Virtual Network Function Use Cases Implemented on SONATA Framework

Cosmin Conțu, Andra Țapu, Eugen Borcoci
University POLITEHNICA of Bucharest – UPB
Bucharest, Romania
Emails: cosmin.contu@elcom.pub.ro, andratapu@elcom.pub.ro, eugen.borcoci@elcom.pub.ro

***Abstract*—Network Function Virtualization (NFV) is a recent and powerful technology capable to support the development of flexible and customizable virtual networks and services, including sliced networks, in multi-tenant, multi-operator and multi-domain environment. Open research issues still exist for architectural, interoperability, design and also related to implementation and experimental aspects. Among several NFV-oriented projects, Service Programming and Orchestration for Virtualized Software Networks (SONATA) is a representative framework. This paper is an extended version of a previous work and develops several Virtual Network Functions (VNF) on SONATA framework. The examples of VNFs include virtual hosts, HTTP server, firewall and a graph of virtual routers. They have been integrated in separate topologies and then chained together into a more complex topology and have been tested using SONATA basis.**

Keywords-Network Function Virtualization; Software Defined Networking; Cloud computing; SONATA; Containernet; Docker.

I. INTRODUCTION

This paper is an extended version of the work [1] [2] and is dedicated to further develop several use cases implementations of *Network Virtual Functions (VNF)* in NFV architecture, illustrating the capability of chaining different functions.

After 2014, *Network Functions Virtualization* (NFV) started to be investigated and developed; today it is recognized as a powerful concept, as well as architecture and technology. It aims to solve some of the current telecommunication world limitations, problems and challenges, like large number of proprietary hardware appliances dedicated to specific services, lack of flexibility and dynamicity, low interoperability, high capital and operational expenditures: capital expenditure (CAPEX), operational expenditure (OPEX), energy consumption and installation space issues [3][4]. Currently, NFV is also seen as a supporting technology in cloud/edge computing domains. NFV decouples the hardware appliances from the network functions that are running over them, by using generic hardware (servers, storage and switches) and running the network functions over virtual machines installed on this generic equipment.

Based on virtualization technologies, NFV allows faster development and deployment (compared to traditional approach) of services composed of network functions that can be implemented in virtualized way. Different virtualized network functions can be defined, instantiated, deployed or moved, while sharing the same infrastructure. They can be created, modified and deleted without needing to physically visit a site to change the hardware supporting those network functions.

The operators' CAPEX and OPEX can be reduced, due to software development (taking advantage of the growing IT industry). Energy consumption reduction is also possible, if a clever power management and migration plan for the virtual machines (VM) is designed.

Software Defined Networking (SDN) [5] is a complementary technology to NFV. The main SDN concept of separating the control plane from the data/user plane creates high flexibility, programmability and network technology abstraction. This approach offers powerful capabilities for the management and control functions. While independent of each other, SDN and NFV can cooperate in order to construct powerful and flexible systems in cloud computing and networking areas. In a general view, NFV and SDN can be seen as "orthogonal": while SDN separates the control and data plane, NFV approach can be used both in the control or data plane to implement different control or forwarding functions as virtual ones.

According to ETSI [6][7], the NFV architecture (see Figure 1) is divided in two main parts: operational and management. The operational part is composed of the functional blocks: *Network Function Virtualization Infrastructure (NFVI)* which contains the physical resources and their abstraction (virtual resources constructed by a virtualization layer); Virtual Network Functions (VNF) which defines different functions that can be composed in services; Operations and Business Support Systems (OSS/BSS). The management part is represented by the Management and Orchestration (MANO) which provides the orchestration and the Life Cycle Management (LCM) of the network functions and infrastructure. Each of the three subsystems in the operational part has a corresponding manager in MANO.

Numerous studies, realizations, projects, proofs of concepts and demos are currently developed in NFV, SDN areas [8][9]. However, many still open research issues exist for such technologies related to different aspects: architectural but also related to use cases, service creation and composition, manageability and resource allocation, virtualization methods, performance obtained in dynamic and mobile environment, scalability, implementation aspects and selection of the software technologies applicable, multi-tenant and multi-domain capabilities and security.

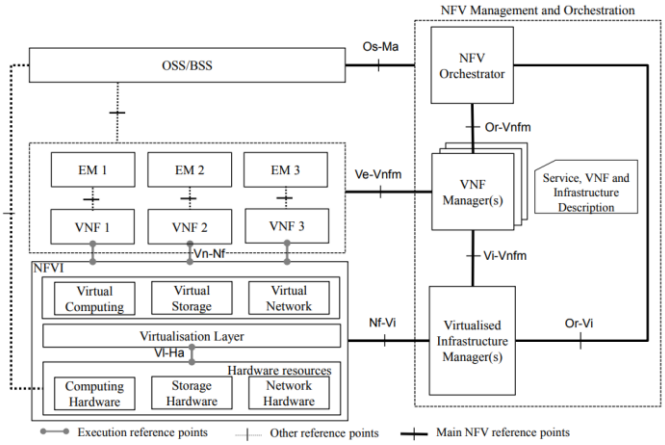

Figure 1. ETSI NFV reference architectural framework [6]

In terms of Development and Operations (DevOps), several problems are recognized to exist [10], like: SDN/NFV infrastructures are not yet stable; Virtual Network Functions (VNFs) are not sufficiently interoperable with orchestrators; multi-vendor environments are not certified; the number of services for which the SDN/NFV framework brings very strong benefits in marketplaces is not yet so large; SDN/NFV combination is difficult and does not offer easy E2E multi-site support; frequently, there is a need for some additional development; key features like network slicing are not yet completely clarified; auto VNF scalability, SP recursiveness, VNF intelligent placements, security, etc., are other open research issues.

Therefore, more extensive experiments with SDN/NFV frameworks are necessary to further clarify different development aspects.

The EU H2020 project Service Programming and Orchestration for Virtualized Software Networks (SONATA) [11] is a relevant example and offers a framework allowing DevOps oriented to SDN/NFV area.

