Small cEllS coordinAtion for Multi-tenancy and Edge services

Grant Agreement No.671596

Topic: H2020-2014-ICT-14
Advanced 5G Network Infrastructure for the Future Internet
Research and Innovation Action

Deliverable D6.3

Service Management and Orchestration functions, including VNF models – Final

Document Number:	H2020-5GPPP-GA No.671596/*WP6/D6.3/30.09.2017*
Contractual Date of Delivery:	*30.09.2017*
Editor:	Irena Trajkovska – Zurich University of Applied Sciences (ZHAW)
Work-package:	*WP6*
Distribution / Type:	Public (PU) / Report (R)
Version:	1.0
Total Number of Pages:	60
File:	*SESAME_Deliverable 6.3_v1.0_Final*

Abstract

SESAME's aim is to build a cloud-enabled platform that supports both radio access and edge computational services at one Point of Presence (PoP). Network Services are supported by Virtual Network Functions hosted in the Light DC, leveraging on technologies like SDN and NFV that allow achieving an adequate level of flexibility and scalability at the cloud infrastructure edge.

The NFV Orchestrator (NFVO), an essential component of CESCM, provides functionalities for managing and orchestrating the resources and network services over the CESC cluster. The NFVO manages a typical Network Function Virtualisation Infrastructure (i.e. processing power, storage and networking), and it also composes service chains (constituted by two or more VNFs located either in one or several CESCs) and manages the deployment of VNFs over the Light DC.

This deliverable describes the deployment procedure of the SESAME Cloud and SDN pilot testbed with a particular focus on the SFC requirements and deployment specifics. It moreover presents the SFC monitoring APIs used in a Prometheus exporter for Netfloc. The monitoring component of the SESAME VIM is integrated with the general monitoring agent of the SESAME CESCM. The automation process of SFC is described through a section dedicated to the SESAME NFVO. This also includes the interfaces between Tenor and Netfloc and the process of automatic service chain deployment.

In the further section, a focus is given on the VNFs that are created and used in SESAME from an SFC point of view, including the network specification of the same, and their functionality within the network service. D6.3 follows up on VNF placement and details on ensuring QoS over multi-tenant CE-RAN. This part also includes an energy-aware VNF placing algorithm and a dynamic replacement of VNFs in order to meet certain SLAs.

The following section describes the virtualization of Radio Access Networks (RAN) and Self-Organizing Network (SON) functions for multi-tenant small cell networks. It also discusses a specific framework for virtualizing RRM/SON functions.

Finally the last part is dedicated to security analysis in the context of 5G networks. It discusses VNF modelling considering different attacks, giving a final focus on SDN as a VNF.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	24.07.2017	Table of Content (ToC) and contribution to sections - ZHAW	Irena Trajkovska - ZHAW
0.2	14.08.2017	Inputs on section 2	Pouria Sayyad Khodashenas – i2CAT
0.3	21.08.2017	Inputs on section 4	Begoña Blanco - EHU
0.4	05.09.2017	Inputs on section 5, updates to Glossary and Reference List in section 7	Jordi Pérez-Romero - UPC
0.5	13.09.2017	Added section 2.2 for SFC monitoring	Irena Trajkovska - ZHAW
0.6	14.09.2017	Reviewed and references added in section 4 – VNF placement and QoE awareness	Begoña Blanco - EHU
0.7	18.09.2017	Added Section Introduction, Abstract and some formatting	Irena Trajkovska - ZHAW
0.8	18.09.2017	Added in a section on the Video Analytics VNF – section 3.5	Mick Wilson - FLE
0.9	18.09.2017	Added content to section 2.4, integrated section with contribution from FLE, edited section 6 headings	Irena Trajkovska - ZHAW
0.10	18.09.2017	Corrections of images	Irena Trajkovska - ZHAW
0.11	25.09.2017	Edited referenced, captions and addressed comments from FLE related to ZHAW sections	Irena Trajkovska - ZHAW
0.12	26.09.2017	Integrated contribution from NCSRD and ORION and some adjustments	Irena Trajkovska - ZHAW
0.13	26.09.2017	Inputs on section 6, updates to authors	Konstantinos Kosmidis - UoB
0.14	26.09.2017	Added references for VA section	Mick Wilson - FLE
0.15	27.09.2017	Updated content, references and figure captions in section 4	Jose Oscar Fajardo - EHU
0.16	27.09.2017	Added Conclusions sections, final edits and pre-final version released	Irena Trajkovska - ZHAW
0.17	27.09.2017	Overall Review	Pouria Sayyad Khodashenas - i2CAT
0.18	27.09.2017	Reference reorder and table update	Irena Trajkovska - ZHAW
0.19	28.09.2017	Overall Review	Athanassios Dardamanis - SMNET
1.0	30.09.2017	Final full editorial and conceptual review. Document ready for submission to the European Commission.	Ioannis Chochliouros - OTE

Contributors

First Name	Last Name	Partner	Email
Irena	Trajkovska	ZHAW	traj@zhaw.ch
Pouria	Sayyad Khodashenas	i2CAT	pouria.khodashenas@i2cat.net
Shuaib	Siddiqui	i2CAT	shuaib.siddiqui@i2cat.net
Begoña	Blanco	EHU	begona.blanco@ehu.eus
Ruben	Solozabal	EHU	Ruben.solozabla@ehu.eus
Jose Oscar	Fajardo	EHU	Joseoscar.fajardo@ehu.eus
Fidel	Liberal	EHU	Fidel.liberal@ehu.eus
Jordi	Pérez-Romero	UPC	jorperez@tsc.upc.edu
Oriol	Sallent	UPC	sallent@tsc.upc.edu
Ramon	Ferrús	UPC	ferrus@tsc.upc.edu
Ramon	Agustí	UPC	ramon@tsc.upc.edu
Julie	Xaio	FLE	hui.xaio@uk.fujitsu.com
Mick	Wilson	FLE	mick.wilson@uk.fujitsu.com
Ioannis	Giannoulakis	NCSRD/ORION	giannoul@iit.demokritos.gr
Konstantinos	Kosmidis	UoB	k.kosmidis@brighton.ac.uk
Athanassios	Dardamanis	SMNET	adardamanis@smart.net.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
3G	Third Generation of Mobile Communications
3GPP	Third Generation Partnership Project
4G	Fourth Generation of Mobile Communications
5G	Fifth Generation of Mobile Communications
AC	Admission Control
ACM	Association for Computing Machinery
API	Application Programming Interface
AR	Augmented Reality
BF	Broadband Forum
CCTV	Closed Circuit Television
CCO	Coverage and Capacity Optimisation
CESC	Cloud-enabled Small Cell
CE-RAN	Cloud Enabled RAN
CESCM	Cloud Enabled Small Cell Manager
CPU	Central Processing Unit
cSON	centralized SON
CV	Computer Vision
CVE	Common Vulnerabilities and Exposures
DB	Database
DC	Data Centre
DDoS	Distributed DoS
DL	Downlink
DoS	Denial of Service
DP	Data Plane
DPDK	Data-plane Development Kit
DPI	Deep Packet Inspection
dSON	distributed SON
DSS	Decision Support System
E2E	End-to-End
EMS	Element Management System
eNB	eNode B
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
E-UTRA	Evolved Universal Terrestrial Radio Access
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
EU	European Union
EuCNC	European Conference on Networks and Communications
FCAPS	Fault, Configuration, Accounting, Performance and Security
FFMPEG	Fast Forward MPEG
FG	Forwarding Graph
fps	frames per second
FW	FireWall
GA	Grant Agreement
GPRS	General Packet Radio Service
GPU	Graphics Processing Unit
GRE	Generic Routing Encapsulation
GS	Group Specification
GSM	Global System for Mobile communications
GTP	GPRS Tunnelling Protocol
GUI	Graphical User Interface

GW	Gateway
H2020	Horizon 2020
HeMS	HeNB Management System
HeNB	Home eNodeB
HTTP	Hyper-text Transmission Protocol
HW	Hardware
HWA	Hardware Accelerator
I/O	Input/Output
ICT	Information and Communication Technology
ID, id	Identifier
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IFIP	International Federation for Information Processing
IoT	Internet of Things
IP	Internet Protocol
ISG	Industry Specification Group
IT	Information Technology
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
JDK	Java SE Development Kit
KPI	Key Performance Indicator
Light DC	Light Data Centre
LAN	Local Area Network
LTE	Long Term Evolution
LXC	Linux Container
MA	Metric Aggregator
MAC	Medium Access Control
MANO	Management and Orchestration
MEC	Mobile Edge Computing
ML	Modular Layer
MME	Mobility Management Entity
MPEG	Moving Picture Experts Group
MS	Management System
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NFVO	NFV Orchestrator
NGMN	Next Generation Mobile Networks
NMS	Network Management System
NS	Network Service
NSD	NS Descriptor
NSH	Network Service Header
ODL	OpenDayLight
OF	Open Flow
OPNFV	Open Platform for NFV
OVS	Open virtual Switch
OVSDB	Open vSwitch Database Management Protocol
PC	Personal Computer
PCAP	Packet Capture
PM	Performance Management
PNF	Physical Network Function
PNFD	PNF Descriptor
PoC	Proof of Concept
PoP	Point of Presence

PPP	Public-Private Partnership
PS	Packet Scheduling
QoE	Quality of Experience
QoS	Quality of Service
RAB	Radio Access Bearer
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request for Comments
RIA	Research and Innovation Action
RNIS	Radio Network Information Service
RO	Robust Optimization
RRM	Radio Resource Management
SC	Small Cell
SCaaS	Small Cells-as-a-Service
SCF	Small Cell Forum
SCNO	Small Cell Network Operator
SDN	Software-defined Networking
SFC	Service Function Chaining
SLA	Service Level Agreement
SME	Small- and Medium-sized Enterprise
SON	Self-Organising Networks
SOTA	State-of-the-Art
SP	Service Provider
SW	Software
TCP	Transmission Control Protocol
TU	Transcoding Unit
TR	Technical Report
TS	Technical Specification
UC	Use Case
UE	User Equipment
UL	Uplink
UMTS	Universal Mobile Telecommunications System
UTRAN	Universal Terrestrial Radio Access Network
VA	Video Analytics
vDPI	virtual Deep Packet Inspection
vFW	virtual FireWall
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VMMF	Virtual Machine Failure Monitoring
VNFCI	Virtual Network Function Component Instance
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	VNF Manager
VNI	VXLAN Network Identifier
VSCNO	Virtual Small Cell Network Operator
vTU	virtual Transcoding Unit
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network
WICM	WAN Infrastructure Connection Manager
WP	Work Package
WT	Warning Threshold

Table of Contents

List of Figures

List of Tables

1 Introduction

The leading activity within the SESAME project related to virtual infrastructure management is the design and development of an Orchestrator that is responsible for the automatic provisioning of virtual resources and the deployment of Network Services (NSs) in coordination with the SESAME VIM. Furthermore, this task involves the modelling of SFC algorithms in order to support network service chaining by using SESAME designed VNFs. In concrete, the focus in the cloud ecosystem of SESAME is to provide an SDN-*enabled* cloud environment, on the top of which SFCs will be provisioned for the VSCNO operators. The NFVO has been described in previous deliverables, along with the details of the SFC algorithm.

In this deliverable the focus is beyond the implementation activities, and more in the integration, deployment and testing of the SFC in the current SDN cloud testbed. Moreover, some important components are discussed, like VNFs and their placement in accordance to SLA definitions in order to "represent" the use case SFC service that is being used, as a validator for the SFC and the integration between components such as: NFVO, CESM, VIM, etc.

We elaborate on the details of VNF placement and replacement, covering several energy-aware aspects and SLA compliance.

Some of the SESAME use case scenarios include VNFs related to the radio part, including virtual RRM and SON functions. The same are further discussed in this deliverable with a closing theoretical elaboration on the security aspects of the VNFs used in the network service.

1.1 Deliverable outline

The present deliverable covers the service function chain deployment testbed, describing the role of TeNOR[1] in automatic service deployment and the role of Netfloc[2] in automatic virtual network graph establishment and data-plane configuration. The main focus here is to bring forward the advances in the activities following the deliverable D6.2, including updated knowhow on related topics. The following sections are included:

- Section 1 offers a brief introductory overview.
- Section 2 described the setup of the SDN environment in the OpenStack testbed and the deployment and testing of the SFC service. It also presents the SDN monitoring component offering the metrics list of the current SFC service. TeNOR and Netfloc interfaces and automatic service deployment is furthermore discussed.
- Section 3 focuses on the SESAME VNFs from the perspective of the network service and describes the specific requirements and the functionality each of them has with respect to the SFC.
- Section 4 includes the VNF placement algorithms and QoE awareness including energy-aware VNF placing algorithm and a dynamic replacement of VNFs in order to "meet" certain SLAs.
- Section 5 deals with virtualization of Radio Access Networks (RAN) and Self-Organizing Network (SON) functions for multi-tenant small cell (SC) networks. It also discusses a specific framework for virtualizing RRM/SON functions.
- Section 6 considers security and modelling of VNFs in the context of 5G networks and under different attacks.

[1] More details about TeNOR can be found at: *https://github.com/T-NOVA/TeNOR/wiki*
[2] See: *http://icclab.github.io/netfloc/*

2 Service Chain deployment and integration

For the demonstration purpose in the SESAME project, an SDN testbed was created within the premises of OTE in order to: (i) showcase VIM and SDN integration in SESAME along with the CESCM – the VNFM and NFVI in particular, *and*; (ii) provide a support for the testing of network services (NSs) in hybrid telco and cloud context. In the following part we will describe some of the details for this implementation, focusing upon the general deployment characteristics and specificities related to the service chain scenario, omitting the detailed step-by-step integration process among the components. This part will be further elaborated in the SESAME Deliverables related to the WP7 activities.

2.1 SESAME Cloud and SDN pilot testbed

The OTE testbed, (as shown in Figure 2-1), used for the Light DC consists of three nodes: Control[3], Compute[4] and Neutron[5] as part of the OpenStack[6] infrastructure. The pilot also contains a 4th node as a standard PC with 4 network cards, out of which 3 are connected to the OpenStack nodes and the 4th one is connected to the management network. This node is used as physical switch and it has also installed OpenvSwitch[7] (OVS) to simulate a real SDN switch. OpenvSwitch is installed and configured on all the nodes in the testbed to provide networking abstraction and connectivity between all the OpenStack nodes and the switch. OVS also enables a full SDN control allowing OpenFlow rules and configurations[8]. For that matter we used a dedicated data network for all the OVS connections, by configuring a second interface on each node and attaching it as a port in the OVS bridge. This was the essential physical interconnection and OVS configuration in order to obtain a starting environment for OpenStack-Netfloc integration.

[3] See: *https://docs.openstack.org/newton/install-guide-ubuntu/nova-controller-install.html*
Also see: *https://docs.openstack.org/mitaka/install-guide-ubuntu/neutron-controller-install.html*
[4] See: *https://ask.openstack.org/en/question/50263/what-is-openstack-compute-node/*
[5] See: *https://wiki.openstack.org/wiki/Neutron*
[6] See: *https://wiki.openstack.org/wiki/Nova*
[7] Also see: *http://openvswitch.org/*
[8] For more informative details see, *inter-alia*: *https://en.wikipedia.org/wiki/OpenFlow*

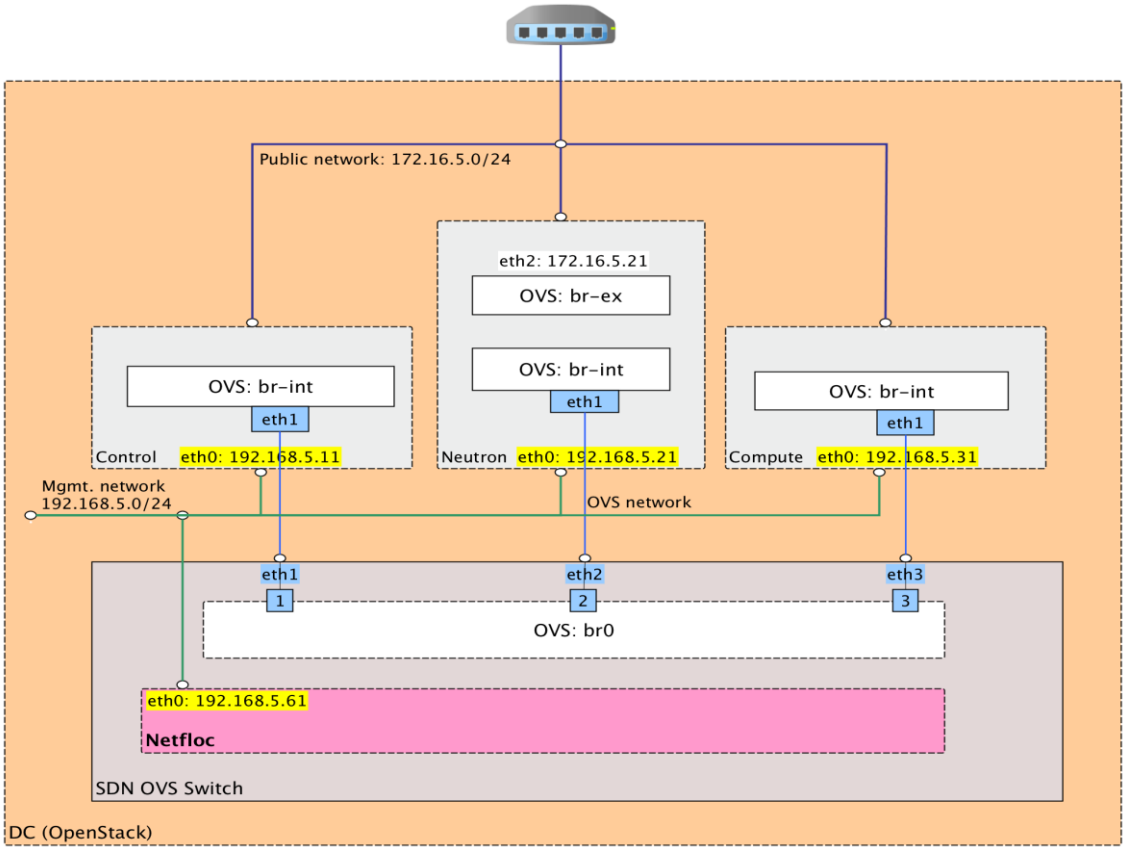


Figure 2-1: Detailed architecture of the OpenStack nodes and Netfloc in the SDN OTE testbed

2.1.1 Prerequisites and infrastructure details

Referring the to Deliverable D6.2, Netfloc was developed a Java-*based* SDN component, bound to the OpenDaylight (ODL) controller[9] in order to "meet" the networking needs for the project. It has been implemented as a Karaf feature[10] that is composed of multiple OpenDaylight libraries, such as: openflow plugin[11], ovsdb[12], neutron api[13], etc. The creation of the SFC algorithms and models in Netfloc started in the year 2015, parallel to the development activities coming from the ODL community. Circumventing the SFC approach based on the Network Service Header, *NSH* header [1] specification that was followed among the ODL community, we have taken an alternative way.

With the purpose to allow SDN-*managed* networking, Netfloc's SFC model was designed as an alternative solution that is simplifying the OpenStack native networking in an SDN integrated scenario. In this respect, Netfloc is the unique control and management entity of the OpenStack VIM, providing end-do-end (E2E) overlay connectivity via OpenFlow-*enabled* management of the virtual and physical switches.

