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 D6.2

Service Management and Orchestration functions, including VNF models

Document Number:	H2020-5GPPP-GA No.671596/*WP6*/*D6.2*/*31.10.2016*
Contractual Date of Delivery:	*31.10.2016*
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:	56
File:	*SESAME_Deliverable 6.2_v1.0_Final*

Abstract

SESAME aims 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 Virtualization Infrastructure (i.e. processing power, storage and networking), and it 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 Service Function Chaining (SFC) as an essential element on NFVO with Virtual Network Function Manager (VIM). After providing a definition of the issue, the deliverable reviews available solutions on the state of the art. Next, based on a developed solution by ZHAW (deliverable leader) it suggests a solution for SESAME.

Meanwhile, to "pave the way" for further NFVO developments, the present deliverable D6.2 reviews some initial idea regarding cognitive management of radio and cloud resources. Moreover, security at NFVO level is discussed.

The contents of this deliverable will be used as reference for the implementation of NFVO in T6.2 and T6.3.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	08/07/2016	Table of Content (ToC) - ZHAW	Irena Trajkovska - ZHAW
0.2	26/09/2016	Inputs on section 3, 4	Irena Trajkovska - ZHAW
0.3	28/09/2016	Inputs on section 2	Cristina Ruiz – i2CAT Pouria Sayyad Khodashenas - i2CAT
0.4	3/10/2016	Inputs on section 5	UPC
0.5	04/10/2016	Inputs on section 2	Hui Xiao - FLE
0.55	04/10/2016	Inputs on section 2	Jose Oscar Fajardo, Fidel Liberal EHU
0.6	07/10/2016	Inputs on section 5	UoB
0.65	11/10/2016	Inputs on section 5	ORION and NCSRD
0.7	14/10/2016	Inputs on section 1, 6	ZHAW
0.8	18/10/2016	Pre-final release	Pouria Sayyad Khodashenas - i2CAT
0.9	24/10/2016	1st review	Atos, EHU, UPC, CNET, I2CAT, ZHAW
0.9	26/10/2016	2nd review	OTE, NCSRD, SMNET
0.10	27/10/2016	Final release	ZHAW, i2CAT
1.0	31/10/2016	Submission to the Commission	OTE

Contributors

First Name	Last Name	Partner	Email
Irena	Trajkovska	ZHAW	traj@zhaw.ch
Pouria	Sayyad Khodashenas	i2CAT	pouria.khodashenas@i2cat.net
Cristina	Ruiz	i2CAT	cristina.ruiz@i2cat.net
Jordi	Ferrer Riera	i2CAT	jordi.ferrer@i2cat.net
Eduard	Escalona	i2CAT	eduard.escalona@i2cat.net
Jordi	Pérez-Romero	UPC	jorperez@tsc.upc.edu
Oriol	Sallent	UPC	sallent@tsc.upc.edu
Hui	Xiao	FLE	hui.xiao@uk.fujitsu.com
Babangida Albaba	Abubakar	UoB	B.Abubakar2@brigton.ac.uk
Haris	Mouratidis	UoB	H.Mouratidis@brighton.ac.uk
Manos	Panaousis	UoB	E.Panaousis@brighton.ac.uk
Jose Oscar	Fajardo	EHU	joseoscar.fajardo@ehu.eus
Fidel	Liberal	EHU	fidel.liberal@ehu.eus
Begoña	Blanco Jauregi	EHU	begona.blanco@ehu.es
Ioannis	Giannoulakis	NCSRD	giannoul@iit.demokritos.gr
Emmanouil	Kafetzakis	ORION	mkafetz@orioninnovations.gr
Athanassios	Dardamanis	SMNET	ADardamanis@smartnet.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
3GPP	The Third Generation Partnership Project
4G	Fourth Generation of Mobile Communications
5G	Fifth Generation of Mobile Communications
AC	Admission Control
AI	Artificial Intelligence
API	Application Programming Interface
BGP	Border Gateway Protocol
BH	Base Header
BSS	Business Support System
CAPWAP	Control And Provisioning of Wireless Access Points
CCTV	Closed-Circuit Television
CESC	Cloud-enabled Small Cell
CESCM	Cloud Enabled Small Cell Manager
CLI	Command-Line Interface
CN	Core Network
CoAP	Constrained Application Protocol
COPS	Common Open Policy Service
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CRUD	Create/Read/Update/Delete
DB	Database
DC	Data Centre
DCD	Dynamic Component Deactivation
DHCP	Dynamic Host Configuration Protocol
DL	Downlink
DNS	Domain Name System
DST, dst	Destination
DVFS	Dynamic Voltage and Frequency Scaling
E2E	End-to-End
EMS	Element Management System
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
EU	European Union
FG	Forwarding Graph
FW	Firewall
GA	Grant Agreement
GPE	Generic Protocol Extension
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
GW	Gateway
H2020	Horizon 2020
HeNB	Home eNodeB
HO	handover
HoT	Heat Orchestration Template
HSS	Home Subscriber Server
HTTP	Hyper-text Transmission Protocol
HW	Hardware
I/O	Input/Output
IaaS	Infrastructure-as-a-Service

ICT	Information and Communication Technology
ID, id	Identifier
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IT	Information Technology
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
LACP	Link Aggregation Control Protocol
LAN	Local Area Network
Light DC	Light Data Centre
LISP	Locator/Identifier Separation Protocol
LTE	Long Term Evolution
MAN	Management and Orchestration
MANO	Management and Orchestration
MD	Metadata
MEC	Mobile Edge Computing
ML	Modular Layer
MLB	Mobility Load Balancing
MME	Mobility Management Entity
MNO	Mobile Network Operator
MRO	Mobility Robustness Optimisation
NAT	Network Address Translator
NE	Network Edge
NETCONF, Netconf	Network Configuration Protocol
NF	Network Function
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NFVO	NFV Orchestrator
NMS	Network Management System
NS	Network Service
NSD	NS Descriptor
NSH	Network Service Header
NVGRE	Network Virtualization using Generic Routing Encapsulation
ODL	OpenDayLight
OF	Open Flow
OPNFV	Open Platform for NFV
OS	Operating System
OSGi	Open Services Gateway initiative
OSS	Operations Support System
OSSM	Orchestrator Security Service Manager
OVS	Open virtual Switch
OVSDB	Open virtual Switch Database
PCEP	Path Computation Element Communication Protocol
PCMM	Packet Cable MultiMedia
PM	Performance Management
PNF	Physical Network Function
PoP	Point of Presence
PPP	Public-Private Partnership
PSC	Physical Small Cell
QoE	Quality of Experience
QoS	Quality of Service
RAB	Radio Access Bearer

RAM	Random Access Memory
RAN	Radio Access Network
REA	Research Executive Agency
REST	Representational State Transfer
RFC	Request for Comments
RIA	Research and Innovation Action
RNIS	Radio Network Information Service
RPC	Remote Procedure Call
RRM	Radio Resource Management
SC	Small Cell
SCF	Small Cell Forum
SCNO	Small Cell Network Operator
SDK	Software Development Kit
SDN	Software-defined Networking
secGW	Security Gateway
SFC	Service Function Chaining
SFF	Service Function Forwarder
SLA	Service Level Agreement
SME	Small- and Medium-sized Enterprise
SNBI	Secure Network Bootstrapping Infrastructure
SNMP	Simple Network Management Protocol
SO	Security Orchestrator
SON	Self-Optimizing Network
SOTA	State-of-the-Art
SPH	Service Path Header
SQL	Structured Query Language
SRC, src	Source
SSC	Security Service CatLog
SSO	SESAME Security Orchestrator
SW	Software
SXP	Source-Group Tag eXchange Protocol
TCP	Transmission Control Protocol
ToC	Table of Contents
TOSCA	Topology and Orchestration Specification for Cloud Applications
TTL	Time to Live
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
URL	Uniform Resource Locator
USC	Unified Secure Channel
VA	Video-Analytics
vDPI	virtual Deep Packet Inspection
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VM	Virtual Machine
VMM	Virtual Machine Monitor
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	VNF Manager

VSCNO	Virtual Small Cell Network Operator
VXLAN	Virtual Extensible LAN
WAN	Wide Area Network
WG	Working Group
WOC	WAN Optimization Controller
WSGI	Web Server Gateway Interface
WP	Work Package
XML	Extensive Markup Language

Table of Contents

List of Figures

List of Tables

---

1 Introduction

In the hybrid Cloud-Telco era, there exist several VNFs deployments of the common LTE functional blocks, e.g. EPC, BSS, HSS, RAN, etc. To enable flow of data between the conventional telco services to the new Cloud-*enabled* infrastructures, Service Function Chaining (SFC) techniques are applied. Software Defined Networking (SDN) mechanisms for Service Function Chaining are the most prominent candidates to enforce traffic steering through a logical network graph and to achieve certain service functionality among the virtualized components. Extending the concept of SFC in the context of the 5G ecosystem requires deeper understanding on the NFV concepts in such a scenario. Carefully identifying the requirements of the specific setup is important in order to choose the most suitable mechanisms and protocols to establish the desired functionality.

The advantage of using SDN in SESAME is that, based on the Open Flow (OF) protocol[1], the routing can be steered over a specific networking path by programmatically applying OF-*based* rules (flows) on the SDN controller or the open virtual switch (OVS)[2] inside the VM hosting the VNF. From a single data center point of view, the SDN controller is placed in a layer between orchestration layer and the VIM. The steering and rule enforcement policy are kept within the controller application logic and enforced over the network that hosts the light DC specific NFVs. If the routing happens across different light DCs, then the SFC approach may alter, depending on the placement of the controller within the given architecture. This has to be in accordance with the networking protocols to be adopted in that case, for example MPLS[3] and BGP[4] as used today for intra data-center routing. Alternatively, a combination of SDN controllers can be employed, one to manage the traffic rules within the cloud, and other for the radio-specific topology. This point is under investigation in SESAME and, hopefully, in the future documents the result of this study will be presented.

The focus of this deliverable is on the VNF models and the services that can be created by composing VNFs in an integral networking service. In the NFV domain, a deployment template is used to describe the network services, by describing the VNFs that it is composed of, and the relations between them. The network service template includes both, that is the resource characteristics of the VNFs in the service, as well as the service specification details. The second includes the connection points of each VNF in the service and the links between them, thus enabling an image of the entire network abstraction model. The VNF models are consumed by the NFVO in order to deploy the specific services in an automated manner. Parameters such as: VNF Descriptor (VNFD), VNF Forwarding Graph Descriptor (VNFFGD), Virtual Link Descriptor (VLD), Physical Network Function (PNF), etc. are just some of the many VNF modeling elements described in the current specifications by working groups such as ETSI[5] and TOSCA[6]. SESAME already started examining different options in D5.1 [1]. An initial live copy of SESAME descriptors has been shared with SESAME consortium, aiming to "form" a SESAME specific descriptors based on ETSI and TOSCA recommendations.

---

1 More information can be found, *for example*, at: https://en.wikipedia.org/wiki/OpenFlow
2 More information can be found, *for example*, at: https://en.wikipedia.org/wiki/Open_vSwitch
3 For more related information see, *for example:* https://en.wikipedia.org/wiki/Multiprotocol_Label_Switching
4 For more related information see, *for example:* https://en.wikipedia.org/wiki/Border_Gateway_Protocol
5 For more details see the general ETSI's website: http://www.etsi.org/
6 More details can be found, *inter-alia*, at: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

As defined by ETSI, NFVO operates on the abstracted level and over the network service descriptions and VNF models. Based on customer specifications, NFVO generates a service deployment template which will be handed to VIM. A detailed interaction between modules will be presented in section 2, but it is worth to mention that SFC is an essential part of the service deployment template. The workflow for this service deployment is as follows: On a client request, the orchestrator "passes" the network service template to a template parser (for example, HoT[7] for OpenStack Heat, [8]TOSCA, etc.). The creation of resources (such as: VMs, networks, ports, connection points, links and finally service chain deployment) is initiated. For the service chain deployment, the orchestrator (TENOR) triggers the APIs of the SDN component (Netfloc) and creates the service chain. For Netfloc this means a creation of network forwarding graph and chaining rules among the VNFs and steering the traffic in the specified chain order. Netfloc as an SDN-*based* SDK will be further described in details, along with the technical details of the chaining mechanism it supports.

This deliverable brings out the specific details of the relations between the SESAME's NFVO and the VNF models it supports. It moreover presents the specific chaining approach in SESAME's SDN component, Netfloc, including an example network chaining service. Brief analysis of supplementary concepts to be considered are finally presented and discussed.

1.1 Deliverable Outline

The present deliverable covers the service management and orchestration functions, in particular - Service Function Chaining and an example SFC composed of several VNFs. It further analyses the current chaining models and mechanisms, by focusing finally upon the SESAME-specific SDN controller and its mechanism to support SFC in a cloud eco-system. The following sections are included:

- Section 1 offers a brief introductory overview.
- Section 2 defines the service function chain in the context of SESAME, describing the role of the SDN controller in a mix cloud-telco environment. It also provides a use case example for a network chain service.
- Section 3 gives a through overview of the state of the art in Service Function Chaining, describing the current mechanisms and solutions among the open source community and in the industry.
- Section 4 focuses on the implementation guidelines of the Service Chain mechanism. It describes the technologies used to provide this service and Netfloc as a main component to offer the chaining functionality and support in the cloud domain.

[7] For more details see, *for example:* http://docs.openstack.org/developer/heat/template_guide/hot_guide.html

[8] "Heat" is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. For more details see, *for example:* https://wiki.openstack.org/wiki/Heat

- Section 5 describes some extra features with respect to cognitive management of radio and cloud resources to consider from an orchestrator point of view. It also discusses some security aspects of a SESAME orchestrator.
- Section 6 summarizes the main conclusions of the deliverable.

2 Definition of Service Chaining Mechanism

To better understand the SFC mechanism in the context of SESAME, let us first take a step back and understand the SFC meaning, in general. To do so, let us focus on the service instantiation process from the VIM perspective (i.e. OpenStack, as selected for SESAME). To deploy a service with OpenStack, one needs to take two main steps: 1) create a network, i.e. a specific and isolated VLAN (assuming that all required plugins are available on OpenStack) for the network service (NS), and; 2) launch instances, i.e. virtual machines (VM) with appropriate images on them, i.e. VNF.

Let us take a closer look at each:
In the first step, some items like the network name, access permission, subnet name, network address (range of available IPs), version of IP protocol (IP4/IP6), subnet Gateway IP, Enable/disable DHCP server, etc. are determined.
In the second step, instances are created by specifying: the name of the instance, its availability zone, flavour, its boot source (image to run on it – which turns a VM into VNF), instance count, security and access to it, network, etc.

Let us name these steps the network service creation. The role on NFVO in this phase is to make the operation automatic (consider the fact that SESAME works over a single Point of Present – PoP). It means that instead of doing all these steps one by one and manually – as one does in OpenStack – an orchestrator sits between the end user (human being - in our case VSCNO) and the VIM and makes the process user friendly and automated. That is, it generates a complete network service descriptor (NSD) from the VSCNO high-level inputs, e.g. defined by using a GUI, and then translates the NSD to a complete and understandable template for the OpenStack (i.e. heat template) with all required information mentioned above (e.g. range of available IPs, etc.). This is useful, especially for wide deployments where many services need to be instantiated and where manually keeping track of all records is difficult. To guarantee the isolation between instantiated NS belonging to different tenants, one way is to dedicate a separate VLAN to each NS and then pass the access permission to a tenant, i.e. VSCNO.
Figure 1 shows the situation. In this case, if a VSCNO requests two NSs, SESAME NFVO will create two separate VLANs each for one NS and pass the access permission to VSCNO. The created network is a logical VLAN (in other words a range of IPs or ports) which can be hosted over a single vSwitch instance.