The main purpose of this paper is to further develop experiments started in [1][2], based on SONATA framework in order to more deeply understand the capabilities of the framework, to test its scalability for using it to develop and test some custom VNFs. This work is an additional step to achieve the goal of creating a network service package which will be uploaded and deployed on SONATA platform.

The paper is organized as follows. Section II is an overview of related work. Section III shortly presents the architecture of SONATA framework. Section IV contains the results of the experiments done with SONATA framework and all the steps taken. Section V presents conclusions and future work.

II. RELATED WORK

This section shortly presents a selective view on some related work dedicated to service development and orchestration in virtualized networks and its relation to SONATA architecture, when applicable. It is split in brief overview firstly on EU-funded collaborative projects, open-source solutions and commercial solution provided.

UNIFY [12] (*EU-funded Collaborative Projects*) proposes an architecture which is similar to those of ETSI-MANO and *Open Networking Foundation* (ONF)-SDN. Its objective is to reduce operational costs by removing the need for costly onsite hardware upgrades, taking advantage of SDN and NFV. Across the infrastructure one can develop networking, storage and computing components, through a service abstraction model. The UNIFY global orchestrator consists of algorithms used for optimization of elementary service components across the infrastructure. The project exposes the fact that all resource orchestration related functionalities existing in a distributed way in the MANO SONATA framework, can be logically centralized, when there is an abstraction combination of compute, network and storage resources.

Even if the main idea of a recursive service platform is applicable both for UNIFY and SONATA, the implementation is different. First, the recursiveness in UNIFY is obtained as a repeatable orchestration layer for each infrastructure design, while within SONATA is implemented as a repeated deployment of a complete SONATA platform. Another difference is related to the service specific functionality: in UNIFY it is added by developer inside a Control Network Function (NF), as a dedicated part of the Service Graph, running in the infrastructure; in SONATA the service functionality is obtained using plugins in the service platform which means that it is not mandatory to be on the same infrastructure where the Virtual Network Function (VNF) is running.

OpenStack [13][26] is an *open source project*, mainly written in Python, that provides an *Infrastructure as-a-Service* solution through a variety of loosely coupled services. Each service offers an API that facilitates the integration. Due to its variety of components, the current version of the OpenStack not only provides a pure *Virtual Infrastructure Manager* (VIM) implementation, but spans various parts of the ETSI-NFV architecture. OpenStack is made up of many different moving parts. Because of its open nature, additional components can be joined to OpenStack in order to meet specific needs. *OpenStack Keystone* [14], for instance, offers authentication and authorization not only to the VIM part, but it can be integrated to other services as well. *OpenStack Ceilometer* [15] provides a pluggable monitoring infrastructure that consolidates various monitoring information from various sources and makes the available to OpenStack users and other services. *OpenStack Tacker* [16] aims at the management and orchestration functionality described by ETSI-NFV.

The overall architecture relies on message buses to interconnect the various OpenStack components. To this end, OpenStack uses the *Advanced Message Queuing Protocol* (AMQP) [17] as messaging technology and an AMQP broker, namely either RabbitMQ [18] or Qpid [19], which sits between any two components and allows them to communicate in a loosely coupled fashion. More precisely, OpenStack components use *Remote Procedure Calls*

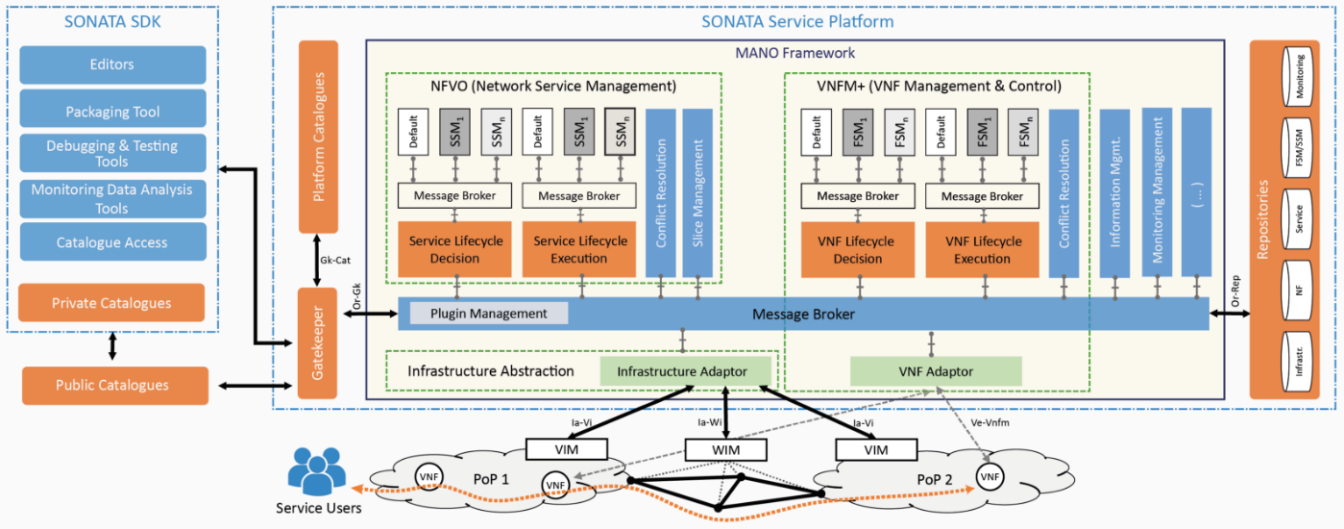

Figure 2. SONATA Framework [21]

(RPCs) to communicate to one another. The OpenStack architecture has been proven to be scalable and flexible. Therefore, it could act as a blueprint for the SONATA architecture.

From SONATA's perspective, OpenStack is used as being supportive and complementary. For the SONATA developers there is the need to have access to a running OpenStack installation to use the capabilities of a VIM for running services from the Service Platform.

Another option for service developers when it comes to SONATA is the SONATA's emulation platform to locally prototype and test complete network service chains in realistic end-to-end scenarios. The emulator of SONATA supports OpenStack-like API endpoints to allow carrier-grade MANO stacks (SONATA, Open Source MANO) to control the emulated VIMs.

