core network to realize the following actions relevant to our Data Plane Slice Scenario:

• Responsible for choosing an SPGW-C for a UE at session establishment on the Network.

• Responsible for authenticating the user (by interacting with the HSS).

Although the MME natively has the ability to select an SPGW-C for session establishment, it only does it on the basis of the Tracking Area Code (TAC), Mobile Network Code (MNC) and Mobile Country Code (MCC) information of the network which only varies if the UE connection comes from different areas. In order to showcase a Network Slice Selection Scenario based on UE Network Service Request, the selection of SPGW-C is handed over to the NSSF. For this, the MME code had to be modified to include a call to a Communication API that contains python code that translates C function requests into Python methods. It also executes a Remote Procedure Call (RPC) for requesting the SPGW-C and SPGW-U Id to the NSSF. In other words, the API holds the communication channel that allows the MME to interact with the NSSF.

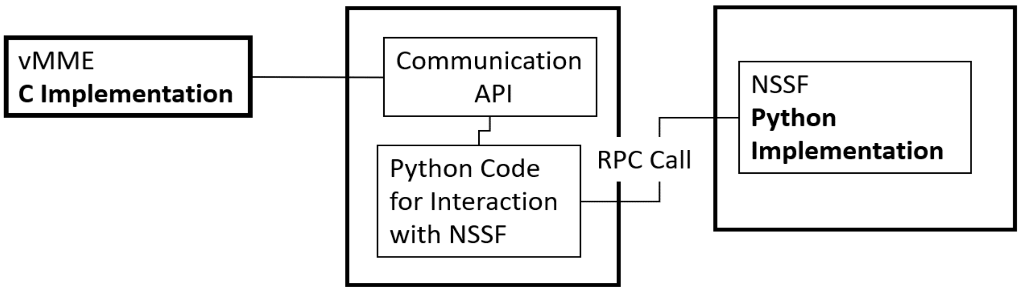


Figure 20. Communication API between MME and NSSF

As mentioned above, the MME is developed in C by Open Air Interface, which requires modification in the *mme_app_select_sgw* procedure to include the ue_context.imsi in the call to the API. The logic of the API will separate the last characters of the IMSI in order to identify the Type of Service that is being requested in the same fashion as explained in *Figure 19*. Furthermore, the *mme_app_send_s11_create_session_req* procedure is modified to include the Data Plane Slice (SPGW-U) IP-Address that was in the response of the NSSF and pack it inside the **bearer_context** as a Packet Data Network Id **pdn_id,** which is forwarded to the serving SPWG-C as part of the session message.

4.4.2 Serving PDN Gateway for the Control Plane SPGW-C.

The relevant functionality of the SPGW-C is to establish a session request for a UE by control message interaction with the MME. It also selects the right SPGW-U from a list of attached PDN in order to initiate PDU session with the UE. In the Open Air Interface original implementation without Network Slicing, the SPGW-C records an **SPGWU_IP_LIST[]** in the attachment order of each SPGW-U. The attachment occurs whenever an SPGW-U "turns on" and is discovered by the SPGW-C. The list is then iterated in the specific order of PDN registration. Once an SPGW-U is selected, an **spgwu_ip.s_addr** is added to the **bearer_context** for PDU session establishment.

In a Network Slice Scenario, Multiple SPGW-C can be connected to a vast number of SPGW-U Data Planes. An SPGW-C is associated to a network service and the connected SPGW-U carry the different QoS for Data Plane Slices, due to this, in order to guarantee a Slice Selection on the basis of UE and Network Service, instead of the above procedure, the SPGW-U logic selection is not handled by the SPGW-C, in its place, the SPGW-C receives the **bearer_context** from the MME with the **spgwu_ip.s_addr** as a **pdn_id.** This IP-Address is used for initiating PDU session directly with the appropriate SPGW-U that was selected by the NSSF.

4.5 XOS Service Definition for Data Plane Slices.

In order to achieve manipulation of SPGW-U (Data Planes) across the multiple levels of the proposed system, the proper definition of VNF attributes must be created in the Management Layer. The SPGW-U XOS Service Data Model has the definitions for the QoS, Network Service and Slice Selection attributes that are used by the whole system. *Table 1* showcases the specific definitions added to the Data Model and *APPENDIX C* has the *xproto* code of the actual Data Model file that represents the XOS service.

Data Model Attributes	**Description**
uprate	Int32 – It represents the uprate QoS in Mb/s
downrate	Int32 – It represents the downrate QoS in Mb/s
service	String – Network Service associated with a Contract. e.g. (HD-VIDEO, WEB)
selectable	Boolean – Sets the Data Plane association to a Network Slice. Used by the NSSF. If **False**, the NSSF will not consider this Data Plane for selection.

Table 1. Data Model Definitions

As can be seen in the above table, **uprate, downrate, service and selectable.** are the attributes defined for the SPGW-U Data Model. The values for uprate, downrate and service are provided by the IBN Tool every time a new contract is created. In the case of the **selectable** attribute, the value is provided by the DPPM during operation. When a TOSCA configuration file is created, it will contain the values for each of these attributes. see *APPENDIX A*

Every XOS Service has a Data Model that represents the abstract state of the system which is defined as an *xproto* file [25]. This type of implementation is particular to the M-CORD Platform for the management of Services.

A **service** attribute is also defined in the SPGW-C XOS Service Data Model with the purpose of achieving a Data Plane slice association based on network service.

5.0 Evaluation and Results

For the evaluation of the proposed functionality, the complete Virtual Mobile Network System that includes a previously developed IBN Tool, M-CORD 4.1, Open Air Interface eNodeB and UE emulator (OAISIM), and Open-Air Interface Core Network was used alongside the implemented modules for Automatic Data Plane Provisioning.

The deployment of the whole system is done in two machines that include a Lab Server for the deployment of CiaB M-CORD 4.1 (XOS, OpenStack, ONOS, Mobile Network VNFs) and a development PC where the IBN Tool (Including the Data Plane Provisioning Module) is running. Both equipment is connected to the laboratory LAN with the Lab Server that houses M-CORD 4.1 having a data-link of 100Mbps.

The evaluation procedure has the following steps. First, a set of contracts for three services are defined using the IBN Tool. This will trigger the creation of Data Plane Network Slices by the Management Layer, each of them configured with proper QoS. Secondly, using the OAISIM, three UE is set up for connecting simultaneously to the Mobile Network where slice selection is verified. Network performance tests are realized in this step. Thirdly, performance tests are realized on the Video Data Plane Slices and Verification of automatic data plane provision is performed in this step. Finally, comparison of results when there are no Network Slices and a Data Plane Provisioning Module does not exist is also presented.

5.1 Network Contract definition in IBN Tool

Table 2 shows the information of three contracts that are defined using the GUI of the IBN Tool. For the VIDEO Contract, two data plane slices are configured using predefined QoS values, Video-1 has an up-rate of 10 Mbps and downrate of 20 Mbps, Video-2 an up-rate of 15 Mbps and downrate of 30 Mbps. The WEB Contract defines three Data Plane Slices with custom QoS. Web-1 has and an up-rate of 5 Mbps and a down-rate of 10 Mbps, Web-2 an up-rate of 2 Mbps and downrate of 4

Mbps, Web-3 an up-rate of 1 Mbps and downrate of 2 Mbps. Finally, a Stress Test Contract is created with an up-rate of 100 Mbps and downrate of 100 Mbps. This contract creates a data plane slice that has the maximum throughput of the CiaB server physical link, its main purpose is to saturate the data plane traffic for the tests that are performed in this chapter. The TOSCA configuration file with the full Data Plane Slices configuration is shown in *APPENDIX A*

Contracts	**QoS**	
	Uprate	**Downrate**
Video	10 Mbps	20 Mbps
	15 Mbps	30 Mbps
Web	5 Mbps	10 Mbps
	2 Mbps	4 Mbps
	1 Mbps	2 Mbps
Stress Test	100 Mbps	100 Mbps

Table 2. Value of QoS defined by the Contracts.

The Management Layer takes the TOSCA information and creates a total of three SPGW-C and six Data Plane Slices that are represented by the SPGW-U VNF. This information is portrayed in *Table 3* where each service defined by the contract takes the form of an SPGW-C and the SPGW-U Data Planes are associated to it based on the *service* attribute defined in the Data Model.

ID	**Service**	**SPGWC-ID**	**SPGWU-ID**
1	Video	SPGWC-1 ID **10.0.8.2**	SPGWU-1 ID **10.0.9.2**
2			SPGWU-2 ID **10.0.9.3**
3	Web	SPGWC-2 ID **10.0.8.3**	SPGWU-3 ID **10.0.9.4**
4			SPGWU-4 ID **10.0.9.5**
5			SPGWU-3 ID **10.0.9.6**
6	Stress-Test	SPGWC-3 ID **10.0.8.4**	SPGWU-4 ID **10.0.9.7**

Table 3. SPGW-C and SPGW-U allocation

5.2 Mobile Network E2E connection and Network Slice Selection.

A mobile network connection starts from the OAISIM VNF. In order to have UE routed to different network slices specific configuration is required for the creation of the UE. *Table 4* shows the IMSI information pertaining to each UE which is used for registration into the network and Data Plane Network Slice Selection and the virtual interface name of the UE that is created by OAISIM.

UE	IMSI
oai1	208930100001111
oai2	208930100001112
oai3	208930100001121

Table 4. IMSI configuration for three UE

The last two digits of the IMSI are used for the slice selection as explained in previous sections. According to this configuration **oa1** and **oa2** are assigned to two different **VIDEO** slices, and **oai3** is assigned to the first **WEB** slice. In order to verify that the Data Plane Slices QoS were correctly deployed and that each user equipment is connected to an isolated Network Slice, two different tests are performed.

5.2.1 iPerf performance test in E2E connection using OAISM (LTE connection).