---

[9] For further information also see:*https://wiki.opendaylight.org/view/OpenDaylight_Controller:Main*
[10] See: *https://karaf.apache.org/manual/latest/*
[11] See: *https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Main*
[12] See: *https://wiki.opendaylight.org/view/OVSDB_Integration:Main*
[13] See: *http://docs.opendaylight.org/en/latest/user-guide/neutron-service-user-guide.html*

The initial version of OpenDaylight that was supported in Netfloc is Lithium[14]. However following a modular design pattern, Netfloc features can be installed as standalone features inside other OLD versions, for example Boron[15], such as in the Figure 2-2.

```
opendaylight-user@root>bundle:list | grep netfloc
105 | Installed |  80 | 1.0.0.SNAPSHOT         | netfloc-api
106 | Installed |  80 | 1.0.0.SNAPSHOT         | netfloc-impl
107 | Active    |  80 | 1.0.0.SNAPSHOT         | netfloc-karaf
108 | Active    |  80 | 1.0.0.SNAPSHOT         | netfloc-features
```

Figure 2-2: List of the Netfloc libraries as OpenDaylight bundles

From a cloud perspective, Netfloc has been tested and supported using OpenStack Kilo[16] and Juno[17] releases.

A main prerequisite for use of Netfloc is that the network needs to be fully SDN-*enabled* through the OVS switches. This includes all the network interfaces of the physical switch that are connected with the interfaces of the OpenStack nodes and configured by using OVS ports. Since Netfloc is created using the Java programming language, installation of Java and Maven[18] are required, to be able to make modifications and improvements in the component, otherwise the rest are obligatory:

- JAVA 7 JDK[19]
- Maven 3.1.1[20]
- OpenFlow 1.3[21] enabled network devices
- OpenvSwitch
- Open Stack basic environment[22] (ex. 3 nodes: compute, control, neutron)

By using the *ovs-vsctl* and *ovs-ofctl* utilities, the virtual interfaces are configured thus allowing a complete overview of the connectivity status and of the OpenFlow flows, as well. In particular, ovs-vsctl provides a high-level overview of the interfaces and the configuration database of *ovs-vswitchd.* The ovs-ofctl command-line tool allows to monitor and administer the openflow switches, providing an overview of the current status of the configuration and the table entries.

A typical network interface configuration related to the OVS ports and interfaces is shown in Figure 2-3:

[14] For further details see: *https://www.opendaylight.org/what-we-do/current-release/lithium*
[15] For further details see: *https://www.opendaylight.org/what-we-do/current-release/boron*
[16] More details can be found at: *https://www.openstack.org/software/kilo/*
[17] More details can be found at: *https://www.openstack.org/software/juno/*
[18] See: *https://maven.apache.org/*
[19] *http://www.oracle.com/technetwork/java/javase/downloads/java-se-jdk-7-download-432154.html*
[20] See: *https://maven.apache.org/docs/3.1.1/release-notes.html*
[21] *https://www.networkcomputing.com/networking/openflow-13-support-why-it-matters/602695705*
[22] See the discussion at: *https://docs.openstack.org/ha-guide/environment.html*

```
auto br-int
 iface br-int inet static
     address 10.0.5.21
     netmask 255.255.255.0
     network 10.0.5.0
     broadcast 10.0.5.255
     dns-nameservers 62.217.124.90 8.8.8.8 212.205.212.205
     gateway 10.0.5.1
     ovs_type OVSBridge
     ovs_ports eth1


auto eth1
iface eth1 inet static
ovs_bridge br-int
ovs_type OVSPort
```

Figure 2-3: OVS network configuration on the integration bridge *br-int*

In the above figure, the integration bridge *br-int*, has a port attached – the *eth1*. Via this port, the OVS of this node is connected to the other OVS components in the infrastructure. In the process of connecting Netfloc with OpenStack there is an additional bridge, the *br-ex* that is created in the Neutron node in order to provider external connectivity service to the OpenStack VIM.

In order for Netfloc to interoperate with OpenStack nodes, it is important to enable SDN support in the infrastructure via the Modular Layer 2[23], ML2 plugin in OpenStack Neutron. The Neutron component in OpenStack creates an abstraction layer of the underlying network in order to provide physical and virtual network resources using several network elements like routers, networks, subnets, floating IPs, etc. Neutron extends the networking service to third parties by offering a variety of plugins. For SDN integration with OpenDaylight-*based* component, we used the ML2 north-bound plug-in[24]. This enabled a fully SDN-*enabled* environment and connected the Neutron networking service with Netfloc.

Since we are using Netfloc as an SDN backend for Neutron, Netfloc is expected to be the only source for Open vSwitch configuration, i.e. the previous configuration by Neutron is no longer valid in the scenario. For that matter, it is necessary to remove existing OpenStack and Open vSwitch (OVS) configurations so that Netfloc can start with a clear configuration. Note: this, as well as the rest of the integration steps between OpenStack and Netfloc coincide with the standard ODL – OpenStack integration process [2].

This process also includes clearing the neutron and openvswitch databases[25],[26]. Another requirement is the installation of the *networking_odl* Python[27] module[28] in order to allow the OpenDaylight driver be in charge of the networking environment. That is configured in the ML2 configuration file. Finally, Netfloc can be connected with OpenvSwitch by setting it up as a manager and controller entity via tcp connection with IP address and ports of the Netfloc node,that has to be configured in each separate node individually in the following manner:

---

[23] Also see: *https://wiki.openstack.org/wiki/Neutron/ML2*
[24] *https://ask.openstack.org/en/question/98780/neutron-ml2-plugin-odl-neutron-northbound-interface/*
[25] Also see, *for example*: *https://wiki.openstack.org/wiki/Neutron/DatabaseMigration*
[26] Also see, *for example*: *https://www.sdxcentral.com/cloud/open-source/definitions/what-is-ovsdb/*
[27] *https://www.python.org/*
[28] See: *https://pypi.python.org/pypi/networking-odl*

ovs-vsctl set-manager tcp:192.168.5.61:6640

ovs-vsctl set-controller br-int tcp:192.168.5.61:6633

Once the initial connectivity is achieved we can make several tests in order to retrieve from Neutron the status of networks/ports (after we create a test case environment in OpenStack).

Netfloc installation is straightforward by downloading the precompiled library and running a startup script. Similarly we have developed a user interface called Netflogi [3] to support the provisioning the management of the chain scenarios. In the SFC window of Netflogi the user can list the deployed Service Chains as well as creating new or deleting old ones. Furthermore, Netflogi enable the visualization of the monitoring characteristics as taken from Netfloc by using the Statistics Manager library from OpenDaylight, as shown in Figure 2-4.

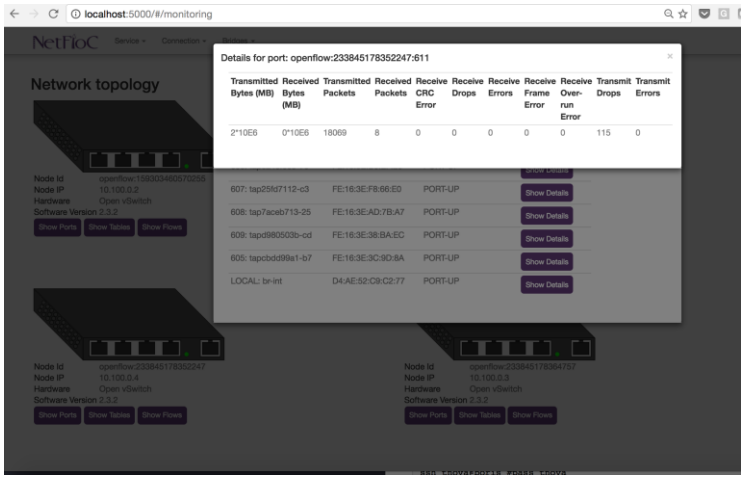


Figure 2-4: Netflogi monitoring details

2.1.2 SFC deployment specifics and demo scenario

Apart from allowing full network connectivity and integration in OpenStack VIM, Netfloc contains a library in order to support traffic steering that is based on the principle of port matching and mac-to-virtual addresses rewriting in order to provide end-to-end SFC support in OpenStack VIM. The SFC library and functionality has been already validated in a wide SFC-WAN scenario within the scope of T-NOVA[29] as a reference project, for the support for intra-datacenter chaining of multiple VNFs like: traffic classifier transcoding unit, watermarking, virtual proxy service, etc. [4].

Within SESAME, an SFC demo was shown in the EuCNC 2016 demonstration booth[30] in cooperation with the COHERENT project [5].

[29] For more details see: *http://www.t-nova.eu/*
[30] Also see: *http://www.eucnc.eu/2016/www.eucnc.eu/indexe637.html?q=node/134*

2.2 SFC monitoring

In order to provide a full picture of the SDN infrastructure, the SDN testbed includes a component for monitoring the network. Apart from the standard sFlow tool[31] for monitoring running on the testbed, there has been developed a component for gathering statistics directly from the monitoring module in Netfloc. This component is based on the statistics gathered form the OpenFlow plugin in OpenDaylight [6]. The statistical data provided from this component includes the following:

> *Individual Flow Statistics, Aggregate Flow Statistics, Flow Table Statistics, Port Statistics, Group Description, Group Statistics, Meter Configuration, Meter Statistics, Queue Statistics, Node Description, Flow Table Features, Port Description, Group Features, Meter Features*

This information has been wrapped up in a library exposing REST interfaces to third party components, such as for example the CESCM in SESAME. This interface is however not direct, rather it is facilitated through a specific plugin – an exporter library for Prometheus [7]. This design allows compatibility with any other component that uses Prometheus and Grafana[32] to attach to the SDN component and represent the data in a generic GUI. It is a standard solution that is compatible with the design characteristics of the CESCM. The Netfloc exporter for Prometheus library [8] has been implemented in Python and it includes modules for scraping the data time series from Netfloc, editing the information and saving the values in predefined metrics. The endpoint is specified in environmental variables and the service is run in the Control node of the SDN testbed.

Overall, there are three type of metrics: Aggregate flow statistics and flow tables statistics per host, statistics per port, and statistics per flow. Figure 2-5 shows the byte-count metric in Netfloc exporter, for each of the nodes in the testbed (compute, control, neutron and netfloc).

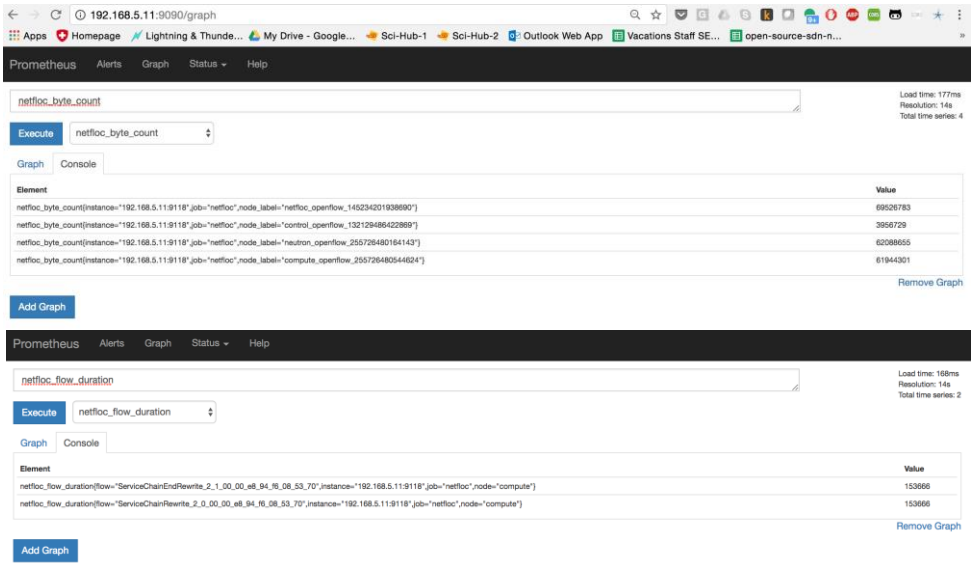


Figure 2-5: Netfloc flow-duration and byte-count metrics in Prometheus

It moreover shows the flows on the netfloc node (the SDN switch) that are programmed by Netfloc for the SFC service. It depicts two flows:

[31] Also see: *http://www.inmon.com/technology/sflowTools.php*
[32] See: *https://grafana.com/*

netfloc_flow_duration{flow="ServiceChainEndRewrite_2_1_00_00_e8_94_f6_08_53_70",instance="192.168.5.11:9118",job="netfloc",node="compute"}

netfloc_flow_duration{flow="ServiceChainRewrite_2_0_00_00_e8_94_f6_08_53_70",instance="192.168.5.11:9118",job="netfloc",node="compute"}

The two flows have priority=20 and this is how the SFC flows are distinguished in Netfloc and programmed on the OVS switch. There is a flow for Service chain rewriting of MAC address on the end of the path (the last OVS bridge in the path of the chain) and the rewrite that happens in all the intermediate OVS bridges. The long number is the MAC address that is being rewritten- the first digit stands as chain identification number – 2 in this case, and the second is the hop rewrite – 1 and 0, since this number decreases in order to uniquely identify the step in the chain.

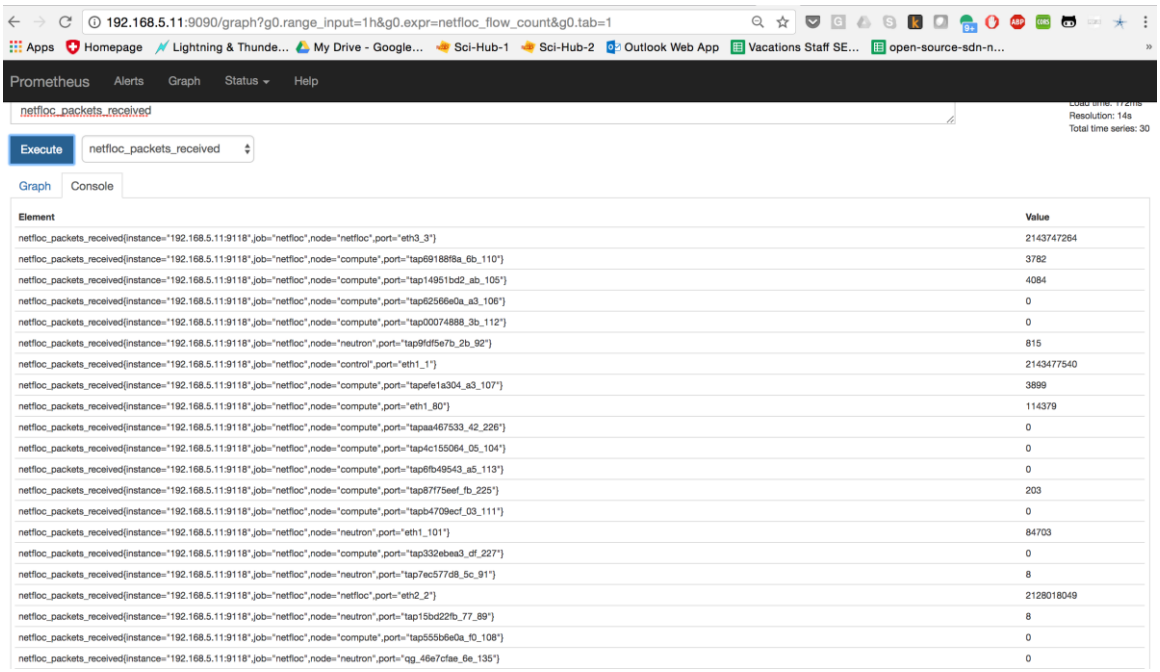


Figure 2-6: Netfloc packets_received metrics in Prometheus for all nodes and ports

The exporter uses labels for the nodes (names, IDs) and the ports in order to show statistics to the specific port of a given node (bytes and packets transmitted, received and matched). Using this information, the network administrator can monitor the correctness of the chain connecting points, the flows being matched and the traffic that passed via those ports. Figure 2-6 lists all the ports in the nodes for the *packets_received* metric. The user can filter out using the labels to show specific metrics (per node/port) and visualize it in Grafana. The data statistics is also shown in the Grafana GUI for better representation of several metrics.

Figure 2-7 shows a graph of the compute node related to statistics about byte count. It also aggregates the same information for the rest of the nodes. The second metric, the active_flows is shown in Figure 2-8 and it relates to the neutron node in the testbed.

Th fact of having all the information transparent in a timely manner provides a full picture of the entire SDN topology and the state of the flows and the packets at a given time. This is important to monitor in integrated NFV environments, as it is an intuitive way to provide management from a higher level on the network. Moreover, in an integrated environment such as SESAME, it

provides a direct interface between CESCM, TeNOR and Netfloc in the way towards automatic network service management. This is facilitated through alerts and alarms that can be triggered in CESCM monitoring in respect so some network violations. Upon such notification the CESCM can trigger TeNOR in order to provide some alteration via Netfloc to the network interfaces or the VNFs. This can be done via Tenor shutting down and spawning a new VNF, or even a new NS.

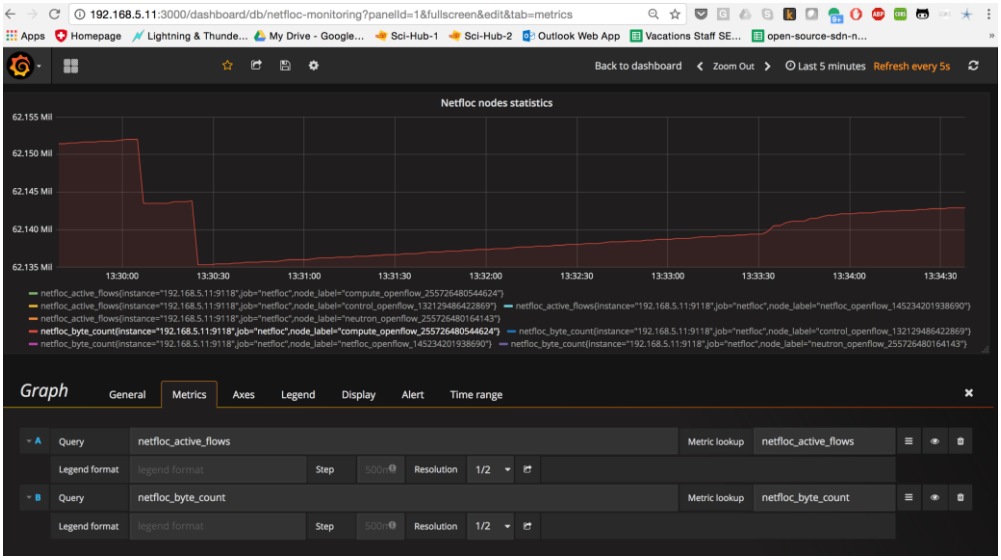


Figure 2-7: Netfloc flow-duration graph in Grafana

Concerning Netfloc, if there is a loop at some interface or too much traffic is generated unexpectedly, TeNOR can call Netfloc's APIs to shut down the service chain and create a new one. Via reactive OpenFlow manipulation, flows can be installed on the OVS to deviate the flows, restart interfaces and prevent some traffic passing through some specific path. Using this monitoring tool allows for security violations to be detected on-time. This way a central entity acting as a unique control/management point is given the responsibility to prevent the integrity of the network service. More details on the integration CESCM monitoring with Netfloc is elaborated in the Deliverable D5.2.