To raise their NFV holding**,** commercial vendors have started to market solutions for the orchestration layer. Even if they created their own NFV context, the first generation of NFV Orchestrators (NFVO) is based off ETSI MANO specifications. But there are also several orchestration solutions developed by established network vendors to further expand a larger NFV ecosystem [20].

From SONATA's perspective, the NFV orchestration concept meets the commercial solutions from the following points: to the complete VNF and network service lifecycles, including onboarding, test and validation, scaling, assurance and maintenance. Vendor marketing material and white papers present their upcoming products as holistic solutions for both service and network orchestration, compatible with current ETSI MANO specifications.

These orchestration solutions are commonly part of a fully integrated NFV management platform, including NFVO, VNFM and extended services such as enhanced monitoring and analytics. For example, IBM's SmartCloud Orchestrator can be integrated with its counterpart solutions, SmartCloud Monitoring and IBM Netcool Network Management System, providing an end-to-end offering.

For this paper, SONATA framework was chosen due to its platform which follows the DevOps approach as well as for its Software Development Tools which help the developers to design, create, debug and analyze network services.

III. SONATA FRAMEWORK

In order to make this paper enough self-contained, this section very shortly presents the SONATA framework architecture [25] along with its objectives, use cases and features along with its correspondence with ETSI NFV framework.

SONATA main goal is to develop a NFV framework that provides to third party developers a programming model and a suite of tool for virtualized services integrated with an orchestration system. SONATA allows the developers to achieve a lower time-to-market of networked services, to optimize and reduce the costs of network services (NS) deployment and to speed-up the integration of software networks in telecommunication industry.

Figure 2 presents the general architecture of SONATA framework which complies with and builds upon the ETSI reference architecture for NFV MANO, contains the following components [27]:

- Service Platform (SP)
- Software Development Kit (SDK)
- Catalogues containing different system artefacts.

The Service Platform (SP) is responsible for management and control of network functions and services and it has the same role as MANO block from ETSI model. It is a modular and customizable environment in which the platform operators can create specific platforms appropriate for their business model, by replacing components of MANO plugins. SP has the following core components: gatekeeper, catalogues and repository, MANO Framework (NFVO and VNFM) and infrastructure abstraction.

MANO framework is the core of SP and provides the management for complex NSs for their entire lifecycle. Same way as developed in ETSI, MANO consists of NFVO and VNFM blocks.

The operations from SONATA involved into lifecycle management are split in two types: service-level and function-level operations, which together represent functionalities of NFVO and VNFM from ETSI's reference architecture. The NFVO in SONATA, as in ETSI model, orchestrates the NFVI resources and manages the lifecycles of network services. The VNFM manages the lifecycle of one or multiple VNF instances of same or different types.

Both in SONATA and ETSI, the VIM controls and manages the virtualized resources (network, storage and compute) in an operator's infrastructure domain. (NFVI-PoP). A VIM can handle a specific or multiple type of NFVI resources. Generally the VIM contains the following functional blocks:

A specialized VIM, called WAN infrastructure Manager (WIM) is used to provide connectivity between endpoints in different NFVI-PoPs.

The catalogues and repositories of SONATA consist of network function and services information like code, executables, configuration data and other requirements. These catalogues are divided into:

1) private (located in SDK and used to store locally developed network services per developer or per project).

2) service platform (holds the data which operates and run network services that can be instantiated using SP. Actually in SP there are same types of catalogues and repositories as defined in ETSI (NS and VNF catalogues; NFV instances and NFVI resources repositories) but also two extra: Service specific managers (SSMs) / Function Specific Managers (FSMs) / catalogue and SSF/FSM repository which offers flexibility for service developers to customize their own services.

3) public (representing the third-party network services which are ready to be used by the service developers and SP operators).

In addition, to MANO block of the ETSI model, in SONATA there can be found the gatekeeper component which is responsible of validating the network services posted into SP in a form of packages by mediating between development and operational tasks [30].

Therefore, the service platform follows the ETSI NFV reference design (see Figure 3) but in the same time adds its own extensions which facilitates multi tenancy support by allowing resource slicing which can be mapped to tenants exclusively.

Comparing to ETSI model, SONATA adds SDK as a new important architectural component. The SDK helps the third-party developers to create complex services composed of multiple VNFs, with a set of software tools and also supports service providers to deploy and manage their created NSs on multiple SONATA SPs.

SDK contains different tools to: generate network functions; emulates trail of services; debug and monitoring; support for DevOps operations of network services [28].

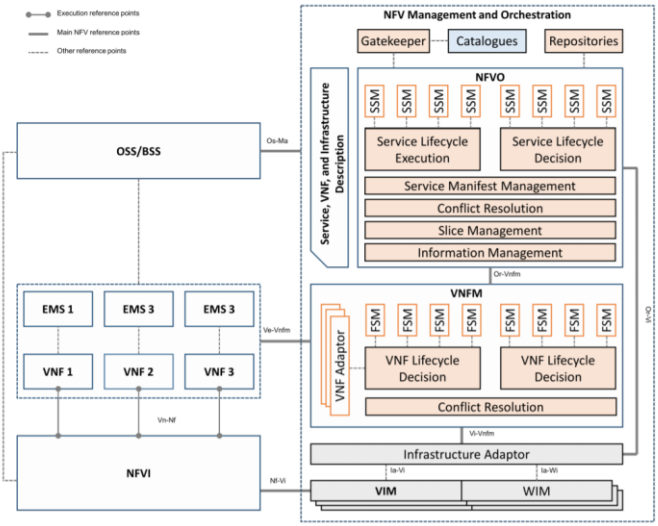

Figure 3. Comparative view: ETSI and SONATA [21]

IV. EXPERIMENTS WITH SONATA

This section presents NFV experiments whose purpose is to test the functionality of different VNFs in various topologies using SONATA framework.