It is also worth noting that in SESAME, *depending on the use case and the available resources on Light DC,* the mentioned isolation method might not be resource-efficient. To improve the situation, considering the fact that the SC VNF is a common point for all NS of a VSCNO, some alternative solutions have to be taken, e.g. having a single VLAN for all NS of a VSCNO.

In this case, one single instance of SC VNF can be used to fulfill the requirements of the VSCNO. However, a solution like this needs deeper study on isolation and performance.
For the moment, we leave it as a possibility, and on the remaining months of the SESAME effort we will investigate it, in more detail.

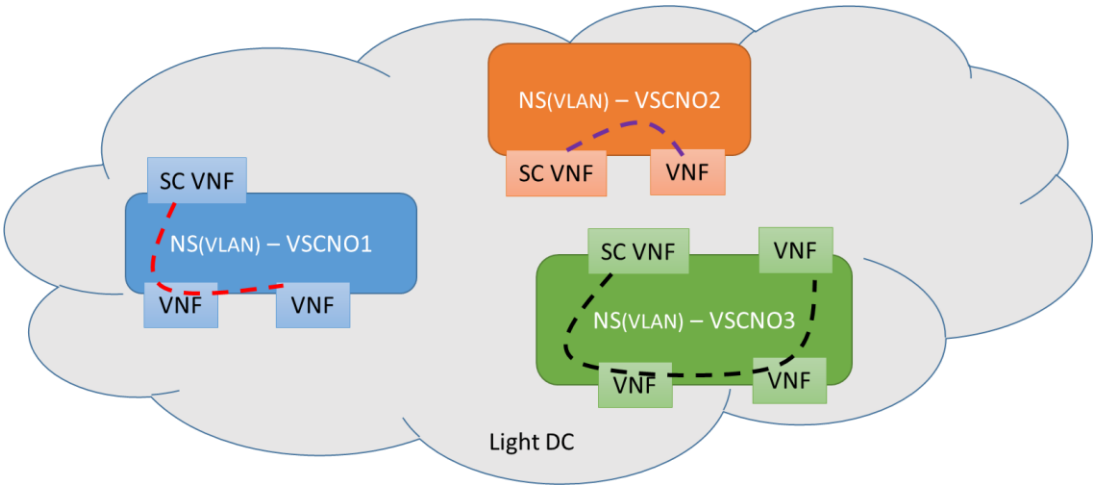

Figure 1: Logical view of three network services of three VSCNOs running over Light DC

In addition to what has been explained above as the "network service creation", there is another important step required to change the instantiated network into a complete network function chain. This important step is the network configuration – i.e. SFC. In this process, which happens in practice at the same time as the network service instantiation (though via a different logical path), NFVO configures the network, i.e. dictates the data flow between VNFs and identifies how data packet will travel from one VNF to another.

To do so, based on the received NSD (which contains VNFFGD) the NFVO generates a template (heat template) which will be communicated to a SDN controller (inside or on the same level as VIM). It helps the SDN controller to determine the destination of data packet at run time. Such interaction happens by implementing a set of rules on the forwarding node (i.e., the vSwitch in our case).

In SESAME, the NFVO with the help of Netfloc a Software Development Kit (SDK) based on OpenDayLight (ODL) – *developed by ZHAW, details are available in later sections 4.2.1* – configures the VLAN network in an automated manner. It means that the VSCNO does not need to be worried about the details of such configuration, because the NFVO will take care of it, based on the extracted data from the NSD.

2.1 SDN Controller over the Mixed Cloud-Radio Environment

Generally speaking, SDN is used to "add" flexibility and programmability to the network forwarding functions, decoupling the control plane and the data plane. In this case, a central controller configures a series of data plane nodes. End-to-end (E2E) forwarding decisions are carried out on the central unit with knowledge of the overall situation, including the status and capabilities of the forwarding nodes and interconnecting links, flow requirements, etc. Once the forwarding decision is made, the forwarding rules are implemented in the data plane forwarding nodes, which perform fast-forwarding actions based on simple rules.

In a general sense, this is like extracting the complexity of distributed routing algorithms from the data plane elements. Now, these routing algorithms are executed in the central controller in a more efficient way, while the forwarding elements can be more general-purpose software/hardware elements that optimise the fast-forwarding operations.

Having said that, it is possible to see that in a mixed cloud-radio environment like SESAME, there are three options to employ SDN controller. Option 1 is to use it only on the cloud domain to enhance data forwarding. This option has been extensively discussed, above. Alternatively, the other option is to "employ" the SDN controller for the radio domain. In this case, control plane functionality includes radio connectivity and mobility management operations. As a result of these procedures, the network configures a data path (tunnel from the eNB to the traffic aggregator in the EPC) for the user plane of each UE following a hierarchical structure. During the mobile service lifetime, the UE and the network interchange information that can lead to modifications in the user plane (moving the user packets to a new tunnel allocated in a different eNB). Traditionally, these operations and forwarding decisions are performed via eNB[9] – EPC communication. However, it is possible to handle it with the help of SDN principles. In order to do the adaptation, the mobile network has to realize a shift in mind set. That is, the control plane information would arrive at a central node (similarly to the role of the MME) and this node would make user plane forwarding decisions "on the fly" with the overall information of the forwarding nodes and the service requirements. In this way, the central controller would be able to implement forwarding rules (they could be similar to the current tunneling approach) to configure the specific data paths within the mobile network. On top of these two options, the third option is the one which merges both mentioned SDN roles in a single unit. Of course, this is a very complicate activity which could be the subject of some ongoing EU-funded research project(s).

Among the three mentioned possibilities above, the specific characteristics of the data plane forwarding considered in SESAME determine which case is "more appropriate" in the context of this project. In summary:

1. In SESAME, the control plane and the data plane of the radio protocol stack are not decoupled and the data path from the UE to the EPC through the eNB is created by the MME.
2. SESAME introduces service VNFs in the RAN though the Light DC. This means that the NFVO creates the network slice for a tenant VSCNO (including both SC VNF and Service VNFs), which includes the specific SFCs associated to the required edge services.

[9] More information about eNodeB -or ENB- can be found, *for example*, at: https://www.tutorialspoint.com/lte/lte_network_architecture.htm

3. The edge cloud is made up of a distributed collection of micro-servers associated to each Small Cell. Therefore, both the VNF placement into specific VMs and the chaining mechanisms for linking those VNFs need to be carefully analysed.

Based on the mentioned specifications, SESAME focuses upon using SDN controller to handle data forwarding over the cloud – i.e. Light DC. Having said that, the radio related activities are handled in a similar way, as of today.

Figure 2 shows a physical – virtual view of the ecosystem which hosts three different VSCNOs over the shared Network Function Virtualized Infrastructure (NFVI).

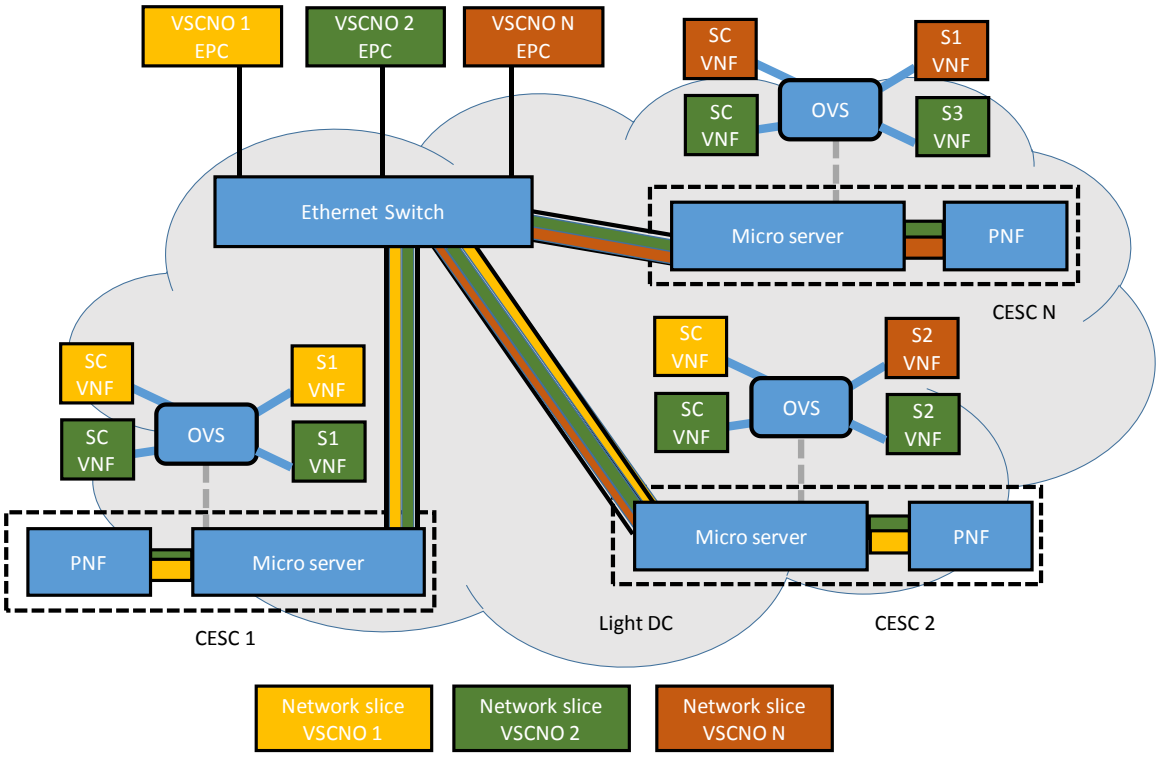

Figure 2: Example network slices for different VSCNOs in the context of SESAME

As stated previously, to form a multi-tenant scenario as presented in Figure 2, NFVO needs to process the functional chaining of VNF requested by VSCNO and to trigger its instantiation to the VIM that manages the NFVI.

Figure 3 represents this case.

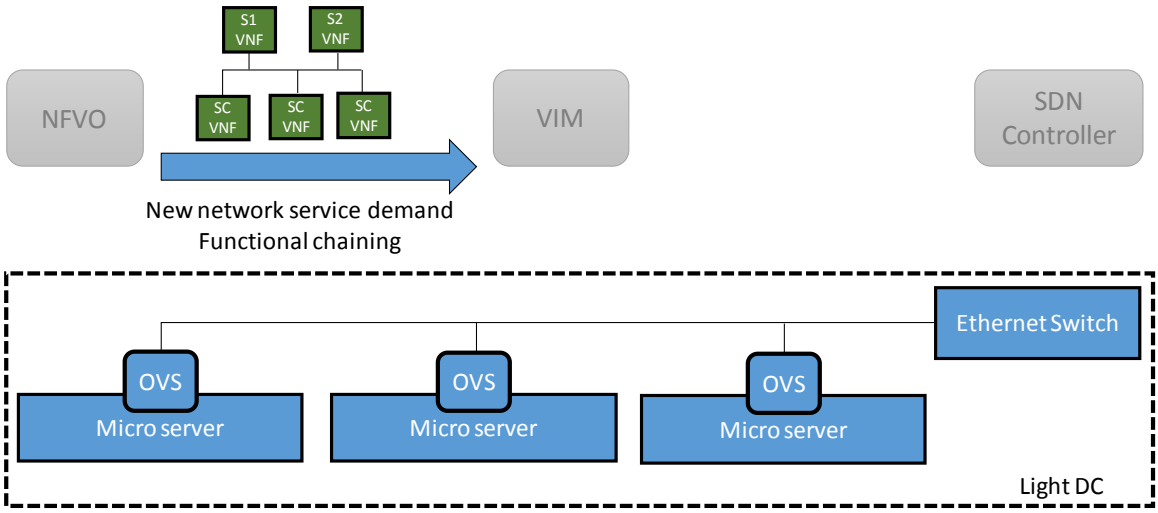

Figure 3: NFVO requests allocation of functionally chained VNFs

Upon the reception of the request, VIM will map the logical request to the actual hardware by instantiation of VNFs, which may run in different micro-servers. The distributed nature of the edge cloud introduces a novel challenge on the hardware resource allocation, which is the subject of another SESAME on-going activity in WP5.

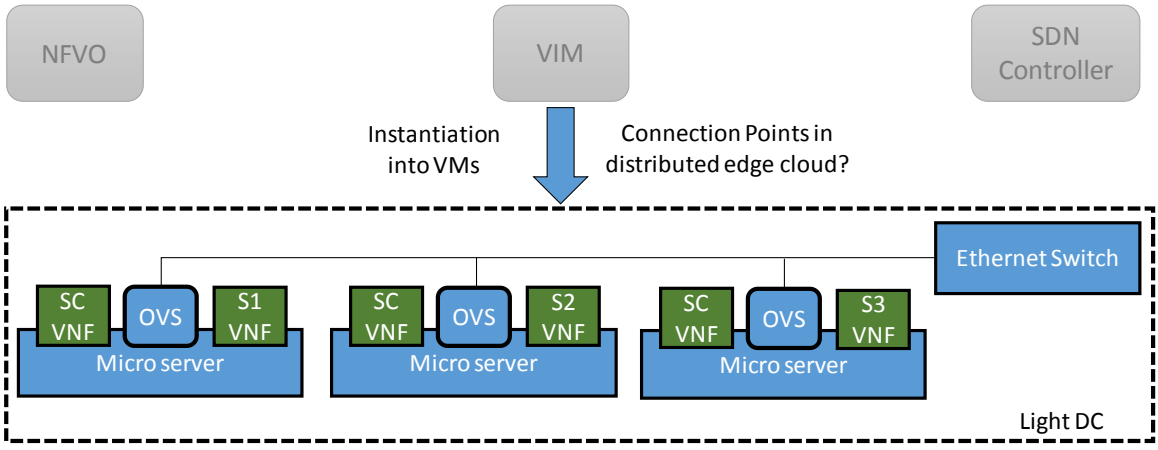

Figure 4: VIM instantiates VNFs

After this step, the created network slice for the VSCNO can be observed as a separate underlying virtual LAN instantiated and "ready to use". However, still different pairs of VNFs need to be connected. In fact, they are the origin and destination of data flows. The SDN controller implements the forwarding rules necessary to "move" data packets according to VNFFG. In this way, the data packets will travel correctly from the first to the last VNF on the NS.

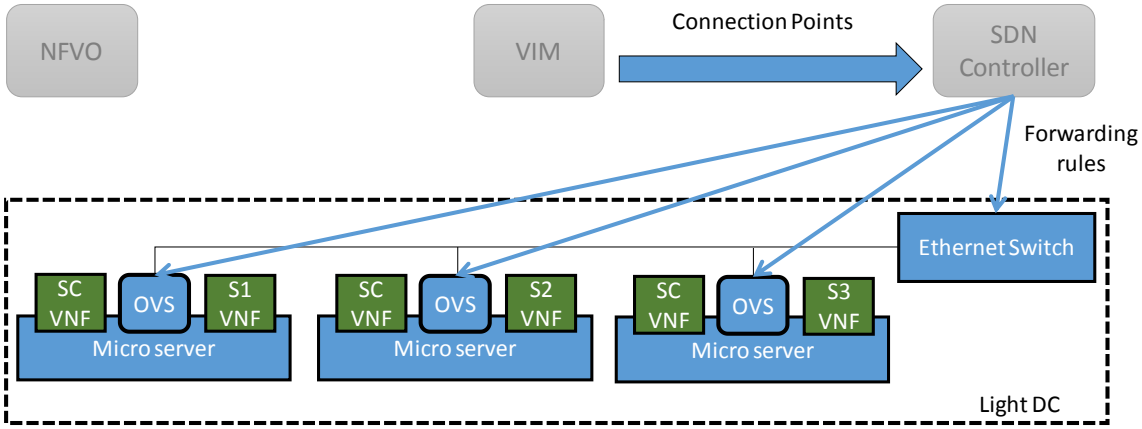

Figure 5: SDN Controller implements the forwarding rules into the forwarding nodes

2.2 Exemplary Use Case: Low Latency Video Analytics

The light DC provides a virtual environment for running a collection of service VNFs at the mobile network edge (NE). Utilizing the edge resources, these service VNFs mainly aim at providing services that can bring contents and services close to the end users, process and filter large amounts of data before they are transferred, perform large-scale analysis of real-time data or allow low latency applications such as real-time video analytics (VA). Therefore, one important type of edge-*based* services are the services that require low end-to-end latencies which are often linked to the usage of Internet of Things (IoT) devices; for example, tactile internet [2], augmented reality and remote device control. Here we suggest using SFC to support some low latency VA-*based* use cases.