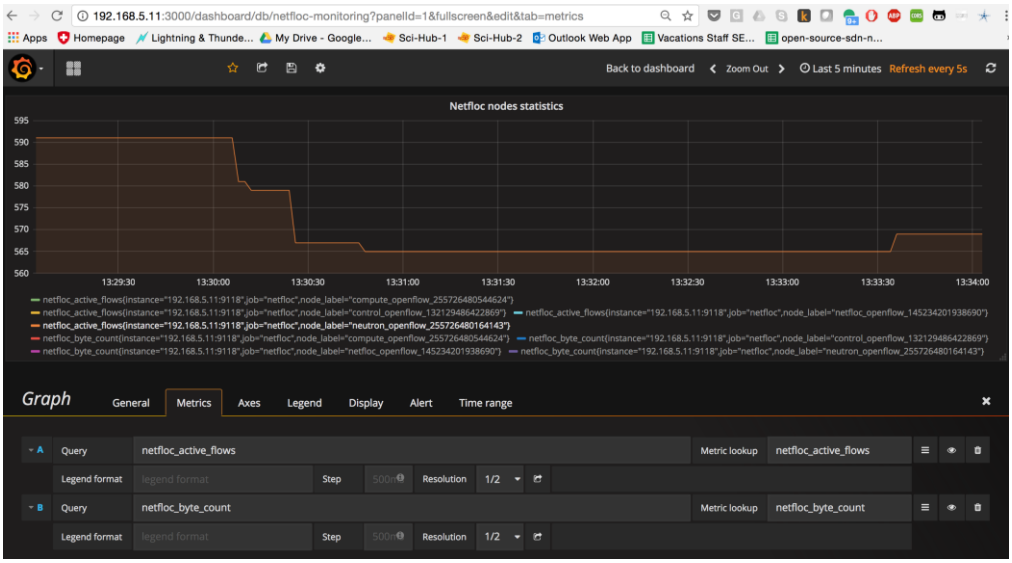


Figure 2-8: Netfloc active_flows graph in Grafana

As previously mentioned, an enhanced monitoring of the entire LightDC networking infrastructure is well facilitated via the sFlow protocol and represented in the sFlow Trend GUI [9]. Figure 2-9 represent some of the data that can be visualized and monitored via this tool, i.e. the traffic on interface 2, using as a filter a given mac source and destination address.

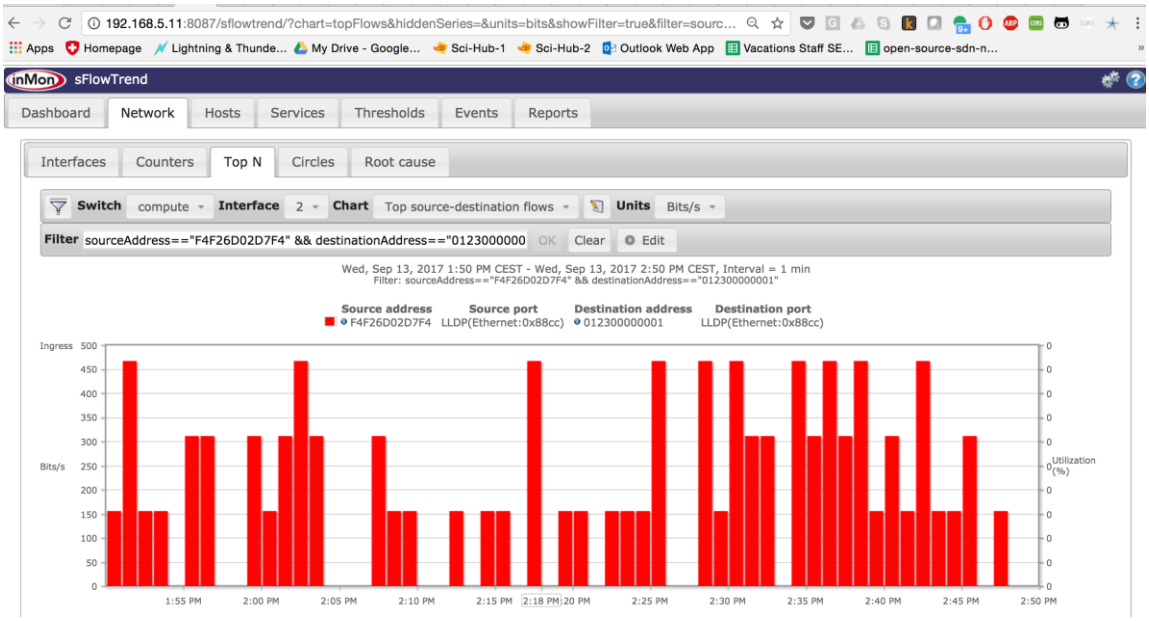


Figure 2-9: sFlow Trend GUI showing traffic on Interface 2 given a source and destination MAC

There is exhaustive information from the sFlow tool as well, which if properly combined with the Netfloc exporter, complements the standard switch monitoring technology with SDN-*based* information gathered from OpenFlow. This can be very useful for many applications and services depending on the specific customer needs, in order to meet certain SLAs defined in the NSD.

The example in Figure 2-10 shows the network information for the control node for interface 3 in terms of both ingress and egress frames including all (unicast, multicast, broadcast traffic, etc.).

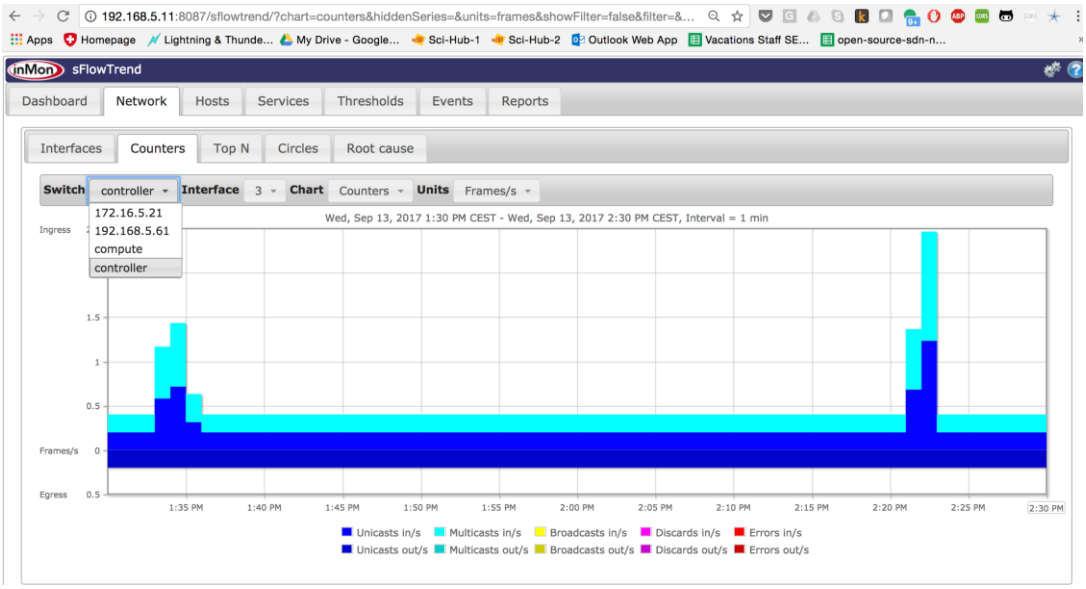


Figure 2-10: sFlow Trend GUI showing ingress/egress traffic on Interface 3 of control node

2.3 SESAME NFVO Role on SFC Automation

This section details the role of SESAME NFVO on the automation of SFC process. That is, automatically setting up an overlay between Open vSwitches (OVSs) using VXLAN [10] as the tunnelling protocol. Of course, the presented solution is not the only way to do SFC automation but it gives a good example, which underlines the NFVO interaction with the rest of SESAME management modules. It is also worth to note that, here we assume that the NS has been instantiated previously and VNFs have been configured appropriately and in accordance with the NS performance requirements (for more information about NFVO, VNFM and VIM interactions on service instantiation and VNF configuration processes, please take a look at [11]). In particular this section details the following subjects:

- Configuration of VXLAN tunnel ports in Open vSwitch (OVS).
- Configuration of OpenFlow entries (OF) in OVS.
- Demonstrate the tunnel in action showing connectivity for 2 tenants over the tunnel, logical separation of traffic between tenants and from the physical underlay network.

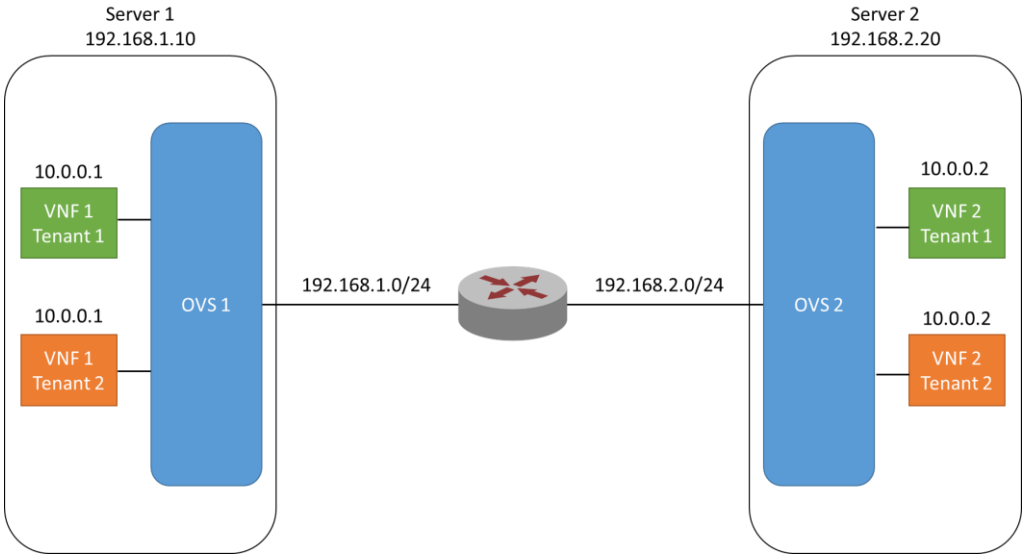


Figure 2-11: Set up overview

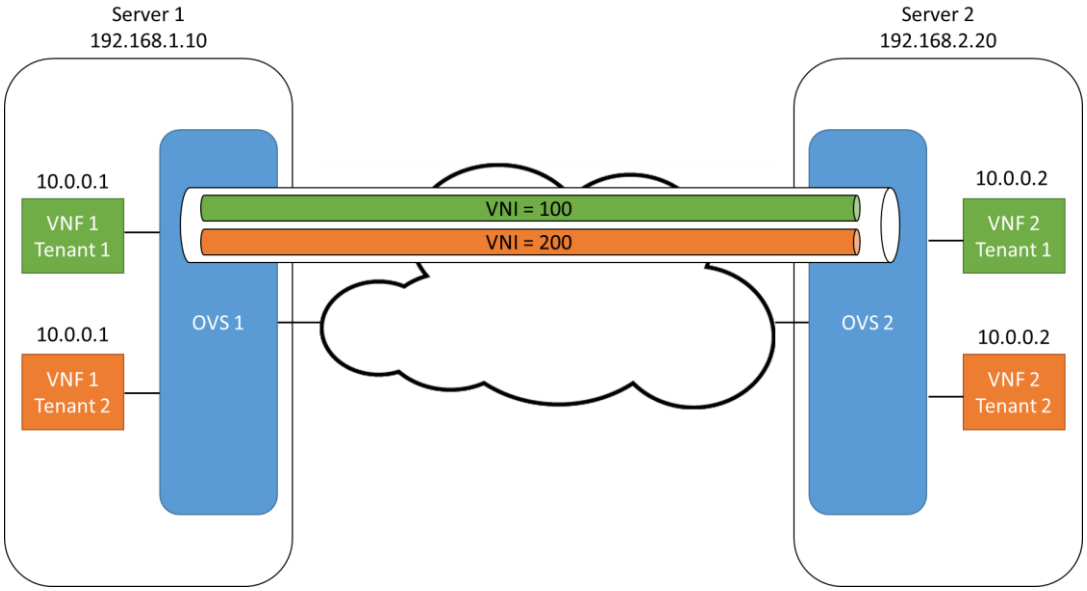


Figure 2-12: VXLAN overlay

Figure 2-11 represents the simple set up used in this section for explanation purposes. In this network topology, there are two virtual servers in two different subnets – represent two separate micro servers in two different CESCs. Server 1 has IP address 192.168.1.10 and server 2 has IP address 192.168.2.20. Since they are in different subnets there is a router in between them for IP reachability, thus forming the light DC. Both, Server 1 and server 2 are using open V switch for networking between VMs / VNFs. Server 1 and server 2 are hosting two tenants. Tenant 1 has VMs / VNFs 1 and 2: VNF 1 is using IP 10.0.01 and VNF 2 is using 10.0.0.2. Then we have tenant 2 with its VMs / VNFs 1 and 2. This tenant is also using IP addresses 10.0.0.1 and .2. These tenants are suing even overlapping MAC addresses between them.

We can set up a VXLAN tunnel, as shown in Figure 2-12, and use tunnel ids to logically separate traffic between tenant 1 and 2. The VXLAN tunnel will also provide layer 2 connectivity despite there being a router between server 1 and server 2. Finally, the tunnel will provide a logical separation from the physical underlay network, which has no knowledge of the tenant IP addressing in use.

To turn up the tunnel between the two OVS, the following steps needs to be taken:

```
RUN IN SERVER 1
===============
sudo su -
cd /vagrant/shared/lesson01
./server1_topology.py
sh ovs-vsctl add-port s1 vtep -- set interface vtep type=vxlan option:remote_ip=192.168.2.20
option:key=flow ofport_request=10
sh ovs-vsctl show
sh ovs-ofctl show s1
sh ovs-ofctl add-flows s1 server1_flows.txt

RUN IN SERVER 2
===============
sudo su -
cd /vagrant/shared/lesson01
./server2_topology.py
sh ovs-vsctl add-port s2 vtep -- set interface vtep type=vxlan option:remote_ip=192.168.1.10
option:key=flow ofport_request=10
sh ovs-vsctl show
sh ovs-ofctl show s2
sh ovs-ofctl add-flows s2 server2_flows.txt
```

Let us review the commands shown above.
The "*ovs-vsctl"* command is used to make change to OVS DB, the configuration database of OVS. The "*add-port"* command adds a port named "*vtep"* (an arbitrary name) to OVS S1/2. "*-- set interface vtep type=vxlan"* specifies the tunnelling type, VXLAN in this case. If for example one decides to use GRE tunnelling, it would be easy to do so by simply changing the type into GRE.
"*Option:remote_ip="* determines the IP address of the other end of the tunnel, i.e. S1/S2. Here we want to use VXLAN Network Identifiers (VNI) to identify logically separate tenant traffics.
With "*Option:key=flow"* no particular VNI number will be specified yet. Of course, it is possible to specify an actual VNI here but then we have to add a tunnel part for every single unique VNI.
"*Option:key=flow"* provides an overloading of the tunnel commands so there is no need for manual set ups.
"ofport_request=" is a helpful option where we are specifying that we want to use open flow port 10 for this port named "*vtep"*. If we do not do this, we do not know exactly what open flow port number we will end-up with which would be bad for OF flow entries later.
"ovs-vsctl show" displays the tunnel port and interface "*vtep*", i.e. VXLAN and the options set just as has been requested.
"ovs-ofctl show" confirms that the port named "*vtep*" is mapped to open flow port 10 as requested.
As the tunnel is configured now, let us load the flow entries to direct traffic. The *"ovs-ofctl add-flows"* is used for this purpose as shown below. The only thing to note is for server 1, S1 should be used while for server 2, S2 is the right input.

```
sh ovs-ofctl add-flows s1 flows.txt
```

An example of flow entries are presented below. In this example just two set of functions are used. "*Table 0"* is used to tag flows with the VXLAN VNI and *"Table 1"* is used to forward packets. Let us start with "*Table 0".* The first flow entry in "*Table 0"* says that for any traffic in open flow port 1, the traffic will get a tunnel ID of 100. Open flow port 1 is where the VNF of the tenant 1 is connected. *"tun_id"* maps the VNI that will be used over the tunnel. Then the flow entry says move on to flow "*Table 1"*. In other words, all traffic from VNF of tenant 1 assigned *"tun_id"* 100 and move on. The second flow entry on "*Table 0"* is similar. It says, all traffic in port 2 which is given to tenant 2 assign *"tun_id"* 200 that will correspond to VNI 200. Finally, there is a default action on "*Table 0"* to go to "*Table 1"*. "*Table 1"* is used for forwarding the packets, so depends on the *"tun_id"* and the destination MAC address specified on *"dl_dst="*, a relevant action will be taken. Note that, even though the MAC addresses are similar flow entries are distinguished by the VXLAN tag. Moreover, there is an explicit default drop rule on "*Table 1".*

Flow entries (from Server 1)
==========
table=0,in_port=1,actions=set_field:100->tun_id,resubmit(,1)
table=0,in_port=2,actions=set_field:200->tun_id,resubmit(,1)
table=0,actions=resubmit(,1)

table=1,tun_id=100,dl_dst=00:00:00:00:aa:01,actions=output:1
table=1,tun_id=200,dl_dst=00:00:00:00:aa:01,actions=output:2
table=1,tun_id=100,dl_dst=00:00:00:00:aa:02,actions=output:10
table=1,tun_id=200,dl_dst=00:00:00:00:aa:02,actions=output:10

table=1,tun_id=100,arp,nw_dst=10.0.0.1,actions=output:1
table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:2
table=1,tun_id=100,arp,nw_dst=10.0.0.2,actions=output:10
table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:10
table=1,priority=100,actions=drop

Flow entries (from Server 2)
==========
table=0,in_port=1,actions=set_field:100->tun_id,resubmit(,1)
table=0,in_port=2,actions=set_field:200->tun_id,resubmit(,1)
table=0,actions=resubmit(,1)
table=1,tun_id=100,dl_dst=00:00:00:00:aa:02,actions=output:1
table=1,tun_id=200,dl_dst=00:00:00:00:aa:02,actions=output:2
table=1,tun_id=100,dl_dst=00:00:00:00:aa:01,actions=output:10
table=1,tun_id=200,dl_dst=00:00:00:00:aa:01,actions=output:10

table=1,tun_id=100,arp,nw_dst=10.0.0.2,actions=output:1
table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:2
table=1,tun_id=100,arp,nw_dst=10.0.0.1,actions=output:10
table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:10

With a simple Ping, it is possible to verify that the VXLANs are correctly set up.

As now we understood how the VXLAN are used to support SFC in multi-tenant scenarios like SESAME, we can explain the role of NFVO. The main purpose of having an orchestrator as an upper management layer in the context of SEF, is mainly to automate the operation. Architectural choices apart, e.g. to have direct NFVO – VIM interface, to have direct NFVO – SDN controller interface or implementing it via VIM, etc., orchestrator is a management layer which sits between the system administrator (SCNO/VSCNO) and underlying infrastructure, i.e. servers and switches, Figure 2-13. In the context of SFC, from one side NFVO provides a use friendly interface to the system administrator where he/she can request network services. From the other side, after processing the admin's request, NFVO extracts the required information for setting up VXLANs, determines the flow entries, and verifiers connectivity between VNFs. This level of automation, especially one wide deployments, is a very important and handy. Furthermore, NFVO can provide automated and constant QoS monitoring and data visualization. This topic is widely discussed in D6.4.