These topologies are represented as custom emulated networks which use Docker [22] containers as compute instances to run VNFs. Moreover, these experiments are developed around SONATA framework and using some specific tools as:

a) Virtual Machine (VM): the experiments are running on a VM of 80GB storage on a 64 -bit Ubuntu distribution ready to use which has been downloaded from SONATA repository [20]

b) Containernet [23]: it is a ramification of Mininet network emulator which allows the developer to create network topologies using Docker containers.

c) Open-source utilities: to create and test the VNFs needed in the proposed topologies, the following collection of utilities has been used: *"iptables"[24], "iproute", "bridge-utils", "traceroute", "inetutils-ping"; "curl"; "squid"; "apache"*.

d) SONATA emulator (son-emu): this is a part of SONATA SDK and it is based on MeDICINE emulation platform. MeDICINE is intended for service developers who can create network service chains and then test them in realistic emulated environments.

A. *(UC1) Simple Virtual Hosts Experiment*

a) *Main objectives:* create two virtual hosts and test their inter-communication.

b) Topology: the topology depicted in Figure 4 contains data centers (DC) in terms of point of presence (PoP) which can be defined as specific emulated hardware by installing docker images which contain the VNFs. In this simple experiment two DCs have been defined, created and used as following:

- Two hosts (dc1 and dc2)

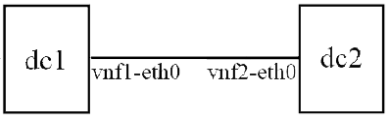

Figure 4. vHosts Experiment Topology

c) *Configuration, tests and results:* first step was to deploy the topology and then instantiate and start the VNFs on each DC (see Figure 5).

```
root@demo:/home/sonata# son-emu-cli compute list
+--------------+-------------+--------------------+------------------+
| Datacenter   | Container   | Image              | Interface list   |
+==============+=============+====================+==================+
| dc2          | vnf2        | vhost-iptables-img | vnf2-eth0        |
+--------------+-------------+--------------------+------------------+
| dc1          | vnf1        | vhost-iptables-img | vnf1-eth0        |
+--------------+-------------+--------------------+------------------+
```

Figure 5. vHosts Experiment compute list

First, using *ifconfig* command on both dc1 and dc2, it can be seen that the IP addresses are in the same network. The IP addresses were set for each data center during the instantiation (see Figures 6 and 7). Afterwards, the "*son-emu-cli network add*" command is invoked, in order to establish the connection between the two datacenters.

```
containernet> vnf1 ifconfig vnf1-eth0
vnf1-eth0 Link encap:Ethernet  HWaddr ca:45:be:12:cb:6f
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
```

Figure 6. vHosts Experiment ifconfig command on vnf1

```
containernet> vnf2 ifconfig vnf2-eth0
vnf2-eth0 Link encap:Ethernet  HWaddr 32:af:20:7c:4d:2b
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
```

Figure 7. vHosts Experiment ifconfig command on vnf2

To check the connectivity between the two data centers, the "ping" command is used from both datacenters (see Figures 8 and 9).

```
containernet> vnf1 ping -c3 10.0.0.2
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=41.388 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=20.545 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=21.240 ms
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 20.545/27.724/41.388/9.666 ms
```

Figure 8. vHosts Experiment ping command from vnf1 to vnf2

```
containernet> vnf2 ping -c3 10.0.0.1
PING 10.0.0.1 (10.0.0.1): 56 data bytes
64 bytes from 10.0.0.1: icmp_seq=0 ttl=64 time=21.580 ms
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=21.156 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=21.199 ms
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 21.156/21.312/21.580/0.191 ms
```

Figure 9. vHosts Experiment ping command from vnf2 to vnf1

B. (UC2) Virtual HTTP Server Experiment

a) *Main objectives:* create a virtual HTTP server which can be accessed from a different host from the same network.

b) Topology: in this topology two DCs have been used (see Figure 10) as following:

- One host (dc1)
- HTTP server (dc2)

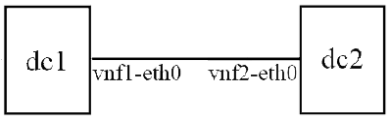

Figure 10. vHTTP_Server Experiment Topology

The two datacenter are in the same network (10.0.0.0/8), dc1 has the IP address 10.0.0.1/8 and dc2 has 10.0.0.2/8. Using apache, a HTTP Server VNF was installed on dc2. Dc1 was used as a virtual host.

c) *Configuration, tests and results:* first step was to deploy the topology and then instantiate and start the VNFs on each DC as can be seen in Figure 11.

```
root@demo:/home/sonata# son-emu-cli compute list
+--------------+-------------+---------------+------------------+
--+
| Datacenter   | Container   | Image         | Interface list   |
  |
+==============+=============+===============+==================+
==+
| dc2          | vnf2        | vhttp-img-new | vnf2-eth0        |
  |
+--------------+-------------+---------------+------------------+
--+
| dc1          | vnf1        | vhost-img-new | vnf1-eth0        |
  |
+--------------+-------------+---------------+------------------+
```

Figure 11. vHTTP_Server Experiment compute list

After the instantiation, the ping command is called between vnf1 and vnf2 in order to verify the connectivity between the two datacenters (see Figure 12).

```
containernet> vnf1 ping -c3 10.0.0.2
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=21.548 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=20.832 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.361 ms
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 20.361/20.914/21.548/0.488 ms
```

Figure 12. vHTTP_Server Experiment ping command from vnf1 to vnf2

On the HTTP Server a resource (html file) will be created in order to be accessed by the virtual host using http protocol. Using *"more"* command on HTTP Server, it can be seen the content of the html file (see Figure 13).

```
containernet> vnf2 more /var/www/site/example.html
root@vnf2:/#
<!DOCTYPE html>
<html>
<head>
        <title>HTTP Call Succeded!</title>
</head>
<body>
<p>HTTP Server Call worked!</p>
</body>
</html>
```