In the example use cases, the real-time video data streamed from video cameras (such as fixed CCTV cameras, or mobile cameras mounted on smartphones, drones or robots) is used by the edge VA VNF as the input data, and VA is applied to the video inputs in real time to carry out meaningful analyses and output the derived analysis results or processed video data to the relevant IoT devices with millisecond level service latency.

Based on what the video data is used for and where the destinations of the output results are, the example use cases are broadly grouped to two categories:
In the first category, the VA VNF supplements the real-time video data with additional information and sends the processed video data to the IoT device that requests the particular service with unnoticeable latency between when the real world video data is generated and when the processed video data is received by the IoT device requiring the service, i.e. augmented reality.
In the second category, the VA VNF performs video content analysis on the real-time video streams to identify the occurrence of a predefined scenario, event or object. Upon the identification of such predefined cases, the VA VNF makes a decision on what actions need to be taken by some selected IoT devices to react to a particular real-time situation. Based on the decision, real-time instructions are sent to the selected IoT devices to configure them in a timely manner for performing the requested tasks. This type of use cases can be considered as the VA-*based* IoT device control services. To ensure the timeliness and validity of the decisions made and the corresponding actions taken, the response time of the selected IoT devices to the real-time situation needs to be controlled to be within millisecond level [2].

For the first category of use cases, the uplink (UL) and downlink (DL) VNF chaining paths are illustrated in Figure 6. In Figure 6, the video data streamed from an IP camera is sent to a specific VSCNO's virtual SC VNF, and then the video data is forwarded to the edge VA VNF to perform real-time video data processing. A UE that requires the augmented reality service will initiate a service request to the VA VNF and, upon reception of the service request from the UE, the VA VNF starts sending the real-time video data with supplementary information to the UE, with unnoticeable latency between the real world situation and when the data is received by the UE.

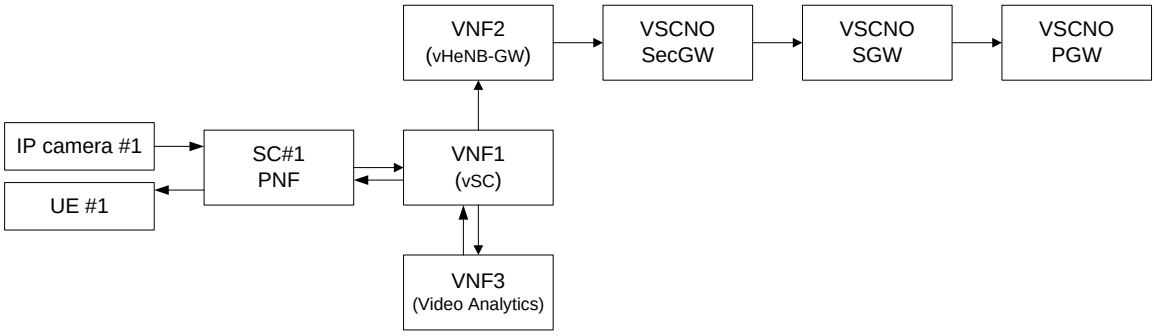

Figure 6: Uplink and downlink VNF chaining paths for category 1 use cases

For the second category of use cases, Figure 7 shows the uplink and downlink VNF chaining paths for the case where the concerned IoT devices are subscribers of the service provided by the VA VNF and already have service level connections with the VA VNF to wait for the VA VNF to provide instructions on what actions to take; whereas Figure 8 shows the uplink and downlink VNF chaining paths for the case where the concerned IoT devices are in RRC_IDLE mode and need to be woken up by the core network to receive the instructions from the VA VNF.

In Figure 7 and 8, a VA VNF carries out video content analysis in real-time by using the video data from an IP camera and, upon the identification of a predefined phenomenon, the VA VNF sends derived instructions based on the real-time analysis results to some selected IoT device(s) so as to respond to the real-time situation in a timely manner.

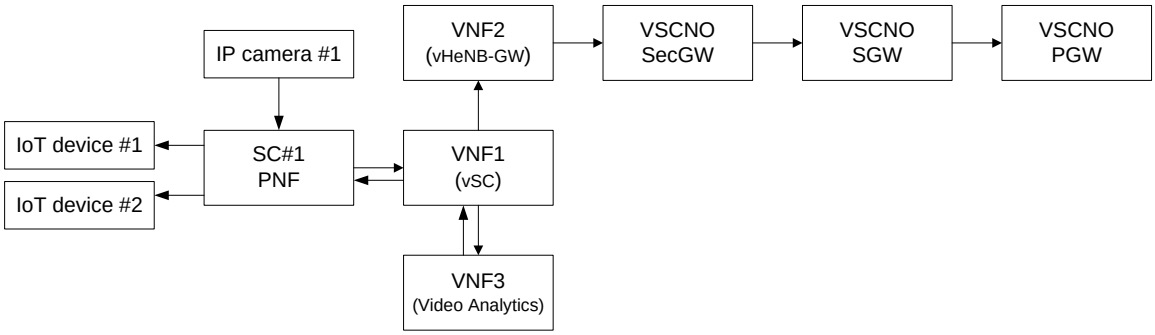

Figure 7: Uplink and downlink VNF chaining paths for the first case of category 2 use cases

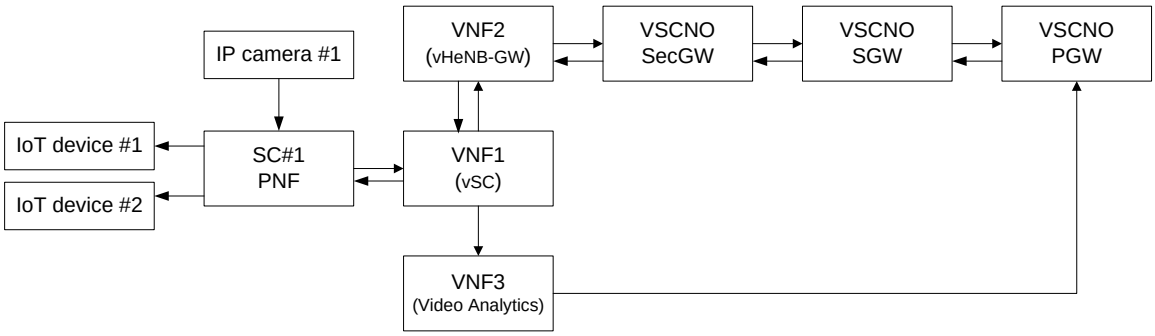

Figure 8: Uplink and downlink VNF chaining paths for the second case of category 2 use cases

The work described in this section is closely related to Section 5 and Section 6 of the Deliverable 6.1, as there is a strong correlation in the two tasks. While in the deliverable 6.1 [3] a more general focus is provided for the SFC within the scope of SESAME, we offer the details in this deliverable from an SDN controller point of view.

3 SFC in the State-of-the-Art

There is significant attention lately on the research for mechanisms to provide traffic steering by using SDN. At current stage, most of the implementations, especially the ones coming from the open source communities, are in a premature phase of development and testing. The approach that will be adopted in SESAME has been following the state of the art developments and it is aligned to the updates on a daily base.

However, the method in SESAME has a specific approach based on the Netfloc SDK, that introduces some advantages in terms of robustness and simplicity of chaining method when compared to the described approaches.

This section describes the leading research works and industry implementation related to Service Function Chaining.

3.1 Juniper's SFC Approach in OpenContrail

Juniper's service chaining solution is based on the OpenContrail controller [4] that has started developing since year 2013. This version came as an SDN alternative to the commercial Contrail solution[10], but does not include a support for OpenFlow. It is based on the Juniper customized and maintained OpenStack distribution, that exposes REST[11] APIs in order to interact with orchestrators on the northbound[12].
The controller components are shown in Figure 9. It has been developed in several programming languages, that is: Python, C++ and JavaScript.

The vRouter module [5] of OpenContrail is an alternative for the OpenvSwitch[13] that also provides routing capabilities. It is a component in charge of data forwarding and has been built as an "alternative" to the Linux bridge in the kernel.
vRouter uses OpenStack standard based solution that allows to talk to MPLS enabled routers (MPLS over GRE/UDP[14] and VxLAN[15]), but it requires a lot of configuration on the underlying network. On the control plane, it uses BGP and Netconf[16].

[10] See, *for example:* http://www.juniper.net/uk/en/products-services/sdn/contrail/contrail-networking/
[11] More information can be found at: https://en.wikipedia.org/wiki/Representational_state_transfer
[12] See, *for example:* https://en.wikipedia.org/wiki/Northbound_interface
[13] See: http://www.openvswitch.org/
[14] See the context of RFC 4023 in: https://tools.ietf.org/pdf/rfc4023.pdf
[15] For more information, see: https://tools.ietf.org/pdf/rfc4023.pdf
[16] See: https://en.wikipedia.org/wiki/NETCONF

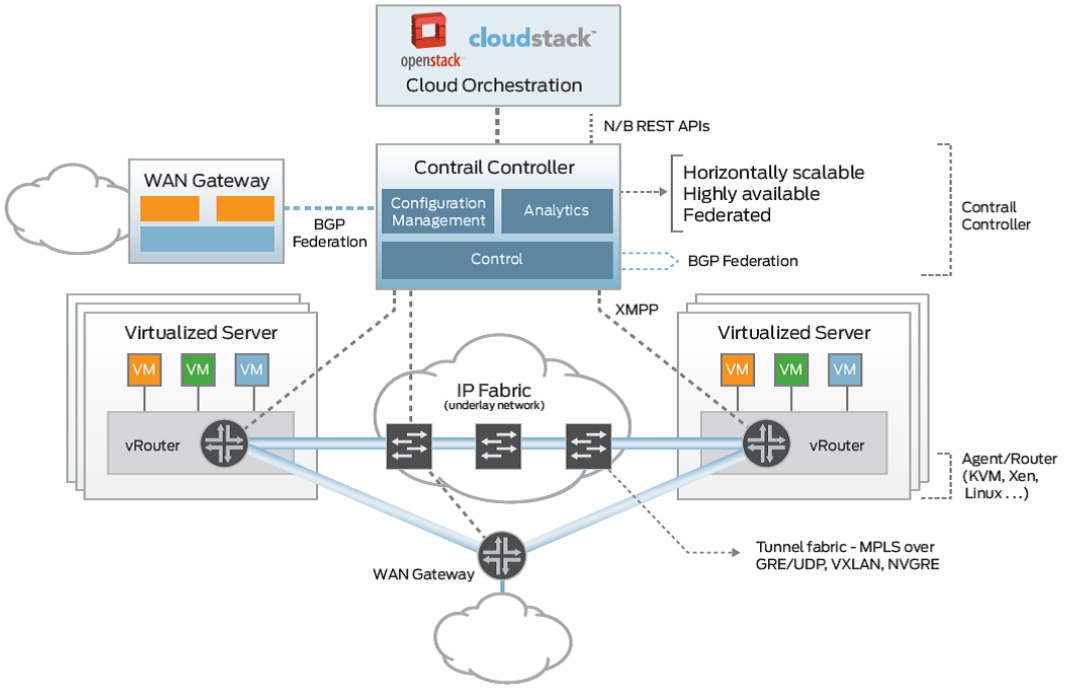

Figure 9: OpenContrail architecture

Source: http://twimgs.com/networkcomputing/news/2013/09/JuniperContrail.pngj

The forwarding policies [6] for the chain are performed via tunnel encapsulation/decapsulation and they are enforced with the flow entries in the vRouter forwarding plane. The flow table contains policies for load balancing, firewall, statistics, etc. Similarly, as of the OpenFlow protocol, there are match criteria in form of N-tuple match on the packets received, and respective actions to be applied, such as: dropping the packet, allowing the packet, or redirecting it to another routing instance.

The policy language in Contrail is similar to the Snort rules language [7]. The traffic steering is done by injecting flows (rules) into the VMs through their virtual interfaces. An example policy rule that establishes a chain from service svc-1 (VNF1) to service svc-2 (VNF2) and allows all traffic from virtual network src-vn to flow to virtual network dst-vn, looks similar to the following:

allow any src-vn -> dst-vn svc-1, svc-2

The configuration of the service chain [8] consists of the following steps: configure virtual network for the chain; specify the chaining template within the same domain; instantiate the network service, and; define a service policy. The chain mechanism consists of establishing a tunnel across the underlay network that spans through the services involved in the chain.
The routing and forwarding is based on a policy that includes the following parameters to be specified:

- Policy name
- Source network name
- Destination network name
- Other policy match conditions, for example direction and source and destination ports
- Policy configured in “routed/in-network” or “bridged/” mode

- An action type called apply_service is used:
- Example: 'apply_service': [DomainName:ProjectName:ServiceInstanceName]

To understand the traffic steering in OpenContrail, (as in Figure 10), data model objects such as "routing instances" and "route targets" are introduced. A virtual network is divided into one or more routing instances, and it is implemented as such in the vRouters. The route targets are used to control routing leaking between routing instances, in the following ways:

- The route targets for routes are manipulated to influence importing and exporting of routing from one routing instance to another routing instance.
- The next hops, labels, or both of the routes are manipulated as they are leaked from routing instance to routing instance to force the traffic through the right sequence of routing instances and the right sequence of corresponding virtual machines.

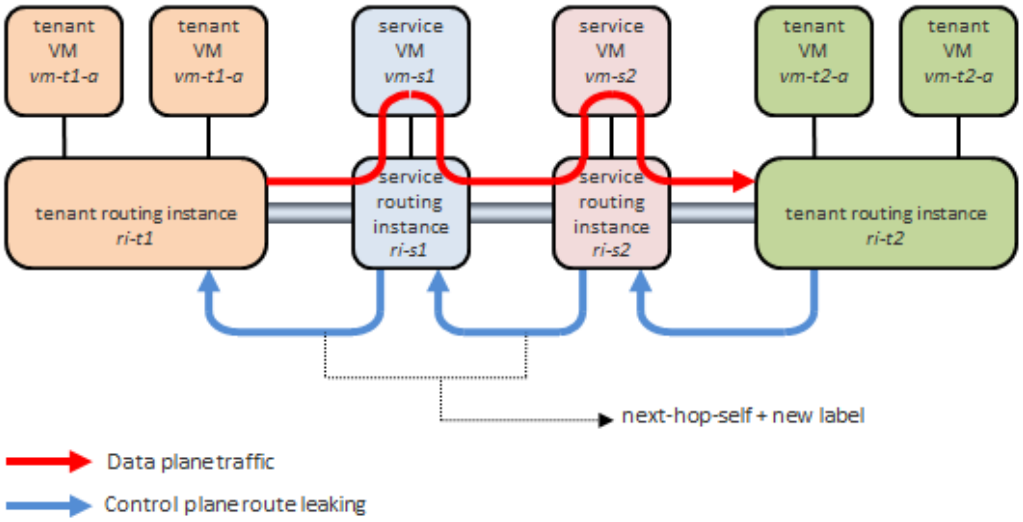

Figure 10: OpenContrail architecture

Figure 10 shows the principle of the chaining explained before:

- The import and export route targets of the routing instances are chosen in such a way that the routes are leaked from routing instance ri-t2 to ri-s2, and then to ri-s1, and then to ri-t1.
- When the service routing instances export the routes, they do a next-hop-self and allocate a new label:
  - The next-hop-self steers the traffic to the server on which the service is hosted.
  - The label steers the traffic to the service virtual machine on that server.

