Securing VNF Communication in NFVI

Shankar Lal *†, Aapo Kalliola*†, Ian Oliver*, Kimmo Ahola‡ and Tarik Taleb†
*Nokia Bell Labs, *Espoo, Finland*
†Aalto University, *Espoo, Finland*
‡VTT Technical Research Centre of Finland, *Espoo, Finland*
Email: *first.last@nokia-bell-labs.com, †first.last@aalto.fi, ‡first.last@vtt.fi

Abstract—**In a modern telco cloud, network functions are performed by groups of single or interconnected virtual machines (VMs), which form virtualized network functions (VNFs). Securing these VNFs is both important and challenging, since the VNFs might be performing some mission critical operations and exchanging sensitive information among each other. The problem lies in the implementation of current cloud infrastructure where security of traffic is not considered at large. The exchanged traffic among VNFs is mostly unencrypted and subject to eavesdropping. In this paper, we present possible approaches along with the implementation setup to solve the problem of securing communication among VNFs. We also discuss performance overhead measurements of our testbed setup along with relevant challenges and directions for future work.**

I. Introduction

Telecommunication industry is moving towards cloud and this transformation is realized in the form of network function virtualization (NFV). NFV is defined as a technology to virtualize all the physical network functions into virtual network functions (VNFs) to form a telecommunication service needed to perform a defined network operation [1] [2] [3]. Traditionally, telecommunication network function would undergo a process of purchasing new equipment, provisioning and installing it at a given location. In addition, software upgrades on these equipments also become tedious due to the physical location of the equipment [4] [5]. NFV provides many great benefits of elasticity, resiliency and agility of the VNF workload but in return it brings security challenges such as virtualization and hypervisor related security issues and attacks on VNF communication etc. [6]. In NFV infrastructure, VNFs can be deployed over a single VM or multiple VMs [7] [8]. In the latter case, each VM will host one component of the VNF. It is likely that these VNFs will contain highly sensitive data and mission critical network operations and traffic exchanged between these VNF instances would require confidentiality [9].

Traditionally, securing the traffic exchanged between different VNF components is solely left on the applications which generate the traffic. It is quite a tedious job, considering the fact that security is not part of the design of most applications and it is taken care at later stages. Let's consider a case whereby two VNFs are set up on different cloud hosts and are connected through external IP addresses. One VNF is running some application process which is collecting the data and the other VNF is hosting a database for data storage. Obviously, they both need to communicate with each other. By default in cloud environments, the traffic between VNFs is in plain text. The security of traffic exchanged among VNFs is taken care by applications running inside the VNFs by using protocols such as SSL/TLS. This is also applicable but it only encrypts the traffic generated by the application. There might be other traffic generated by the VNFs which can derive meaningful result and could be subject to hostile eavesdropping and interference. Thus encrypting all the traffic coming in and out from VNFs is highly needed.

Cloud nowadays is geographically sparse which means that there can be some cloud nodes located at different geographical locations around the globe. Given the fact that traffic exchanged among VNFs instantiated over multiple cloud nodes is generally in plain text (e.g., in case of OpenStack-based cloud), it is the responsibility of cloud providers to encrypt the exchanged traffic within the cloud.

In this vein, the approaches proposed in this paper provide end to end encryption between cloud hosts and also among VNFs. One of the proposed approaches modifies networking component of OpenStack cloud i.e. software bridges. It is a fact that many VNFs use single-root input output virtualization (SR-IOV) interface of the cloud host for high performance network throughput and cause packets to completely bypass the software bridges. The solution presented in this paper is not applicable in that scenario.

The rest of this paper is structured as follows. Section II presents the related work. Section III discusses the testbed experiment which includes OpenStack networking modification and our proposed solutions. In Section IV, we present the implementation along with its performance evaluation. Section V concludes the paper recapping the main findings of the envisioned experimental setup and also suggesting future work to improve the performance of the setup.

II. Related Work

Multi-tenancy is one important feature of cloud computing which enables different cloud customers to use computing resources of a public or private cloud. The main emphasis of this architecture is that each tenant and its associated data is isolated and invisible to the other tenants hosted in the same cloud. NFV will also support multi-tenancy where VNFs from different operators can be run in the same NFV domain. This, however, poses threat of various attacks such as isolation failure among VNFs and also with cloud nodes. Let's consider an attack scenario, whereby a group of mission critical VNFs are communicating with each other and a malicious VNF breaks the isolation layer with its cloud host and performs hostile traffic sniffing or eavesdropping. It is possible to sniff or steal data from the cloud host when guest machines have bypassed the isolation with their hypervisor. The guest machines can access the most confidential information such as monitoring the traffic exchanged among other guest machines. This is caused by the vulnerabilities present in the hypervisors. This requires that the isolation mechanism should be hardened and links among cloud nodes should be secured in order to secure the VNFs communication.

In multi-tenant cloud environment, one way of separating the traffic of guest VNFs from different operators hosted on single NFV domain, is to use virtual local area network (VLAN) tagging. Each virtual network gets unique VLAN ID and only VNFs connected to this virtual network can access each other, thus preventing access from other VNFs running in different networks. VLAN security depends upon the security of cloud provider's network e.g. routers and switches (both physical and virtual). In general, a cloud provider has complete control over hosted VNFs and if desired, the cloud provider can see all data residing in the VNFs, including sensitive ones (e.g., cryptographic keys). Therefore, cloud model is based on trust and cloud provider must be a trusted party without having any malicious intents. Whilst this could be true, there are still chances that cloud providers could be careless or incompetent to implement robust security mechanisms. For example, they could make some configuration errors resulting from human mistakes such as putting one operator's VNF on some other operator's VLAN thus enabling attacks as mentioned above [10].