Figure 13. vHTTP_Server Experiment checking the local file from vnf2

With the *"curl"* command called from vnf1, the resource is fetched from vnf2 (see Figure 14) and it can be seen that it the same html file that was previously created on vnf2, as expected.

```
containernet> vnf1 curl 10.0.0.2:80/example.html
root@vnf1:/#
<!DOCTYPE html>
<html>
<head>
        <title>HTTP Call Succeded!</title>
</head>
<body>
<p>HTTP Server Call worked!</p>
</body>
</html>
```

Figure 14. vHTTP_Server Experiment curl command from vnf1 to vnf2 (http server)

C. (UC3) Virtual Firewall Experiment

a) *Main objectives:* create a virtual firewall which has the purpose to block –if requested - some particular traffic flows between two hosts.

b) Topology (Figure 15): it consists of three DCs have been used as following:

- Two hosts (dc1 and dc2)
- Firewall (dc3)

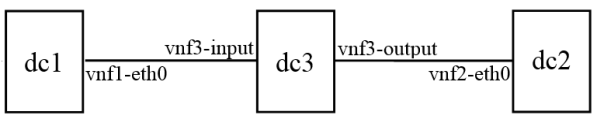

Figure 15. vFw Experiment Topology

The subnet 10.0.0.0/8 has been used together with the *"bridge-utils"* utility on dc3 to make the communication between dc1 and dc2 possible. Utility *"iptables"* has been used to create the *"DROP"* rule for the traffic which is forwarded by dc3.

c) *Configuration, tests and results:* first step was to deploy the topology and then instantiate and start the VNFs on each DC as can be seen in Figure 16.

```
root@demo:/home/sonata# son-emu-cli compute list
+--------------+-------------+-----------------+-----------------+
| Datacenter   | Container   | Image           | Interface list  |
+==============+=============+=================+=================+
| dc2          | vnf2        | ubuntu:trusty   | vnf2-eth0       |
+--------------+-------------+-----------------+-----------------+
| dc3          | vnf3        | vfw-iptables-img | input,output    |
+--------------+-------------+-----------------+-----------------+
| dc1          | vnf1        | ubuntu:trusty   | vnf1-eth0       |
+--------------+-------------+-----------------+-----------------+
```

Figure 16. vFw Experiment compute list

Further, the "DROP" rule has been added for vnf3 and the connectivity between the two hosts (vnf1 with 10.0.0.7 on interface vnf1-eth0 and vnf2 with 10.0.0.5 on interface eth2) has been tested.

If the "DROP" rule is removed, it can be seen in Figure 17 that the two hosts can communicate with each other:

```
containernet> vnf3 iptables -D FORWARD -j DROP
containernet> vnf1 ping -c3 vnf2
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=61.7 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=62.0 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=62.0 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 61.781/61.947/62.045/0.118 ms
```

Figure 17. vFw Experiment ping without "DROP" rule

When "DROP" rule is added then the whole traffic between the 2 hosts does not exist anymore. This rule is exposed in Figure 18.

```
containernet> vnf3 iptables -A FORWARD -j DROP
containernet> vnf1 ping -c3 vnf2
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
--- 10.0.0.5 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2025ms
```

Figure 18. vFw Experiment ping with "DROP" rule

D. (UC4) Virtual Routers Graph Experiment

a) *Main objectives:* create a small network of virtual routers which will forward traffic through a network graph between three hosts from three different subnets.

b) Topology (Figure 19): it consists of six DCs using two different docker images, one for the virtual routers and another for virtual hosts.

- Three hosts (dc1, dc2 and dc3)
- Three routers (dc4, dc5 and dc6)

Routing tables (containing static routes) have been made for the entire topology using *"iproute"* utility. The hosts are assigned within the subnets 11.0.0.0/8, 12.0.0.0/8, 13.0.0.0/8 and the subnets between routers are 10.0.0.0/8 (dc4-dc5), 20.0.0.0/8 (dc5-dc6) and 30.0.0.0/8 (dc4-dc6).

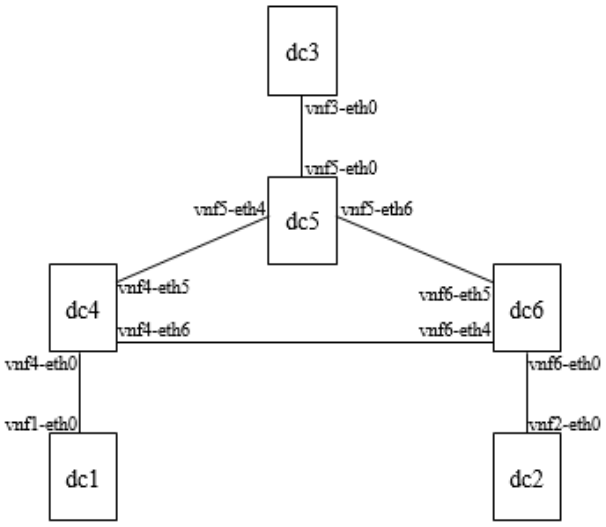

Figure 19. vRouters Graph Experiment Topology

c) *Configuration, tests and results:* after deploying the topology, the VNFs were instantiated and started on each DC and the links between them were also added as illustrated in Figure 20.

```
root@demo:/home/sonata# son-emu-cli compute list
+-------------+------------+------------------+-----------------------------+
---------+
| Datacenter  | Container  | Image            | Interface list              |
        |
+=============+============+==================+=============================+
=========+
| dc6         | vnf6       | vnat-iptables-img | vnf6-eth0,vnf6-eth5,vnf6-eth4 |
.s1-eth5 |
+-------------+------------+------------------+-----------------------------+
---------+
| dc4         | vnf4       | vnat-iptables-img | vnf4-eth0,vnf4-eth5,vnf4-eth6 |
.s1-eth5 |
+-------------+------------+------------------+-----------------------------+
---------+
| dc5         | vnf5       | vnat-iptables-img | vnf5-eth0,vnf5-eth4,vnf5-eth6 |
.s1-eth6 |
+-------------+------------+------------------+-----------------------------+
---------+
| dc2         | vnf2       | vhost-iptables-img | vnf2-eth0                  |
        |
+-------------+------------+------------------+-----------------------------+
---------+
| dc3         | vnf3       | vhost-iptables-img | vnf3-eth0                  |
        |
+-------------+------------+------------------+-----------------------------+
---------+
| dc1         | vnf1       | vhost-iptables-img | vnf1-eth0                  |
        |
+-------------+------------+------------------+-----------------------------+
```

Figure 20. vRouters Graph Experiment compute list

Another way to visualize, as in Figure 21, and monitor the state of the topology and output of *son-emu-cli* is through web-based emulator dashboard.