This test focus on using iPerf [26] to stress the bandwidth of the Data Plane Slice assigned to user equipment. The test setup follows a simple Client-Server setup where an external Lab Web Server has three instances of iPerf running on port 5001, 5002 and 5003 respectively. As each client is represented by a virtual network interface that exists on the OAISIM Virtual Machine, the iPerf command running on the UE needs to specify the proper network interface where the traffic is going to be channeled. e.g. (**iperf –c <Server-IP> -p<Server-Port> -B <Interface-Name>**) where **<Server-IP>** refers to the IP of the machine that has the iPerf server running e.g. (**220.149.42.102**), **<Server-Port>** to one of the ports specified for the server and **<Interface-Name>** refers to the name of the interface that represents the UE e.g.

(oai1). This is done for the three interfaces in order to test simultaneous traffic to the iPerf Server. In order to simulate continuous traffic, the test is modified to run for 60 seconds in just one iteration per test.

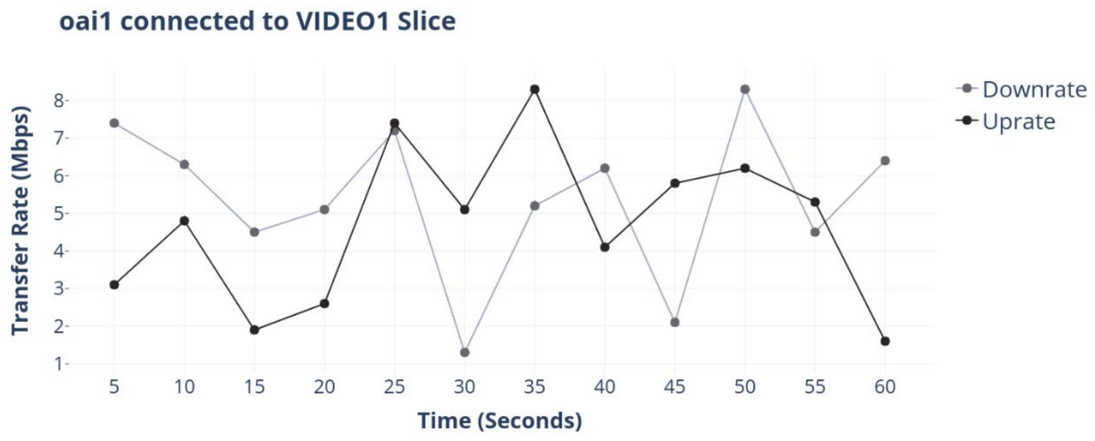


Figure 21. UE1 – VIDEO1 Slice Performance Test (OAISIM)

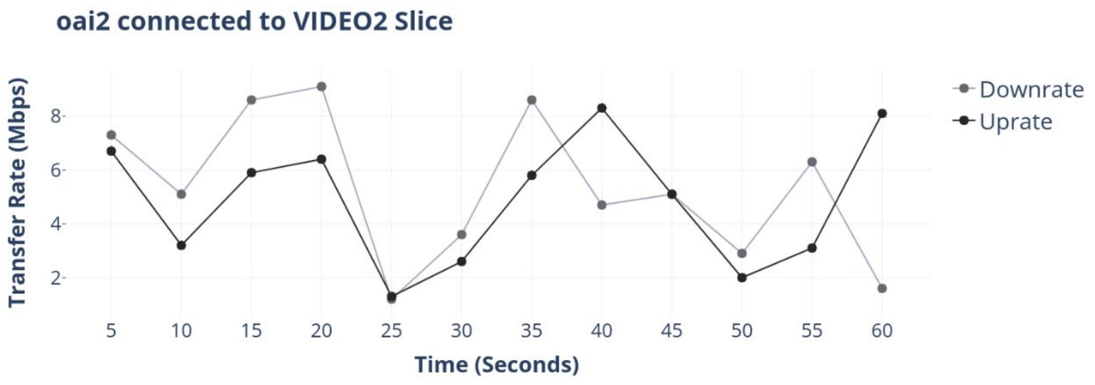


Figure 22. UE2 – VIDEO2 Slice Performance Test (OAISIM)

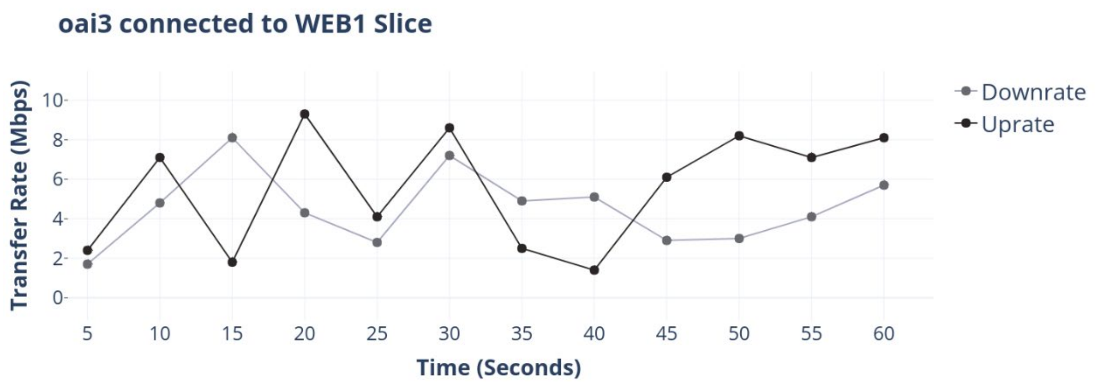


Figure 23. UE3 – WEB1 Slice performance test (OAISIM)

The above results were taken by using the **Wireshark** [27] application on the device that is running the iPerf Server. By configuring Wireshark to listen to a specific interface and ports, it is possible to log the transfer rate of the iPerf tests. The graphs show mixed results in terms of **bandwidth stability**, this is due to the use of an *emulated eNodeB* that is not capable enough for handling multiple virtual user equipment that stresses the transfer rate of the **GTP (GPRS Tunneling Protocol) – Tunnel** created by the LTE connection emulation. Even though it is evident that **Network Slice Selection** is working as intended as each UE is assigned to a different data plane slice, the above results cannot showcase the type of tests that are necessary for the research of this thesis. In order to work with proper bandwidth results, a workaround that simulates and E2E connection with proper transfer rates is proposed.

5.2.2 iPerf performance test using SSH Tunnels for simulating an E2E connection.

To achieve the desired transfer rates specified in the QoS for the High-Level contracts of each slice, the use of **SSH-Tunneling** is required. SSH Tunneling (also known as SSH Port Forwarding) is a feature of SSH which forwards encrypted connections between a local and remote system [38]. It is natively supported by Ubuntu distributions which makes it ideal for the scenario that this research is presenting. SSH-Tunnel is used for each of the virtual interfaces that represent the UE that are connected in the mobile network and replaces the GTP-Tunnel provided by the EPC and OAISIM in order to achieve stable data transfers.

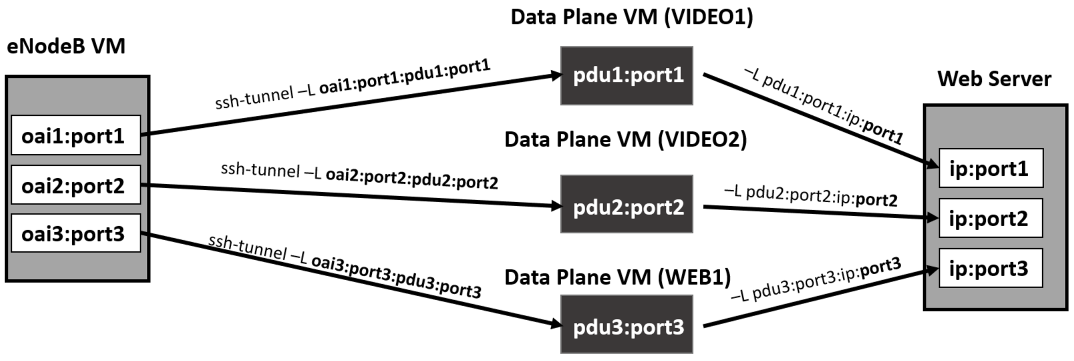


Figure 24. SSH-Tunneling between Virtual UE, Data Plane Slices and Web Server

As can be seen in *Figure 24*, once the virtual equipment is created during the E2E connection, **ssh-tunnels** are configured for each of them in order to facilitate the performance tests and reaching the desired QoS rates specified for each Data Plane. There are multiple ways that an ssh-tunnel can be achieved, in this case, a tunnel needs to be created between three hosts, it is necessary to specify the interfaces and ports that are going to be forwarded during the tunnel configuration. For example, for doing a tunnel between **oa1 interface** (UE1) and the Web Server (For Iperf Tests), the tunnel needs to pass through the VIDEO1 data plane slice. The command will be as follows: ***ssh -L oai1<ip-address>:<port>:pdu1<ip-address>:<port> VIDEO1 ssh -L pdu1<ip-address>:<port>:WebServer<ip-address><port> -N WebServer.*** With this command, the virtual IP-address of the UE1 is used as a tunnel starting point and the WebServer IP-address as the ending point of the tunnel. In order to replicate the iPerf test, the WebServer IP-address needs to be changed to the UE1 IP-address as all the traffic that goes through that interface and port will be redirected to the WebServer, e.g. (**iperf -c <oa1-IP> -p <ssh-port>**) there is no need to bind to the interface anymore. The results of this test can be seen in the following graphs.