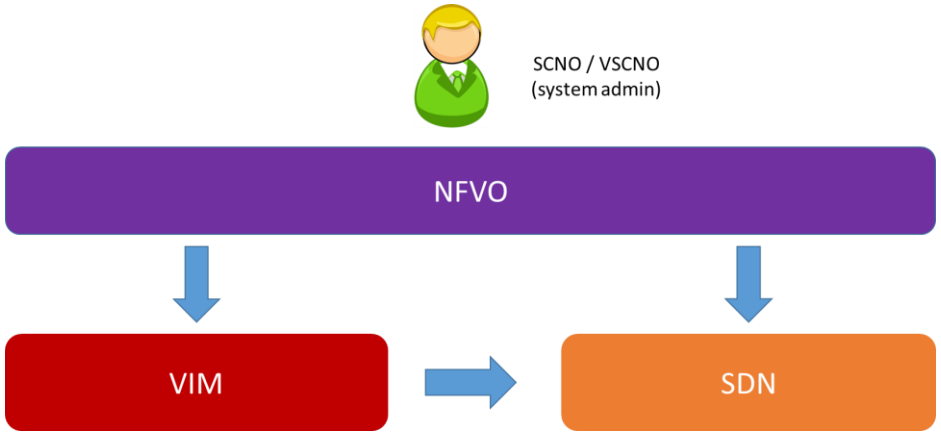


Figure 2-13: NFVO role on SFC

2.4 SESAME NFVO and Netfloc

2.4.1 Network service specification

TeNOR, as a SESAME NFVO, is a component that provides, *among others*, orchestration functionality to the SESAME cloud and also interaction with components such as Netfloc and the CESCM in order to provide top-down deployment of a network service, from the CESCM portal through Netfloc.

Related to the Network service deployment, TeNOR includes a template called Network Service Descriptor (NSD), (see Figure 2-14) that includes all the specific parameters related to the network service – VNFs, their virtual network connections, number of instances, SLAs, QoS details, etc. More on TeNOR and the specific details on the service implementation can be found in Deliverables D5.1 and D6.1.

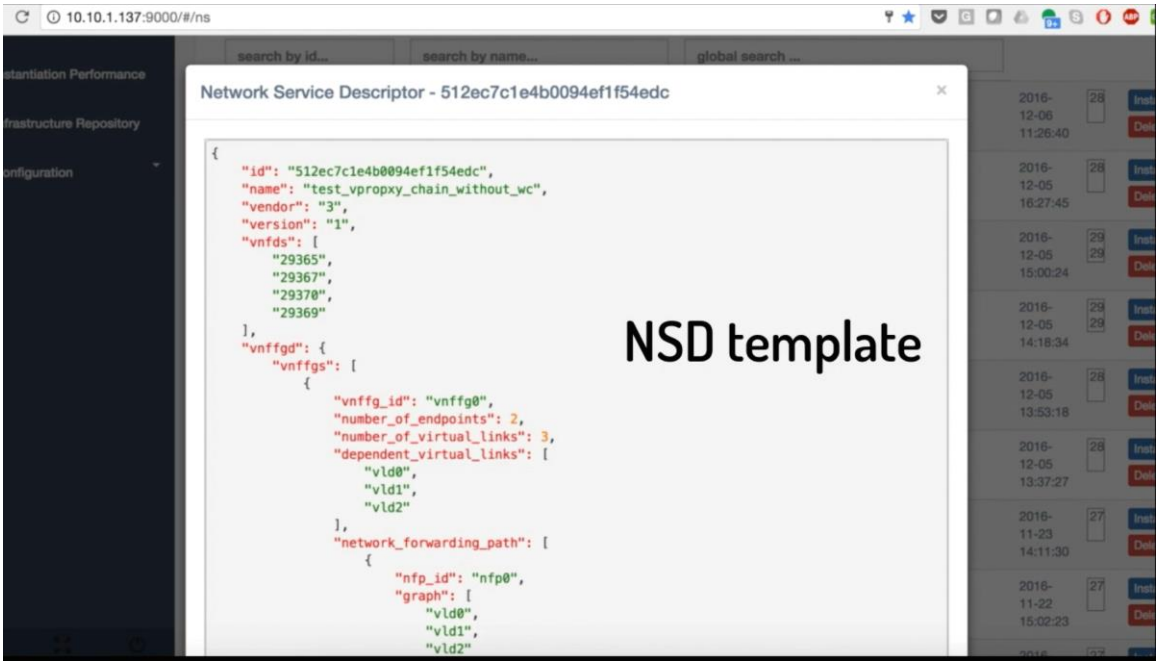


Figure 2-14: Network service descriptor used by TeNOR

2.4.2 Interfaces of TeNOR and Netfloc

With the purpose of providing seamless service chain functionality, Netfloc exposes REST APIs on the northbound, to third party developers and components such as the SESAME Orchestrator. In SESAME, TeNOR configures an SFC call to Netfloc in a service specification manifest, for the automatic deployment of the network service. For example, when the user requests a chained service whose traffic passes via several VNFs (telco and cloud-based), it is TeNOR that coordinates the deployment of resources as a per service specification, triggering Netfloc API for SFC service deployment *a posteriori*.

TeNOR also enables the insertion of the VNFs in the data path between the two end-points of the chain path, following the correct chaining sequence, as defined in the network service specification that is requested by customer.

The Service Chain component in Netfloc together with the Flow patterns, are triggered by an external API call. The API definition is generated via YANG models[33] and stored in the ODL data store. When a Service Chain is registered via the REST/Java API in a form of ordered list of Neutron ports, a Flow Chain Pattern is applied to the Network Path that includes the specific ports and interfaces.

Apart from the REST APIs, Netfloc provides a plugin for OpenStack Heat[34] in order to facilitate automatic deployment of all-in-one services. The SESAME NFVO uses a HoT YAML template[35] translation to ETSI specific VNFFGD notation [12] in order to provide service-specific OpenStack resources and call the Netfloc Chain create API, both in the same iteration.

Figure 2-15 shows the APIs exposed by Netfloc on the northbound. The CREATE, DELETE and LIST service chains are listed under the REST specification[36]. TeNOR uses those interfaces in order to deploy the chain in the OpenStack-SDN environment by passing the Neutron port list of the VNFs to be chained in the API call.

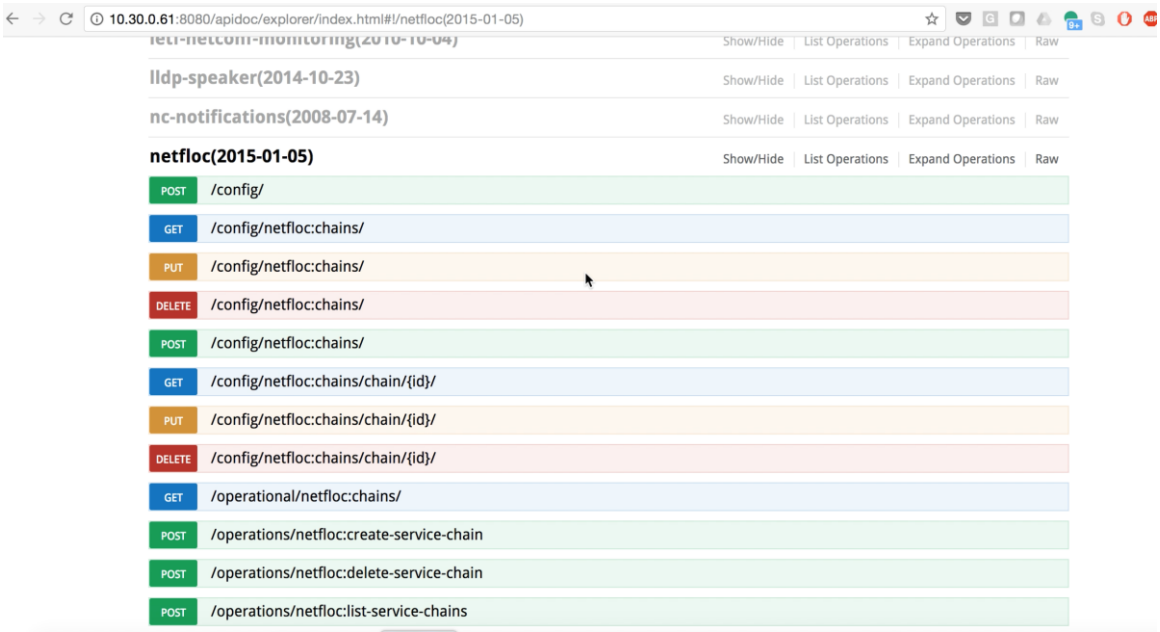


Figure 2-15: The RESTCONF interface contains the Netfloc service chain APIs

2.4.3 Automatic Service Chain deployment

On startup, Netfloc configures an overlay network in the VIM that connects physical to virtual ports. The Neutron API on the Northbound[37] feeds Netfloc with OpenStack-*related* data for the network elements such as: ports, floating ips, etc. The ovsdb plugin on the Southbound[38] is responsible for passing information about the physical infrastructure (nodes and switches) and OVS information related to topology.

---

[33] For more informative details see, *for example*: https://en.wikipedia.org/wiki/YANG
[34] For further details see: https://wiki.openstack.org/wiki/Heat
[35] For further information see: https://wiki.openstack.org/wiki/Heat/YAMLTemplates
[36] See: https://en.wikipedia.org/wiki/Representational_state_transfer
[37] http://docs.opendaylight.org/en/stable-carbon/developer-guide/neutron-northbound.html
[38] See, *for example*: http://docs.inocybe.com/dev-guide/content/_ovsdb_southbound_plugin.html

When traffic enters the VIM, that has been previously steered by the SC-VNF, Neutron is the first incoming node that accepts that ingress traffic and steers it to the rest of the nodes in the architecture. Depending on the use case, the traffic is then redirected to a certain VM (VNF) residing in one of the compute nodes.

Figure 2-16 shows two SESAME services composed of the following VNFs: vTU, vWatermark and the vDPI (vTC). It shows the connections between the VNFs established by Netfloc, after the traffic enters in the cloud environment. The Neutron node accepts the traffic and through OpenFlow rules the traffic is steered to the first VNF, in this case the vTC – entering the eht1 interface (ingress). The traffic is then classified and sent out on one of the egress ports towards the vTU and the vWatermark *a posteriori*.

The monitoring component, including the Prometheus exporter is integrated into the general CESM monitoring GUI and exposes SDN- and SFC-*related* data. Netflogi is another way to represent in a graphical manner the current status of the Netloc's network view and the statistics related to the service. As Figure 2-16 shows, there is a continuous monitoring activity performed during this process with the Netfloc's monitoring component being in charge. This component and the interfaces it has with the CESCM are described in the D6.3 and D5.3.

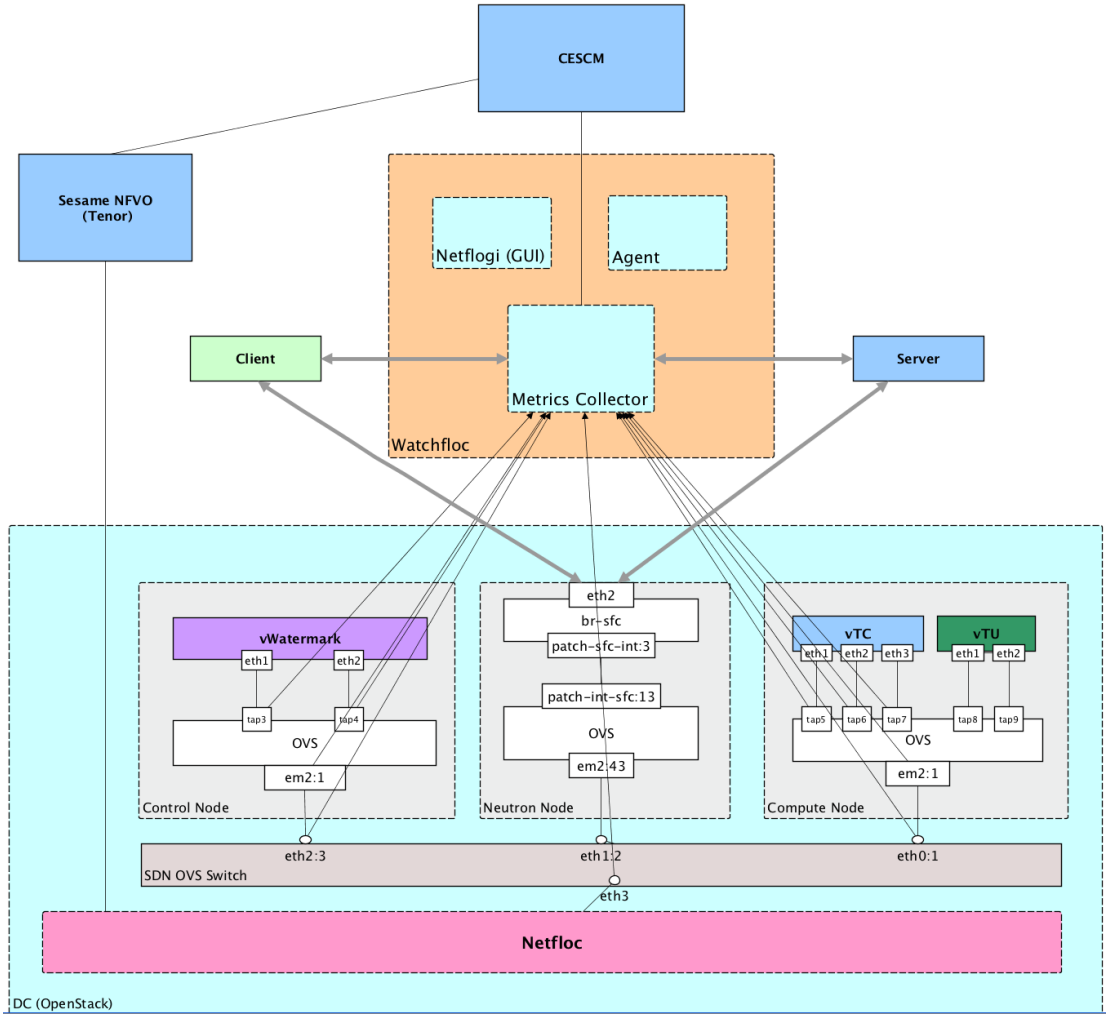


Figure 2-16: SFC use case architecture and Netfloc monitoring component

3 SESAME VNFs and their role in the Network Service

This section includes an overview of each of the VNFs used in the SESAME for the purpose of service chain. We focus on the elements required to establish the service chain, i.e. the interfaces (ingress, egress), the VNFs' functionality in context of the network service and finally we show some evaluation measurements.

3.1 vFW - virtualized Firewall

3.1.1 Features and description in the context of an SFC use case

The virtualized Firewall described and developed in SESAME project serves as a virtualized appliance to filter traffic directed to the UE, operating between the EPC and an eNB[39]. The current vFW implementation is based on Ubuntu 14.04 operating system[40] and the widely used OpenvSwitch software switch to build a stable, programmable and open-flow compatible firewall service. The vFW service consists of one VNF, which requires 3 interfaces to operate properly on the SESAME environment. The first interface is the management interface, which provides a set of capabilities for the VNF. Firstly, it provides remote access and communication with the VNFM in order to be able to manage its lifecycle through the TeNOR orchestrator. Additionally, the management interface operates as the monitoring interface, which is the interface to communicate with the SLA monitoring.

Furthermore, the other 2 interfaces serve as the LAN and WAN interfaces of a common firewall. In order to evaluate the performance of the developed vFW in the SESAME project, a set of tests was run using real captured data in an OpenStack environment. The experimental setup was 3 VMs, one sender of the traffic, the vFW in the middle and the client/receiver at the other end. The sender replayed in maximum speed the captured PCAP[41] file, and the performance of the vFW was monitored and captured. The vFW was tested in 2 different configurations in order to assess the correlation between the packet throughput and CPU/RAM allocation. Figure 3-1 shows the performance assessment results.

[39] See: *https://en.wikipedia.org/wiki/EnodeB*
[40] See: http://releases.ubuntu.com/14.04/
[41] See: *https://en.wikipedia.org/wiki/Pcap*

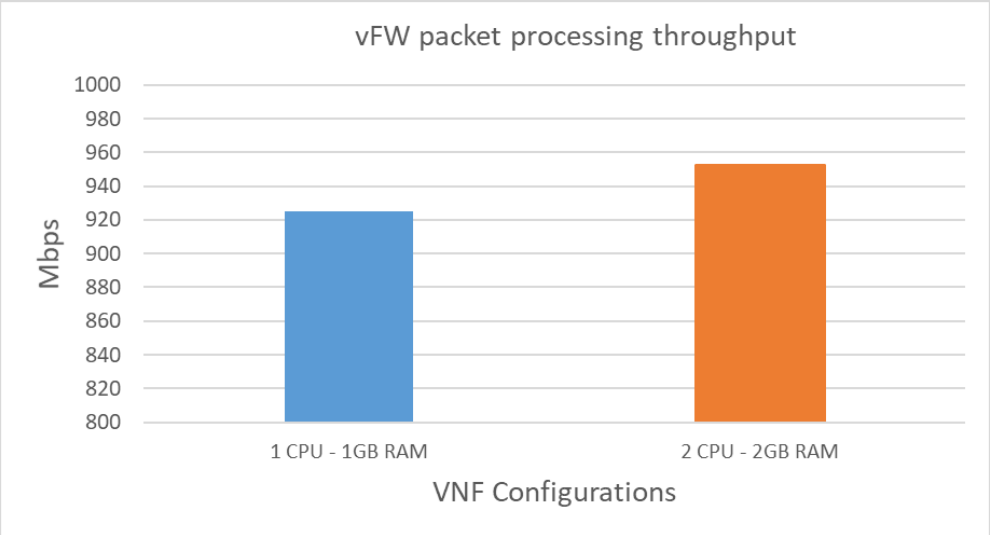


Figure 3-1: vFW performance results for packet processing in Mbps, based on 2 configuration setups.

The results show that the vFW can achieve processing speeds close to 1Gbps, which is more than enough for a SESAME test-case as the typical LTE backhaul does not surpass 170Mbps. Additionally, the CPU allocation can improve the performance of the vFW, but not in a considerable scale.

3.2 vDPI - virtualized Deep Packet Inspection

3.2.1 Features and description in the context of an SFC use case

The VNF described and implemented in SESAME project is based upon a variety of technologies to achieve advanced packet capturing, as well as traffic identification. The use case under test for nDPI[42] was to test it under low-power platforms to assess its performance, when computing resources are not available in ampleness. The platforms chosen where the Intel's DPDK-in-a-box[43] mini-PC, the Sistelbanda's B2239B ARM board (developed especially for the SESAME project), and the Raspberry Pi 3[44]. As the hosting platforms do not offer excessive computing power but aim to build a power efficient environment, the virtualization platform of the VNF that was chosen was the Docker[45], in comparison to OpenStack that has high resource demands.

The Docker is a form of Linux container[46] (LXC), which provides a self-contained execution environment that provides isolated CPU, memory, block I/O, and network resources based on sharing the kernel of a host operating system. In order to investigate the pros and cons of the container technology, the vDPI was developed also as an independent container application. The

[42] See: *https://github.com/ntop/nDPI/wiki/Supported-Protocols*
[43] Also see the context discussed by D. Hunt at: *https://dpdksummit.com/Archive/pdf/2016USA/Day02-Session16-DaveHunt-DPDKUSASummit2016.pdf*
[44] Also see the context in: *https://www.raspberrypi.org/products/raspberry-pi-3-model-b/*
[45] For further details see: *https://www.docker.com/*
[46] See: *https://en.wikipedia.org/wiki/LXC*

forwarding and the inspecting of the traffic are also performed using nDPI, and they are modified accordingly to function properly in a containerized environment. The results of the comparison for the 3 different platforms can be seen in Figure 3-2.