In the newest version of OpenContrail (v3.0)[17], the concept of port tuples has been introduced, which represents groups of virtual machine interfaces belonging to a same machine – this could be a VM, container or a physical appliance.

[17] More details about OpenContrail (v3.0) can be found *for example,* at: http://www.juniper.net/techpubs/en_US/contrail3.0/information-products/topic-collections/release-notes/contrail-release-notes-3.0.pdf

3.2 OpenStack SFC

The SFC approach in OpenStack has been introduced in the Liberty release[18] and it is currently under development [9]. This mechanism is bound to the SFC support in Neutron[19], which is currently done via the Service VMs (VNFs).
This implementation comes from the notion of Neutron ports[20] as a sequence of connection points among the VNFs VMs that are involved in the chain. From here, a service function path is represented via:

- Ordered set of neutron port pairs that define the sequence of the functions, ex: [{'p1': 'p2'}, {'p3': 'p4'}, {'p5': 'p6'}]
- Set of classifiers for the traffic flows of the chain

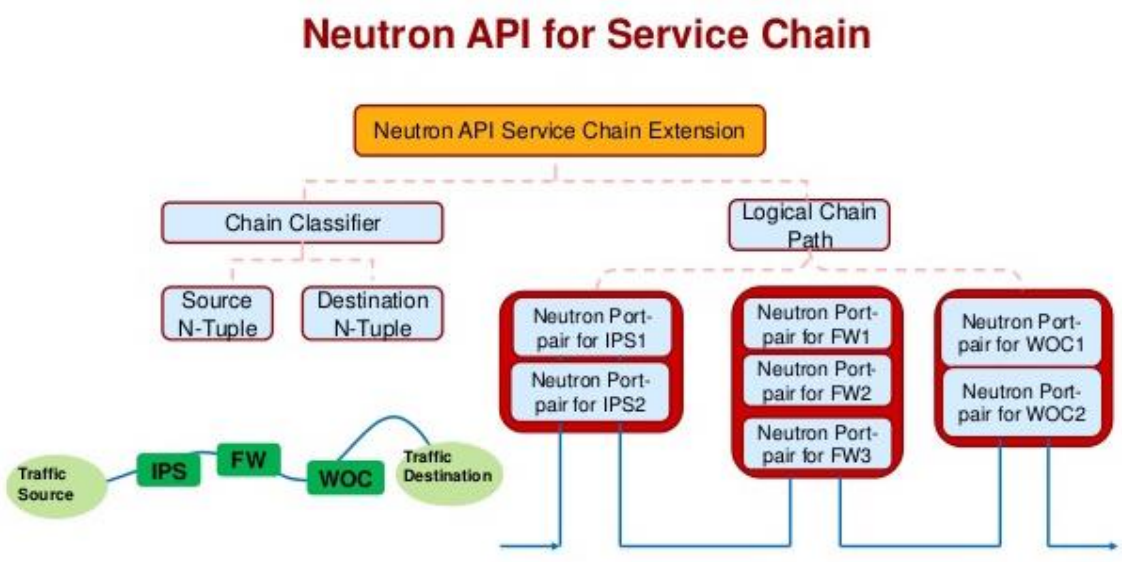

Figure 11: Example showing the Neutron API extension for service chain based on port pairs

Figure 11 shows the service chain objects introduced in the Neutron API[21]. It presents the Port Pairs groups for different IPSs and FWs. The Chain Classifier is a n-tuple or ingress and egress port pairs for packet matching based on source/destination IP address, TCP/UDP ports, protocol, IP version, source/destination neutron ports.

Once the ports have been specified and the chain order defined, the flow classifiers are applied to specify the traffic that can enter the chain. The reference implementation will be based on OpenvSwitch with flow table entries that override the default MAC based forwarding and instead forward frames based on criteria defined via the Neutron SFC API[22]. This enables the use of this API for control plan forwarding to third party SDN implementations based on Neutron and SFC capabilities (e.g. Contrail, Nuage[23], etc.). Such approach make it tightly coupled to SDN Controllers versions that integrate with the Neutron [10].

---

[18] For more relevant information, see: https://releases.openstack.org/liberty/index.html
[19] See, *for example:* http://docs.openstack.org/developer/networking-sfc/system_design%20and_workflow.html
[20] Some more detailed information can be found at: https://wiki.openstack.org/wiki/Neutron/APIv2-specification
[21] More information can be found, *inter-alia,* at: https://jujucharms.com/neutron-api/
[22] See: https://blueprints.launchpad.net/neutron/+spec/neutron-api-extension-for-service-chaining
[23] For more details see, *for example:* https://www.openstack.org/summit/openstack-paris-summit-2014/session-videos/presentation/nuage-networks-openstack-neutron-and-private-clouds

This SFC mechanism is based on tunnelling mechanisms in OpenStack. The OVS Agent is in charge for the creation of tunnels for SFC traffic between the Compute nodes, where the VNFs reside. The tunnel includes the following attributes:

- Name
- Local tunnel IP address
- Remote tunnel IP address
- Tunnel Type: VXLAN, GRE

The OVS Agent moreover performs traffic classification and service function forwarding via the integration and tunneling bridge of the hosts using n-tuple rules and chain hop index. The MPLS label is used to transport the chain path identifier and the MPLs TTL to transport the chain hop index, in the form of IPv4 packet encapsulation, described in Figure 12.

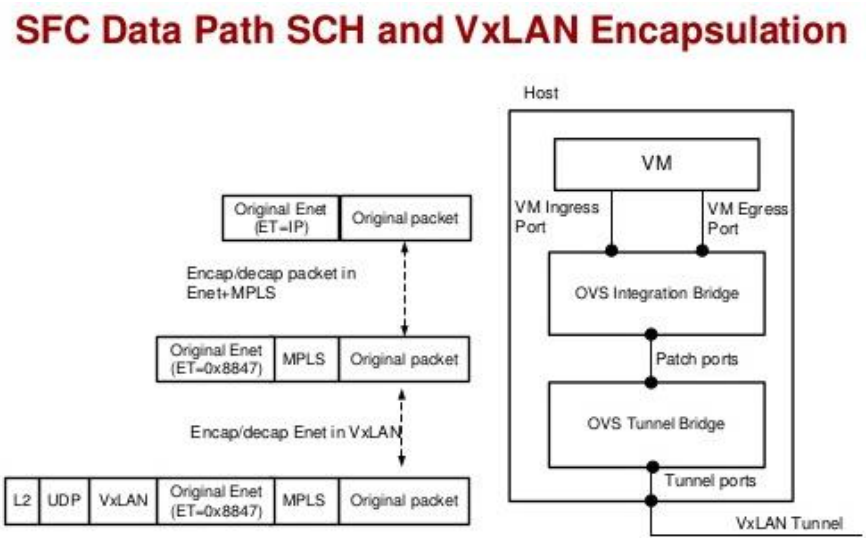

Figure 12: Packet encap/decap in OpenStack SFC based on MPLS

The networking-sfc milestones are related to several changes across the entire networking stack in Neutron and OVS, which include the following changes shown through the components in Figure 13:

- Northbound Intent Based Service Chaining API and Intent Engine[24].
- Neutron API Extension for Service Chaining [11].
- Neutron DB and Driver Manager.
- Neutron Common Service Chaining Driver API[25].
- Service Registration API.
- OVS Driver and Agent extension including Service Function Forwarder (SFF) proxy in theOVS bridge.

[24] See, *for example*: https://blueprints.launchpad.net/neutron/+spec/openstack-service-chain-framework

[25] See, *for example:* https://blueprints.launchpad.net/networking-sfc/+spec/openstack-common-service-function-chaining-driver-api

- CLI and Horizon.

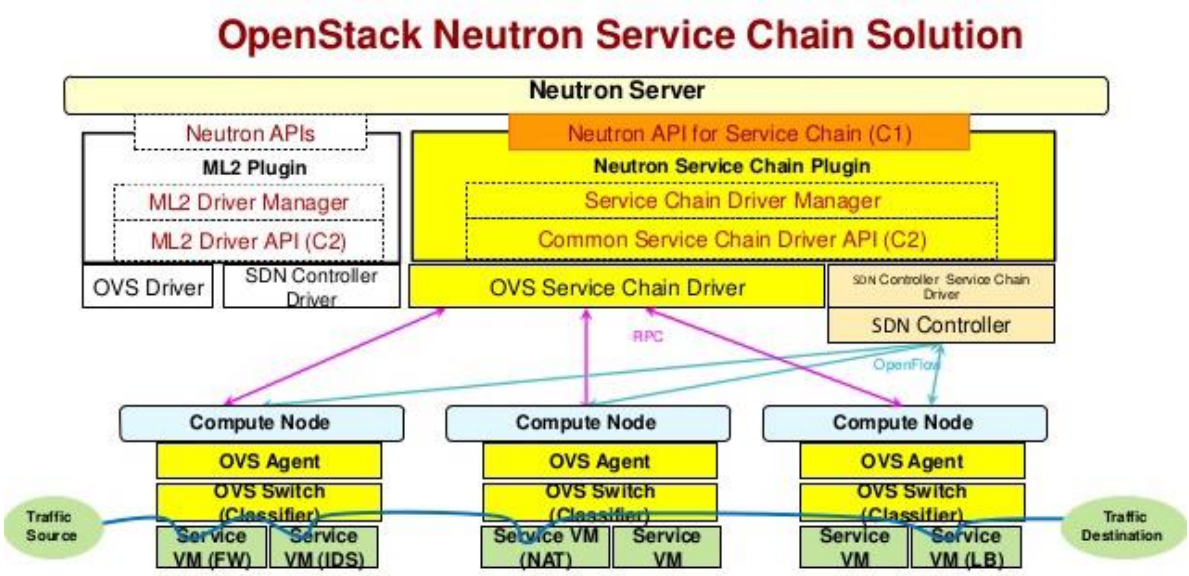

Figure 13: Diagram showing all the modules to be implemented and altered for SFC support in OpenStack

Currently the OpenStack SFC project ([12], [13]) has a work in progress status and the way to test it is via devstack[26] in the Mitaka release[27] of OpenStack. It is also based on a dedicated Open vSwitch version 2.5.0[28]. From the above said, it is currently not possible to fully rely on the SFC implementation, except for testing purposes. The described projects show promising features in terms of standard-*based* chaining solution that will further support third party vendors; however, benchmarks are not yet available to be assessed, at the time of writing of this deliverable.

[26] More related information can be found, *for example*, at: http://docs.openstack.org/developer/devstack/
[27] More detailed information can be found at: https://releases.openstack.org/mitaka/index.html
[28] See, *for example:* http://openvswitch.org/pipermail/announce/2016-February/000081.html

3.3 Tacker Chaining Approach based on OpenDayLight and OpenStack SFC

Tacker [14] is an OpenStack project targeting the creation of a unique tool for management and orchestration of VNFs and network services. Tacker is based on the ETSI MANO Architectural Framework [15], more specifically the VNFM and NFVO components (Figure 13), and uses TOSCA[16], as a specification for the description model of the VNFs.

Currently, Tacker does not yet support SFC, rather only the management of VNFs. The entire plan for extending Tacker for the SFC support is described in [17], while visual representation and use-cases are presented in [18].

The main idea is to extend the current architecture of Tacker, by providing SFC drivers that are based on the OpenDayLight SFC solution [19] and the networking-sfc in Neutron [20] (as discussed above). This project's idea came as a direct requirement from the community to define a network chain on an orchestration level through an abstract representation, which will then be rendered down in the networking level. The specification of the traffic steering will be defined in a VNFFG Forwarding Graph [21] that will contain the VNFs connection points and the path between them in order to enforce the traffic steering.

To be able to support SFC, several changes will have to be made in the Tacker architecture. Some of them include:

- Tacker Server plugin inclusion to support VNFFG.
- Tacker Client to permit CRUD operations[29].
- Drivers for VNFFG.Drivers for classification, or mechanism drivers that will be initially based on 'netvirtsfc' in ODL.
- Extensions to tacker-horizon to enable defining SFCs.

The classification for the chaining will be based on Match Criteria that will be a "synergy" between both, the match criteria supported in ODL and the networking-sfc project.

[29] See, *for example:* https://docs.oracle.com/cloud/latest/globalcs_gs/OESWH/Standard_CRUD_Operations.htm

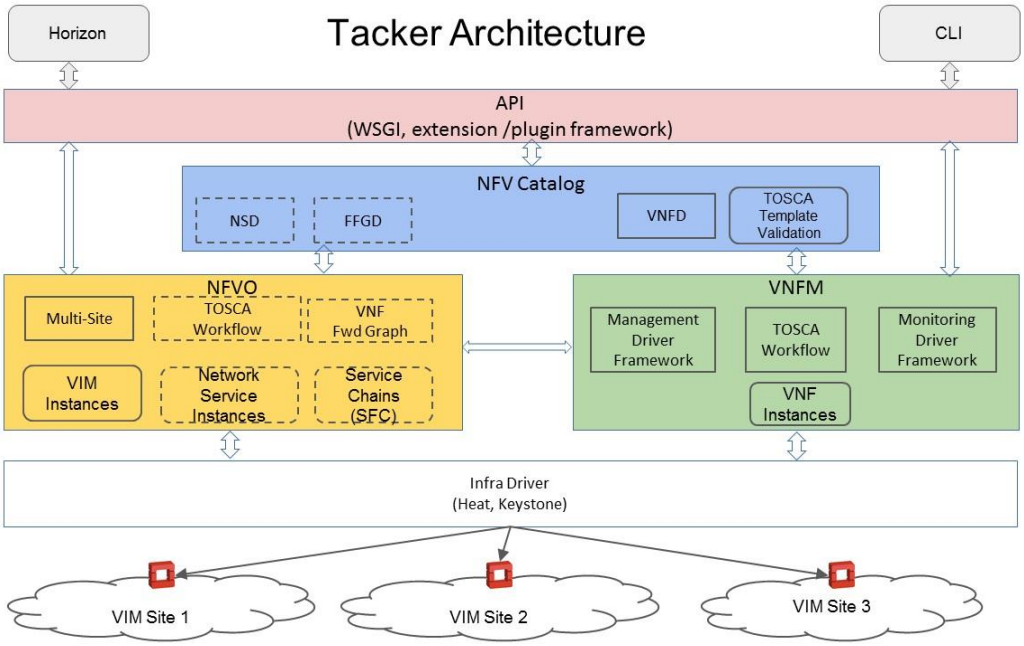

Figure 14: Tacker architecture and components including SFC modules

SFC based on Network Service Header (NSH)

The SFC mechanism that will be supported in Tacker, is based on the SFC approach in OpenDayLight and OpenvSwitch that are using the NSH specification, initially introduced in September, 2015 by Cisco [22].