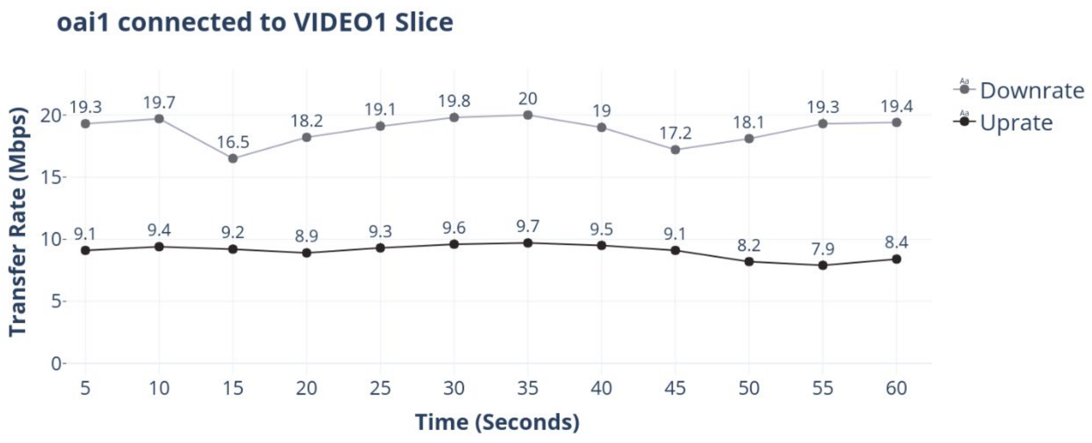


Figure 25. UE1 – VIDEO1 Slice Performance Test (SSH-Tunnel)

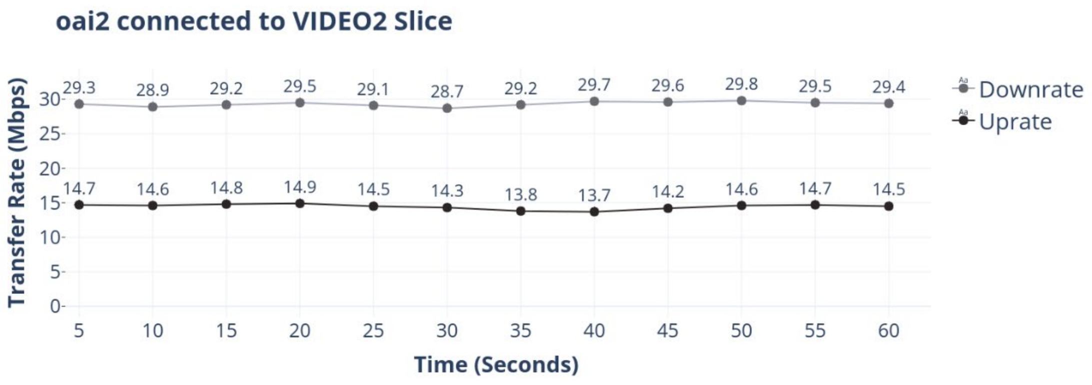


Figure 26. UE2 – VIDEO2 Slice Performance Test (SSH-Tunnel)

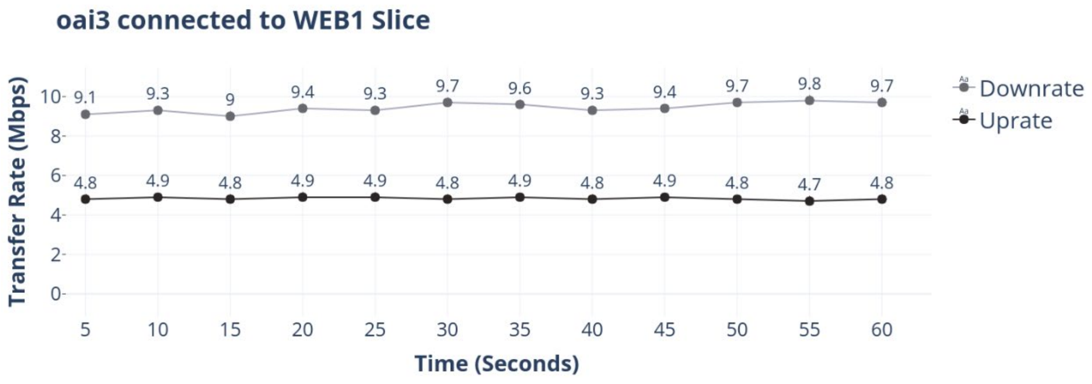


Figure 27. UE3 – WEB1 Slice performance test (SSH-Tunnel)

By exchanging the GTP-Tunnel to an SSH-Tunnel the results are closer to the QoS Policies specified for each Data Plane Slice. It is important to clarify that this is just a workaround in order to achieve good transfer rates for the tests that are relevant to Data Plane slices due to lack of good eNodeB emulation. From this point forward the remaining tests performed for this Thesis are achieved by utilizing SSH-Tunnels.

5.2.3 Video Streaming with Adaptive Bitrate (ABR) Test.

The best scenario for show-casing the isolation of network slice traffic is by testing the capabilities of video streaming on the UE. Adaptive bitrate streaming is a technique used in streaming multimedia over computer networks that are mostly based on HTTP and designed to work efficiently over large distributed HTTP networks such as the Internet [28]. Basically, the source content is encoded at diverse-multiple bit rates, then each of the different bit rate segments is divided into small parts. The client is aware of the available streams at differing bit rates, by a manifest that is encoded into the video file. When video streaming starts, the client requests the segments from the lowest bit rate stream (lowest Quality). If the client finds the download speed is greater, then it will request the next higher bit rate segments until it achieves maximum quality. If the network performance degrades, the bitrate will also go to a lower tier to avoid pausing the stream. [28]. This technique is used widely on HTTP based video streaming on the Internet. In order to achieve a real-life scenario for Data Plane Network Slicing, video files encoded in multiple bitrates using HTTP live streaming ABR technique (HLS) [29] are placed on the same Web Server where the previous iPerf test was performed. For the UE to be able to consume the Video, the **ffplay** tool that is part of the **FFmpeg** [30] package was installed on the OAISIM virtual machine. An important function of this tool is the capability of running without displaying Video by passing the option **-nodisp** during the execution command.

The tests are performed using **ffplay** while routing traffic through **oai1** interface and doing a simultaneous iPerf test from **oai2** in a similar fashion as the previous performance tests. A second test is performed by using **ffplay** while routing traffic to **oai2** and using iPerf on **oai1** interface. The main reason for doing the tests in this fashion is that **ffplay** cannot be used simultaneously while connecting using two different network interfaces and we need a way to inject traffic in both slices at the same time in order to showcase that traffic does not interfere between each other.

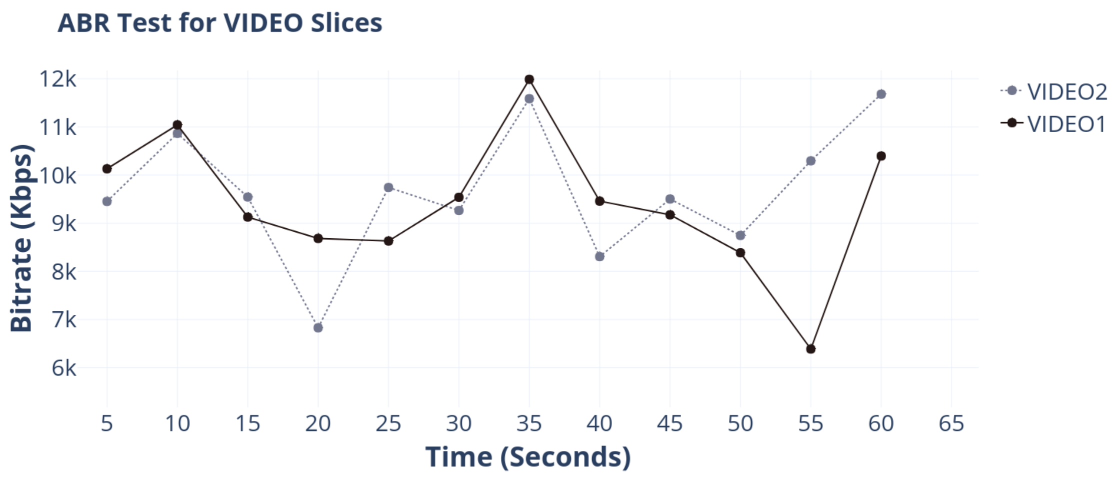


Figure 28. Adaptive Bitrate Test for VIDEO1 and VIDEO2 Slice

The above graph shows the results of video streaming using **ffplay** for both Video Slices. The variation of the bitrate is similar for both tests, as the same Video File is being used for the streaming. Even though in some cases the bitrate dipped below **7000Kbps**, it kept the resolution of 1920 x 1080p through the length of the test.

Normally Full-HD streaming requires a minimum of **4000kbps** which is way below the results presented in the graph. Both Video slices have enough down-rate QoS to keep a good quality stream.

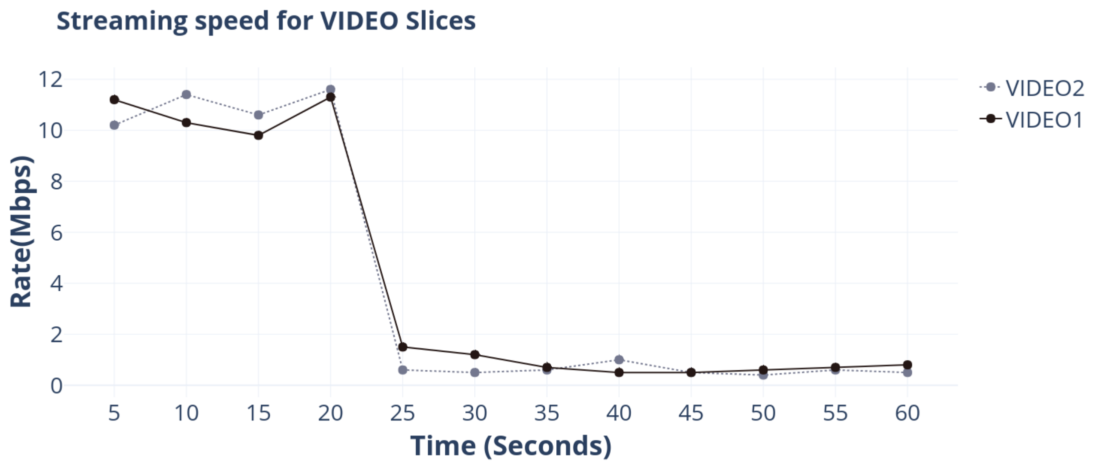


Figure 29. Streaming Speed Test for VIDEO1 and VIDEO2 Slice

An interesting result can be seen on the streaming speed test using **ffplay.** As soon as the Video Streaming starts, the speed rate stays on the range of **11 Mbps** for both Video Slices, but after around 25 seconds of streaming, the speed rate goes down and normalize in between **1Mbps** and **500Kbps.** The main reason for this to happen is the buffering that the video player requires to do at the beginning of the stream, once the buffer is full, the speed rate goes down to the values shown in the graph. It is important to take notice that neither slice went up more than **12Mbps** for the rate of the streaming, this is due to the quality of the Video File that is being streamed, as 1080p Full HD video does not require that much bandwidth to work with, both VIDEO slices have more than enough room to handle high-quality Video Streaming from multiple UE without problem.