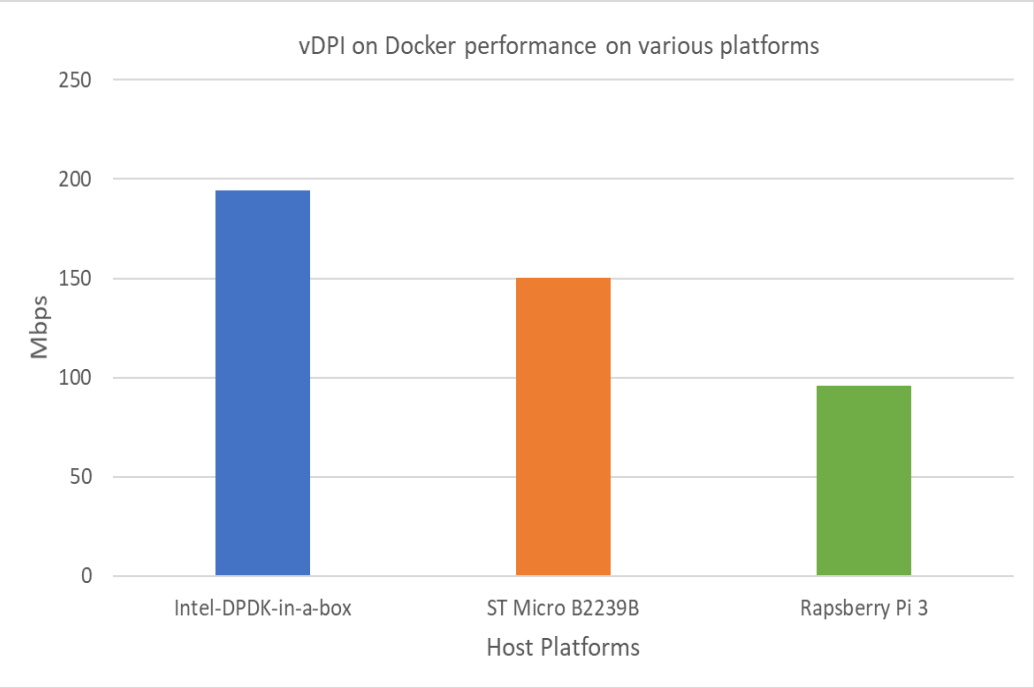


Figure 3-2: vDPI on Docker performance results for packet processing in Mbps, based on 3 different platforms.

The results illustrated in Figure 3-2 show the competitive performance of Sistelbanda's ARM board to Intel's x86 based DPDK-in-a-box. The Raspberry board performs considerable worse than the other 2 platforms, but still maintains a satisfiable rate for the needs of a telecom backhaul to serve as an Edge Light DC. It is important to mention Sistelbanda's board performance, which well covers the needs of a Light DC, and can even approach x86 numbers.

3.3 vWatermark - virtualized Watermark

3.3.1 Features and description in the context of an SFC use case

The virtualized Watermark VNF is designed in SESAME in order to process a video stream and embed it with a predefined watermark. In the use case of SESAME, the scenario for using a vWatermark, is based upon a set up of multiple operators, and each one was a differentiation of their video content. The video server streaming the video signal to the UE registered to each operator, transmits a "clean" video stream, with no features added. The stream is forwarded through the testbed and is first processed by the vGTP, in order to be stripped of the GTP header and have its flow stored for further processing. The stripped flow is then passed to the Light DC of the edge, hosting the vWatermark VNF, which waits for video streams to process them. After the embedding of the video stream is done, the video flow is forwarded back to the vGTP to be re-encapsulated with the GTP headers, so as to be ready to be handled by the eNB. The outcome video stream has the embedded operator logo, as the each VNF is set up and operating based on the requirements of the operator.

In terms of service function chaining the vWatermark VNF has been implemented to function in cooperation with the vVideoAnalytics VNF. The processed video stream out of the 2 VNFs included the annotation of the detected object and the watermark of the corresponding operator.

A working demo of the functional chain of the 2 VNFs was demonstrated in the EuCNC Conference of 2017[47].

3.4 vVideoAnalytics - real-time Video Analytics

3.4.1 Features and description in the context of an SFC use case

Two real-time video analytics-*based* VNFs (VA VNFs) have been designed and developed to investigate the two very import KPIs that 5G system aims to address, i.e. low end-to-end latency and reduction in the backhaul consumption. These two real-time VA VNFs are intended for augmented reality (AR) services and video analytics-*based* real-time remote control of IoT devices (Smart IoT) services respectively. These applications both require low end-to-end service latency and consume large amounts of real-time video streaming data to perform the desired video analytics tasks.

The commonality between the two VA VNFs is that they both take the real-time video data streamed from video cameras, such as fixed CCTV cameras or mobile cameras mounted on smart phones, drones or robots, as input data. Video analytics is applied to the video input in real time to carry out meaningful analyses with the output of the derived analysis results or enhanced video data to the relevant receiving devices. However, the difference is in the case of AR service, the output data is the processed video data with additional information added to the input video data, whereas in the case of Smart IoT service, the output data is the derived video analysis results or instructions for autonomous IoT device control.

In terms of the main features of the two developed VA VNFs, both are based on the continuous tracking of a predefined object of interest. For the AR service, the additional annotations regarding the object of interest are added to each video frame, based on the continuous object tracking. For the Smart IoT service, the monitoring camera or cameras of an object of interest are remotely controlled in order to continuously track the object of interest.

The object tracking function is a common function used by both VA VNFs consisting of the following functional components: object detection, object recognition, object tagging and object location identification, which are the primary functionalities supported by the AR VA VNF. The Smart IoT VA VNF, on top of the object tracking function, has further functions such as situation identification, control decision-making and control decision execution to realize the situation-aware remote control of IoT devices. The implementations of all these functional components are in Python, mainly based on the usage of Open Source Computer Vision (OpenCV) libraries [13] and Caffe libraries [14] depending on the type of object being tracked and the real-life scenarios, and have been tested on x86 platforms generating desired analysis results.

To achieve the real-time video streaming from the video source to the VA VNF, and from the VA VNF to the relevant receiving devices, the widely used FFMPEG [15] is adopted at the video source and the VA VNF to pipeline the live video data and integrate them to a video stream flow, which has been tested on x86 platforms generating desired performance results.

[47] Also see: http://www.eucnc.eu/?q=node/134

3.4.2 Requirements in the context of an SFC use case

From the SFC's point of view, the incoming and outgoing data of the VA VNFs should be pure IP packets for the VA VNFs to perform their video analytics functions. The VA VNFs need to be chained with a function that is able to decapsulate and re-encapsulate the GTP tunnels, so that the VA VNFs can obtain the inner pure IP packets that carry the video data. After processing the video data the VA VNFs can forward the processed video IP packets to this function module in order for it to route the processed video data for the correct receiving device through the associated GTP tunnel. As an SFC example, the current integration of the AR VA VNF with the SESAME platform is illustrated in Figure 3-3.

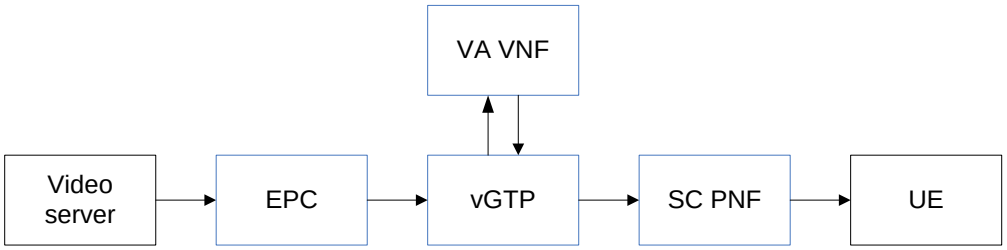


Figure 3-3: SFC of the AR VA VNF

From the VNF performance's perspective, to ensure the VA-*based* services are provisioned to the relevant end-devices with unnoticeable delays compared to the real-life situations, the required end-to-end service latencies need to be within 100 ms; this is the time between when a video frame is sent from a streaming camera to the mobile edge VA VNF and when the processed video frame or the derived instruction corresponding to the sent video frame arrives at a receiving UE. Additionally, the video frame processing rate at the VA VNFs, i.e. the number of frames being processed per second, should be higher than -or equal to- the streaming camera's frame rate, in order to avoid that the end-to-end latency of each frame increases with the number of the frames being processed. To be specific, if the streaming camera's frame rate is 30 fps, then the processing of each video frame at the VA VNF needs to be completed within 33 ms.

Therefore, in order to "meet" these stringent requirements on end-to-end service latency and video data processing time, the resources (such as CPU, memory and transmission data rate to/from the VA VNF) provided to the VA VNF need to be dynamically adjusted, to cope with the time varying traffic load and the type of required VA tasks.

For this purpose, some important monitoring parameters to guarantee the SLA of the video analytics-*based* services are discussed in the following section.

Related to the platform choice x86 *vs.* NXP, the VA VNFs were limited to an x86 implementation only.

3.4.3 SLA monitoring

The vVideoAnalytics VNF forms part of a SFC that will support user services such as Augmented Reality and, in this context, the key performance indicator that need to be monitored, managed and controlled by the SESAME infrastructure is the end-to-end latency necessary to meet the end-users quality of experience estimated at around 100ms. The measured latency of the resulting SFC will need to account for the network latency both in the uplink (UL) from the video source and from the downlink (DL) to the video display. In addition, in order to accurately measure the end-to-end latency, the SESAME infrastructure will need to monitor the processing delay imposed by each VNF in the service chain.

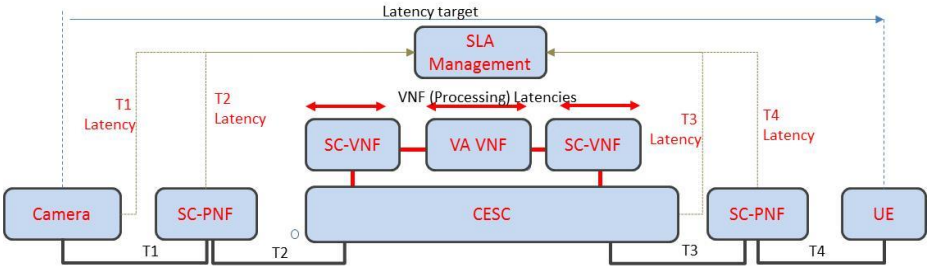


Figure 3-4: Management for end-to-end latency

The end-to-end service is depicted in Figure 3-4. In order to monitor the end-to-end latency, a number of monitoring points are proposed in the SESAME infrastructure:

- UE reporting (video source) for the uplink frame delay over the radio access interface assuming a wireless connected device.
- Small Cell reporting the downlink frame delay from video transmission to the UE handling the video display.
- Processing delay incurred by each of the VNFs included in the service chain, when handling individual service video frames averaged over a specified time period. For the vVideoAnalytics service, this will cover the processing delays incurred by the SC VNF (both for the uplink and downlink transmission) and the vVideoAnalytics VNF.

The vVideoAnalytics VNF, in order to prevent degradation in the service, needs to -at least- guarantee that the average frame processing time does not exceed the video frame arrival rate of 33ms in the current implementation. However, depending on the delays in other parts of the end-to-end service chain, the processing time allocated for video analytics may be reduced by the SLA management component.

For a latency-constrained service, each VNF in the service chain will need to monitor the processing time for the flow through the VNF. If this exceeds some predefined threshold averaged over a specified period, the VNF should notify the SLA manager.

In the case that such a threshold violation, the SLA management will need to assess the overall impact on the end-to-end latency target. If there is no impact, the SLA manager may decide to do nothing unless further degradation of the VNF latency is reported. One option for the SLA manager is to request that the resources be scaled up for the VNF in question.

In general, the VNF will support multiple flows and, in such cases, the SLA manager could apply access control strategies -or a priority schemes- for handling new service requests.

4 VNF placement and QoE awareness

4.1 QoS over Multi-Tenant CE-RAN

Ensuring the QoS per tenant base, assuring the SLA, is an important aspect in the service lifecycle management. QoS assurance demands establishing a feedback loop consist in three main steps:

- **MONITORING:** A phase in which performance metrics are collected from the radio/cloud/software elements (e.g. SC physical network function, VMs, etc.) and handed over to the next step (decision-making). Depending on the NS deployment and nature the metrics to be collected may vary each time.

- **DECISION-MAKING:** A phase in which performance metrics collected in the previous step are processed. Depending on the situation and available resources, a decision will be taken to ensure the level of QoS (with the help of a dedicated algorithm). Besides available resources, in a multi-tenant scenario, the decision making process needs to take into account the status of other tenants.

- **REACTION:** Upon making a decision, the management/orchestration system needs to coordinate the interaction with the other lower level modules such as Element Management System (EMS), VNFM and VIM to react appropriately.

Implementing this QoS loop means adding the radio dimension to the standard cloud orchestration system defined by the European Telecommunications Standards Institute (ETSI), and/or to shift the traditional radio network management mentality towards a cloud-*oriented* mind-set. For instance, leveraging on the radio traffic profile over a certain period (e.g. a day, a week, etc.), it would be possible determining when and where cloud services need to be scaled up/down in a Point of Presence (PoP).

Decision-maker could be placed in several parts along the control plane (VIM, VFNM and NFVO) and the NMS in the management plane [16]. When a change on the architecture is processed, a feedback should be given to the learning module in order to analyze the results and infer enhanced decision rules.

Starting at the lower level on the proposed architecture, SDN controller should provide QoS. Networking between VNFs of the chain is an important task in the cloud architecture. Intercepting traffic and forwarding to the correct NS is a challenge resolved by the SDN controller. By taking advantages of standard protocols such as OpenFlow, SDN controller can prioritize traffic according to QoS levels.

At a higher level, mapping mechanism deals with the allocation of the softwarized components of a NS into the resources of the CESC cluster; in other words, this implicates where to instantiate the VNF chain that composes a NS. Placement algorithm checks the available virtual resources in the NFVI and the instantiation requirements. Bearing that in mind allocates the resources in the optimal location.

The placement algorithm resides inside the VIM. As described in SESAME D5.3, the proposed placement mechanism minimizes power consumption subject to latency constrains. This placement algorithm finds the optimal placement in order to achieve best energy-aware provisioning in compliance with the service level agreement.

By default it allocates the new NS using the available resources in the Light DC. Due to its high cost, it is not recommended to dynamically recompute the algorithm, in order to find actively the best placement for all the existing NS. Reordering VNFs while executing is possible thanks to live-migrations techniques inside the cloud enabled SCs. To achieve a fast migration, some requirements have to be taken when designing the cloud architecture [17]. Figure 4-1 depicts the intended replacement algorithm.

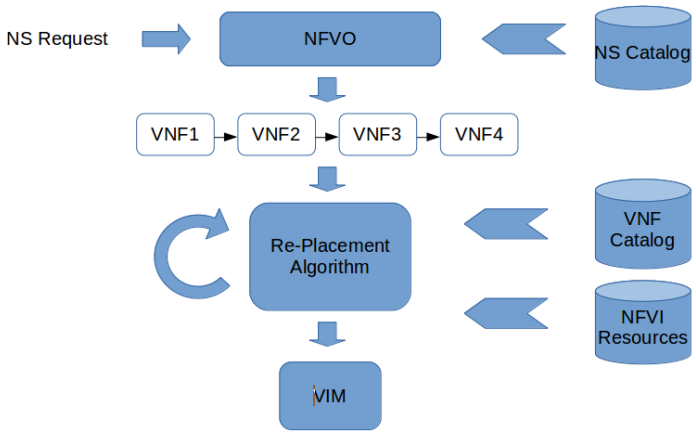


Figure 4-1: Replacement algorithm

The algorithm execution and placement allocation time is critical. The proposed algorithm deploys new instances over the cloud resources available at the RAN side. In other words, placement of service VNFs that conforms to the aforementioned edge services are allocated in the remaining infrastructure, in order to not re-compute the whole optimal location of all VNFs. There must be a commitment between the handover introduced in the system to live-migrate some services and the benefits of QoE obtained.

In the CESCM, VNFM is in charge of the lifecycle management of the VNFs. VMFM is able to apply policies for NS-level rescaling and reconfiguration to achieve high resource utilization. General purpose clouds introduce automatic rescaling algorithms. Generally, a minimum and maximum number of intestacies is given to the VMFM, in order to select the optimum QoE. In some cases, VNF must be rescaled so it generates a new template indicating the VIM the new architecture, thuh letting this “entity” to compute the best placement for the requested instances.

The Metric Aggregator (MA), as its name indicates, is the responsible for combining and filtering the collected monitored parameters and associates them with the running services over the SESAME platform. MA continuously processes the collected monitoring values for the QoS or SLA evaluation. Depending on the use case (UC), the algorithm used for this purpose might be a simple threshold checking logic or a complex multi-attribute decision-making process. Bearing in mind its fast processing time, we select the threshold checking procedure. To this end, a threshold, i.e. Warning Threshold (WT), is defined for each of the monitored metrics, aiming to detect the critical values per case and trigger a warning alert for the specific failure component. Note that, WTs are defined in a way that the multi-tenant capabilities of the system are taken into account. In this sense, the proposed solution is inflexible, i.e. values of WTs will not adapt dynamically according to the real-time needs of the system. Having said that, SESAME targets introducing a more complete solution in the project lifetime.

In conjunction with a more complex data process, SESAME envisions a module denoted as the Decision Support System (DSS), as shown in Figure 4-2. The main responsibility of DSS is to

detect the level of severity on the QoS evaluation process done in MA and decide whether a reactive -or a proactive- action is needed. Basically, such a decision will be made based on the high-level SLA agreements made with VNOs. Such a high-level agreement will indicate points such as at what PoPs a VNO will be present, how much of overall IT resources are dedicated to a VNO, etc., as presented in [18]. With the help of DSS, the SESAME solution, in addition to the *per NS performance metrics* will bring in the high-level SLA agreements into calculations.

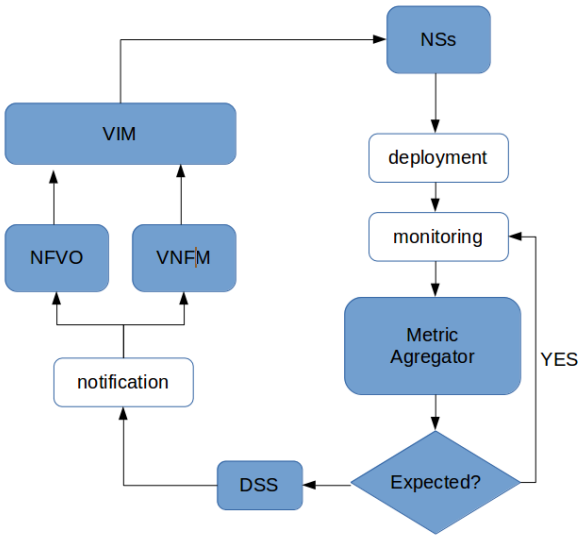


Figure 4-2: QoS feedback loop

NFVO is the responsible module to perform the appropriate reaction based on the analysis made in the previous steps. There is a set of possible reaction mechanisms, including: (i) reconfiguring the flow of data in a NS (i.e. changing the SDN rules); (ii) migrating the NS within the PoP, from one PoP to another; (iii) scaling up/down the whole NS (i.e. instantiation of a parallel service or terminating a running one); (iv) scale in/out of VNF (i.e., adding more resources to a VNF). Depending on the nature of requested process (proactive/reactive), nature of warning (radio/cloud), available resources, possibilities of VNFs indicated in the VNFD by developers, NS agreement with VNO depicted on NSD, the NFVO selects and applies a reaction mechanism. Figure 4-2, depicts also the workflow of the reaction process. That is, NFVO has the possibility to interact with the EMS (managing the radio parameters of SC or SCs) and/or VIM (managing the cloud infrastructure – e.g. OpenStack). In this sense, EMS and VIM are the enablers' components for the adaptation service of NFVO on cloud and/or radio domain. It is worth noting that, in a more advanced scenario, EMS and VIM can also be enhanced with "self-x" features, aiming to enable them to make local decisions for a CESC or a set of CESCs (subset of Light DC). From the infrastructure owner perspective, besides the status of each provided service, the overall system performance is also highly relevant; therefore, MA is also able to expose general information about the system status. This information helps the infrastructure owner to perform a temporary capacity upgrade if the expected use of resources does not meet planned terms, e.g. due to overloaded use of services/users. The data can be seen visually from the CESCM Portal through, for example, a customized dashboard.

