Network Function Virtualization Experiments using SONATA Framework

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

***Abstract* — Network Function Virtualization (NFV) represents a novel and strong technology to support the development of flexible and customizable virtual networks in multi-tenant and multi-domain environment. Open issues still exist for architectural, interoperability, design and also related to implementation and experimental aspects. This paper presents two experiments in which a virtual firewall and a graph of virtual routers have been integrated in two different topologies and have been tested using SONATA framework.**

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

I. Introduction

Network Functions Virtualization (NFV) is an emerging powerful concept, as well as a 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 [1][2]. 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, 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 deployed or moved using the same infrastructure, created, modified and deleted without needing to physically visit a site to change the hardware supporting those network functions.

The 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) [3] is a complementary technology to NFV. The main concept of separating the control plane from the data 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.

According to ETSI [4][5], the NFV architecture is divided into four main 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; *Management and Orchestration* (MANO) which provides the orchestration and the lifecycle management of the network functions and infrastructure; *Operations and Business Support Systems* (OSS/BSS).

Numerous studies, realizations, projects, proofs of concepts, demos are currently developed in NFV, SDN areas [6][7]. There are still open issues which exist for such technologies and these are related to architectural aspects, to use cases, service creation and composition, manageability, virtualization methods, performance obtained in dynamic and mobile environment, scalability, implementation aspects and selection of the software technologies applicable, multi-domain features, security.

In terms of *Development and Operations* (DevOps), [8] several problems are recognized to exist, 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) [9] is a relevant example and offers a framework allowing DevOps oriented to SDN/NFV area.

The main purpose of this paper is to develop experiments based on SONATA framework in order to understand the capabilities of the framework, to test its scalability for using it to develop and test some custom VNFs.

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, opensource solutions and commercial solution provided.

UNIFY [10] (*EU-funded Collaborative Projects*) architecture 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 the 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 specific 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 mandatory not to be on the same infrastructure where the Virtual Network Function (VNF) is running.

OpenStack [11] 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* [12], for instance, offers authentication and authorization not only to the VIM part, but it can be integrated to other services as well. *OpenStack Ceilometer* [13] 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* [14] 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) [15] as messaging technology and an AMQP broker, namely either RabbitMQ [16] or Qpid [17], sits between any two components and allows them to communicate in a loosely coupled fashion. More precisely, OpenStack components use *Remote Procedure Calls* (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 Orchestrator (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 [18].

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, NFVI 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.

III. SONATA Framework

In order to make this paper self-contained, this section very shortly presents the SONATA framework architecture [19] along with its objectives, use cases and features.