5.3 Automatic Data Plane Provisioning when a Slice is Saturated.

The aim of the evaluation and test results is to verify that the Network Monitoring Agent is triggering the Data Plane Provisioning Module operation when a Data Plane is saturated with network traffic. Due to the current limitations posed by OAISIM, the system becomes unstable when more than three UE are created for connection tests.

Because of this, a re-arrangement for the current UE is necessary to force them to connect to a single VIDEO Slice. The new configuration of the UE can be seen in *Table 5*.

UE	IMSI
oai1	208930100001111
oai2	208930100001211
oai3	208930100001311

Table 5. New IMSI configuration for connecting to a single Video Slice

The above configuration shows that the three UE is assigned to a single VIDEO slice (VIDEO1). Due to this, there is no requirement for multiple Video slices for this test. The new arrangement of slices can be seen in the table below. An advantage of working with the IBN Tool is that Data Plane Slices can be easily managed from the Application Layer. Deleting or creating new slices does not pose a big downtime during the tests.

Contracts	**QoS**	
	Uprate	**Downrate**
Video	10 Mbps	20 Mbps
Web	5 Mbps	10 Mbps
	2 Mbps	4 Mbps
	1 Mbps	2 Mbps
Stress Test	100 Mbps	100 Mbps

Table 6. Single Video Slice

In order to trigger the functionality of the DPPM, VIDEO1 Slice must reach the peak of traffic and saturate the Network Utilization of its "physical" link. The QoS set for this slice have a **down-rate** of **20Mbps**, and maximum **link rate** that can support is **40Mbps.** As explained in previous sections, the **link rate** limitation for each Data Plane Slice is forced and particular for achieving the results of this thesis. As only three UE can be connected at the same time, it is necessary to saturate the link before the 3rd UE connects. The test procedure is as follows. **oai1** and **oai2** connect first to the mobile network and are assigned to VIDEO1 Slice. The iPerf tool is used for both UE virtual interfaces with the purpose of filling the bandwidth limit of the Data Plane. The Network Monitoring Agent is constantly probing the sFlow-RT collector for the status of all the data planes. After approximately 15 seconds of the iPerf Test, the DPPM successfully triggers its functionality and creates a new Data Plane Slice and disables the selection of the saturated Data Plane. At this point, **oai3** connects to the Mobile Network and starts a similar Video Streaming Test using the **ffplay** tool. Streaming is achieved without the interfering of **oai1** and **oai2** traffic test as the last UE is connected to a different Data Plane in a transparent fashion.

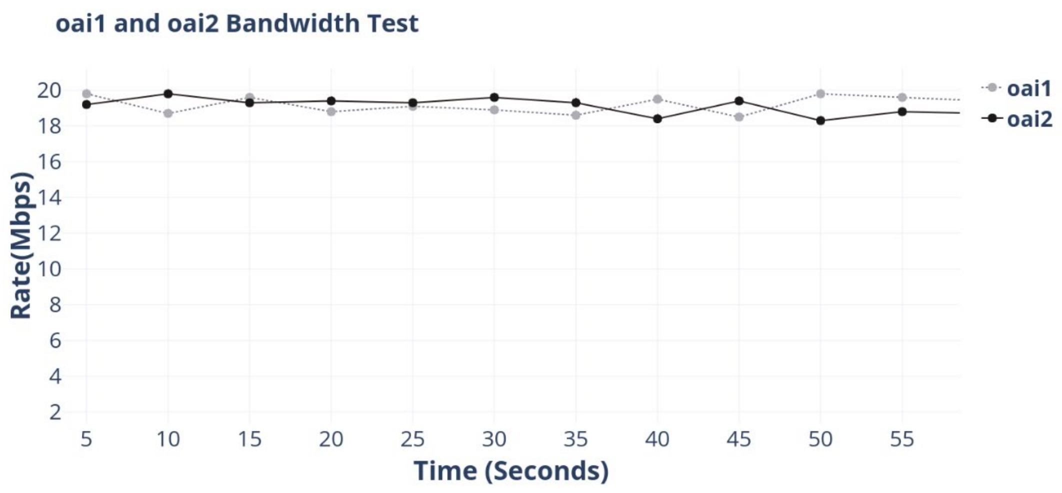


Figure 30. iPerf bandwidth stress test for VIDEO1 Slice

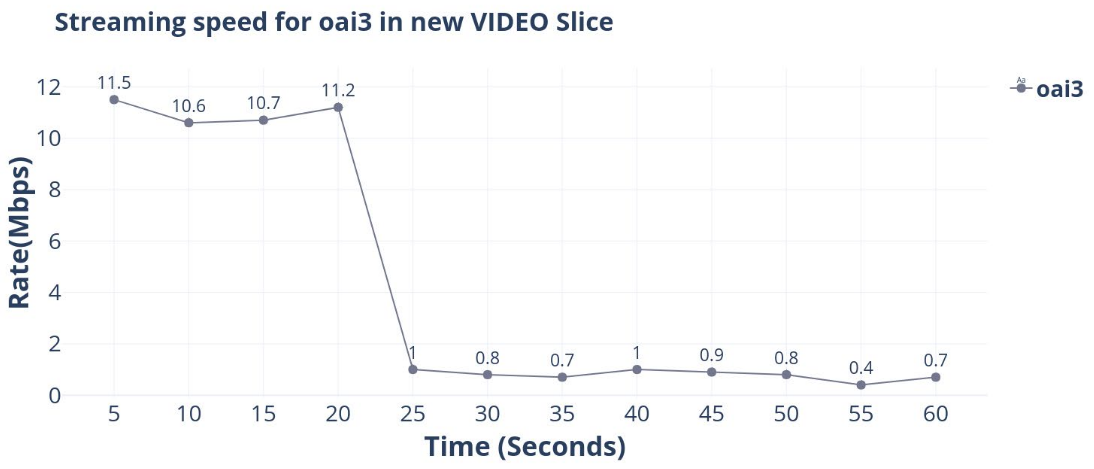


Figure 31. Streaming Speed Test for oai3 UE in new VIDEO1 Slice

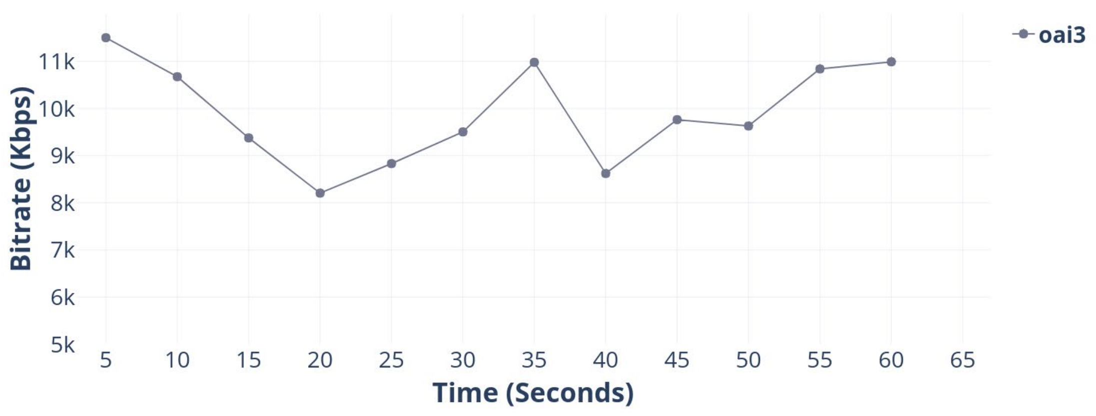
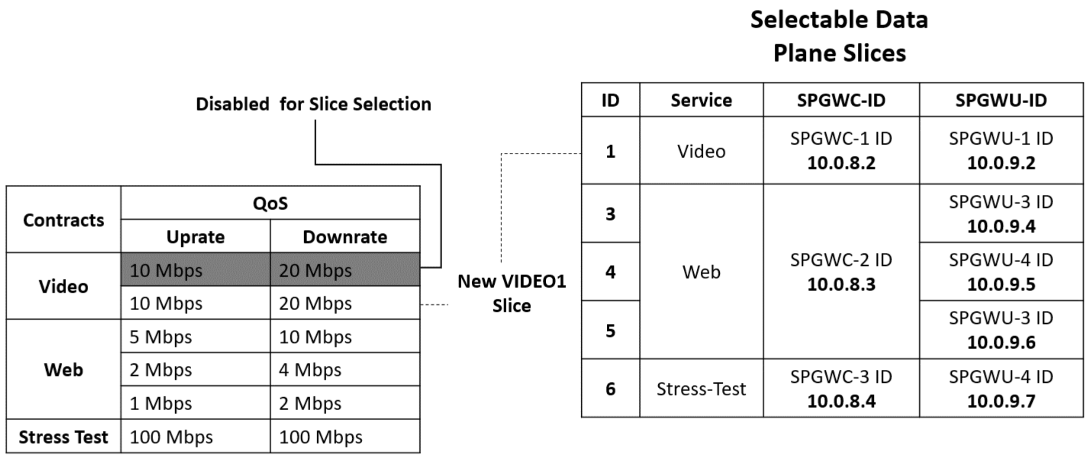


Figure 32. Adaptive Bitrate Test for oai3 UE in new VIDEO1 Slice

As seen in *Figure 31 and 32* the results of the Video Streaming test are similar to the ones performed in the previous section. Full video quality is achieved in both Bitrate and Stream Speed because the **oai3** user equipment is assigned to a complete **new** VIDEO1 Slice. This slice keeps the QoS and Data Model details of the **saturated** VIDEO1 Slice and is automatically provisioned by the DPPM in order to serve a new UE connection. The **saturated** VIDEO1 Slice is not included in the Network Slice Selection process by the NSSF due to its **selectable** property being disabled by the DPPM. Figure 33 has the representation of how the current slices are viewed by the Mobile Network System.