4.2 Placing a requested NS

In this section, results of the energy-aware VNF placement system for a CE-RAN environment proposed in D5.3 are analysed. We use constraint programming to solve the discrete optimization problem of resource assignment, combined with robust optimization techniques (RO) to develop an energy-aware VNF placement policy. Thus, our approach differentiates from others in the use of RO, which allows us to generate a suboptimal placement policy that enables the introduction of uncertainty. In particular, we introduce service demand uncertainty in order to capture the variable nature of traffic flows size and per service job execution burden.

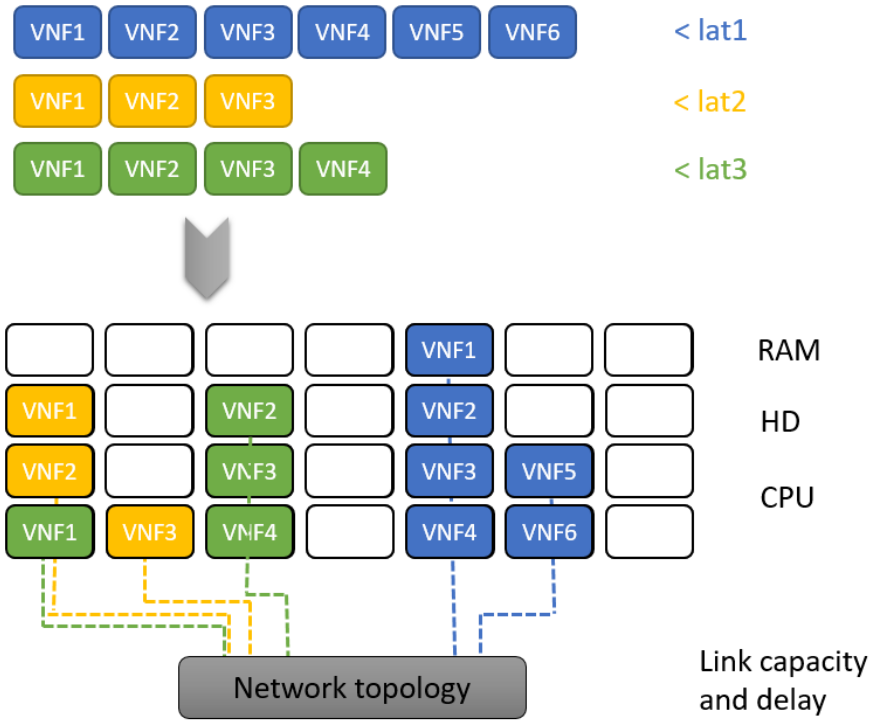


Figure 4-3: Placement algorithm

The placement algorithm (see Figure 4-3) inspects the VNF catalog for the parametrization of the VNF instances needed to match the SLA, and checks the available resources provided through the virtualized infrastructure. VNFs are characterized by the use of resources (i.e. number of cores, RAM and storage) according to the aggregated bitrate (i.e. characterized aggregated traffic flows of a NS) served by the NS they belong to. Moreover, the service latency of each VNF also depends on the traffic load supported by the VNF instance. Lastly, virtualized resources may also include other hardware appliances, such as hardware accelerators (HWA), that can improve the performance of a VNF in terms of latency. This way, these additional appliances help matching SLA with the tenant, but at a higher energy cost.

All the VNFs composing the service chains of the requested network services must be allocated in the available virtualized resources, including a given number of CPUs, hardware appliances, RAM and storage space and network bandwidth. We assume that each VM runs in a single core that is devoted to the execution of a single VNF instance, but one VNF instance can scale from one to many VMs upon the service requirements of the NS. Besides, the transmission latency along the network topology between two VNFs allocated in different micro-servers depends on the traffic load of the involved links and the size of the flow to be transmitted. With this information, the placement algorithm assigns each VNF instance to one or more VMs with the

objective of minimizing the energy consumption of the whole system while matching the latency constraints for the NS.

Combinational optimization problem: The mathematical VNF placement problem model presented is a combinatorial or discrete optimization problem equivalent to the well-known knapsack problem[48]. This means that the decision problem is NPcomplete, while the optimization problem is NP-hard [19]. The proposed optimization model has been solved using constraint programming as a powerful paradigm for solving combinatorial search problems [20].

We used Minizinc [21], a free and open-source constraint modeling language, to model the optimization and constraint satisfaction VNF placement problem in a high-level, solver-independent way. Then, Minizinc compiles this model into FlatZinc[49], a solver input language that is understood by a wide range of solvers. In our case, Gecode solver [22] provided the best performance in the search of the optimal solution.

Robust Optimization: Many existing placement works assume perfect knowledge on input parameters pointing, *thus*, to the formulation of a deterministic optimization problem. Unfortunately, in the real world, the estimated values of the input parameters may differ due to biased data, unrealistic assumptions or numerical errors, affecting the obtained optimal solution and its performance. This potential deviation of the nominal or expected input may lead to the violation of the problem constraints and, *therefore*, make the obtained solution suboptimal or even unfeasible. The uncertainty of modeling parameters has been traditionally handled through stochastic programming and sensitivity analysis, but Robust Optimization techniques have recently arised as a powerful tool to manage the impact of uncertain input sets.

Here a robust constraint-*based* placement solution is proposed that deals with service demand uncertainty. With this RO approach, we consider a trade-off between the problem optimization and the protection from deviations caused by input parameters uncertainty. This uncertainty in service demand has an impact on several components on the formulation previously presented. On the one hand, we consider that a VM which is running a specific VNF requires an expected use of CPU and, *thus*, a certain constant power consumption due to CPU. Nevertheless, the uncertainty in traffic demand may cause uncertainty in the CPU requirement/use and, *consequently*, on its energy consumption and processing delays. On the other hand, both energy consumption and delays in switches and links depend on that demand uncertainty as well.

Service Constraints: Service constraints are imposed by the Key Performance Indicators (KPI) of the corresponding Service Level Agreement (SLA) between the tenant and the CE-RAN operator. In our work, the SLA KPIs include the following network parameters: maximum accepted latency for the NS, aggregated user bit rate and a robust protection parameter.

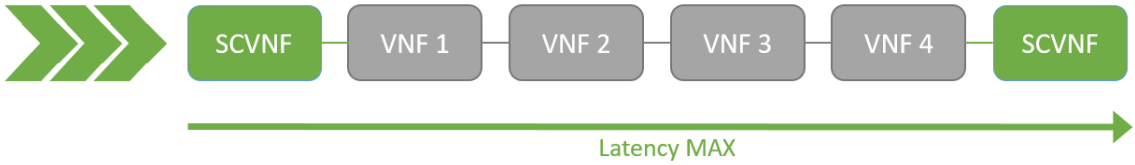


Figure 4-4: Network Service Model

[48] Also see, *for example*: https://en.wikipedia.org/wiki/Knapsack_problem
[49] See: http://www.minizinc.org/downloads/doc-1.3/flatzinc-spec.pdf

The robust protection parameter is therefore related to the expected peak traffic demand deviation and will be used to overestimate the allocation of virtualized resources, in order to be prepared for eventual user peak demands.

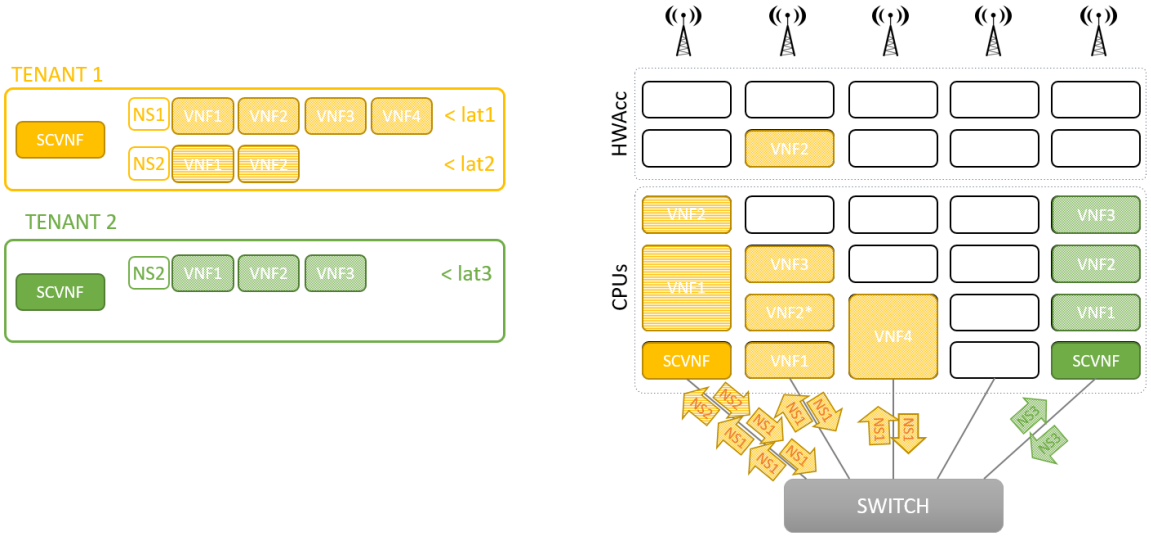


Figure 4-5: Model design

In the proposed model two NSs of the same tenant share the same radio VNF - SC VNF (SCVNF). In this way, the common radio operations are performed in a coherent way between the different edge services of a tenant, and the data paths of the specific service instances per tenant are split. The SCVNF of each tenant will be shared by all de NSs of this tenant, which means that the use of resources of the SCVNF must then consider the sum of the aggregated flows of the NSs of the same tenant.

Energy model: We consider a 6 SCs deployment, e.g. to cover a football stadium for periodical flash events. The SCs are connected in a star topology through an Ethernet switch. Each SC is composed of 8 cores, 16 GB RAM, 100 GB of storage and 2 HWA (e.g. GPU). We also consider 2 NSs that can be offered at 3 service flavours (bronze, silver and gold) that determine the latency constraint for each NS and the nominal aggregate user bit rate served.

	Gold	Silver	Bronze
Flow deviation	20%	10%	0%
Max latency NS1	100 ms	120 ms	150 ms
Max latency NS2	80 ms	90 ms	110 ms

Figure 4-6: Microserver characterization

In the following, Figure 4-7 and Figure 4-8 show the experimental energy model employed for the experiments. This value shows the VNF modelling of the VNFs that compose the proposed NSs for the user traffic demand that will be employed in the evaluation experiments. The user traffic values include the nominal bit rates of services NS1 and NS2 (130 Mbps and 90 Mbps,

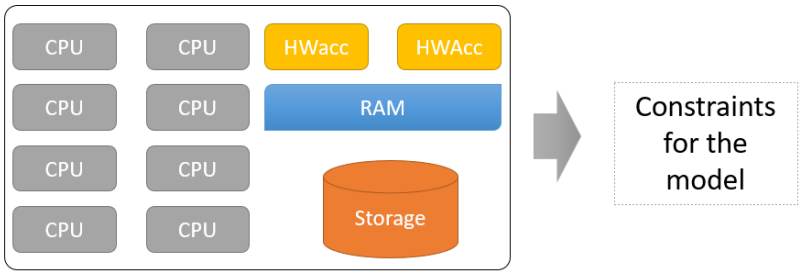
respectively), as well as the values when a 20% protection parameter is applied to the nominal values (156 Mbps and 108 Mbps, *respectively*) in order to study the effect of robust optimization techniques on the placement decision.

As a final remark, the placement behavior of VNFs can be altered by the use of HWA. The assignment of a HWA to a VNF implies that the service latency of the correspondent instance is reduced 3 times, at a cost of incrementing the power consumption by 10. It is important to note that when the VNF instance needs more than 1 CPU to execute without the assistance of a HWA, *then*, if the placement algorithm assigns it to a HWA, the VNF will need a single core.

		VNF1	VNF2	VNF3	VNF4	VNF5	VNF6
Nº of cores per instance	90 Mbps	1	2	1	1	1	1
	108 Mbps	1	2	2	1	1	1
	130 Mbps	1	2	2	1	1	1
	156 Mbps	1	2	2	1	1	1
CPU usage	90 Mbps	30%	20%	90%	50%	50 %	20%
	108 Mbps	40%	30%	60%	60%	60%	30%
	130 Mbps	40%	30%	60%	60%	60%	30%
	156 Mbps	40%	30%	90%	40%	20%	30%
Service latency	90 Mbps	10 ms	15 ms	30 ms	35 ms	20 ms	25 ms
	108 Mbps	10 ms	15 ms	35 ms	40 ms	25 ms	30 ms
	130 Mbps	10 ms	15 ms	35 ms	40 ms	25 ms	30 ms
	156 Mbps	12 ms	16 ms	40 ms	50 ms	30 ms	35 ms

Table 1: VNF characterization for the service flavours

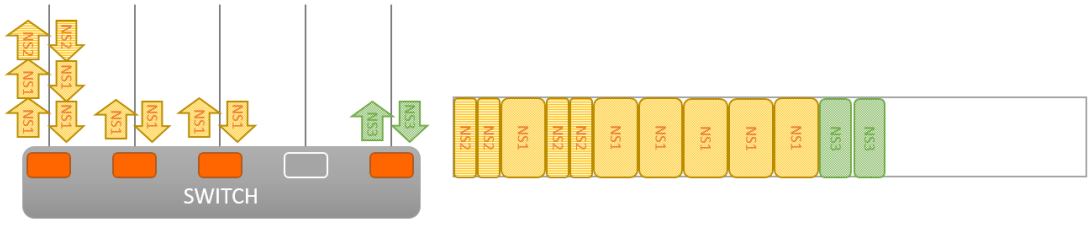
The power consumption of the cores that run VMs depends of the aggregated user traffic served by the correspondent VNF, and this relationship may be non-linear. It must be also understood that when a VNF instance scales to a higher number of cores, the individual load of the involved CPUs decreases, leading to a lower power consumption per core.



CPU Usage	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Power consumption (W)	5	30	40	55	70	85	100	120	130	139	145

Figure 4-7: Energy model

Additionally, the energy consumption of the networking devices (e.g. an Ethernet switch) depends on the aggregated flow that traverses the device to forward data packets between SCs and the number of active ports.



Traffic capacity	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Power consumption (W)	0	1	2	3	4	5	7	10	12	14	15

Figure 4-8: Switch characterization

The proposed model must to minimize the power consumption of the overall set, which entails the power consumption of the CESCs and the switch. And it must be satisfied meeting the hardware constrains, link capacity constraints (i.e. link capacity of 1000Mbps with the switch) and NSs latency constrains.

NS latency is a model constraint composed by the latency of the VNF in addition to the link latency generated in the inter-VNF connection. Both depend of the aggregated user traffic served (as illustrated in Figure 4-9).

The simulation results end with the optimal VNF energy-aware placement for the two operator NSs meeting the design constraints:

	TENANT 1		TENANT 2	
	NS1	NS2	NS1	NS2
Service type	Gold	Gold	Gold	Silver
Max flow	130 Mbps	90 Mbps	130 Mbps	90 Mbps
Max latency	100 ms	80 ms	100 ms	90 ms
Deviation	20%	20%	20%	10%
Peak flow	156 Mbps	108 Mbps	156 Mbps	99 Mbps
Latency	70 ms	69 ms	91 ms	82 ms
Energy	6943 W			

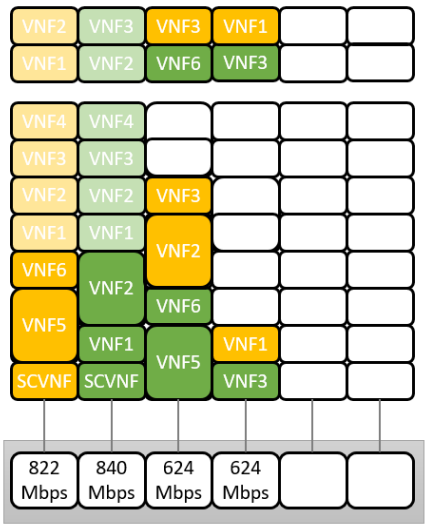


Figure 4-9: Placement results on empty cloud-enabled SC cluster

In case that a new NS is requested, the placement may not change the allocation of the placed VNFs. The cost of live-migrating a VNF is expected high, so the placement algorithm must find an optimal solution in order to allocate the requested NS using the available resources.

	TENANT 1		TENANT 2		TENANT 3
	NS1	NS2	NS1	NS2	NS1
Service type	Gold	Gold	Gold	Silver	Bronze
Max flow	130 Mbps	90 Mbps	130 Mbps	90 Mbps	130 Mbps
Max latency	100 ms	80 ms	100 ms	90 ms	150 ms
Deviation	20%	20%	20%	10%	0%
Peak flow	156 Mbps	108 Mbps	156 Mbps	99 Mbps	130 Mbps
Latency	70 ms	69 ms	91 ms	82 ms	130 ms
Energy	10330 W				

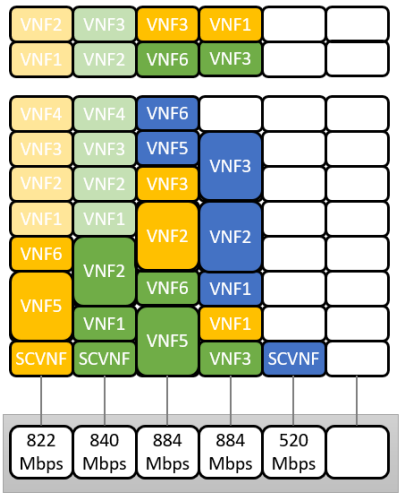


Figure 4-10: Placement results of a new service on cloud-enabled SC cluster

4.3 Dynamically replace VNFs to meet SLA

In case of a breach of the KPI agreed in the SLA, the placement algorithm has the decision power to reallocate the operating NS. In this case a failure of a CESC is simulated, caused for example by the fall of the microserver or the connection link. Figure 4-11 illustrates the example where a dynamical reallocation only of the VNF of the fallen microserver is proposed. The placement algorithm must find the optimal allocation for the affected VNFs in the available resources.