In this example, the dc4 vRouter has two routes to dc6, with different generic metrics:

- via interface vnf4-eth5, with metric 20;

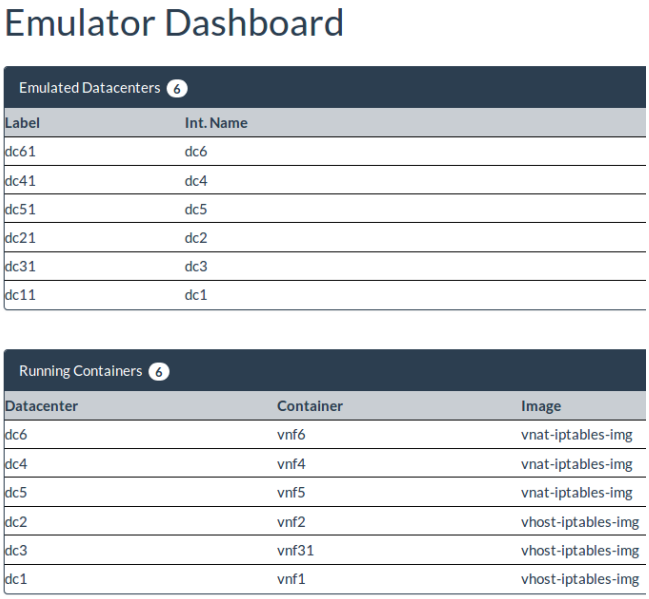

Figure 21. vRouter Graph Experiment emulator dashboard (partial view)

- via vnf4-eth6 with metric 10 (same settings were made respectively on dc6 since static routing is in place).

A shortest path route selection is supposed.

To verify the functionality of the experiment, a traceroute between dc1 and dc2 hosts has been made and it can be seen in Figure 22 that the traffic has been forwarded through the route with the lowest metric (10).

```
containernet> vnf1 traceroute vnf2
traceroute to 12.0.0.1 (12.0.0.1), 30 hops max, 60 byte packets
1  11.0.0.2 (11.0.0.2)  20.589 ms  20.560 ms  20.552 ms
2  30.0.0.2 (30.0.0.2)  82.123 ms  82.116 ms  82.109 ms
3  12.0.0.1 (12.0.0.1)  123.580 ms  123.574 ms  123.569 ms
```

Figure 22. vRouters Graph Experiment traceroute metric 10

If the interface vnf6-eth4 is down and the link between dc4 and dc6 is stopped, it can be observed in Figure 23 that traffic will be forwarded through the route with metric 20 (the only one now remained) when a traceroute between dc1 and dc2 is made again.

```
containernet> vnf6 ifconfig vnf6-eth4 down
containernet> vnf1 traceroute vnf2
traceroute to 12.0.0.1 (12.0.0.1), 30 hops max, 60 byte packets
1  11.0.0.2 (11.0.0.2)  22.001 ms  22.025 ms  22.028 ms
2  10.0.0.2 (10.0.0.2)  43.172 ms  43.165 ms  43.157 ms
3  20.0.0.1 (20.0.0.1)  84.176 ms  84.168 ms  84.160 ms
4  12.0.0.1 (12.0.0.1)  125.179 ms  125.171 ms  125.163 ms
```

Figure 23. vRouters Graph Experiment traceroute metric 20

Although the above experiments are rather simple, they illustrate a complete implementation successful sequence of steps, i.e., to define, instantiate and then run VNF-based

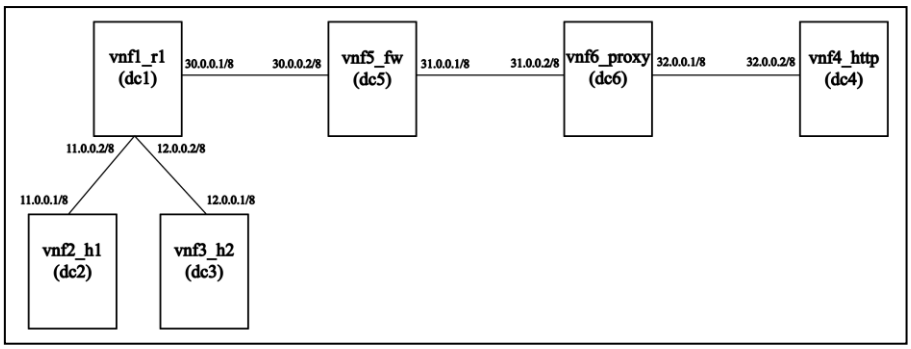

Figure 24. SONATA Framework [21]

topologies on the complex SONATA framework. Modification of the operational parameters is also demonstrated.

E. Multiple Chained VNFs Experiment

1) Main objectives: create a network topology whose purpose is to instantiate a chain of VNFs with SONATA platform. These VNFs roles are : hosts, routers, firewall, proxy, http server, all virtual.

2) Topology: the topology contains data centers (DC) in terms of point of presence (PoP) which can be defined as specific emulated hardware by installing docker images which contain the VNFs. In this experiment (Figure 24) there have been used six DCs as following:

a) Two hosts (vnf2_h1 and vnf3_h2).
b) One router (vnf1_r1).
c) One firewall (vnf5_fw).
d) One proxy server (vnf6_proxy).
e) One http server (vnf4_http).

Routing tables (containing static routes) have been made for the entire topology using *"iproute"* utility. The hosts are assigned within the subnets 11.0.0.0/8, 12.0.0.0/8; the subnet between router and firewall is 30.0.0.0/8; between firewall and proxy is 31.0.0.0/8 and between proxy and http server is 32.0.0.0/8.