Contracts	QoS	
	Uprate	Downrate
Video	▨ 10 Mbps	▨ 20 Mbps
	10 Mbps	20 Mbps
Web	5 Mbps	10 Mbps
	2 Mbps	4 Mbps
	1 Mbps	2 Mbps
Stress Test	100 Mbps	100 Mbps

ID	Service	SPGWC-ID	SPGWU-ID
1	Video	SPGWC-1 ID **10.0.8.2**	SPGWU-1 ID **10.0.9.2**
3	Web	SPGWC-2 ID **10.0.8.3**	SPGWU-3 ID **10.0.9.4**
4			SPGWU-4 ID **10.0.9.5**
5			SPGWU-3 ID **10.0.9.6**
6	Stress-Test	SPGWC-3 ID **10.0.8.4**	SPGWU-4 ID **10.0.9.7**

Figure 33. Representation of new VIDEO1 slice and disabled VIDEO1 slice

It is important to remark that the "**disabled**" VIDEO1 slice is fully operational and the UE connected to it will continue to have network connection until they un-register from the network. But this slice will not serve any new UE connection request and after a period of time will be assigned to the Unused Data Plane Repository. In its place, the **new** VIDEO1 slice is essentially a twin or replica with the same QoS policies and Data Model attributes.

5.4 Results in a Non-Sliced scenario and no Automatic Data Plane Provisioning.

The benefits of the proposal made in this thesis are completely visible if comparison with results obtained by not including an NSSF and a DPPM is showcased. The test is divided into two categories. Firstly, tests performed in a single Data Plane Slice are presented, and secondly, tests performed in a Network Sliced Scenario with no Automatic Data Plane Provision are presented.

5.4.1 Non-Sliced Scenario.

For this test, a single Data Plane exists for serving three simultaneous UE. The test focus only on the **downrate** for iPerf and Video Streaming. The Data Plane Slice used for this scenario is the **Stress Test** with 100Mbps, all three UE are connected to this Data Plane Slice. Two UE (**oai1** and **oai2**) perform an iPerf test to the Lab Web Server and oai3 perform **ffplay** video streaming.

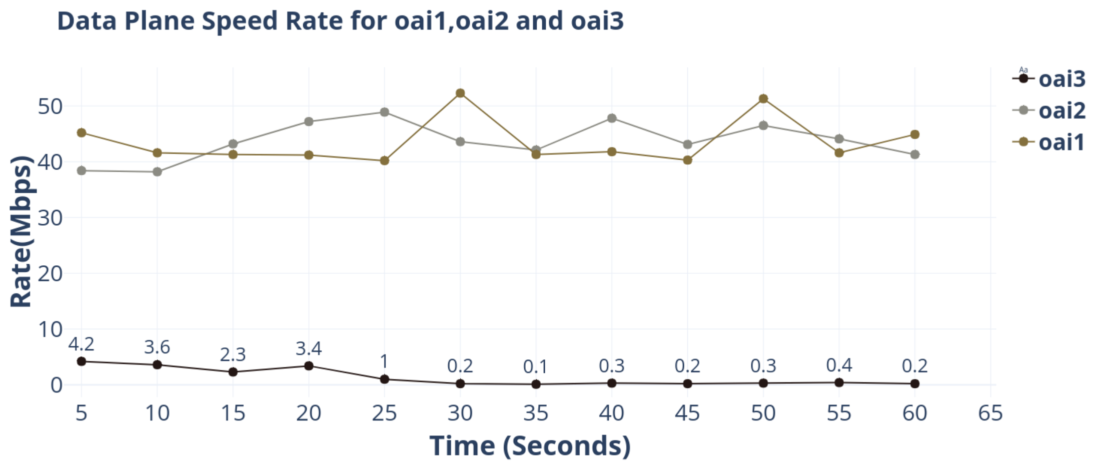


Figure 34. Streaming Speed Test for oai1, oai2 and oai3 in Stress Test Slice

The behavior of the test shows that iPerf tries to occupy the maximum traffic as possible during execution. The physical link to the network only allows a maximum of 100Mbps, so the rate is divided between the three UE. oai1 and oai2 network rate hover in between the 40 to 50Mbps range. The interesting result can be seen for oai3 which is streaming video by using the ffplay tool. It tries to take as much bandwidth as possible but is not able to compete with the iPerf test, and as can be seen, it cannot reach the same speed rate of previous streaming tests. Of course, when the buffer is filled, the speed rate goes down to less intensive values. The effect of the reduced bandwidth for video streaming can be seen in the following graph.

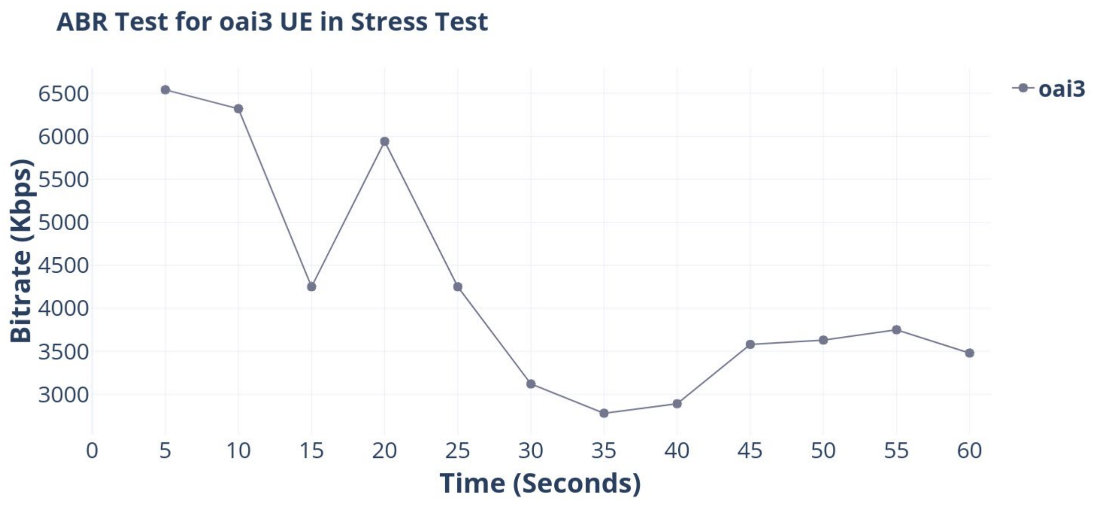


Figure 35. Adaptive Bitrate Test for oai3 UE in Stress Test Data Plane

As can be seen, the saturation of the Data Plane does not allow optimal Video Streaming performance. The Overall Bitrate achieved during this test is less than half than the previous test done in Data Plane Slices that had less downrate (20Mbps) in comparison with the current Data Plane(100Mbps). Even though the video manages to play in 1080p (Full-HD) during the first few seconds of the stream, the speed rate was so constrained that the video switched automatically to 720p (HD) quality in order to keep streaming without having to re-buffer the video. These tests make evident that if there are no different Network Slices tailored for multiple UE and network services, even if the data link has enough capacity for high-quality video streaming, the performance of the UE connection will reduce if demanding network consumption activities e.g. (File downloading, P2P) are mixed into a single Data Plane.

5.4.2 Non-automatic Data Plane Provisioning Scenario.

The following test is performed in a similar fashion as section 5.3 tests related to Data Plane Provisioning. The main difference for this case is that, instead of VIDEO1 Slice, the **Stress Test** slice is used. Again, **oai1** and **oai2** perform iPerf tests with the purpose of filling the bandwidth limit of the Data Plane. Because there is no Automatic Data Plane Provisioning, when **oai3** connects to the Mobile Network it is assigned to the same Data Plane as **oai1** and **oai2.** The test shows result far worse than in the case of Non-Sliced Scenario. As the iPerf test was already running before the **oai3** UE connected into the network, the moment that the video stream started there was not enough bandwidth to reach 1080p Full-HD quality, and the stream started in 720p. Also, during the test, the bitrate went down to the low **1000kbps** switching the quality of the stream to Standard Definition.