	TENANT 1		TENANT 2		TENANT 3
	NS1	NS2	NS1	NS2	NS1
Service type	Gold	Gold	Gold	Silver	Bronze
Max flow	130 Mbps	90 Mbps	130 Mbps	90 Mbps	130 Mbps
Max latency	100 ms	80 ms	100 ms	90 ms	150 ms
Deviation	20%	20%	20%	10%	0%
Peak flow	156 Mbps	108 Mbps	156 Mbps	99 Mbps	130 Mbps
Latency	70 ms	69 ms	91 ms	82 ms	130 ms
Energy	10330 W				

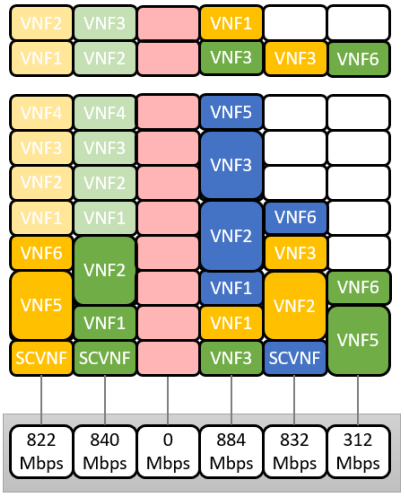


Figure 4-11: Replacement results of unavailable SC on cloud-enabled SC cluster

As long as many allocations take place, the cost function represented by the total power consumption separated from the optimal results done by a replacement of the overall NSs. Notice that in this example a 22% power optimization is obtained. The decision of a replacement to improve the QoE is left to the operator, which must evaluate also the cost of the live-migrations.

	TENANT 1		TENANT 2		TENANT 3
	NS1	NS2	NS1	NS2	NS1
Service type	**Gold**	**Gold**	**Gold**	**Silver**	**Bronze**
Max flow	130 Mbps	90 Mbps	130 Mbps	90 Mbps	130 Mbps
Max latency	100 ms	80 ms	100 ms	90 ms	150 ms
Deviation	20%	20%	20%	10%	0%
Peak flow	156 Mbps	108 Mbps	156 Mbps	99 Mbps	130 Mbps
Latency	91 ms	74 ms	95 ms	73 ms	150 ms
Energy	7967 W				

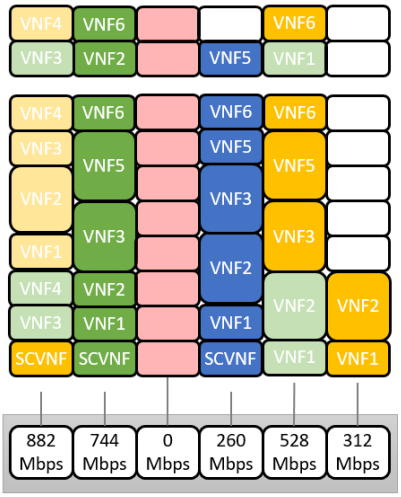


Figure 4-12: Optimal replacement results on cloud-enabled SC cluster

A nearly full resource occupation could end in an impossibility to place a requested NS with the agreed KPIs using the available resources. In such scenario, a global reallocation could fit the situation. However, in the case of an impossibility to agree with the QoS a service rejection must be done. In that case, service in NS contracted with the low KPI may be disrupted in favour of a correct service continuation of the higher KPI NSs.

5 Virtualization of RRM/SON functions for multi-tenant small cell networks

5.1 Introduction

The deployment of multi-tenant Radio Access Networks (RAN) in future Fifth Generation (5G) systems is expected to be fuelled by the inherent flexibility associated to the introduction of Network Function Virtualization (NFV) technologies in the RAN through the software implementation of network functions running on general purpose computing/storage resources [12]. The advent of the NFV paradigm provides an inherent capability to add new functionalities, extend, upgrade or evolve existing functionalities and to customize the RAN on a per-tenant basis. Given that 5G systems are envisaged to support a wide range of business models and vertical markets with very diverse performance and functional requirements [23], RAN customization arises as an essential capability for 5G.

The customization of a RAN on a per-tenant basis requires the implementation of "network slicing" mechanisms, which enable the deployment and operation of multiple logical networks over a common physical network infrastructure [23],[24]. By ensuring isolation among "network slices" (i.e., one tenant's traffic do not affect the performance observed by another tenant) [24], customisation to best serve the needs of specific applications (e.g. mobile broadband, Internet of Things applications) and/or tenants (e.g. enterprises, service providers, content providers, etc., across different sectors such as public safety, utilities, smart city, automotive, etc.) is greatly facilitated [25].

RAN customisation is strongly related to Radio Resource Management (RRM) and Self-Organizing Network (SON) functions, also denoted in the context of SESAME as Self-X functions, [25],[26]. A given set of RRM/SON algorithms implements and enforces a set of pre-established policies (e.g. mobility policies, service usage policies) and operational constraints (e.g. capacity limitations).

In this context, the design and implementation of RRM/SON solutions for 5G face radical changes compared to the legacy 3G/4G *status quo*. On the one hand, a shared radio interface among multiple users from diverse tenants requires advanced RRM/SON algorithms able to ensure a fair and efficient usage of the radio resources. On the other hand, RRM/SON functions can be implemented in a highly flexible and customizable way by exploiting the NFV paradigm, "opening the door" for more open markets where third parties could provide RRM/SON solutions that can be flexibly implemented in the network. Therefore, there is the opportunity for substantial innovation in the way that RRM/SON functions are conceived and brought to the market.

The provision of RRM/SON as virtualized functions for small cell networks is identified as one of the possible use cases by the Small Cell Forum [27]. In [28], virtualization of RRM/SON functions is considered in the context of a cloud-RAN, emphasizing the benefits of centralization, while [29] presents some considerations in relation to the implications of the functional split between virtualized and physical functions with respect to some RRM functions. Overall, [27]-[29] reflect industrial interest and fuel further work in this direction, particularly in the area of multi-tenant networks, where no tangible and publicly available outcomes have been produced to date.

Based on the above, this section presents a framework for implementing virtualized RRM/SON functions in multi-tenant small cell networks based on the SESAME architecture. The framework relies on the use of a VNF for handling the acquisition, processing and collection of different measurements in the RAN, denoted as Radio Network Information Service (RNIS). The management of the involved VNFs by means of the orchestrator is discussed in Deliverable D6.4,

where also specific examples of RRM/SON VNFs are provided, assessing the orders of magnitude of the requirements of the involved Virtual Network Functions (VNFs), in order to illustrate the potentials of the considered framework [30].

5.2 Virtualization of RRM/SON functions

The dynamic management of the radio resources in a cellular network is carried out by a set of RRM functions aimed to ensure the efficient use of the resources while, *at the same time*, meeting the Quality of Service (QoS) requirements of the services provided to the users. Examples of these functionalities include the Admission Control (AC), Mobility Control (i.e. handover), Congestion Control, Packet Scheduling (PS), etc. Each RRM function is implemented as a decision-making logic. Traditionally, RRM algorithms are vendor-specific.

The parameters of each RRM function should be properly configured to achieve an efficient operation of this function. This configuration can be static, e.g. specified by the Element Management System (EMS) when the radio network is activated, or it can be dynamically and automatically modified at runtime by means of a Self-Organizing Network (SON) function. In general, SON refers to a set of features and capabilities for automating the operation of a network [26]. SON functions can automatically tune global operational settings of the radio cells (e.g., maximum transmit power, channel bandwidth, electrical antenna tilt) as well as parameters corresponding to RRM functions (e.g., admission control threshold, handover offsets, etc.).

From an architectural perspective, SON functions can be: (i) centralized SON (cSON), meaning that the SON algorithm is executed at the network management level (e.g. at the EMS); (ii) distributed SON (dSON), meaning that the SON algorithm resides at the small cells, or; (iii) hybrid SON, meaning that part of the functions are distributed and part are centralized.

Given that RRM/SON mechanisms have a strong and direct impact on the RAN performance, the development of RRM/SON algorithms adds an important value to the vendor equipment.

The introduction of virtualization technologies enables the possibility of implementing RRM/SON functions as virtualized services [27]. Virtualizing RRM/SON functions in scenarios like those of SESAME has multiple potential benefits, such as: (i) scalability and flexibility in how RRM/SON functions are deployed within a CESC cluster, thanks to the capability of adding/removing the services as required, where and when needed and to upgrade them without hardware/firmware changes of the radio equipment; (ii) ease of integration of RRM/SON solutions from multiple vendors if common interfaces are adopted, which facilitates a more open market where third-parties could provide RRM/SON components to be flexibly plugged into the network; (iii) higher efficiency in power consumption by centralizing functions in some micro servers of the Light DC; (iv) more flexibility in accommodating variable computation requirements of the RRM/SON functions by scaling up/down the amount of resources allocated to them across the LightDC micro-servers.

5.3 Radio Network Information Service

The implementation of RRM/SON functions as VNFs requires that radio network related information available within the small cells be accessed by these functions. In order to avoid separate access to the same information from multiple deployed RRM/SON functions, a VNF to interact with the different components embedding the small cell (SC) functionality is considered here. In this way, this VNF handles the acquisition, processing and collection of different measurements to monitor the status of the RAN and the User Equipment (UE) and exposes

them to the rest of RRM/SON functions. The exposed information could contain a wide range of metrics such as measurements and statistics information related to the user plane, information related to UEs served by the cells, cell load, throughput measurements, etc., delivered at the relevant granularity (e.g. per UE, per cell, per period of time). Moreover, the radio network information could be profiled according to the specific needs of each RRM/SON function.

In general, most of the measurements to be collected for RRM/SON functions are Layer 2 measurements [31]. Therefore, within the prototyping framework in SESAME project, which assumes a functional split at the S1 interface[50] so that the SC PNF includes the whole radio protocol stack, these measurements are performed within the SC PNF. Therefore, these measurements can be obtained in SESAME from the SC PNF by means of Performance Management (PM) reports in the form of eXtensible Markup Language (XML) files including the measurements specified in [32] through a management interface such as the TR-069 [33]. The PM file upload is controlled by the parameters defined in [34], which allow specifying the destination (e.g. the address of the VNF for measurements collection) and the periodicity of the file upload. On the other hand, while the interaction between the VNF for measurement collection and the SC PNF could be conditioned by vendor specific implementation of the SC PNF management interface, the approach for this VNF to expose the radio network information is the adoption of open interfaces, in line with the on-going work in European Telecommunications Standardization Institute about the Radio Network Information Service (RNIS) defined in [35] for MEC applications. Based on this, in the following we adopt the terminology RNIS to refer to the abovementioned VNF.

In a multi-tenant scenario, the RNIS should in general provide both an aggregated view of a cell, composed of measurements aggregated for all the tenants, and an individual tenant view, composed of measurements that are split on a per-tenant basis and characterize the performance associated to the UEs of a single tenant.

5.4 Framework for virtualizing multi-tenant RRM/SON functions

Beyond the general benefits of virtualization mentioned above, the design and implementation of virtualized RRM/SON functions in multi-tenant scenarios provides additional degrees of flexibility by facilitating the capability of customizing the RAN on a per-tenant basis depending on which RRM/SON functions are instantiated for each tenant.

An RRM/SON function can be instantiated on a per-tenant basis as long as isolation mechanisms are in place with respect to the resources/parameters managed by that specific function [25]. Isolation means that the decisions taken by a tenant's RRM/SON function do not impact on the performance observed by other tenants. If isolation is not ensured, a common RRM/SON solution is necessary for all the tenants.

For example, let us consider a Coverage and Capacity Optimisation (CCO) SON function that dynamically adjusts the tilt of a cell. Since a change in the tilt will impact on all the users in the cell regardless of the tenant they belong to, this function must be common to all tenants. In this case, this function will typically be implemented in a *tenant-agnostic* way, meaning that only aggregated measurements (e.g., power measurements received from all the UEs in a cell regardless of the tenant they belong to) from the RNIS will be used as inputs.

Instead, let us consider an Admission Control (AC) function that decides to accept or reject the Radio Access Bearers (RABs) of the UEs of the different tenants. If the radio resources used by the different tenants are isolated (i.e., each tenant uses a different orthogonal set of

[50] See: http://lteguide.blogspot.gr/2011/11/s1-interface.html

time/frequency resources), it is possible to instantiate different tenant-specific AC functions, so that each tenant can apply its own admission criteria. In this case, the AC will get a *tenant-specific* view of the measurements provided by the RNIS (e.g., amount of free physical resources in relation to the total set of resources allocated to this tenant).

As an additional example, let us consider a Packet Scheduling (PS) function that manages a set of time/frequency resources that are shared among several tenants. The PS decision making logic must be a common function, although the implementation of the function will typically be *tenant-aware* (i.e., it will be based on both aggregated and per-tenant measurements), so that the assignment of physical resources to a specific UE will take into account both how many resources are available as well as how many resources have already been occupied by other UEs from the same tenant.

Through the above examples, Figure 5-1 presents the considered framework for virtualizing RRM/SON functions in multi-tenant small cells. It particularises the general SESAME architecture of [36] with a focus on the VNFs that compose the RRM/SON services. The figure depicts the tenant-specific RRM/SON VNFs in different colours for tenant T1 and T2 (i.e. red and green, respectively), while the common RRM/SON VNFs, i.e. the VNFs that support procedures that are common to all the tenants, are depicted in brown. Furthermore, Figure 5-1 illustrates that the RNIS will provide the necessary measurements to the different VNFs.

The different RRM/SON VNFs and the RNIS of Figure 5-1 are executed at the Network Function Virtualisation Infrastructure (NFVI) constituted by the Light DC. The specific RRM/SON functions that can be implemented as VNFs depends on the functional split that specifies the subset of the Small Cell (SC) functionality that is embedded in the SC PNF and the subset that is run externally as VNFs. Different functional splits are discussed in [27] following the layering architecture of the radio interface protocol stack.

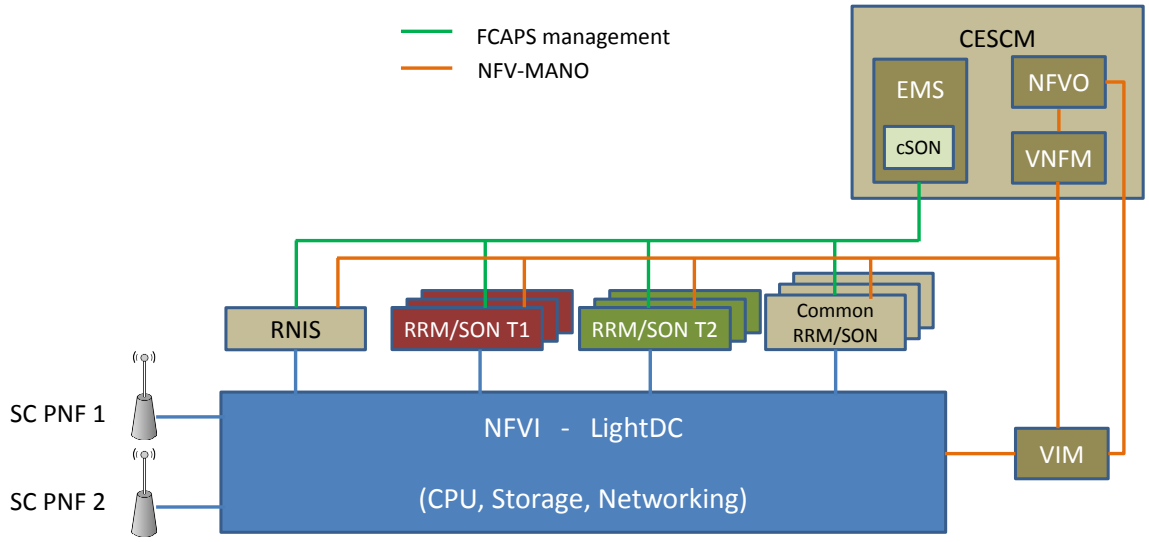


Figure 5-1 Virtualized RRM/SON functions in a multi-tenant environment

The management of the RNIS VNF and the RRM/SON VNFs depicted in Figure 5-1 involves two different frameworks, namely the Fault, Configuration, Accounting, Performance and Security (FCAPS) management and the NFV-MANO.

The FCAPS management[51] is performed by the EMS and includes a set of functions for configuration and re-configuration of the operational parameters of the RNIS VNF and the different RRM/SON functions (e.g. adjusting parameters and thresholds of the AC, handover, etc.) or for the collection of Performance Management (PM) measurements characterizing the operation of these functions (e.g. number of successfully established RABs, etc.).

The NFV-MANO involves the management functionalities provided by the NFVO, VNFM and VIM to support the lifecycle management of these VNFs (e.g. instantiation, scaling, and termination) as well as of the entire Network Services (NS) in which the VNFs are chained as components across the Light DC. Further details about the management and orchestration for the RRM/SON VNFs are presented in Deliverable D6.4 [30].

[51] For further detail also see: *https://en.wikipedia.org/wiki/FCAPS*

6 Security analysis

Security challenges of 5G networks, as investigated in [37], are not only due to virtualisation, but also due to the different security requirements identified at different network domains. Therefore, an effective security management system is required to address these challenges. The security management system should be able to provide robust authentication and authorisation to control any unauthorised access and abuse to the Northbound interface[52] and controllers. The Northbound interface is an abstraction layer to the internal modules of the Network Function Virtualisation Orchestrator (NFVO). It is the interface that allow communication to the NFVO from multiple tenants or external modules. Multiple tenants communicate to the NFVO for service request and alteration. Therefore, to guarantee confidentiality, security and integrity in the multi-tenant environment, tenant isolation should be instigated at different network domain.

The SESAME security analysis, has been carried out at two different levels, requirements level and threat level. For the former, the SecTro[53] security modelling tool was used to identify the SESAME system architecture as well as to analyse the most important security requirements. For the latter, the SESAME potential security threats were identified and analysed using the STRIDE threat modelling[54]. The identified threats were then in cooperated in to the SecTro model for in-depth threat analysis and the security mitigations. Moreover, the output of that was presented in terms of a dossier that includes all the relevant information. All that analysis has been presented in the Deliverable D5.3 and it is not repeated here. This section is instead focused on NFV and VNFs.

NFV has many advantages to network operators [41]. Some of them are reduced energy consumption and equipment costs, and increased service agility and possibility to optimize the network configuration and/or topology on the fly.

It is comprised of two main functional blocks: the Network Functions Virtualization Infrastructure (NFVI) on the left side of Figure 6-1, and the NFV Management and Orchestration (NFV MANO) on the right side of Figure 6-1.

[52] For further informative details see, *for example*: *https://en.wikipedia.org/wiki/Northbound_interface*
[53] See: *http://www.sense-brighton.eu/research/sectro-tool/*
[54] For further details see: *https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx*

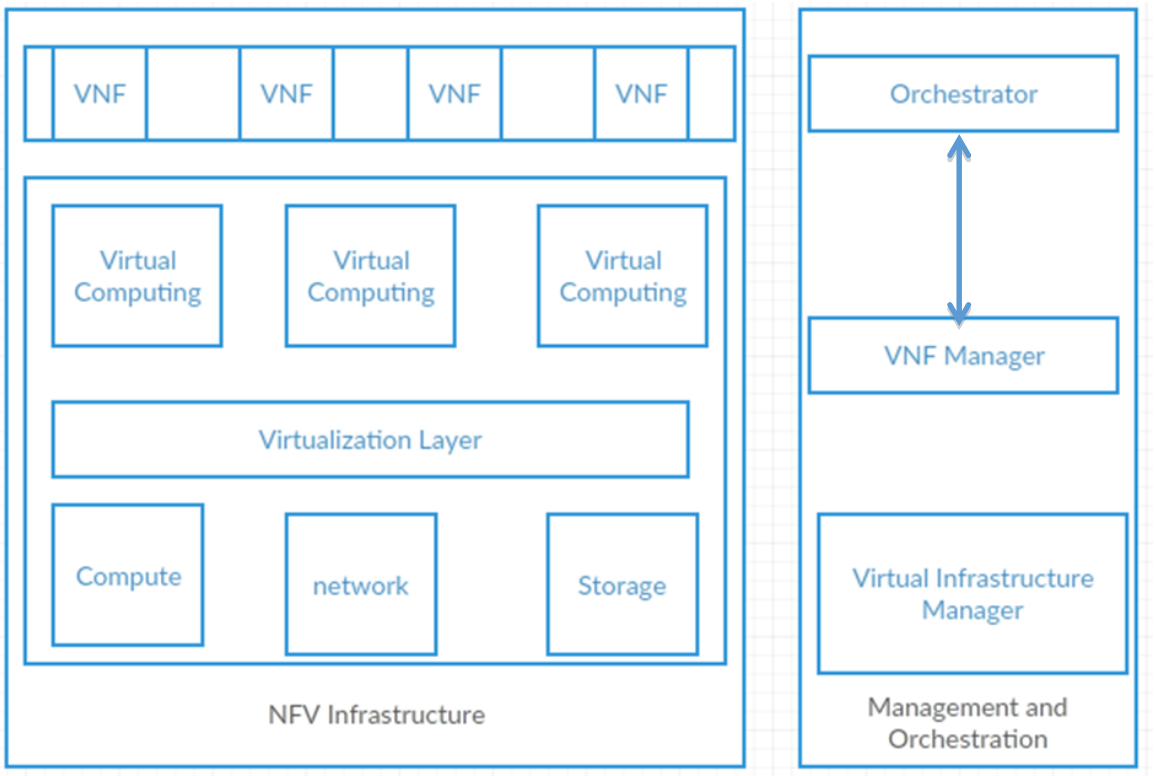


Figure 6-1: NFV MANO Management and Orchestration framework

The NFVI is the set of hardware and software components which build up the environment in which the Virtualized Network Functions (VNFs) are deployed [42], while the NFV MANO [43] is in charge of managing the lifecycle and chaining of the VNFs in order to provide the needed Network Services [44].