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

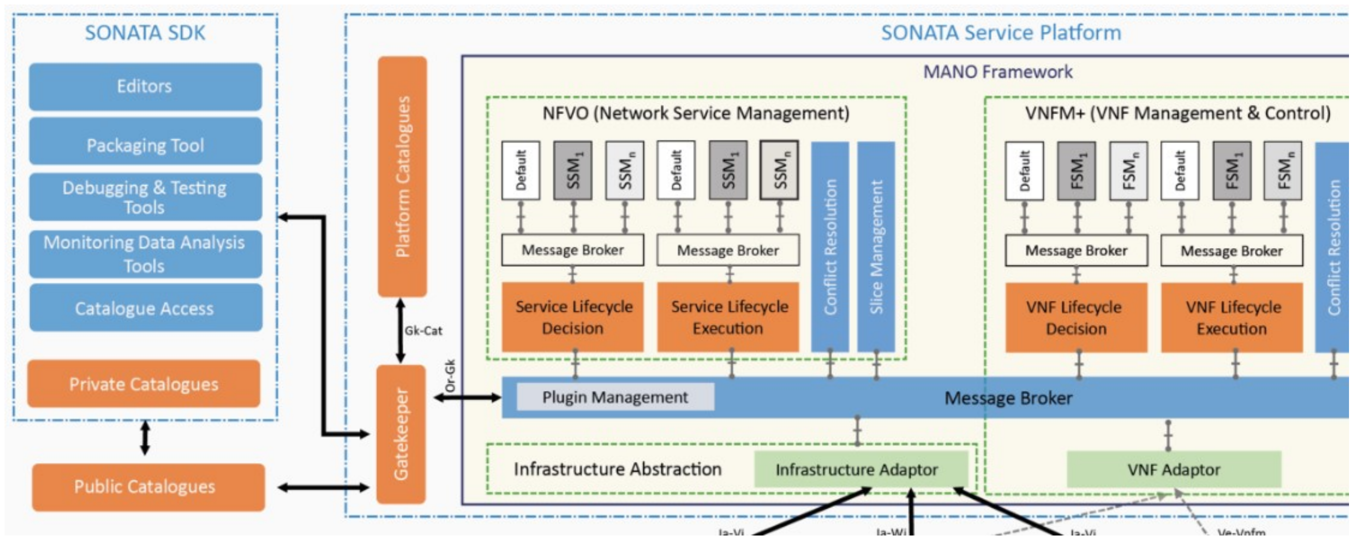


Figure 1. SONATA Framework [19].

The general architecture of SONATA framework, as it can be seen in Figure 1, contains the following components: Software Development Kit (SDK), the Service Platform (SP) and different catalogues in which one can find different system artefacts.

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.

The Service Platform (SP) is responsible for management and control of network functions and services. 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. This environment is also flexible from service developers' perspective which can customize their own services through *Function Specific Managers* (FSMs)/*Service specific managers* (SSMs). Service platform is a component where the users are created and authorized, NS and function descriptors are validated and stored.

The Catalogues consist of network function and services information like code, executables, configuration data and other requirements. These catalogues are divided into private, service platform and public catalogues.

SONATA runs directly on the top of an infrastructure which may belong to the service platform operator or to a third-party operator. To assure the communication between SP and infrastructure, the Virtual Infrastructure Managers (VIMs) are used (example: OpenStack) whose role is to abstract the infrastructure resources.

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 [20] 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 [18]

b) Containernet [21]*:* it is a ramification of Mininet network emulator which allows to create network topologies using Docker containers.

c) Opensource utilities: to create and test the VNFs needed in the proposed topologies, the following collection of utilities has been used: "*iptables*"[22], "*iproute*", "*bridge-utils*", "*traceroute*", "*inetutils-ping*".

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

A. *Virtual Firewall Experiment*

a) *Main objectives:* create a virtual firewall which has the purpose to block the traffic between two hosts.

b) Topology: the topology explained in Figure 2 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 three DCs have been used as following:

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

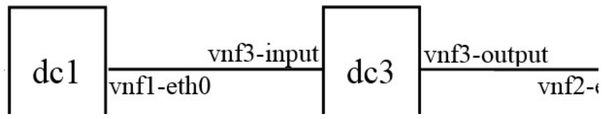
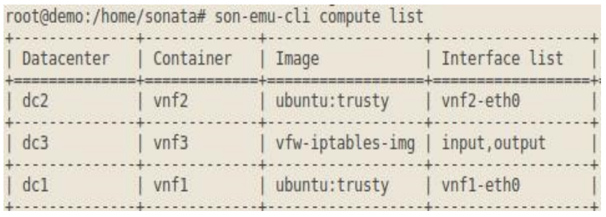


Figure 2. vFw Experiment Topology

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) *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 3:

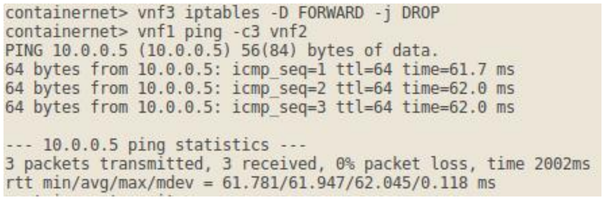


```
root@demo:/home/sonata# son-emu-cli compute list
+--------------+-------------+------------------+------------------+
| Datacenter   | Container   | Image            | Interface list   |
+==============+=============+==================+==================+
| dc2          | vnf2        | ubuntu:trusty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

Figure 3. 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 4 that the two hosts can communicate with each other:

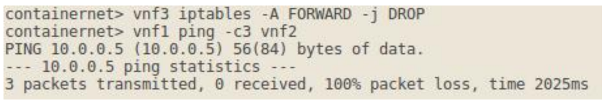


```
containernet> vnf3 iptables -D FORWARD -j DROP
containernet> vnf1 ping -c3 vnf2
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=61.7 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=62.0 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=62.0 ms

--- 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 4. 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 5:



```
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 5. vFw Experiment ping with "DROP" rule

B. 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 6): 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)

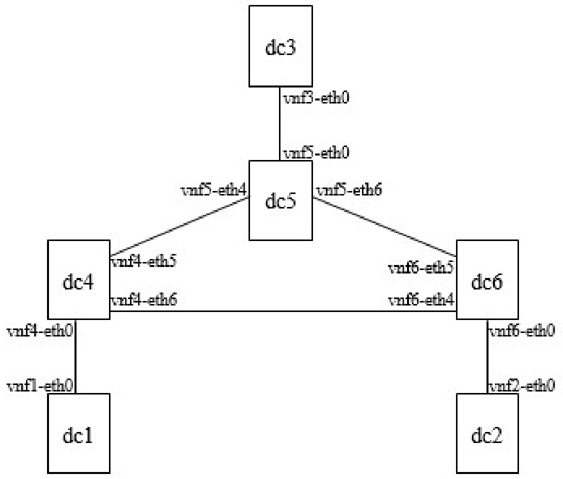
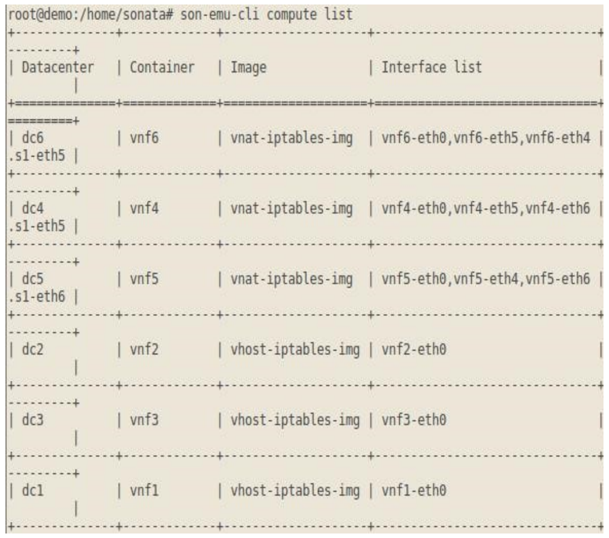


Figure 6. vRouters Graph Experiment Topology

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).

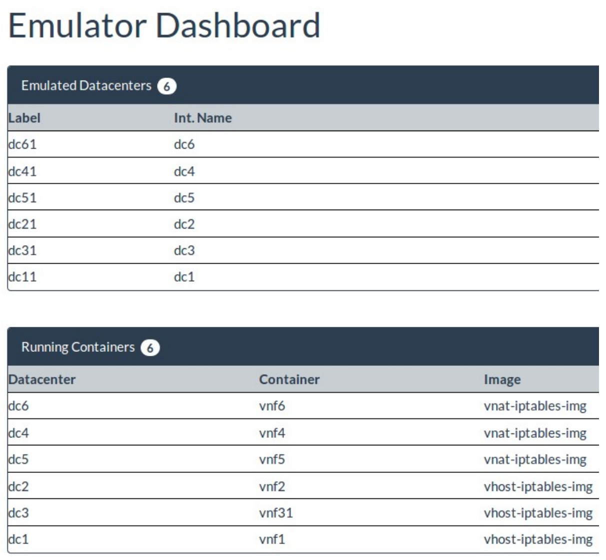
c) *Tests and results:* after deploying the topology, the VNFs were instantiated and started on each DC and the links between them were also added as ilustrated in Figure 7:



```
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 7. vRouters Graph Experiment compute list

Another way to visualize, as in Figure 8, and monitor the state of the topology and output of *son-emu-cli* is through web-based emulator dashboard:



Emulator Dashboard

Emulated Datacenters 6

Label	Int. Name
dc61	dc6
dc41	dc4
dc51	dc5
dc21	dc2
dc31	dc3
dc11	dc1

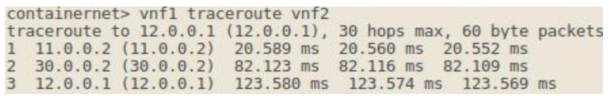
Running Containers 6

Datacenter	Container	Image
dc6	vnf6	vnat-iptables-img
dc4	vnf4	vnat-iptables-img
dc5	vnf5	vnat-iptables-img
dc2	vnf2	vhost-iptables-img
dc3	vnf31	vhost-iptables-img
dc1	vnf1	vhost-iptables-img

Figure 8. vRouter Graph Experiment emulator dashboard (partial view)

For dc4 vRouter there are two routes with different generic metrics: the route via interface vnf4-eth5 has metric 20 and via vnf4-eth6 has metric 10. (same settings were made respectively on dc6 since static routing is in place). A shortest path route selection is supposed.

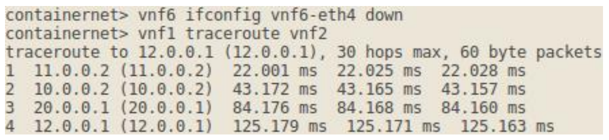
To verify the functionality of the experiment, a traceroute between dc1 and dc2 hosts has been made and it can be seen in Figure 9 that the traffic has been forwarded through the route with the lowest metric (10):



```
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 9. 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 10 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 10. vRouters Graph Experiment traceroute metric 20

Although the above experiments are rather simple, they illustrate a complete successful sequence of steps to define, instantiate and then run VNF-based topologies on the complex SONATA framework. Modification of the operational parameters are also demonstrated.

V. CONCLUSIONS AND FUTURE WORK

This paper presented two NFV experiments using SONATA SDK framework in which it was tested the functionality of two VNFs: a virtual firewall which blocks and filters the traffic between two endpoints and a graph of virtual routers configured to be able to route traffic according to metrics in a static routing configuration.

For the development of these experiments, SONATA architecture has been chosen for multiple reasons: complexity framework, appropriate platform to develop VNFs and to test, explore and emulate virtual networks and topologies.

Beyond the results of these two experiments presented in section IV, there can be proved also the fact that SONATA can:

- offer an open source simulation environment which can be transformed as well into a production environment for the developers who have the need of it
- be a flexible and dynamic test platform and a good support in NFV area
- be able to reduce costs by removing the need of dedicated hardware

As future work, several other experiments will be done using more complex topologies for testing the scalability of SONATA framework. Other area of experiments development is intended to use the emulator within the SONATA NFV eco system to create multiple chained VNFs and descriptors grouped as “network service packages” which will be deployed and uploaded on the SONATA NFV platform and emulator.

Following this direction, after the completion of the proposed future experiments and getting a deeper knowledge of SONATA framework capabilities, the final scope would be to develop and test new VNFs.

REFERENCES

[1] 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].

[2] 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.

[3] 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.

[4] 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].

[5] 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].

[6] 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.

[7] 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].

[8] 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].

[9] 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

[10] 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].

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

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

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

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

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

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

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

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

[19] 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].

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

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

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

[23] 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)