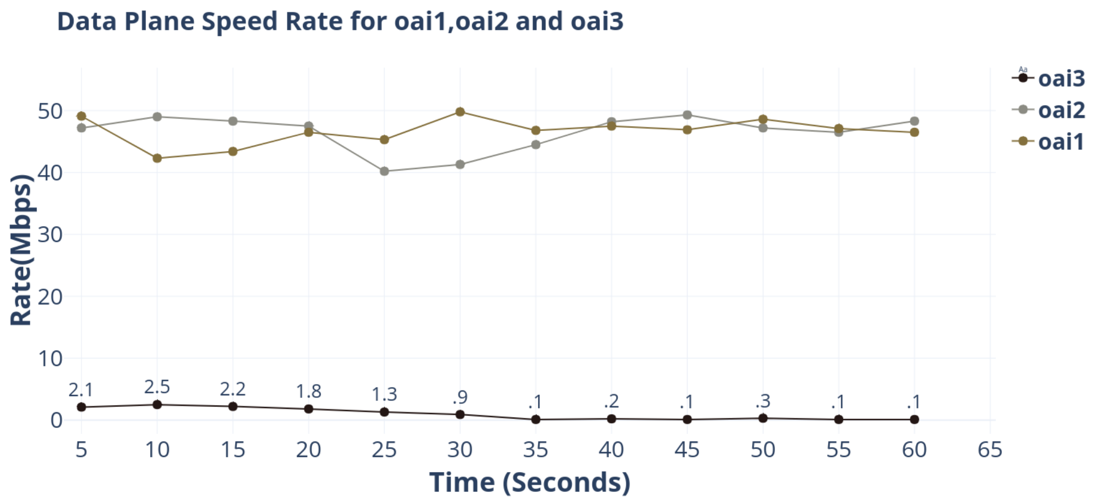


Figure 36. Streaming Speed Test for oai1, oai2 and oai3 in Stress Test Slice in non-Automatic Data Plane Provisioning Scenario

Figure 36 and 37 results show that even if Network Slicing and Network Slice Selection is enabled, if the network is not capable to adapt to the current situation, it risks on assigning new UE connection to already saturated Data Planes, thus diminishing the user experience.

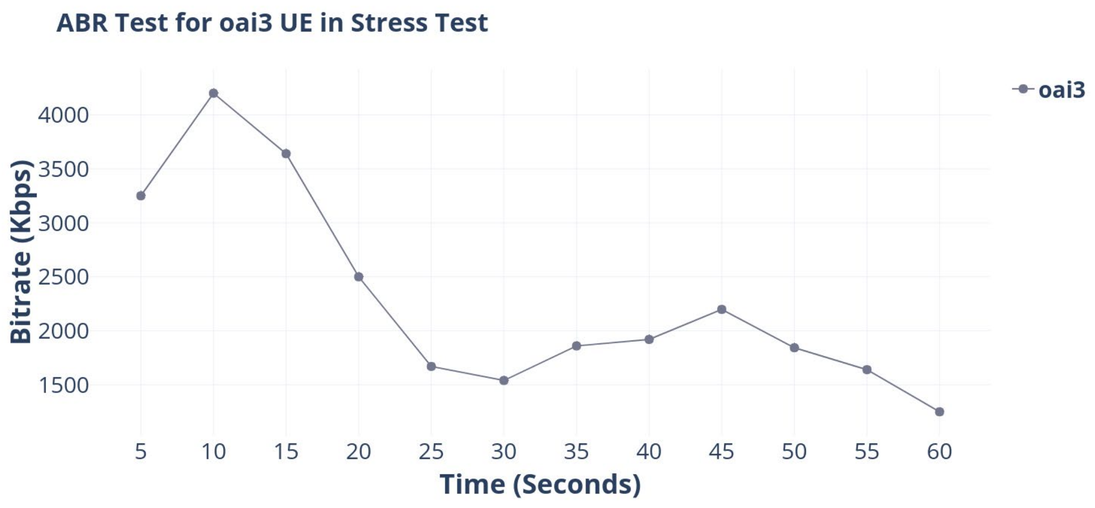


Figure 37. Adaptive Bitrate Test for oai3 UE in Stress Test Data Plane in non-Automatic Data Plane Provisioning Scenario

5.5 Performance Test when the System is Overloaded.

The Data Plane Provisioning Module is designed to avoid over-provisioning of Data Plane slices in cases where there are no system resources available. An ideal scenario will never over commit resources of a Compute Node above the recommended values (see *section 4.1*). The results presented below are particular to the testbed of this thesis where compute resources are constrained, and Data Plane overprovision can be easily achieved. To reach this stage, sixteen Data Plane slices have been manually provisioned using the orchestrator UI (XOS). The focus of this test is not to showcase End-to End connection, but to verify the bandwidth performance of the compute node when 16 Data Planes are doing an iPerf test simultaneously.

Figure 38 has the condensed results of the *average* of ten iPerf tests related to the increasing number of Data Plane Slices. The test starts with 2 Data Planes and increments by 2 until it reaches the 16 Data Planes. As the focus is the performance of the compute node that houses the Data Plane Slices, bandwidth is measured for the whole compute node link (100mbps max) instead of individual data planes, as there are no QoS policies in place.

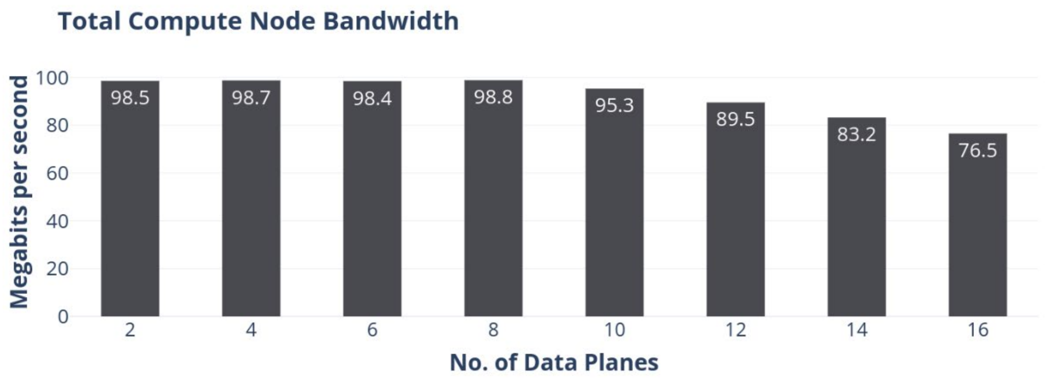


Figure 38. Total compute node bandwidth in relation to the number of Data Plane Slices that are deployed.

The above figure shows that provisioning Data Plane Slices without considering the resources of the Compute Node, has a direct relation to network performance, especially if the compute node is also a virtual entity, as nested virtualization has an inherent impact on performance. According to the graph, the compute node bandwidth starts to degrade around the mark of 10 data plane slices, until it reaches the lowest possible value with 16 Data Plane Slices.

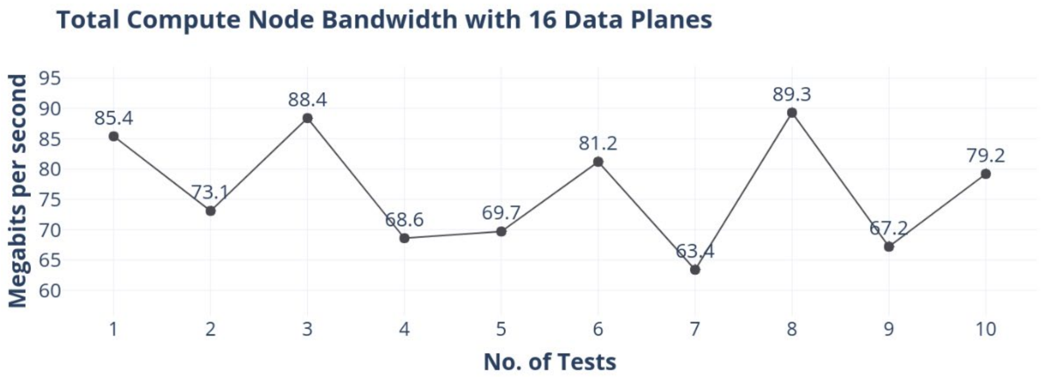


Figure 39. Total compute node bandwidth with 16 Data Planes.

Figure 39 shows the case when 16 Data Planes are doing an iPerf test simultaneously. This graph represents the last record from *Figure 38* before calculating the average values of the data transmission. As can be observed, due to overprovision of slices, the compute node is overloaded and cannot provide steady transfer rates for even simple iPerf tests.

6.0 Conclusion and Future Work

6.1 Conclusion

The shift to virtualization technologies has brought with it a big change in the way Mobile Networks are managed and deployed. Software Engineers have a bigger role to play as network logic has moved from dedicated hardware to commodity equipment. This aspect is made evident with the increased proliferation of Open Source Projects [9,31,32] that aim to bring new/optimized functionality to the network. In this thesis, a mechanism for Automatic Data Plane Provisioning integrated into a Network Sliced Virtual Mobile Network system is proposed in order to leverage the network load during consumption of network services.

The benefits of Automatic Data Plane Provisioning have been detailed in the literature. It not only provides a way for monitoring the status of the mobile network during operation but also brings a level of automatic decision for adapting to the consumption of network services. The clearest example of showcasing the added value of this mechanism is the Video Streaming scenario. According to [33], Netflix consumes 15% of all internet bandwidth in the world, next is followed by HTTP Media Streaming with 13.1% and in Third place is Youtube with 11.4%. The top three places for global internet bandwidth consumption is related to Video Streaming. By 2018, there were 3.7 billion mobile internet users in the world and the topmost demanding mobile traffic service is Youtube video with 35% of the total use of mobile internet [34]. This data makes evident the need for a mechanism for adapting to high consuming network services.

Taking into consideration HTTP media streaming services in the evaluation tests done for showcasing the functionality of Automatic Data Plane Provisioning, it is easy to observe that the network can be prepared for cases when UE wants to consume High-Quality video services. This is done by setting up Data Planes when the network load goes above the standard QoS policies that were set up by the Network Contracts. Also, during the comparison test, when there is no Automatic Data Plane Provision mechanism it is clear that the network performance degrades heavily when multiple UE are consuming high bandwidth network services.