VNFs can be vulnerable to diverse attacks from inside, outside and between VNFs. Virtualized Network Functions could be the source of an attack being a malware developed and programmed to perform attacks or it can be the target of an attack as a software component and part of an infrastructure.

This section will look into potential security issues in VNFs. Moreover, it will investigate and introduce security challenges in SDN virtualized as VNFs, while it high points some research directions in NFV that needs more study in order to become suggestion for mitigating security incidents.

6.1 VNF modelling with respect to security issues

Security challenges become a serious concern for researchers and industry, despite the advantages of NFV. Just one incident can harm and bring the entire network down. In a virtualized environment in NFV, VNFs are an important asset of NFV architecture when introducing security issues. As VNFs can be virtualized functions that implemented on vendor's software, a third party involvement, and trust management between VNFs.

There are three main types of attack: insider attack, outsider attack, and between VNFs attack but also there are some other interesting attacks discussed below. Also, there is an investigation and discussion of security concerns on SDN virtualized as a VNF.

6.1.1 Return-oriented-programming-based attacks

Riddle and Chang [45] introduce an attack on the Xen[55] hypervisor that with the usage of return-oriented-programming permits and lets the adversary to escalate their VMs to a privileged state.

In the context of NFV, these attacks could be used to read or modify the memory of, take control of, or deny resources to VNFs co-resident with a malicious VNF meaning something that disrupts the action of several Network Services, or even deploy more malicious VNFs.

6.1.2 Insider Attack

Insider attack occurs inside VNFs framework. Adversaries exploits vulnerabilities in software in order to gain unauthorized access. Those attacks can compromise the whole NFV components. Most common types of these attacks are known exploiting known vulnerabilities from outdated software on NFVs, software (SW) crashes and incorrect software validation of VNFs.

6.1.2.1 Security issues in VNFs software

One of NFV advantages is to build open source software and deploy them on virtualized environments [46]. However, software is subject to diverse security attacks and caution is required before deploying any software into virtualized NFV environments. An adversary can exploit vulnerabilities in the deployed software, these vulnerabilities include implementation flaws and design flaws [47],[48].

Another difficulty is coping with a vendor who is considered unreliable, which may leads to weak security features and difficulty in communication [47]. One more important part of the security in deployed software is their update and upgrade procedures. Software updates will aid to fix bugs and other errors in order to keep software secured from attackers who can exploit vulnerabilities [49].

Software validation

Another important part of VNFs and their software is validation. A validation of VNF software has specific procedures starting with installation and ending with launching of a VNF, according to the ETSI [49]. Furthermore, validation highlights that the loaded code is authentic and has not been altered, infected and modified by unauthorized actions [49].

Software crashes

The case of crashes can be considered as another scenario for security issues. For example, if the software operates on VM crashes, the hypervisor requires and makes sure there is no alteration to the existing authorizations [49].

6.1.3 Outsider Attack

An outsider attack means when a third party from outside has an unauthorized access to VNFs and NFVs by a remote access, performing Denial of Service (DoS) and software attacks.

6.1.3.1 A Third-party network

VNFs infrastructure can be managed from outside access such as a third-party network [51]. A third party controls the specific VNFs through a portal [48]. Therefore, significant security issues

[55] For more details also see, *inter-alia*: https://en.wikipedia.org/wiki/Xen

are raised not only in the VNFs themselves, but also NFV generally will suffer in terms of security threats [48].

Incidents happen when a third-party is malicious. The infrastructure gets exposed by performing network attacks, such as DDOS, and software attacks such as taking advantage of vulnerabilities in the deployed virtualization software [48].

6.1.3.2 Denial of Service (DoS)

Denial of Service (DoS) attacks and incidents are a significant and important threat to NFV environments. There are several DoS incidents on providers and their services hosted in the Cloud, like Bitbucket[56], a web-*based* hosting service company hosted by Amazon that was victim of massive DDoS (Distributed DoS) attacks [52]. The danger of DDoS is even more considerable given the background of NFV because it could also affect unfocused and non-targeted services that are hosted on the same physical host. VNFs are software components providing network functions, so they are likely to be vulnerable to software programming bugs: it could be possible to bypass firewall restrictions or to take advantage of a buffer overflow to execute arbitrary code. CVE-2012-2663[57] for iptables and CVE-2006-5276[58] for Snort[59] are examples of such vulnerabilities and they give a first perception of threats that target typical NFs that are deployed in NFV infrastructures

6.1.4 Between VNFs Attacks

6.1.4.1 Trust between VNFs

Establishing and managing trust between VNFs is extremely significant to provide security confidentiality, integrity and availability [53]. It is also significant both to provide and prevent incidents between entities such as spoofing and tampering [54].

The following are different forms of trust between VNFs [55]:

- Trusting to carry out operations that have indirect influence on data.
- Trusting the information correctness output between software objects.
- Trusting between software objects to accomplish the correct operations.

The Virtual Network Function Component Instances (VNCIs) build the trust relationships between VNFs [55]. Once the VNFs are established, they require a guarantee that other entities which they might react with can be trusted in order to execute certain operations [55]. Therefore, without trusting relationships, entities are not able to validate their own operations or other entities that they interact with. Also the Service Provider (SPs) are not able to assure that the service which they provided will work as they expected [55]. However, establishing a longer chain of trust with additional software remains one of the security challenges [56].

6.1.4.2 Sharing host

Sharing underlying infrastructure among multiple hosted virtual machines is a significant security issue in VNFs. Adversaries can perform malicious actions and attacks using shared resources between VMs [52]. Another way if the attacker finds a way through one of the hosted

[56] See: *https://bitbucket.org/*
[57] For further information also see: *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2663*
[58] For further information also see: *http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2006-5276*
[59] For more information about Snort also consider: *https://www.snort.org/*

VMs like a "noisy neighbour" in order to consume a large number of resources [52]. Therefore, there is a need to isolate and protect different VMs in case if one of them has been compromised. However, providing a secure isolation among shared resources is complicated [51].

6.1.5 SDN as a VNF

There are many different positions that SDN can be located in the context of NFV framework. However, this research focused on identifying possible security issues in the case of SDN virtualized as VNFs. Security concerns can be related to their deployments and the communications and interfaces between them.

6.1.5.1 SDN application

It is responsible for network resources customization including: allocation, automation, and management of the services [59]. Unauthenticated and unauthorized application or malicious application causes failure to the network. Therefore, specified and verified application should be guaranteed and checked all the time to prevent exploit of such vulnerability.

6.1.5.2 SDN controller

This is a middle component between the applications and the network resources. It controls and relays the configuration that received from application to the network resources [56], [57]. However, compromised SDN controller might modify the VNF forwarding graph. Another security issue could be the Denial of Service attack, where the attack sends packets to consume the networking resources, threating the integrity and confidentiality and availability of the system [58]. While the controller traffic should be verified.

6.1.5.3 SDN resources

The SDN network resources are responsible for routing and packet processing. The data plane such as router and switch could be implemented as VNF. Every virtualized network resources should ensure the configurations from authentic SDN controller.

6.1.5.4 Interfaces

The SDN components: application, controller and resources communicate each other with standardized interfaces, making necessary to be secured using resistant protocols.

6.2 Resume of the security analysis

This section researched high level and theoretical attacks that may impact Network Functions Virtualization in order to have a visualization of their potential threats. There is no evidence of any attack that specifically targeted an NFV infrastructure, so there was a search for CVE Identifiers and attacks related to Cloud computing and hypervisors and analysed the impact of such attacks on VNF.

NFV have a non-negligible number of weaknesses, as a result the security of NFV MANO and hypervisors require particular attention. This section has provided a focused presentation of such weaknesses across three main types of attack: insider attack, outsider attack, and between VNFs attack. Furthermore, it discussed issues with two possible positions: SDN controller as a VNF and SDN switch or a router as a VNF. All these are important to gain an understanding of security issues for NFV.

7 Conclusion

This deliverable wraps up the Service Function Chaining work in SESAME, describing in details the deployment procedure of Netfloc in the SESAME VIM. Development advances of the Netfloc's Prometheus exporter component are described, offering further specifics on the metrics and a description of the SDN monitoring framework. The deliverable moreover describes the interfaces between Netfloc, CESCM and the SESAME orchestrator and all the details related to automatic deployment of network service. Several VNFs are described in a reference to the network service, along with the VNF placement algorithms and QoS control for SLA compliance. Knowledge on the radio part is also covered through virtualization of RRM and SON functions in multi-tenant small cell networks. Finally a security analysis is discussed with respect to different types of attacks when SDN controller is represented as a VNF.

8 References

[1] NSH header specification: https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/. Accessed on 25.09.2017.

[2] OpenStack-OpenDaylight integration: http://superuser.openstack.org/articles/open-daylight-integration-with-openstack-a-tutorial/. Accessed on 25.09.2017.

[3] Netflogi, the Netfloc's GUI: https://github.com/icclab/netflogi. Accessed on 25.09.2017.

[4] T-Nova SFC demo with Netfloc: https://blog.zhaw.ch/icclab/t-nova-final-year-demo-and-project-review-aftermath/

[5] SESAME-COHERENT demo at EuCNC 2016: https://blog.zhaw.ch/icclab/sesame-hurtle-netfloc/

[6] OpenFlow statistics plugin used by Netfloc monitoring module. Accessed on 25.09.2017: https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Statistics

[7] Prometheus monitoring system: https://prometheus.io/. Accessed on 25.09.2017.

[8] Netfloc exporter for Prometheus: https://github.com/icclab/netfloc-prometheus-exporter. Accessed on 25.09.2017.

[9] inMon sFlow Trend GUI: http://www.inmon.com/products/sFlowTrend.php. Accessed on 25.09.2017.

[10] Internet Engineering Task Force (IETF) (2014). *RFC 7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*, August 2014. https://tools.ietf.org/html/rfc7348

[11] SESAME H2020/5G-PPP Project: *Deliverable 6.1: Orchestrator Architecture Design and Interfaces Specification*. June 2016.

[12] European Telecommunications Standards Institute (ETSI) (2014). *ETSI GS NFV-MAN 001 v1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration*. December, 2014.

[13] OpenCV.org, "*OpenCV API Reference*," [Online]. Available at: http://docs.opencv.org/3.0-beta/modules/refman.html.

[14] BerkeleyVision.org, [Online]. Available at: http://caffe.berkeleyvision.org/.

[15] "FFmpeg" [Online] Available at: https://www.ffmpeg.org/, 2017.

[16] B. Blanco, J.O. Fajardo, and F. Liberal (2016). Design of Cognitive Cycles in 5G Networks. Cham. Springer International Publishing, 2016, pp.697–708. [Online]. Available at: https://doi.org/10.1007/978-3-319-44944-9 62

[17] OpenStack Live-migration. [Online]. Available at: https://docs.openstack.org/admin-guide/compute-configuring-migrations.html

[18] P. Khodashenas, C. Ruiz, J.F. Riera, J. Fajardo, I. Taboada, B. Blanco, F. Liberal, J. Lloreda, J. Perez-Romero, O. Sallent, et *al.* (2016). Service provisioning and pricing methods in a multi-tenant cloud enabled RAN. In *Proceedings of the 2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp.1-6. IEEE, Berlin, Germany, October 31, 2016 – November 02, 2016.

[19] J. Gil Herrera, J.F. Botero (2016). Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management, 13*(3), 518–532.

[20] F. Rossi, P. van Beek, and T. Walsh (2006). *Handbook of constraint programming*. Elsevier, 2006.

[21] MiniZinc. http://www.minizinc.org/, August 2017

[22] C. Schulte, G. Tack, M.Z. Lagerkvist (2017). *Modeling and Programming with Gecode.* CreativeCommons. Available at: http://www.gecode.org/doc-latest/MPG.pdf

[23] R. El Hattachi and J. Erganian (2015, February). *5G White Paper*, NGMN Alliance. Available at: https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf

[24] The Third Generation Partnership (3GPP) (2016, December). *3GPP TR 38.801 v1.0.0: Study on New Radio Access Technology; Radio Access Architecture and Interfaces (Release 14).*

[25] O. Sallent, J. Pérez-Romero, R. Ferrús, and R. Agustí (2017). On Radio Access Network Slicing from a Radio Resource Management Perspective. *IEEE Wireless Communications* (in press), April 2017.

[26] J. Ramiro, and K. Hamied (2012). *Self-Organizing Networks. Self-planning, self-optimization and self-healing for GSM, UMTS and LTE.* John Wiley & Sons.

[27] Small Cell Forum (2016, January). *SCF 159.06.02: Small Cell Virtualization: Functional Splits and Use Cases*.

[28] Radisys Corporation (2014, June). *Evaluating Cloud RAN Implementation Scenarios*, *White Paper.*

[29] E. Westerberg (2016, July). 4G/*5G RAN architecture. How a split can make the difference.* Ericsson Technology Review.

[30] P.S. Khodashenas (editor), "*Orchestrator Prototype*", Deliverable D6.4 of SESAME, September, 2017.

[31] 3GPP (2016, March). *3GPP TS 36.314 v13.1.0: "Evolved Universal Terrestrial Radio Access (E-UTRA); Layer 2 – Measurements.*

[32] 3GPP (2016, December). *3GPP TS 32.425 v14.1.0: Performance Management (PM); Performance measurements Evolved Universal Terrestrial Radio Access Network (E-UTRAN)*.

[33] Broadband Forum (BF) (2013, November). *TR-069 CPE WAN Management Protocol*.

[34] 3GPP (2016, June). *3GPP TS 32.592 v13.1.0: Information model for Type 1 interface HeNB to HeNB Management System (HeMS)*.

[35] ETSI (2016, March). *ETSI GS MEC 003 v1.1.1: Mobile Edge Computing (MEC); Framework and reference architecture.*

[36] I. Giannoulakis (editor), "*SESAME Final Architecture and PoC Assessment KPIs*", Deliverable D2.5 of SESAME, December 2016.

[37] V. Vassilakis, E. Panaousis, and H. Mouratidis (2016). Security challenges of small cell as a service in virtualized mobile edge computing environments. In *Proceedings of 2016 IFIP International Conference on Information Security Theory and Practice,* pp.70-84. Springer, Heraklion, Greece, September 26-27, 2016.

[38] H. Mouratidis, N. Argyropoulos, and S. Shei (2016). Security requirements engineering for cloud computing: The secure tropos approach. In D. Karagiannis, H.C. Mayr, and J. Mylopoulos (Eds.), *Domain-Specific Conceptual Modelling*, pp.357-380. Springer.

[39] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack (2006). Threat Modelling: Uncover Security Design Flaws Using the STRIDE Approach, pp.68-75. CMP Media Inc., San Francisco, CA, US.

[40] D. Kreutz, F. Ramos, and P. Verissimo (2013). Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp.55-60. ACM, Hong Kong, China, August 12-16, 2013.

[41] ETSI (2014, October). *ETSI-ISG-NFV: Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action*. Available at: *http://portal.etsi.org/NFV/NFV_White_Paper.pdf*.

[42] ETSI (2014, December). *ETSI-ISG-NFV: Network Functions Virtualization; Architectural Framework.* Available at: *https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/-01.01.01_60/gs_NFV002v010101p.pdf*.

[43] ETSI (2014, December). Network Function Virtualization; Management and Orchestration. Available at: *https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.-01.01_60/gs_NFV-MAN001v010101p.pdf*.

[44] ETSI (2014). *Network Function Virtualization; Terminology for Main Concepts in NFV*. Available at: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/-01.02.01_60/gs_NFV003v010201p.pdf.

[45] A.R. Riddle and S.M. Chung (2015). A Survey on the Security of Hypervisors in Cloud Computing. In *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops.* IEEE. Colombus, OH, US. June 29, 2016 – July 02, 2015.

[46] C. Price and S. Rivera (2012). *OPNFV: An Open Platform to Accelerate NFV*, *White Paper*. OPNFV.

[47] M.D. Firoozjaei, J.(P.) Jeong, H. Ko, and H. Kim (2016). Security challenges with network functions virtualization. *Future Generation Computer Systems, 67*, 315-324.

[48] ETSI (2015, December). *ETSI Group Specification: Network Functions Virtualization (NFV) NFV Security; Problem Statement*.

[49] ETSI (2014, December). *ETSI Group Specification: Network Functions Virtualization (NFV) Virtual Network Functions Architecture*.

[50] Alcatel-Lucent (2014). *Providing security in NFV - Challenges and Opportunities*. *Alcatel-Lucent White Paper. Technical Report.* Alcatel-Lucent.

[51] T.-S. Chou (2013, June). Security Threats on Cloud Computing Vulnerabilities. *International Journal of Computer Science & Information Technology (IJCSIT), 5*(3), 79-88.

[52] F. Reynaud, F.X. Aguessy, O. Bettan, M. Bouet, and V. Conan (2016). Attacks against Network Functions Virtualization and Software-Defined Networking: State-of-the-art. In *Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pp.471-476. IEEE, Seoul, South Korea, June 06-10, 2016.

[53] Huawei Technologies Co. (2014). *White Paper - Huawei Observation to NFV*.

[54] ETSI (2014, December). *ETSI Group Specification: Network Functions Virtualization (NFV) NFV Security; Security and Trust Guidance*.

[55] R. Mijumbi, J. Serrat, J.L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Survey Tutorials, 18*(1), 236-262.

[56] ITU-T (2014, June). *Recommendation ITU-T Y.3300: Framework of software-defined networking*. ITU-T, Geneva, Switzerland.

[57] Understanding the SDN architecture - definition -. (n.d.). Retrieved on November 09, 2016, from: https://www.sdxcentral.com/resources/sdn/inside-sdnarchitecture/.

[58] S. Shin, and G. Gu (2013). Attacking Software-Defined Networks: A first feasibility study. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp.165-166. ACM Press, Hong Kong, China. August 16, 2013.

[59] A. Aljuhani, and T. Alharbi (2017). Virtualized Network Functions security attacks and vulnerabilities. In *Proceedings of the 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp.1-4. IEE, Las Vegas, NV, US. January 09-11, 2017.