NSH defines encapsulation mechanism to support SFC and it has been through 10 version changes during the past year, being currently archived in the current draft of the SFC Working Group: 'draft-ietf-sfc-nsh-10.txt'.

The NSH header is inserted in the packets of the chain, to enforce end-to-end flow of traffic on the top of the physical network path. Since NSH is not transport based, is has to be encapsulated by another protocol and the current ways to do this is by using VxLAN+GPE[30] or Ethernet. Once the VNF receives the packets, it decrements the NSH header. The NSH version of OVS was initially supported in the Lithium version[31] of OpenDayLight.

As depicted in Figure 15, NSH has a fixed-length of 4 byte Base Header and a 4 byte Service Path Header. After that come two NSH metadata payload options, which consist of four mandatory 4-byte (16 byte total) Context Headers (NSH MD Type 1) or optional variable-length Context Headers (NSH MD Type 2) [23].

[30] See, *for example:* https://tools.ietf.org/html/draft-ietf-nvo3-vxlan-gpe-03
[31] More detailed information can be found at: https://www.opendaylight.org/lithium

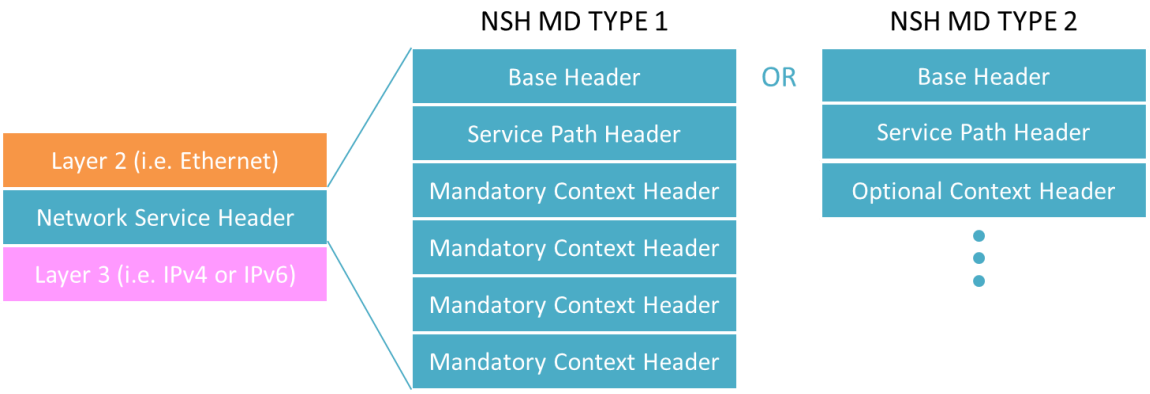

Figure 15: NSH header over IP with Metadata (Options 1 and 2)

To apply the NSH-*based* chaining, a dedicated version of OVS is required (NSH-*aware*) which introduces overheads because of the additional packet header to be send back and forth over the chain path. Moreover, there is not yet a standardized version of the OVS, but rather patches by Cisco and Intel to allow this functionality.

Interesting insight of the NSH adoption for service chain among the operators, has been presented in Figure 9 of the White Paper "*Service chaining in carrier networks*" [24].

Finally, the Tacker community is constantly increasing and many developments have been made since the introduction of the sfc project. However, the work in progress status and the tight dependence on the advancements of two different projects, make it error prone and immature for stable deployments at current stage.

4 Implementation Guidelines

4.1 Selected Technologies

One of the key innovations in SESAME is the synergy between the telco and cloud eco-systems. The baseline infrastructure of the network services is the cloud. In SESAME, the OpenStack VIM will be leveraged to offer the virtualized resources and service compositions on the top of those. OpenDayLight has been the choice of SDN controller within the SESAME eco-system. The following is a description of the mentioned projects and their relation to the SESAME-*defined* network services.

4.1.1 OpenStack

OpenStack is a collaborative open source project, covering the management of storage, network and compute management with a focus on the IaaS part of the cloud stack. It is organized in a modular framework consisting of several integrated and side projects. One of the essential components (projects) is Neutron, the networking service.

The Neutron project 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. The Neutron service has also several plugins available to externalize the OpenStack networking service to third party projects. One of them is the Modular Layer 2 (ML2) north-bound plug-in. The ML2 plugin extends the Neutron integration to third party services and projects. In this context, Neutron can be leveraged with SDN controllers in order to enhance the control of the network with additional SDN functionality. The OpenDayLight controller is currently relying on the ML2 plugin in order to wire up to the Neutron networking service. More details on OpenStack Neutron and networking can be found in the following references ([25], [26]).

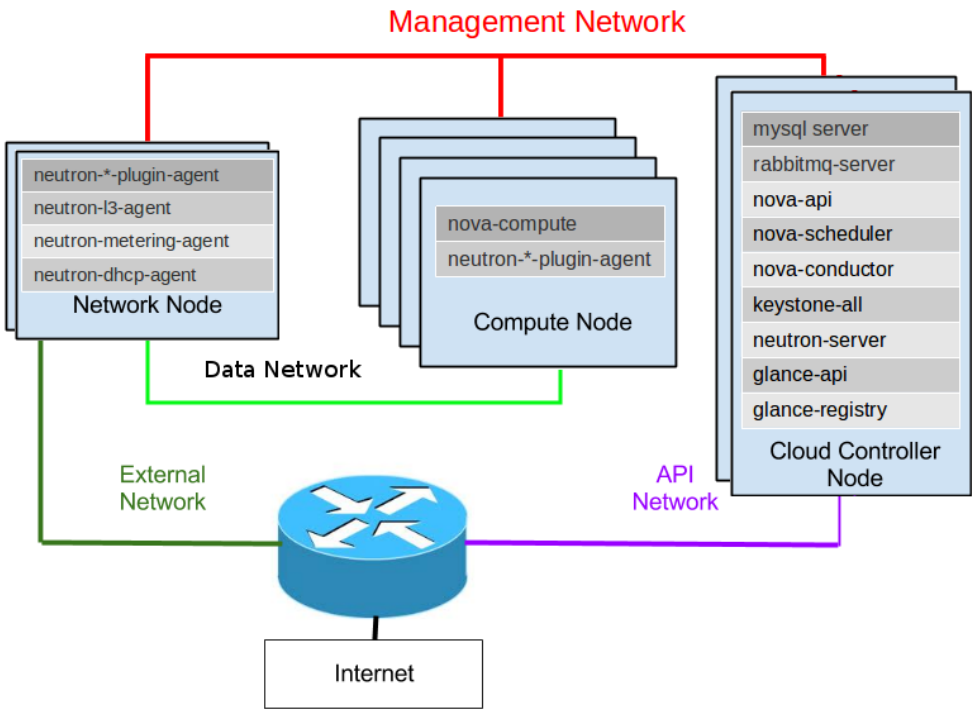

Figure 16: Neutron services and the network connectivity

Figure 16 shows the Neutron project components (plugin agent, l3-agent, dhcp-agent and metadata-agent) and the networking wiring with the rest of the OpenStack services (management, data, external and API networks).

4.1.2 OpenDayLight

OpenDayLight [27] is an open source project that provides SDN programmability framework for network operators and third-party developers, and it is currently being the *de facto* controller among the SDN community. This is seen in the large community that actively cooperates in all of the ODL integral projects, especially in the main controller project on Github [28]. OpenDayLight was started as a project from the Linux Foundation including big company names [29] as active supporters to date with the main aim to advance the SDN and NFV adoption among the community. Similar to OpenStack, ODL has a modular design with variety of networking projects, standards and protocols supported and actively being developed. ODL allows the inclusion of north or southbound projects, standards and protocols due to its extensibility. ODL is built upon the Apache Karaf[32] – a small OSGi[33] based runtime that provides a lightweight container for the deployment of various components and applications. Though, the Karaf runtime one can import different bundles in the controller environment to achieve a specific functionality.

The rationale of having OpenDayLight as an open source SDN controller choice for the SESAME project is the following:

- It is currently the biggest open source collaborative SDN project with huge support from the community and new updates on a daily basis, to "meet" the most important requirements of the SDN developers.
- It is fully integrated with the VIM choice of the project being OpenDayLight through the ML2 plugin on the northbound.
- It offers complete support of southbound protocols like, OpenFlow for the flow manipulation and OVSDB for the OVS networking infrastructure management, SNMP, etc.
- Neutron integration has matured over the last three versions (Hydrogen, Helium and Lithium, Beryllium and currently Boron) in documentation[34] and in stability, which is a prerequisite for a stable and reliable "abstraction" on top of standard OpenDayLight features.
- There are already use case examples and test applications built on the top of the ODL framework related to use cases as defined in SESAME (for example, the service function chaining project).
- Netfloc, the SDK component to be applied in the SDN support of the NFV services is fully OpenStack- and ODL-based, tested and validated in such environment ([30], [31]).
- It is aligned to standards (Tosca, Heat, ETSI, OpenFlow, OVSDB, etc.) and Open Source communities (Tacker, OpenStack, OpenDayLight, OPNFV). This is essential to a large scale European project such as SESAME, in order to deliver work with a potential to provide progressive development over the years, and offer relevant contribution to the community.
- Tested and working Service Function Chaining (SFC) library in Netfloc, based on the ODL controller.

Figure 17 shows the integral components of the ODL project in the current Boron release[35], including the supported northbound applications and southbound protocols.

---

[32] See: *http://karaf.apache.org/*
[33] For more details, see: *https://en.wikipedia.org/wiki/OSGi*
[34] See: *https://wiki.opendaylight.org/view/Release_Plan*
[35] See: *https://wiki.opendaylight.org/view/Release_Plan#Boron*

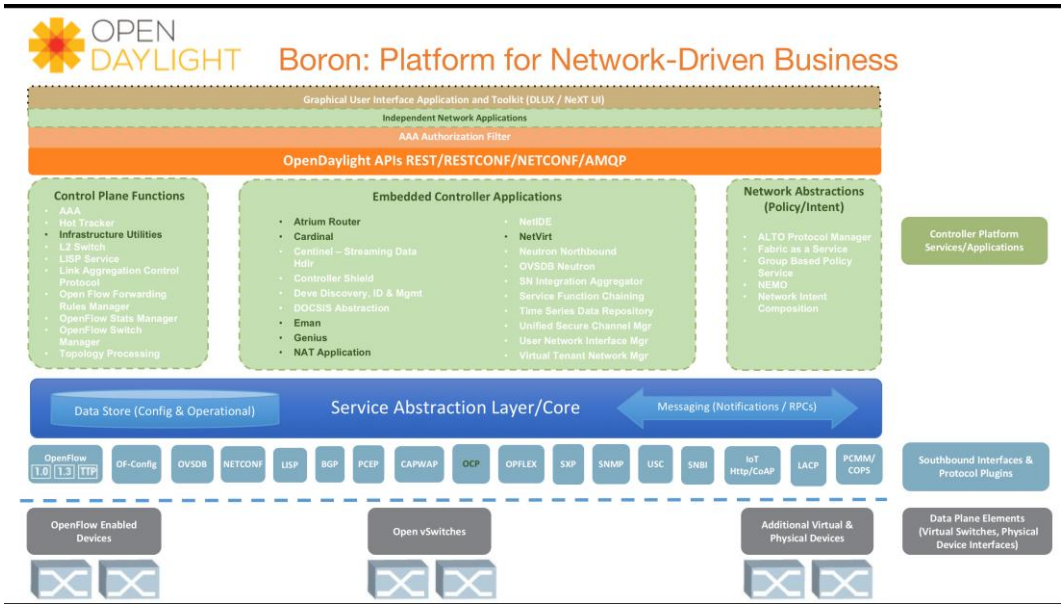

Figure 17: Projects and components in OpenDayLight Boron

4.2 SESAME Approach

To fully implement and support SFC, the conventional protocols plus some SDN adaptations can be used, or novel dedicated mechanisms can be adopted for a full compliance with the related standards. One of the disadvantages of the second approach (by using, for example, the SFC NSH specification header), is the protocol encapsulation complexity and the throughput overheads on the traffic within the chain.

Overall, the current SFC solutions are based on two "key" approaches: packet-*based* and flow-*based* traffic steering. The former enforces the traffic steering by packet header manipulation (ex. packet tagging or rewrites). The later uses the OpenFlow protocol to make traffic redirection based on flow-to-port mapping.

Cisco initiated the NSH specification and it was immediately after leveraged by the OpenDayLight (ODL) community for the ODL SFC native project. Network Service Header (NSH) is introduced in order to enforce end-to-end traffic as an overlay connection above the service chain path. The problem of such solution is that it alters the datagrams; this can potentially cause a problem in the case where the VNF that runs on some of the virtual machine (VM) hops along the chain, requires the datagrams in their original structure. One example is a virtual function such as vDPI (virtual deep packet inspection) that requires the packets in their original structure in order to enforce a correct behaviour.

The advantage of applying SDN for traffic steering is the use of Open Flow (OF) protocol rules (flows) on the SDN controller or the virtual switch (OVS) inside the VM hosting the VNF. This is the second approach to implement SFC, based on flow programming rules, that leaves the packets "untouched" while applying actions on the OF ports of the switches, in order to gear the desired route of the packets in the chain. OpenFlow routing logic is simplistic when compared to NSH, as it mitigates additional headers on the top of the already existing ones [32] (for example, OpenStack native networking and SDN-*based* networking).

To fully support this solution, the underlying network environment has to be fully SDN-*enabled* with OVS switched capable to run the chain rules along the full virtual network graph. The resulting routing flows need to be maintained to reflect alterations in the function chain (e.g. a VNF altering the packet header could invalidate the end-to-end chain).

4.2.1 Netfloc SDK based on OpenDayLight

Netfloc is an open source implementation of SDK for SDN hosted on Github [30] and maintained by ZHAW. It has an architecture based on the OpenDayLight controller for SDN support, where Netfloc registers itself to ODL features via OSGI and depends mostly on the neutron-northbound, ovsdb southbound feature and OpenFlow plugin projects. Components such as Service Chains Patterns and Flow Patterns are maintained by an external API generated by YANG models[36] and stored in the ODL data store. The forwarding plane is based on a standard OVS implementation.

Netfloc comprises of set of libraries aimed to facilitate the development and use of SDN based applications. The SFC library has emerged in Netfloc as a direct requirement from the SFC

[36] For more details, see: *https://en.wikipedia.org/wiki/YANG*

application defined within the research community, chosen to show-case in a simplest way the application of SDN methods to cloud networks by using VNF models and functions.
The key concepts in the design of Netfloc are: (i) The network data representation as an abstracted overlay on the top of OpenStack networking, *and;* (ii) the flow control on the top of the specific network abstraction.

Network Data Abstraction

Netfloc retrieves information from different ODL service interfaces in order to integrate with OpenStack-*based* VIM and enable full SDN compatibility. This data is combined and represented through a network graph, which can be traversed in order to find and establish connections. Such model facilitates the network management enabling straightforward control of the network properties and services created upon its libraries.
Being standards agnostic makes it generic enough to "meet" the needs of components/services support, whereas modularity by design makes Netfloc more extensible in the support of new features and models (for example, Yang service definitions).

Flow Programming Service

Netfloc follows an application-*driven* design supported by OpenFlow standard messages. OF is powerful protocol, yet it does not provide context over more than one switch. One of the goals with the Netfloc's programming service is to provide that context and to allow users to program flows on top of multiple SDN switches.