The mechanism relies heavily on the deployment of the M-CORD Platform [9] and the use of an IBN Tool [11] for the definition of High-Level Contracts and configuration of QoS for Network Slices. Although it follows independent functionality (sFlow and sFlow-RT) in order to Monitor and calculate the Link Utilization of the Data Plane Slices, it still requires integration with the Virtual Mobile Network System presented in this literature in order to apply the TOSCA configuration into the physical system and also interact with the Network Slice Selection Function. This is not to be considered a weak point, as the M-CORD platform is constantly maturing, and more partners are interested in the functionality that it provides. Also, the use of TOSCA configuration is standard for many operators and open source Network Orchestrators [35]. Which means that migration or portability of the Automatic Data Plane Provisioning functionality is possible.

6.2 Future Work

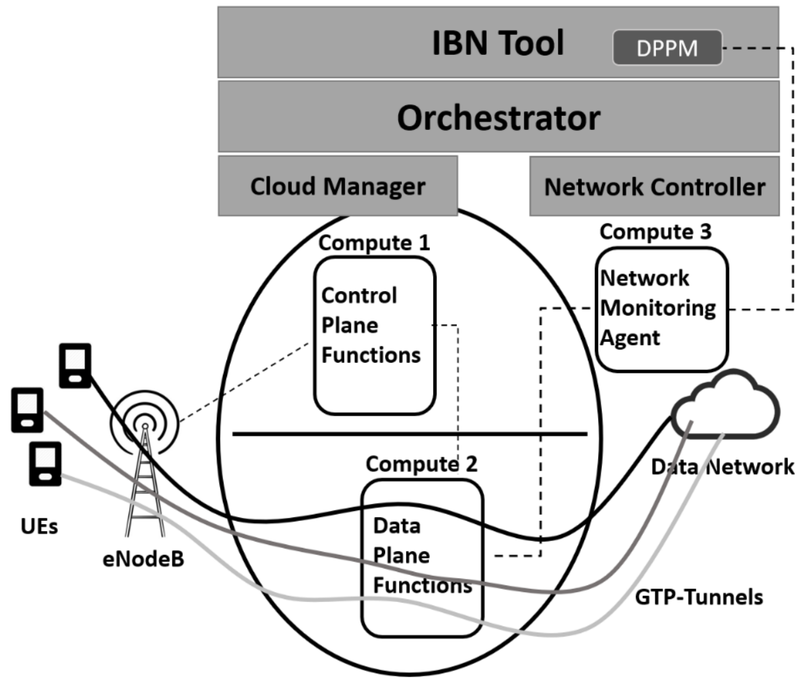


Figure 40. Abstracted view of the system

Limited compute resources are the main constraint that the solution presented in this thesis could face for the deployment of Data Plane Slices. *Figure 40* shows an abstracted view of the proposed system considering the three virtual compute nodes that are created. As the testbed is housed in a single physical server, the real limit of how many compute nodes can be created is dependent on the RAM and CPU cores of the physical system (Currently 64 Gigabytes of RAM with 40 CPU cores) this is barely enough for creating 3 compute nodes that house multiple Virtual Machines that represent the VNF of the EPC and Data Plane Slices. If physical resources were limitless, multiple virtual compute nodes could be created in order to load balance the provision of Data Plane Slices. The load balancing is inherent to the orchestrator and follows a *LeastUsedResources* approach in order to select a compute node for the deployment of *VMs*. Also, the selection of Data Plane Slices is still be handled by the NSSF regardless of the compute node where the slices are provisioned, as each compute node is part of the Virtual Tenant Network (VTN) that is managed by the Orchestrator together with the Cloud Manager and Network Controller. *Figure 41* shows a scenario where multiple compute nodes house the data plane slices for connecting to the Data Network.

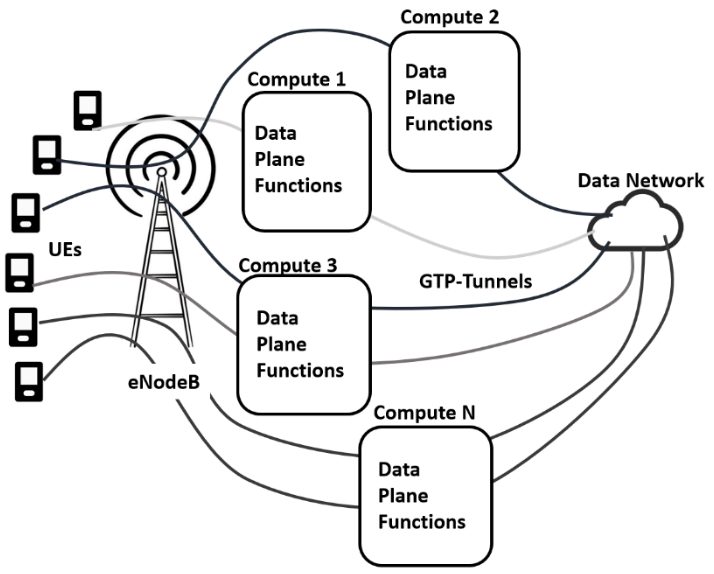


Figure 41. Multiple compute nodes for data plane slices

The above figure focuses only on the compute nodes that handle the deployment of data plane slices. Although this scenario may be possible if the physical resources are infinite, is not feasible and not cost-effective. That is why better management of existing resources is required.

The thesis proposal requires interaction with a Resource Manager in order to calculate the available physical resources for accommodating the Data Plane Slices. This consideration exists as part of the work done in the Thesis, but it will benefit more if the Resource Manager had the functionality of providing an instantiation of resources in real time. Currently, it relies on calculations done by taking into account the physical resources of the Compute Node and static values for the type of VMs that are created. In a real scenario, Virtual Network Functions could be encased inside containers which require fewer system resources for operation and in some cases, they are able to upscale or downscale their resources on the basis of the current system load. This poses a challenge for the current method of calculating resources, which will require additional functionality to be added to this module in order to correctly measure the Data Plane Slices resource requirements.

REFERENCES

[1] Indika A. M. Balapuwaduge and Frank Y. Li, “Cellular Networks: An Evolution from 1G to 4G”, Centre for Integrated Emergency Management, July 2018

[2] Yousaf Bin Zikria, Sung Won Kim, Muhammad Khalil Afzal, Haoxiang Wang and Mubashir Husain Rehmani “5G Mobile Services and Scenarios: Challenges and Solutions” Sustainability, 2018

[3] Adel Nadjaran Toosi, Redowan Mahmud, Qinghua Chi and Rajkumar Buyya “Management and Orchestration of Network Slices in 5G, Fog, Edge and Clouds” Fog and Edge Computing: Principles and Paradigms (Book), pp 79-101, 2019.

[4] Intel “Evolved Packet Core (EPC) for Communications Service Providers”, Solutions Reference Architecture, Revision 1.2 May 2016.

[5] 3GPP, TS 23.501 V15.0.0, Technical Specification Group Services and System Aspects; System Architecture for the 5G System; Release 15, Dec 2017

[6] Christos Bouras, Anastasia Kollia, Andreas Papazois, “SDN & NFV in 5G: Advancements and Challenges”, 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), April 2017

[7] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Steven Latre, Marinos Charalambides, and Diego Lopez, “Management and Orchestration Challenges in

Network Functions Virtualization”, IEEE Communications Magazine, Volume: 54, Issue: 1, pp 98-105, January 2016.

[8] ETSI, ETSI GS. NFV-MAN 001: Network Function Virtualisation (NFV): Management and Orchestration, 2014.

[9] Open Network Foundation. Mobile-Central Office Rearchitected as a Datacenter (M-CORD) v4.1, [Online; accessed 06-2019] From https://guide.opencord.org/cord-4.1/

[10] Javier Diaz Rivera, Talha Ahmed Khan, Mehmood Asif, Wang-Cheol Song.

“Network Slice Selection Function on M-CORD”, KNOM Review '18-02 Vol.21 No.02, 2018

[11] Asif Mehmood, Javier Diaz Rivera, Talha Ahmed Khan, Wang-Cheol Song, “An intent-based mechanism to create a network slice using contracts”, Proceedings of Symposium of the Korean Institute of Communications and Information Sciences, 2018. (pp. 180-181).

[12] Ala’a Al-Habashna, Gabriel Wainer, Stenio Fernandes, “Improving Video Streaming over Cellular Networks with DASH-based Device-to-Device Streaming” International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), September 2017

[13] Vinod Kumar Choyi, Ayman Abdel-Hamid, Yogendra Shah, Samir Ferdi, Alec Brusilovsky, “Network Slice Selection, Assignment and Routing within 5G Networks”, IEEE Conference on Standards for Communications and Networking (CSCN), 2016.

[14] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J. Ramos-Munoz, Javier Lorca, and Jesús Folgueira, “Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges”, IEEE Communications Magazine, Volume: 55, Issue: 5, pp. 80-87 May 2017.

[15] Peter Rost, Christian Mannweiler, Diomidis S. Michalopoulos, Cinzia Sartori, Vincenzo Sciancalepore, Nishanth Sastry, Oliver Holland, Shreya Tayade, Bin Han, Dario Bega, Danish Aziz, and Hajo Bakker, “Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks”, IEEE Communications Magazine,Volume: 55, Issue: 5, pp 72-79, May 2017.

[16] Aman Jain, Sadagopan N S, Sunny Kumar Lohani, Mythili Vutukuru, “A Comparison of SDN and NFV for Re-designing the LTE Packet Core”, IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2016.

[17] Tulja Vamshi Kiran Buyakar, Anil Kumar Rangisetti, Antony Franklin A, and Bheemarjuna Reddy Tamma, “Auto Scaling of Data Plane VNFs in 5G Networks”, 13th International Conference on Network and Service Management (CNSM), 2017.