3) Configuration, tests and results.

a) First step was to deploy the topology and then instantiate and start the VNFs on each DC as can be seen on Figure 25.

```
root@demo:/home/sonata# son-emu-cli compute list
+--------------+------------+-----------------+-----------------------------+
| Datacenter   | Container  | Image           | Interface list              |
+==============+============+=================+=============================+
| dc6          | vnf6_proxy | vproxy-img-new  | vnf6-eth5,vnf6-eth4         |
+--------------+------------+-----------------+-----------------------------+
| dc4          | vnf4_http  | vhttp-img-new   | vnf4-eth6                   |
+--------------+------------+-----------------+-----------------------------+
| dc5          | vnf5_fw    | vfw-iptables-img| vnf5-eth1,vnf5-eth6         |
+--------------+------------+-----------------+-----------------------------+
| dc2          | vnf2_h1    | vhost-img-new   | vnf2-eth0                   |
+--------------+------------+-----------------+-----------------------------+
| dc3          | vnf3_h2    | vhost-img-new   | vnf3-eth0                   |
+--------------+------------+-----------------+-----------------------------+
| dc1          | vnf1_r1    | vnat-iptables-img | vnf1-eth2,vnf1-eth3,vnf1-eth5 |
+--------------+------------+-----------------+-----------------------------+
```

Figure 25. Topology compute list

b) Second step is meant to prove that the routing is working, and it has been tested with "traceroute" utility between vnf2_h1 and vnf6_proxy (Figure 26).

```
root@vnf2_h1:/# traceroute 31.0.0.2
traceroute to 31.0.0.2 (31.0.0.2), 30 hops max, 60 byte packets
 1  11.0.0.2 (11.0.0.2)  22.552 ms  22.884 ms  22.872 ms
 2  30.0.0.2 (30.0.0.2)  146.628 ms  146.606 ms  146.582 ms
 3  31.0.0.2 (31.0.0.2)  186.763 ms  186.741 ms  186.719 ms
```

Figure 26. Traceroute command between vnf2_h1 and vnf6_proxy

c) Third step represents the functionality of the VNF proxy squid which acts as an intermediary passing the clients (vnf2_h1 and vnf3_h2) requests to the http server (vnf4_http). In the presence of the proxy server, there is no direct communication between the clients h1 and h2 and the http server (Figure 27).

```
root@vnf2_h1:/# ping -c3 32.0.0.2
PING 32.0.0.2 (32.0.0.2): 56 data bytes
--- 32.0.0.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Figure 27. Ping command between vnf2_h1 and vnf4_http

Instead, the client connects to the proxy server and sends requests for a resource file that resides on http server (Figure 28).

```
root@vnf2_h1:/# ping 31.0.0.2
PING 31.0.0.2 (31.0.0.2): 56 data bytes
64 bytes from 31.0.0.2: icmp_seq=0 ttl=62 time=187.837 ms
64 bytes from 31.0.0.2: icmp_seq=1 ttl=62 time=185.059 ms
64 bytes from 31.0.0.2: icmp_seq=2 ttl=62 time=185.054 ms
^C--- 31.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 185.054/185.983/187.837/1.311 ms
```

Figure 28. Ping command between vnf2_h1 and vnf6_proxy

The proxy server handles this request by fetching (with the *"curl"* command) the required resource (proxy_test.html file) from the http server and forwarding the same to the client (Figure 29).

```
root@vnf3_h2:/# curl -x http://31.0.0.2:3128 32.0.0.2:80/proxy_test.html
<html>
<body>
<p>Success! HTTP Server call using Proxy VNF worked!</p>
</body>
</html>
```

Figure 29. Curl command from vnf3_h2 to vnf6_proxy

d) Last step is meant to show the functionality of firewall VNF as it blocks the TCP traffic between h2 (12.0.0.1) and http server through proxy but it allows the rest of traffic, for example ICMP (Figures 30, 31 and 32).

```
root@vnf5_fw:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  12.0.0.1             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figure 30. Drop rule added on vnf5_fw

```
root@vnf3_h2:/# curl -x 31.0.0.2:3128 32.0.0.2:80/proxy_test.html
curl: (7) Failed to connect to 31.0.0.2 port 3128: Connection timed out
```

Figure 31. Curl command from vnf3_h2 to vnf6_proxy after adding the Drop rule

```
root@vnf3_h2:/# ping 31.0.0.2
PING 31.0.0.2 (31.0.0.2): 56 data bytes
64 bytes from 31.0.0.2: icmp_seq=0 ttl=62 time=224.957 ms
64 bytes from 31.0.0.2: icmp_seq=1 ttl=62 time=224.172 ms
^C--- 31.0.0.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 224.172/224.565/224.957/0.392 ms
```

Figure 32. Ping command from vnf3_h2 to vnf6_proxy after adding the Drop rule

The above experiment proves the capability of SONATA to emulate a multiple chained VNFs complex topology. All VNFs successfully communicated with each other, creating a functional, complex network topology.

V. CONCLUSIONS

This paper presented the results of multiple experiments with different VNFs treated separately and a more complex topology containing a multiple chained VNFs using the emulator (part of the SDK) from the SONATA framework.

The single-VNF experiments: virtual hosts which demonstrates the connectivity between two datacenters, virtual http server from which a html file was fetched, virtual firewall to filter the traffic between two hosts and virtual routers that are able to route traffic between several networks, were successfully completed.

Using all the above VNFs, together with a proxy VNF, a more complex topology was created, having as a goal to prove that the functionality of all the VNFs from the VNF chain are preserved and can work together.

The tests have successfully proved that the access to http server through proxy server worked without a known route and also that firewall filtered the inbound traffic to proxy by blocking a certain network.

This paper accomplished the proposed objective to successfully test various single-VNF and multiple chained VNF topologies using the emulator from SONATA SDK.

As future work, new experiments will be done by creating "network service packages" containing multiple chained VNFs and having a lifecycle management, coordinated by the MANO framework. The network service packages will be uploaded and tested with the Service Platform, part of SONATA framework.