4.2.2 Network Service Chain Implementation Details

The request of network service chain in Netfloc is facilitated via REST/Java API[37]. A Service Chain is composed of an ordered list of Neutron ports in OpenStack. Netfloc then creates an underlying chain path implementation on the top of the Network Graph and, with this, traffic redirection is established in a holistic end-to-end manner. The MAC Rewrite Pattern for SFC are described in the following part, to give a clear picture of the SFC mechanism and serve as a "guideline" for further activities related to implementation and integration of Netfloc within SESAME.

Traffic Classification

The solution for traffic classification uses two fields in the MAC address: one reserved for the chain ID, and the other as a hop counter along the chain. After traffic classification, the destination MAC address is rewritten to the following format:

Destination MAC = <ID - Chain Identification>:<N - Number of Chain Hops>:00:00:00:00
The resulting virtual MAC address is matched along the Chain Path to forward the classified traffic to the Nth VNF in the chain:

Destination MAC = <ID>:<N>:00:00:00:00/00:00:FF:FF:FF:FF

[37] See, *for example: http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/*

Hop Rewrite

On each egress bridge of a specific VNF, there is a rewrite flow that matches the egress port of the VNF and the virtual destination MAC address and decreases the Number of Chain Hops by 1.

This way the forwarding is always clearly determined. Forwarding to the next VNF is done via the following matching until N reaches 0.

Destination MAC = <ID>:<N - 1>:00:00:00:00/00:00:FF:FF:FF:FF

Endpoint Rewrite

After the last VNF in a chain, the packets are resubmitted to the VNF egress bridge (or sent to an according table). Before reaching the endpoint host the destination MAC address is rewritten to its origin with the following match:

Destination MAC = <ID>:00:00:00:00:00/00:00:FF:FF:FF:FF

Destination IP = <Destination IP Address of specific Host>

The controller is aware of the MAC to IP mapping via the Neutron API and it preserves internally the original src/dst addresses in order to recover them in the last step of the chain. The packet then gets resubmitted and matches on standard L2 forwarding plane.

4.2.3 Network Service Chain APIs

Both the support of Restonf [33] - *based* APIs (as in Figure 18), and Java-from-yang generated APIs is implemented by Netfloc. These can be accessed and invoked using Postman[38] or the OpenDayLight specific apidoc repository on the following URL:

[Neftloc-URL]/apidoc/explorer/index.html#!/netfloc(2015-01-05).

Create, List and Delete of network service chain are currently supported APIs.

[38] See: *https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Flows*

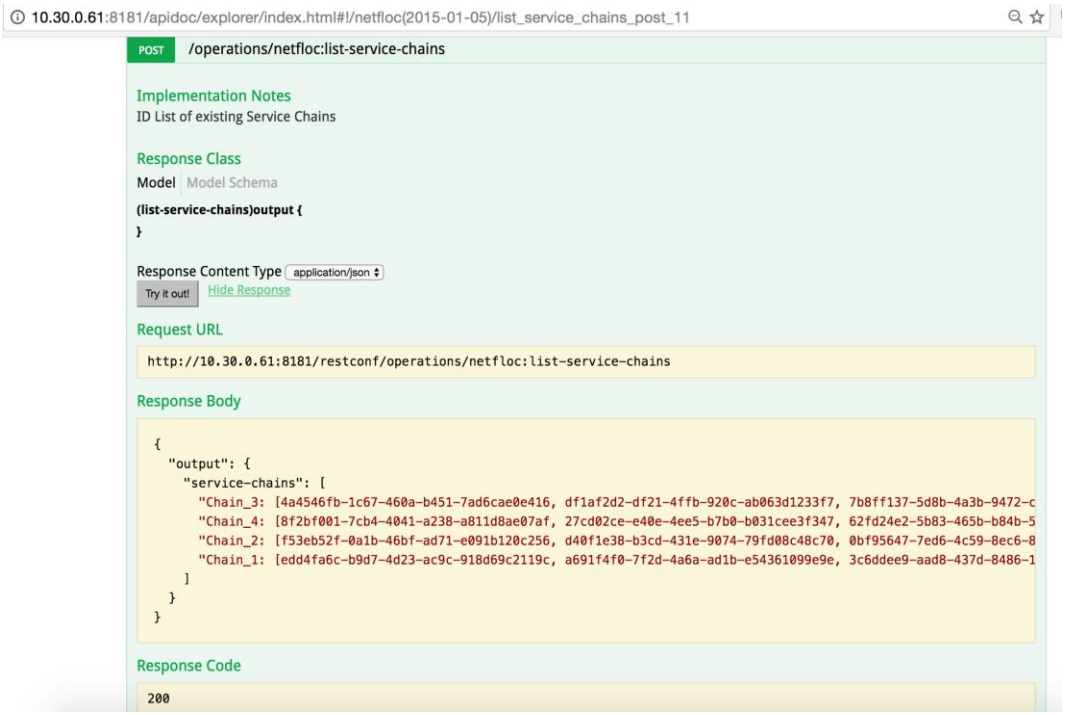

Figure 18: Service Chains List API based on RESTCONF RPC

As an example, List chains API is an RPC call to the Operational data store. It returns the information about the Chain Owner and the Connection points of the chained Virtual Network Functions (VNFs).

This call and the respective chains data is shown in Figure 18.

5 Extra Features to consider

In practice, NFVO plays an important role on the automated service monitoring, scaling and reconfiguration. Based on the inputs of the prior SESAME deliverable D6.1 [3], there are two ways to react upon such an event: (i) Increase the resources of VNFs or instantiate new instances (scale up/down, in/out) and; (ii) modify the SFC, i.e. by changing the data flow depends on, e.g. a radio triggered situation (radio traffic variation over a time period).

In practice, the later situation can be handled in two ways: (i) Make before break approach: create a new instance of NS with updated network configuration and after it gets provisioned, shut down the old (currently running) NS instance, and; (ii) modify the SFC "on-the-fly", i.e. update SDN rules instead of taking a make before break approach. Considering the limited resource at the edge of mobile network, the second option seems as "more reasonable" for scenarios like SESAME. That is, SESAME SFC mechanism will try to address it within the lifecycle of the project.

Nevertheless, implementation of any of the options mentioned above demands introduction of comprehensive monitoring data sets, data processing and decision-making methods, and reaction mechanism deployed on the NFVO/VIM level.

Cognitive knowledge plays an important role, especially on data analysis and decision-making processes. In general, this topic covers a wide range of activities, but in the context of SESAME they can be classified, based upon:

1- Trigger of event: In the context of SESAME, this might be a radio or cloud trigger.
2- Type of reaction: Over radio resources (e.g. radio resource optimization actions) or over cloud resources (done in-line with the ETSI suggestions in two ways: (i)- modification of SFC, i.e. by changing the data flow between VNFs/VMs, and; (ii)- scale in/out – up/down resources over NFVI).

In the following, we will present some initial ideas to introduce cognitive methods for radio (section 5.1) and cloud (section 5.2) resource management in SESAME ecosystem.
Besides, since security is a critical issue on both radio and cloud worlds, SESAME needs to take it into account. To this end, section 5.3 reviews it in the context of SESAME project.

5.1 Cognitive Management of Radio Resources

The availability of computational capabilities at the edge enables the possibility of including monitoring and analytics functionalities implemented as Mobile Edge Computing (MEC) applications to perform resource optimisation tasks.

The ETSI MEC documentation [34] already identifies this possibility, by classifying the MEC use cases around three categories: (i) Consumer-*oriented* services, which benefit directly the end user (e.g. gaming, remote desktop applications, augmented reality, etc.); (ii) operator and third party services (e.g. active device location tracking, big data, security and safety, etc.), and; (iii) network performance and QoE improvements, which aim at improving the performance of the network, either via application-*specific* or generic improvements (e.g. content/DNS caching, performance optimization, video optimization, etc.). Examples of use cases belonging to this latest category of network performance optimization include, among others, the Radio Access Bearer (RAB) monitoring, the SLA management, or the mobile backhaul optimization.

In relation to real-time monitoring a relevant MEC service defined in [35] is the Radio Network Information Service (RNIS), which exposes up-to-date radio network information to applications, such as measurements and statistics information related to the user plane, information (e.g. UE (User Equipment) context and radio access bearers) related to UEs served by the radio node(s) associated with the mobile edge host, changes on information related to UEs served by the radio node(s) associated with the mobile edge host, etc. This radio network information is provided at the relevant granularity (e.g. per User Equipment or per cell, per period of time) and can be used for network optimization tasks.
The use of real-time analytics functionality running at the edge (to support automated decisions and enhance the management systems), is also identified by Small Cell Forum (SCF) in document [36] through different use cases. Two relevant examples of these use cases are summarized in the following:

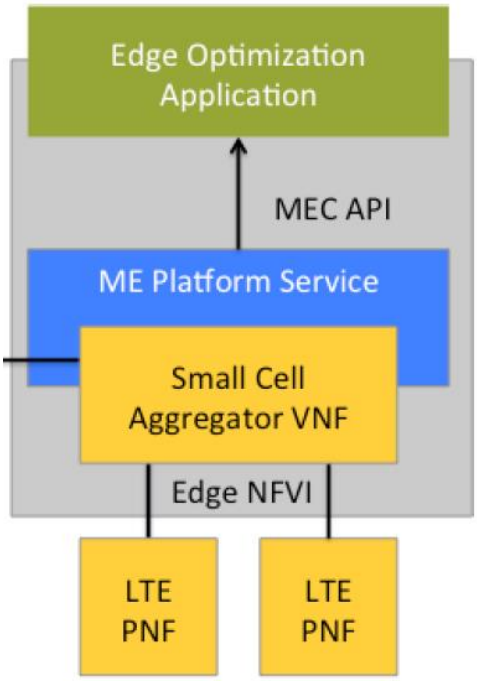

Figure 19: Edge optimization approach (from [36])

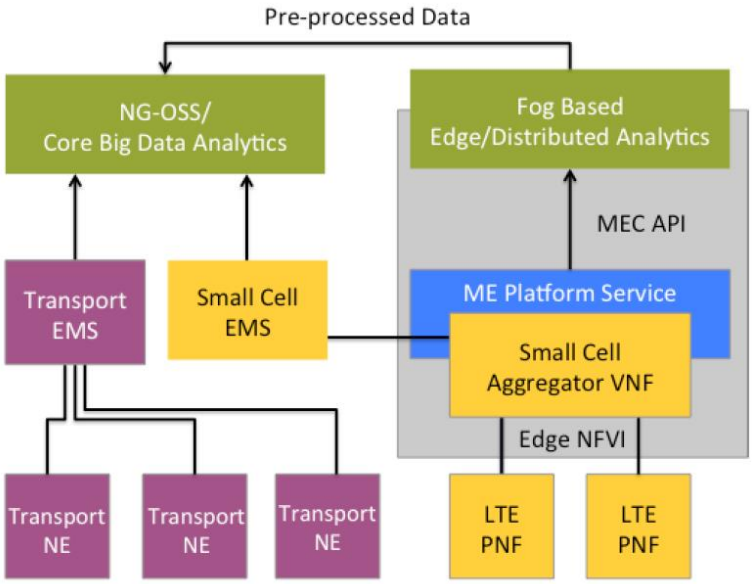

Figure 20: Integration of distributed analytics into a core/big data architecture (from [36])

- *MEC and HetNet optimization:* This use case considers the possibility that a SON application subscribes to the MEC RNIS service and receives radio network information exposed through a MEC API. This facilitates that the SON function can be implemented in a distributed fashion by using SON agents sitting locally in the mobile edge NFVI platform. This is illustrated in Figure 19, where the edge optimization application corresponds to a SON function running at the edge NFVI. It receives, from the MEC API, the radio network information collected by the RNIS (i.e. the ME Platform Service) and corresponding to the different small cells.

- *Mobile Edge and distributed analytics:* This use case discusses a distributed analytics platform that enables distributed applications to operate on vast amounts of local data, instead of using centralized cloud data centers and backhauling data over wide area networks (WANs). Edge computing then enables distributed NFVI (compute, storage and networking) to be used to support real-time analytics based on sources of local data that can be enriched with context-awareness and filtered before delivering only relevant data back to the centralized big data analytics tools, as illustrated in Figure 20.

Based on the above foundations, this section intends to explore the implementation of an analytics framework in the form of a MEC network service that enables the cognitive management of the SESAME architecture relying on the available computational capabilities at the Light DC. It is devised as a network service that provides the functionalities of the general Artificial Intelligence (AI)-*based* framework presented in Deliverable D3.1 [37] for driving "Self-X" decisions.
In particular, it extracts knowledge from different network data and uses it to "drive" the optimisation of both the radio access (i.e. through the "Self-X" functions defined in WP3) and the NFVO operation (e.g. the VNF placement decisions, the service scaling decisions, etc.).

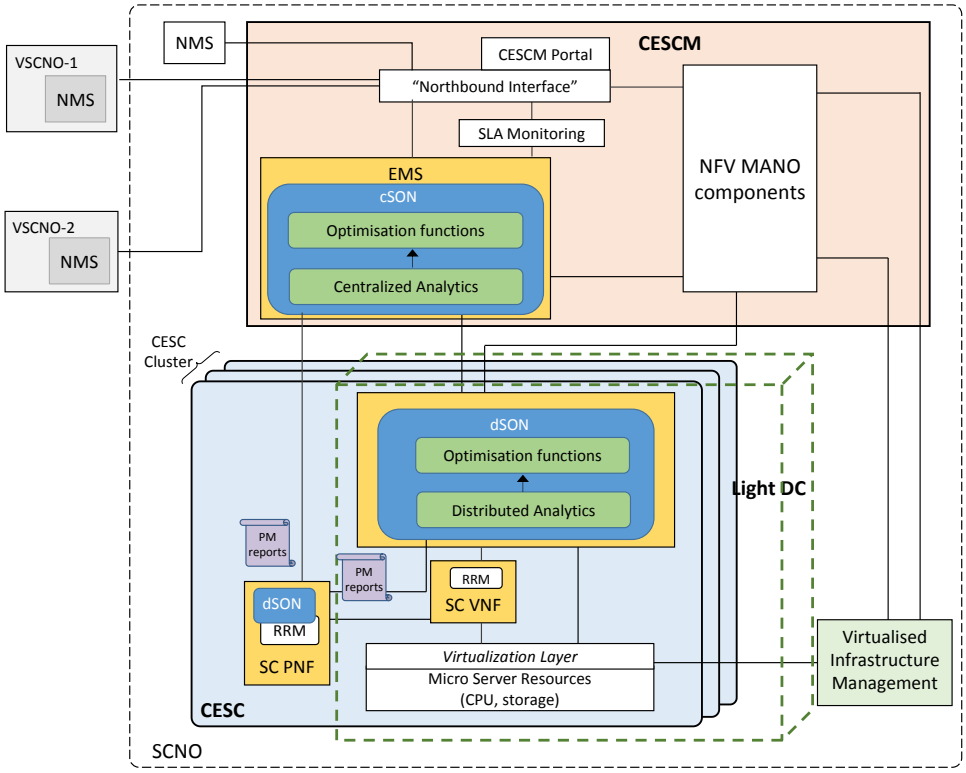

Figure 21: SESAME architecture (simplified view) in relation to "Self-X" and analytics framework

Analytics Framework in the SESAME Architecture

Figure 21 depicts a simplified view of the SESAME architecture with the main focus placed on the incorporation of the analytics functionalities for supporting "Self-x" functions. As it was discussed in [37], "Self-X" functions can be centralized (cSON) and run at the EMS (either the PNF EMS or the SC EMS, depicted for simplicity in the figure as a single box) or they can be distributed (dSON) and be placed at the CESCs. In this case, dSON functions can be part of the SC PNF or they can be implemented as VNFs and run at the Light DC.