[18] Niels L. M. van Adrichem, Christian Doerr and Fernando A. Kuipers, “OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks”, IEEE Network Operations and Management Symposium (NOMS), 2014

[19] Shihabur Rahman Chowdhury, Md. Faizul Bari, Reaz Ahmed, and Raouf Boutaba, “PayLess: A Low-Cost Network Monitoring Framework for Software Defined Networks”, IEEE Network Operations and Management Symposium (NOMS), 2014.

[20] Muhammad Afaq, Shafqat Rehman, Wang-Cheol Song, “Large Flows Detection, Marking, and Mitigation based on sFlow Standard in SDN” Journal of Korea Multimedia Society. pp. 189-198, 2015.

[21] Nèstor Bonjorn López Ferran Cañellas Cruz, “VIM Adaptation Layer for CORD”, M.Sc. Thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, 2018.

[22] Eurecom, Open Air Interface Alliance. Open-Air Interface Project. [Online; accessed 06-2019] from https://gitlab.eurecom.fr/oai/

[23] sFlow.org, "sFlow - Making the Network Visible". [Online; accessed 06-2019] from https://sflow.org/

[24] Nevyan Neykov, “Real-time detection and mitigation of flood attacks in SDN networks”, 2017.

[25] Protocol Buffers. “A language-neutral, platform-neutral extensible mechanism for serializing structured data”. [Online; accessed 06-2019] from https://developers.google.com/protocol-buffers/

[26] iPerf. “A tool for measuring TCP and UDP network performance”, [Online; accessed 06-2019] from https://iperf.fr/ *Software.

[27] The Wireshark Foundation, “Wireshark”, [Online; accessed 06-2019] from https://www.wireshark.org/ *Software.

[28] Saamer Akhshabi; Ali C. Begen; Constantine Dovrolis. “An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP” In Proceedings of the second annual ACM conference on Multimedia systems (MMSys '11). New York, NY, USA. 2011.

[29] Pantos, R.P. "HTTP Live Streaming draft-pantos-HTTP-live-streaming-19". Network Working Group, 2016.

[30] FFmpeg, “ffplay”, [Online; accessed 06-2019] from: https://ffmpeg.org/ffplay.html. *Software.

[31] ONF, “ONOS -A new carrier-grade SDN network operating system”, [Online; accessed 06-2019] from https://onosproject.org/.

[32] OpenStack, “Tacker - OpenStack NFV Orchestration”, [Online; accessed 06-2019] from: https://wiki.openstack.org/wiki/Tacker

[33] Sandvine, “The Global Internet Phenomena Report”, 2018.

[34] Sandvine, “The Mobile Internet Phenomena Report”, 2019.

[35] sdxcentral “TOSCA Orchestration for Service Providers " [Online; accessed 06-2019] from https://www.sdxcentral.com/networking/nfv/definitions/tosca-orchestration/

[36] “OpenStack Overcommit Ratio” [Online; accessed 06-2019] from: https://docs.openstack.org/arch-design/design-compute/design-compute-overcommit.html

[37] CISCO, “Best Practices in Core Network Capacity Planning”, White Paper [Online; accessed 06-2019] from https://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728551.html

[38] Bill Brassfield, “Advanced SSH Tunneling”, White Paper [Online; accessed 06-2019] from https://docplayer.net/10733625-Advanced-ssh-tunneling-by-bill-brassfield-dev-ops-technical-consultant-taos.html

APPENDIX A – TOSCA Definition of Slices

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: created by platform-install, need to add M-CORD services later

imports:
   - custom_types/xos.yaml
   - custom_types/slice.yaml
   - custom_types/site.yaml
   - custom_types/image.yaml
   - custom_types/flavor.yaml
   - custom_types/network.yaml
   - custom_types/networkslice.yaml
   - custom_types/vspgwuservice.yaml
   - custom_types/vspgwuvendor.yaml
   - custom_types/vspgwutenant.yaml
   - custom_types/vspgwcservice.yaml
   - custom_types/vspgwcvendor.yaml
   - custom_types/vspgwctenant.yaml

topology_template:
  node_templates:


    m1.medium:
      type: tosca.nodes.Flavor
      properties:
        name: m1.medium

    image-oai:
      type: tosca.nodes.Image
      properties:
        name: image-oai

    service#vspgwu:
      type: tosca.nodes.VSPGWUService
      properties:
        name: VSPGWUService

    oai_vspgwu:
      type: tosca.nodes.VSPGWUVendor
      properties:
        name: oai_vspgwu

    service#vspgwc:
      type: tosca.nodes.VSPGWCService
      properties:
        name: VSPGWCService

    oai_vspgwc:
      type: tosca.nodes.VSPGWCVendor
      properties:
        name: oai_vspgwc

# OAI Service instances
    serviceinstance#vSPGWC_Web:
      type: tosca.nodes.VSPGWCTenant
      properties:
        name: Web
        service: web
      requirements:
        - vspgwc_vendor:
            node: oai_vspgwc
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwc
            relationship: tosca.relationships.BelongsToOne
```

```
# OAI Service instances
    serviceinstance#vSPGWC_StressTest:
      type: tosca.nodes.VSPGWCTenant
      properties:
        name: StressTest
        service: stresstest
      requirements:
        - vspgwc_vendor:
            node: oai_vspgwc
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwc
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWC_Video:
      type: tosca.nodes.VSPGWCTenant
      properties:
        name: Video
        service: video
      requirements:
        - vspgwc_vendor:
            node: oai_vspgwc
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwc
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWU_Video1:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: Video1
        service: video
        selectable: true
        uprate: 10
        downrate: 20
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWU_Video2:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: Video2
        service: video
        selectable: true
        uprate: 15
        downrate: 30
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne
```

```
# OAI Service instances
    serviceinstance#vSPGWU_Web1:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: Web1
        service: web
        selectable: true
        uprate: 5
        downrate: 10
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWU_Web2:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: Web2
        service: web
        selectable: true
        uprate: 2
        downrate: 4
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWU_Web3:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: Web3
        service: web
        selectable: true
        uprate: 1
        downrate: 2
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne

# OAI Service instances
    serviceinstance#vSPGWU_StressTest:
      type: tosca.nodes.VSPGWUTenant
      properties:
        name: StressTest
        service: stresstest
        selectable: true
        uprate: 100
        downrate: 100
      requirements:
        - vspgwu_vendor:
            node: oai_vspgwu
            relationship: tosca.relationships.BelongsToOne
        - owner:
            node: service#vspgwu
            relationship: tosca.relationships.BelongsToOne
```

APPENDIX B – OVS configuration Leaf-Spine

```
e2974151-9340-42fc-94b2-64edb1387373
    Bridge "leaf2"
        Controller "tcp:10.100.198.201:6653"
            is_connected: true
        Port "leaf2-spine2"
            Interface "leaf2-spine2"
                type: patch
                options: {peer="spine2-leaf2"}
        Port "leaf2"
            Interface "leaf2"
                type: internal
        Port "leaf2-spine1"
            Interface "leaf2-spine1"
                type: patch
                options: {peer="spine1-leaf2"}
        Port "vnet8"
            Interface "vnet8"
        Port "vnet3"
            Interface "vnet3"
    Bridge "leaf1"
        Controller "tcp:10.100.198.201:6653"
            is_connected: true
        Port "vnet11"
            Interface "vnet11"
        Port "leaf1-spine2"
            Interface "leaf1-spine2"
                type: patch
                options: {peer="spine2-leaf1"}
        Port "leaf1"
            Interface "leaf1"
                type: internal
        Port "leaf1-spine1"
            Interface "leaf1-spine1"
                type: patch
                options: {peer="spine1-leaf1"}
    Bridge "spine1"
        Controller "tcp:10.100.198.201:6653"
            is_connected: true
        Port "spine1"
            Interface "spine1"
                type: internal
        Port "spine1-leaf2"
            Interface "spine1-leaf2"
                type: patch
                options: {peer="leaf2-spine1"}
        Port "spine1-leaf1"
            Interface "spine1-leaf1"
                type: patch
                options: {peer="leaf1-spine1"}
    Bridge "spine2"
        Controller "tcp:10.100.198.201:6653"
            is_connected: true
        Port "spine2"
            Interface "spine2"
                type: internal
        Port "spine2-leaf2"
            Interface "spine2-leaf2"
                type: patch
                options: {peer="leaf2-spine2"}
        Port "spine2-leaf1"
            Interface "spine2-leaf1"
                type: patch
                options: {peer="leaf1-spine2"}
    ovs_version: "2.5.3"
```

APPENDIX C – xproto Data Model

```
option name = "vspgwu";
option app_label = "vspgwu";

message VSPGWUService (Service){
    option verbose_name = "Virtual Serving Gateway User Plane Service";
}

message VSPGWUVendor (XOSBase){
    option verbose_name = "Virtual Serving Gateway User Plane Vendor";

    required string name = 1 [help_text = "vendor name", max_length = 32, null = False, db_index = False, blank = False];
    required manytoone image->Image:+ = 2 [help_text = "select image for this vendor", db_index = True, null = False, blank =
False];
    required manytoone flavor->Flavor:+ = 3 [help_text = "select openstack flavor for vendor image", db_index = True, null = False,
blank = False];

}

message VSPGWUTenant (TenantWithContainer){
    option verbose_name = "Virtual Serving Gateway User Plane Service Instance";
    required string name = 1 [help_text = "Network Service", max_length = 32, null = False, db_index = False, blank = False];
    required bool selectable = 2 [help_text = "Enable/Disable selection of Data Plane by NSSF", default = True, null = False,
db_index = False, blank = True];
    required int32 uprate = 3 [help_text = "Uprate QoS in Mb/s", default = 5, null = False, db_index = False, blank = False];
    required int32 downrate = 4 [help_text = "Downrate QoS in Mb/s", default = 10, null = False, db_index = False, blank = False];
    optional manytoone vspgwu_vendor->VSPGWUVendor:vendor_tenants = 3 [help_text = "select vendor of choice, leave blank for slice
default", db_index = True, null = True, blank = True];

}
```