Further work should be developed in attempt to solve still existing, specific open issues of the complex SONATA framework, like those mentioned in [29]: NFV Orchestration development- including LCMs for virtual functions and Service Specific Managers (SSM); interfacing the SDK to the Service Platform; scalability and flexibility of the monitoring framework; network slicing capabilities of SONATA (concept still not clear); cooperation of the service platform with recursive architectures, service function chaining via Docker- based VIM (not yet mature); continuous integration and delivery (CI/CD) methodology and others.

REFERENCES

[1] A. Ţapu, C. Conţu, E. Borcoci, "Network Function Virtualization Experiments using SONATA Framework", The International Symposium on Advances in Software Defined Networking and Network Functions Virtualization SOFTNETWORKING 2018.

[2] A. Ţapu, C. Conţu, E. Borcoci, "Multiple Chained Virtual Network Functions Experiments with SONATA Emulator", The Twelfth International Conference on Communications COMM 2018.

[3] NFV White paper: "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1". Available from: https://portal.etsi.org/NFV/NFV_White_Paper.pdf [retrieved: February, 2018].

[4] R. Mijumbi et al., "Network function virtualization: State-of-the-art and research challenges", IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 236-262, 1st Quart. 2016.

[5] B. N. Astuto, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", Communications Surveys and Tutorials, IEEE Communications Society, (IEEE), 2014, 16 (3), pp. 1617 – 1634.

[6] NFV White paper: "Network Functions Virtualisation (NFV) ,Network Operator Perspectives on Industry Progress. Issue 1".Available from: https://portal.etsi.org/NFV/NFV_White_Paper2.pdf [retrieved: February, 2018].

[7] ETSI GS NFV 002: "Network Functions Virtualisation (NFV); Architectural Framework". Available from: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf [retrieved: February, 2018].

[8] S. Van Rossem et al, "Deploying elastic routing capability in an sdn/nfv-enabled environment", 2015 IEEE Conference on Network Function Virtualization and Software Defined Network, pp. 22-24, 2015.

[9] ETSI Plugtests Report: "1st ETSI NFV Plugtests, Madrid, Spain, 23rd January–3rd February". Available from: https://portal.etsi.org/Portals/0/TBpages/CTI/Docs/1st_ETSI_NFV_Plugtests_Report_v1.0.0.pdf [retrieved: February, 2018].

[10] J.Martrat, "SONATA approach towards DevOps in 5G Networks", SDN World Congress, 2017, Hague. Available

from: http://sonata-nfv.eu/content/sonata-approach-towards-devops-5g-networks-0 [retrieved: February, 2018].

[11] S. Dräxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin, and G. Xilouris, "Sonata: Service programming and orchestration for virtualized software networks," in 2017 IEEE International Conference on Communications Workshops (ICC Workshops), May 2017, pp. 973–978

[12] Mario Kind et al. "Deliverable 2.2: Final Architecture". Available from: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY%20Deliverable%202.2%20Final%20Architecture.pdf [retrieved: February, 2018].

[13] The OpenStack Project. OpenStack: The Open Source Cloud Operating System. Available from: http://www.openstack.org/ [retrieved: February, 2018].

[14] The OpenStack Project. Openstack keystone developer. Available from: http://www.openstack.org/developer/keystone [retrieved: February, 2018].

[15] The OpenStack Project. Openstack ceilometer developer. Available from: http://docs.openstack.org/developer/ceilometer [retrieved: February, 2018].

[16] The OpenStack Project. Openstack tacker: An open nfv orchestrator on top of openstack. Available from: https://wiki.openstack.org/wiki/Tacker [retrieved: February, 2018].

[17] OASIS. Advanced messaging queuing protocol. Available from: https://www.amqp.org/ [retrieved: February, 2018].

[18] Pivotal Software. RabbitMq - Messaging. Available from: https://www.rabbitmq.com [retrieved: February, 2018].

[19] Apache Software Foundation. Qpid.Available from: https://qpid.apache.org/ [retrieved: February, 2018].

[20] Containernet and SONATA Emulator Demo. Available from: https://github.com/sonata-nfv/son-tutorials/tree/master/upb-containernet-emulator-summerschool-demo [retrieved: February, 2018].

[21] SONATA. D2.2 Architecture Design.Available from: http://sonata-nfv.eu/sites/default/files/sonata/public/content-files/pages/SONATA_D2.2_Architecture_and_Design.pdf [retrieved: February, 2018].

[22] Docker - Build, Ship, and Run Any App, Anywhere. Available from: https://www.docker.com/ [retrieved: February, 2018].

[23] Containernet. Available from: https://containernet.github.io/ [retrieved: February, 2018].

[24] The netfilter.org "iptables" project.Available from: http://netfilter.org/projects/iptables/ [retrieved: February, 2018].

[25] M. Peuster, H. Karl, and S. v. Rossem: "MeDICINE: Rapid Prototyping of Production-Ready Network Services in Multi-PoP Environments". IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, pp. 148-153. doi: 10.1109/NFV-SDN.2016.7919490. (2016)

[26] Daniel Grzonka, "The Analysis of OpenStack Cloud Computing Platform: Features and Performance" in Journal of Telecommunicationation Technology, 3/2015, pp. 52-57.

[27] Sevil Draxle et al., "SONATA: Service Programming and Orchestration for Virtualized Software Networks", 2017 IEEE International Conference on Communications Workshops (ICC Workshops).

[28] Steven van Rossem et al., "A Network Service Development Kit Supporting the End-to-End Lifecycle of NFV-based Telecom Services", 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).

[29] T. Soenen, S.Van Rossem, W.Tavernier, F.Vicensy, D.Valocchiz, et al., "Insights from SONATA: Implementing and Integrating a Microservice-based NFV Service Platform with a DevOps Methodology", https://biblio.ugent.be/publication/8562744

[30] The SONATA Gatekeeper, Available from: http://sonata-nfv.eu/sites/default/files/sonata/public/content-files/article/SONATA_Gatekeeper_SDNWorld_3.pdf [retrieved: August, 2018].