The analytics functions shown in Figure 21 correspond to the knowledge discovery functions of the AI-*based* framework of [37]. They are fed by different performance measurements carried out at the CESCs, i.e. the Performance Management (PM) counters defined in Appendix A of

[37], which provided as PM reports in the form of XML files conforming to the format specified in 3GPP 32.435[39]. From a more general perspective, other data sources can also be considered, such as the radio network information exposed by the RNIS service defined in ETSI MEC, in case that this service is available, the data included in UE measurement reports or even information coming from other service VNFs such as the vDPI, which can provide information about the users' applications.

The analytics functions carry out different processing tasks which generate knowledge used by both the cSON and the dSON optimisation functions. This leads to distinguishing between the centralized analytics functions which run at the EMS and support the cSON decisions, and the distributed analytics functions which run over the light DC and whose outcomes support the dSON decisions and, possibly, also the cSON decisions.

This split is facilitated by the computational capabilities available at the edge (light DC) and provides two main advantages over a purely centralized analytics. *First*, it allows that dSON functions can be fed by more detailed knowledge about its local environment obtained by the analytics tools, thus leading to faster and more powerful optimisation decisions. This would be aligned with the SCF use case previously shown in Figure 19. *Second*, the availability of distributed analytics provides a first level of local data processing which reduces the computational burden of the centralized analytics and can also reduce the amount of data to be delivered to the centralized analytics, because the PM and/or measurement reports can be replaced by lighter reports, including only pre-processed data. This would be aligned with the SCF use case previously shown in Figure 20.

It is worth mentioning that the main focus of this work is placed on the distributed analytics and the dSON functions that are implemented as MEC services at the Light DC. A similar concept could be applicable for the dSON functions running at the SC PNF, but this is out of the scope of this work.

The implementation of the distributed analytics framework making use of VNFs offers several advantages. *First*, it offers an inherent flexibility to easily update the capabilities of the framework (e.g. to extract new models or to apply different algorithms). *Second*, it offers the capability to customize the analytics functions on a per-tenant basis.

Distributed analytics and optimisation network service

The multiplicity of different tasks and knowledge models that can be extracted by the analytics functions as well as the type of inputs needed by each optimisation functions, suggests that the distributed analytics and optimisation network service should follow a modular design splitting it into smaller functions that carry out specific tasks. This is illustrated in Figure 22, which shows the functional decomposition of the distributed analytics and optimisation network service.

[39] See: http://www.tech-invite.com/3m32/tinv-3gpp-32-435.html

The distributed analytics includes a collection of VNFs[40] that carry out the data pre-processing and knowledge discovery stages of the AI-*based* framework in [37]. Each VNF is responsible for extracting a different type of knowledge model by analysing specific types of input data. For example, one VNF can be in charge of learning a model that characterizes the time-domain traffic patterns of the cells in the CESC cluster, another VNF can be in charge of identifying the spatial traffic distribution in the area, etc.
The outcomes of each VNF of the distributed analytics are fed into one or more distributed "Self-X" functions, e.g. Mobility Robustness Optimisation (MRO), Mobility Load Balancing (MLB), optimisation of Admission Control (AC), that are also implemented as VNFs. Similarly, Figure 22 also illustrates that the outputs of the distributed analytics VNFs can also be used by the cSON optimisation functions or by the centralized analytics. In such a case, the distributed analytics would provide the knowledge models resulting from the local processing of the PM reports at the Light DC.

As illustrated in Figure 22, it is possible to chain distributed analytics VNFs in order to carry out more complex tasks that combine multiple functionalities (e.g., VNF2 in Figure 22 is fed by VNF1).
To illustrate the above aspects with a specific example, let us take as a reference the use case on learning mobility patterns presented in section 3.4.2 of [37] and let us discuss *how it would be implemented according to the functional components shown in the example of Figure 22*. This use case intends to analyse multiple trajectories followed by different users in a given area in order to derive a database of prototype trajectories (i.e., trajectories followed by many users) and then uses this database for predicting the trajectory of specific users (for details on the process the reader can be referred to the specific scope of reference [37]).

This would lead to the following VNFs:

a) VNF1 learns the mobility patterns. This is done in two steps: *First*, it processes the measurement reports provided by multiple users and transforms them into a set of trajectories. *Second*, it processes these trajectories by means of a clustering process that provides the database of prototype trajectories.
b) VNF2 performs mobility prediction to anticipate the future positions of selected users (e.g. high priority users, users demanding high bit rates, etc.). It takes as input the current positions of these specific users and predicts their future positions, based on the prototype trajectories that have been built by VNF1.
c) The results of the mobility prediction carried out by VNF2 are used by two "Self-X" functions in the example of Figure 22, namely the MLB and the optimisation of the AC. For example, if it is predicted that a high priority user will enter in the coverage area of a certain cell, these functions can take actions over the load of the different cells and the admission threshold parameters, *respectively*, to avoid a dropping for this user.
d) The results of the prototype trajectories learnt by VNF1 are used in the example of Figure 22 by the MRO function, which may decide to tune the HO parameters in a cell depending on these trajectories. Similarly, the results of VNF1 also feed the centralized analytics, so that the trajectories in the local area of a CESC cluster can be combined with the trajectories in other areas thus providing a wider vision of the mobility existing in the scenario. Finally, the cSON optimisation functions can also use the results of VNF1 in order to, e.g. coordinate the adjustments made by the distributed MRO functions.

[40] While the approach shown in the figure assumes multiple VNFs for implementing the distributed analytics functions, other possibilities can be explored, such as having a single VNF with multiple VNFCs each one extracting a different type of model, or having groups of VNFs each one composed by multiple VNFCs.

Considering also a wider scope beyond the "Self-X" functions that refer to the radio side, the outcomes of the analytics framework could also be used for supporting the decisions made by the NFVO. According to the NFVO architecture presented in [3], functions that can benefit from the obtained knowledge models include: (i) the VNF placement (e.g., by observing how the traffic is spatially distributed in a certain area and by predicting the mobility of the users, it is possible to decide which is the best location of a given service VNF inside the CESC cluster for enhancing the QoS perception of the users) and; (ii) the SFC scaling (e.g., by observing the traffic variations along time in the different CESCs it is possible to increase or decrease the amount of CPU, RAM and storage resources used by a service or to instantiate parallel VNFs for better service provisioning to the users).

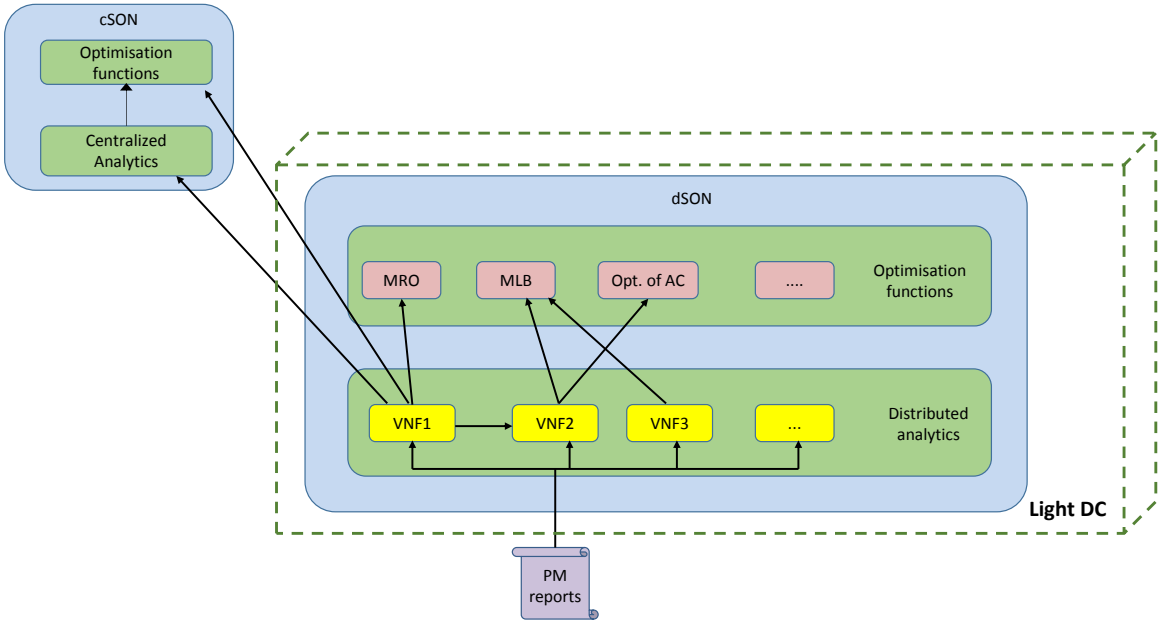

Figure 22: Components of the distributed analytics and optimisation network service

5.2 Cognitive Management of Cloud Resources

Since costs associated to management and maintenance in cloud resources of the data center are not negligible, it is essential to design and deploy energy-efficient technologies for building "eco-friendly" data centers. Auspiciously, the energy consumption problem has already attracted enough attention and several efforts focus on improving the energy efficiency of data center from different points of view. These include processor energy efficiency, storage power management and network power management. For our purposes, we will focus upon the technologies used at the virtualization and cloud levels.

Virtualization Level

Virtualization introduces an abstraction layer between an Operating System (OS) and hardware. Physical resources can be split into a number of logical slices called Virtual Machines (VMs). Each VM can accommodate an individual OS creating for the user a view of a dedicated physical resource and ensuring the performance and failure isolation between VMs sharing a single physical machine. Virtualization allows one to create several VMs on a physical server; and therefore, it reduces the amount of hardware in use and improves the utilization of resources.

Following the rapid development of virtualization technologies such as VMware[41], Xen[42], KVM[43] and OpenVZ[44], more and more data centers apply such techniques to build new architectures supporting cloud computing due to severe benefits that arise, like e.g., improving resource utilization, reducing costs, simplifying server management (see, e.g., [38]). Furthermore, server consolidation, coupled with live migration of virtual machines, consist crucial methods for achieving load balancing and energy saving. Server consolidation allows multiple virtual servers running in a single physical server simultaneously and is the main approach to improve the data center efficiency. This is because server consolidation allows more physical servers to be turned off via the "live migration" of the virtual machines to other unsaturated physical servers. Although virtual machine technology can improve the energy efficiency in data centers, the overheads caused by virtualization and the efficiency of consolidation and migration strategies need to be investigated.

Virtualizing resources helps to boost performance and provide increased fault tolerance. Also, the benefits of virtualization include the capability to move VMs between physical hosts through VM migration procedures and the support of software and hardware heterogeneity. The technique of VM migration at run-time provides the ability of dynamic VM consolidation that applies at the data center level.

Normally, the virtualization layer lies between hardware and the Operating System, and comes into effect by a Virtual Machine Monitor (VMM). This VMM is responsible for the resource multiplexing and also controls the distribution of physical resources to VMs.
Typically there are two ways in which a VMM can participate in power management (see, for example, [39]):

1. The VMM may act as OS which is power-*aware*: In this mode, the OS will monitor the overall system performance and will apply dynamic voltage and frequency scaling

[41] See: http://www.vmware.com/
[42] See: https://www.xenproject.org/
[43] See: http://www.linux-kvm.org/page/Main_Page
[44] See, *for example:* https://en.wikipedia.org/wiki/OpenVZ

(DVFS) or other partial or complete Dynamic Component Deactivation (DCD) management algorithms to the system elements.

2. The VMM may control the power management techniques applied by the guest OS using the application-level and power management commands of different VMs. In this way, it can enforce changes in the hardware power state, or centrally pose system-wide power limits.

Concerning the migration of a virtual machine, there are two migration methods: suspend/resume migration and live migration. Due to the long downtime in suspend/resume migration, live migration is widely used in cloud environment. Pre-copy technology is prevailing live migration technology to perform live migration of virtual machine, which is used in the mainstream virtual machine monitors such as VMware, Xen, KVM and OpenVZ. There are four metrics to quantify the migration algorithms: downtime, total migration time, total data transmitted and workload's QoS. In order to minimize the downtime, pre-copy approach first copies memory pages to the destination machine recursively while keeping VM service still available. When the applications' writable working set becomes small and reaches the threshold, the virtual machine is suspended and only CPU state and dirty pages in the last round are sent out to the destination machine.

Cloud Level

Cloud computing naturally leads to energy-efficiency by providing the following characteristics:

- Economy of scale due to elimination of redundancies.
- Improved utilization of the resources.
- Location independence – VMs can be moved to a place where energy is cheaper.
- Scaling up and down – resource usage can be adjusted to current requirements.
- Efficient resource management by the Cloud provider.

There are various data center level solutions, as described in [39], that focus on combining the workload to fewer servers and deactivating the idle servers, thus boosting the utilization of resources and reducing power and energy consumption. Workload consolidation is normally a complex issue, since tough consolidation and variability of workloads may yield a severe performance degradation of applications. This effect could also lead to increased response times, timeouts, and faults. As stated also in [39], one of the important demands for a Cloud computing facility is to give reliable and measurable QoS. This could for example be defined in terms of Service Level Agreements (SLAs) that characterize features (like, e.g., throughput, response time and latency provided by the system). Indeed, virtualization technologies can provide isolation between VMs that share the same physical computing node.
However, due to reasons like aggressive consolidation and variability of the workload, some VMs may not get the requested amount of resources which will cause performance loss in terms of increased response time, time outs or even failures.
Thus, Cloud providers should deal with energy-performance trade-offs and minimization of energy consumption, and at the same time to continue meeting the QoS constraints.

Also, for effective performance evaluation and comparison of algorithms it is essential to define performance metrics that capture the relevant characteristics of the algorithms. One of the objectives of dynamic VM consolidation is the minimization of energy consumption by the physical nodes, which can be a metric for performance evaluation and comparison.

However, energy consumption is highly dependent on the particular model and configuration of the underlying hardware, efficiency of power supplies, implementation of the sleep mode, etc. A metric that abstracts from the mentioned factors, but is directly proportional and can be used to estimate energy consumption, is the time of a host being idle, aggregated over the full set of hosts. Using this metric, the quality of VM consolidation can be represented by the increase in the aggregated idle time of hosts. Meeting QoS requirements is highly important for Cloud computing environments. QoS requirements are commonly formalized in the form of SLAs, which can be determined in terms of characteristics such as minimum throughput or maximum response time delivered by the deployed system. Since these characteristics can vary for different applications, it is necessary to define a workload independent metric that can be used to evaluate the QoS delivered for any VM deployed on the cloud infrastructure.

5.3 Security

Cloud computing technology has gained wider acceptance in all areas of computing including telecommunication networks, where network and computing infrastructures and resources are virtualised for effective utilisation. One of the focus areas of the SESAME project is the use of cloud technology to virtualised Physical Small Cell (PSC) to provide telecommunication services to Mobile Network Operators (MNOs).

Recently, the Management and Orchestration Working Group (WG) of the European Telecommunications Standards Institute, Network Function Virtualisation (ETSI NFV) defined the Management and Orchestration Reference Architecture.
However, at the end of the first phase of the work, security has not been properly defined in that architecture [40]. At the start of phase 2, the ETSI NFV architecture became more standardised by defining the interfaces between the involved entities inside the Reference Architecture. However, even in the new architecture, security is not considered.
Nevertheless, security should be given utmost consideration and it should be defined as an essential part of multi-tenant NFV environment, in order to increase trustworthiness on the telecommunication networks running on virtual environment.

There have been some efforts to include security in the extended ETSI NFV Reference Architecture. The work by [41], argued that controlling NFV environments through automation and orchestration can only be accomplished if security orchestration is automated in the dynamic NFV environment.
Authors defined the location of potential security functions, of the orchestrator, and *how it will be managed in hybrid networks.* Hybrid networks refer to the physical and virtual network parts. Figure 23, shows the deployment and configuration of the extended ETSI orchestrator.

However, to effectively automate and manage the orchestrator security, the security configuration logs should be recorded. The recorded data can be analysed by using an intelligent algorithm to predict any future security bridge. The current literature fails to discuss and deal with that issue.

SESAME Security Orchestrator

The main function of orchestration in SESAME is to automate and manage the dynamic NFV environment. To achieve effective management of the NFV environment, security must be considered at the initial design of the orchestrator. The security orchestration function is added to the extended ETSI NFV Reference Architecture and it is referred to as SESAME Security Orchestrator (SSO). The SSO function is responsible for automatically managing all relevant security configurations within the NFV environment.

SSO, as a central management point, aims to provide overall understanding of the security issues in the orchestrator through the security records stored in the Security Service CatLog (SSC). It also manages the configuration and organisation of all security functions within the orchestrator as well as the alerts on any relevant security issues that may arise. All the security configurations status, threats and alerts are stored in the SSC.

Moreover, to effectively achieve the automated security system in the orchestrator, Orchestrator Security Service Manager (OSSM) function is introduced. The OSSM is responsible

of monitoring and collecting security service status as well as analysing security data gathered, and alerts SSO of any security threat or bridge in the orchestrator. Based on the information received by OSSM, the SSO makes decision and alerts NFVO on any security bridge. The NFVO will then order for service suspension and request for SSO to re-configure securing, by setting on that service. Upon confirmation of the security configuration by SSO, the NFVO will then order of service resumption.

Security Implementation

The implementation of security on orchestrator varies, depending on the implementation of the orchestrator *per se*. It could be in a single domain, multiple domains with cross domain trust or multiple domains with no trust relationship or outside an active directory environment, such as workgroup [42]. Microsoft recommended to use the Active Directory approach for authentication in System Centre, although, orchestrator does not require an Active Directory environment. In the absence of Active Directory, different authentication mechanisms must be use. These may include the use of SQL authentication between runbook servers and the orchestrator database.

The Security Orchestrator should establish security rules depending on the security requirements imposed to the orchestrator. Some of these rules may be traffic separation and data protection according to the regulations, standards, guidelines and best practices of telecommunication cloud security.

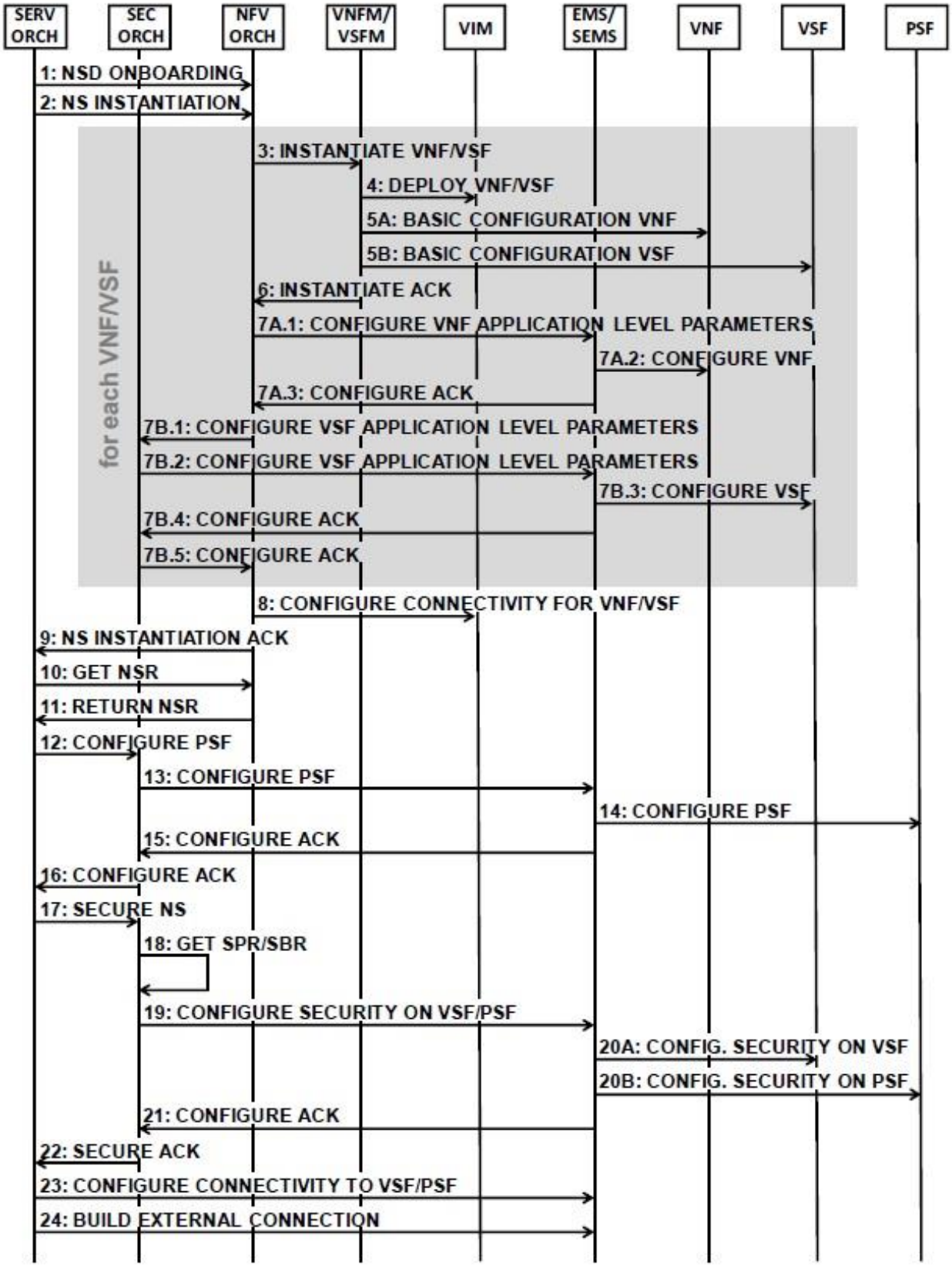

Figure 23: Extended ETSI NFV Orchestrator workflow

Figure Data transmission in the SESAME security orchestrator, including control commands, should be encrypted to provide secure communication. Also any changes in security configuration should be recorded in the SSC. This will be used by OSSM in identifying whether the action was authorised and where the security bridge initiated.

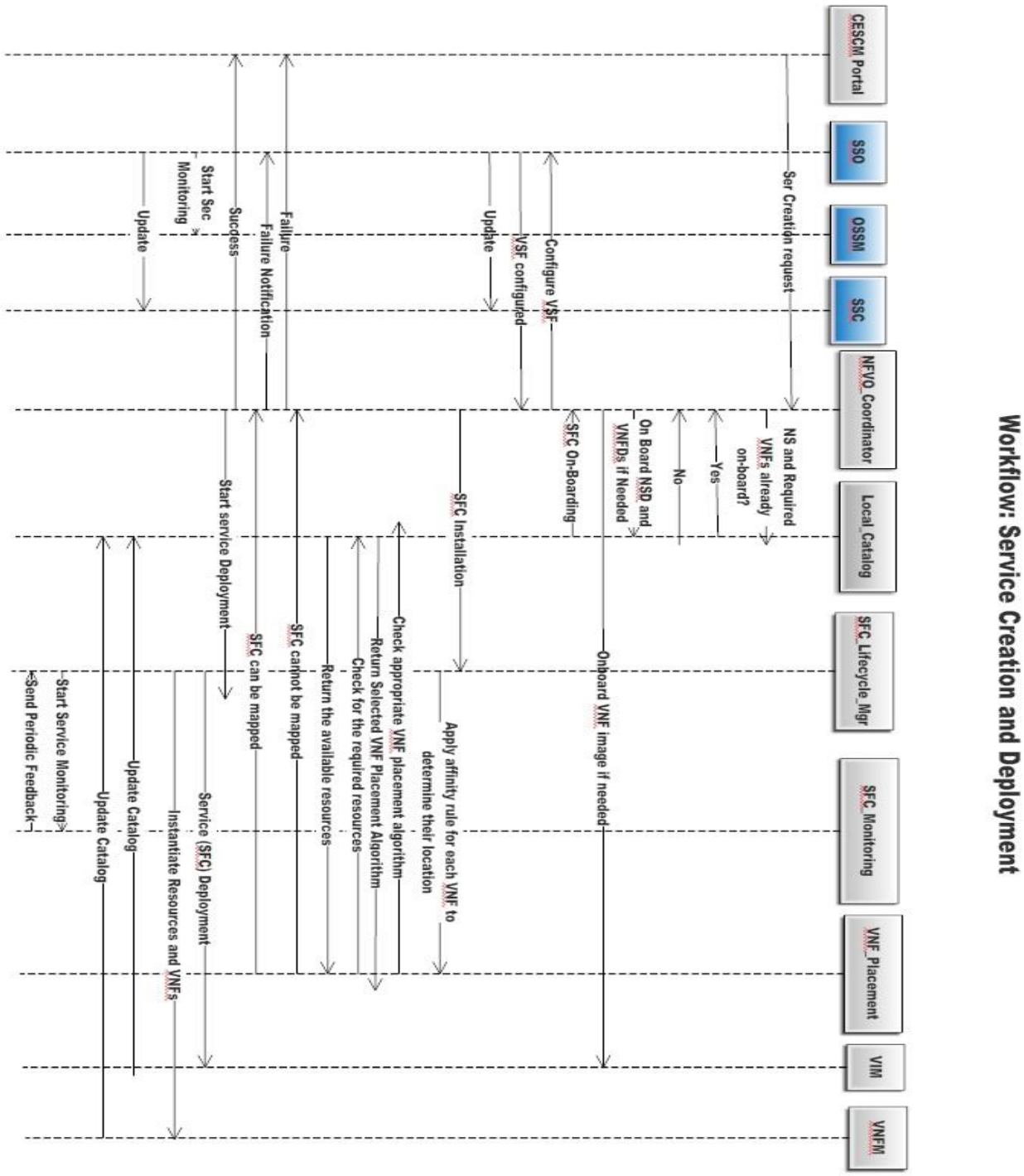

Figure 24: Workflow: Service Creation and Deployment [41]

6 Conclusions

This document has presented an overview of the VNF modeling in SESAME and how those models are consumed by the NFVO and the SDN controller, Netfloc. It has described the SDN controller role within the mixed cloud-telco environment and the ways to deploy an SDN-*based* networking services across the mixed infrastructure.
The service chaining mechanism in Netfloc was highlighted as a mechanism driver to deploy the SFC among the VNFs deployed in the cloud.

An example of a video service chain was described in details, to showcase the advantages of applying the SDN technology to achieve value added services. Employing innovative technologies in development is essential for SESAME's placement among the leading 5G-PPP projects that "share" common building blocks and technologies with the Cloud environment.
Being able to "bridge the gap" among this two worlds by using a common approach, is a challenging work and precisely these points were discussed and elaborated in this document. An initial solution for service function chaining, that has been adopted and tested is also as defined in the initial work description of this task.

A state-of-the-art and comparative analysis has been presented to "serve" as orientation of the undertaken activities and the status of the ongoing deployments in the scientific world and the industry.

Finally, an analysis of cognitive methods for managing cloud and radio resources has been presented, as requirements to have in consideration for the deployment of the SESAME orchestrator and of its services.

Security was briefly tackled as an important aspect in the design of the NFVO's management and orchestration functions.

7 References

[1] "Description of CESC description model", SESAME Deliverable 5.1 of the SESAME project, June, 2016.

[2] https://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000230001PDFE.pdf

[3] "Orchestrator Architecture Design and Interfaces Specification", SESAME Deliverable 6.1 of the SESAME project, June, 2016.

[4] https://github.com/Juniper/contrail-controller

[5] https://github.com/Juniper/contrail-controller/wiki/A-guide-to-'vRouter'-command-line-utilities-(work-in-progress

[6] http://www.juniper.net/us/en/local/pdf/whitepapers/2000535-en.pdf

[7] http://www.seren.net/documentation/unix%20utilities/Snort.pdf

[8] https://www.juniper.net/techpubs/en_US/contrail2.0/topics/task/configuration/service-chaining-vnc.html

[9] https://github.com/openstack/networking-sfc/blob/master/doc/source/api.rst

[10] http://docs.openstack.org/developer/networking-sfc/portchain-ovs-driver-agent.html

[11] http://docs.openstack.org/developer/networking-sfc/api.html

[12] https://wiki.openstack.org/wiki/Neutron/ServiceInsertionAndChaining

[13] https://github.com/openstack/networking-sfc

[14] https://wiki.openstack.org/wiki/Tacker

[15] http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf.

[16] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[17] https://github.com/openstack/tacker-specs/blob/master/specs/archive/liberty/tacker-sfc.rst

[18] https://docs.google.com/presentation/d/18AGaiysVgHOd_fIHVpObMO7zUkMjJGOQ98CUwkxU1xo/edit#slide=id.ge8ecbd4d4_44_13

[19] https://github.com/opendaylight/sfc

[20] https://github.com/openstack/networking-sfc/blob/master/doc/source/api.rst

[21] http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[22] https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/

[23] http://getcloudify.org/2016/02/24/sdn-mwc-open-source-network-cloud-etsi-vnf-nfv-orchestration-automation.html

[24] *http://www.qosmos.com/wp-content/uploads/2015/02/Service-Chaining-in-Carrier-Networks_WP_Heavy-Reading_Qosmos_Feb2015.pdf*

[25] *http://abregman.com/2016/01/06/openstack-neutron-troubleshooting-and-solving-common-problems/*

[26] *https://github.com/openstack/neutron*

[27] https://www.opendaylight.org/

[28] https://github.com/opendaylight

[29] https://www.opendaylight.org/membership/

[30] https://github.com/icclab/netfloc

[31] https://blog.zhaw.ch/icclab/data-model-definion-in-netfloc-and-data-representation-in-netflogi/

[32] https://getpocket.com/a/read/870369383

[33] https://tools.ietf.org/html/draft-ietf-netconf-restconf-05

[34] ETSI (2016, March). *ETSI GS MEC 002 "Mobile-Edge Computing (MEC); Technical Requirements" v1.1.1*.

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

[36] SFC (2016, May). *SCF 172.07.01: "Integrated HetNet architecture framework".*

[37] "CESC Prototype design specifications and initial studies on Self-X and virtualization aspects", Deliverable D3.1 of the SESAME project, June, 2016.

[38] E. Mohammadi, M. Karimi, S. R. Heikalabad, (2011). "A Novel Virtual Machine Placement in Cloud Computing" in A*ustralian Journal of Basic and Applied Sciences,* 5(10), pp.1549-1555.

[39] A. Beloglazov, et *al.* (2011). "A taxonomy and survey of energy-efficient data centers and cloud computing systems", *Advances in Computers, vol.82,* pp. 47-111.

[40] ETSI (2014). *ETSI and NFV, "Network Functions Virtualisation (NFV); Management and Orchestration," ETSI GS NFV-MAN 001.*

[41] B. Jaeger (2015). "*Security Orchestrator*" in 2015 IEEE Trustcom/BigDataSE/ISPA, pp. 1255–1260.

[42] K. Meyler, P. Zerger, M. Oh, A. Bengtsson, and K. Van Hoecke (2014). "*System Center 2012 Orchestrator Unleashed*". Pearson Education, Inc.