As per cloud security alliance (CSA), attacks against communication of virtualized network functions are complicated due to reasons that their traffic do not flow over the network anymore but through the hardware backplane. This limits conventional network monitoring and security tools in place. CSA suggests to implement virtualized security appliances to tackle these issues. Incorporating security tools inside VNFs is another option [6].

III. Testbed experiment

In this section, we provide two approaches for securing VNF communication in an OpenStack-managed cloud environment. In this proof of concept demonstration, we made some modifications to OpenStack networking layer and incorporated our solutions as explained below.

A. Modifying OpenStack Networking Layer

Each compute node in OpenStack contains networking components such as virtual switches and virtual bridges etc. Virtual switches are Open vSwitch and are managed by Open vSwitch agent. The bridges used are Linux bridge, integration bridge, tunnel bridge and VLAN bridge. Linux bridges handle mainly the security groups since Open vswitch and iptables do not have such functionality. Figure 1 shows OpenStack's underlying network connectivity [11].

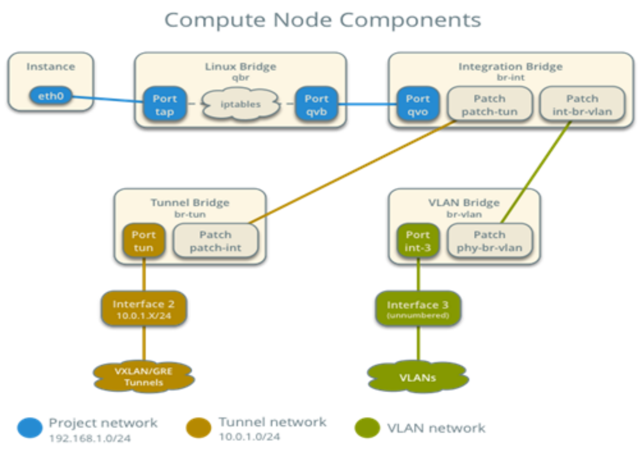

Fig. 1. OpenStack networking diagram [11]

B. Proposed Solutions

We consider two different scenarios for securing communication among VNFs. In the first scenario, the VNF operator creates and manages its own virtual private network (VPN) server to establish secure traffic tunnels among its VNFs. All the cryptographic keys would then be under control of the VNF operator. This is needed when the VNF traffic is highly confidential and the underlying cloud infrastructure is not reliable. In another scenario, cloud provider is trusted and does all the modifications needed to secure the communication among the VNFs so that there is no need to modify the VNF images. Both proposed solutions are explained as below:

- **Approach 1: Dedicated vVPN server per operator**
  VPN service can also be virtualized like other network functions. A VNF operator could setup its own VPN servers to secure communication of their VNFs as shown in Figure 2. This approach has basically two requirements. The first requirement is that the VPN server should have information about all VNFs such as host name, IP address and network interfaces. Thus VPN servers maintain routing tables to know where all VNFs are located in order to create end to end encryption tunnels. The VPN server also generates the cryptographic keys for every client i.e., VNF. The keys could be managed by public key infrastructure (PKI) connected with vVPN server. The other requirement in this approach is that the VNF images need to be customized to include VPN client tool and already acquired cryptographic keys from VPN server. The VPN tunnel will be activated at startup when VNF images are booted as depicted in Figure 3.
  This setup eliminates any changes which should be carried out in NFV and somehow ensures that VNF operators could setup end to end encryption tunnels and gain control over their traffic. In order to avoid the VPN server becoming a single point of failure, a high availability deployment model should be considered such

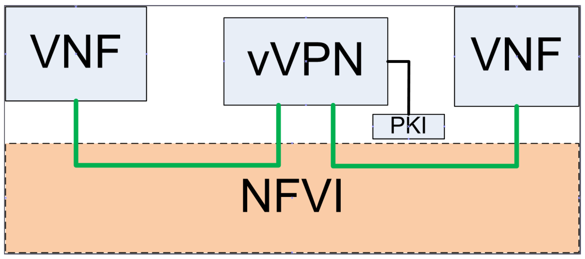

Fig. 2. Virtualized VPN server setup

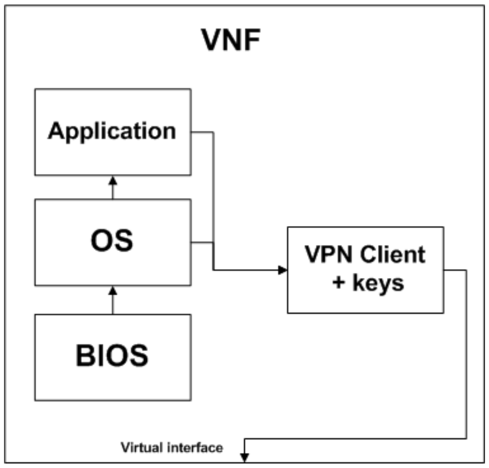

Fig. 3. VNF image with integrated VPN client

as spinning up new instances of VPN server on demand.

- **Approach 2: VPN service provided by the NFV** This setup provides comparatively easier method for VNFs to start secure communication between each other without worrying about the image customization such as installing VPN client tool and performing key management. In this setup, NFV components such as compute and network nodes, host VPN service to provide traffic encryption. One node acts as a VPN server and hosts VPN portal for the purpose of key generation and management, while all other nodes only host the VPN client. The operator is provided with a VPN portal to generate cryptographic keys, select the destination VNF/(s) and start the VPN tunnel. Once the VPN tunnel is established between compute nodes, all the communication generated by the VNFs will be encrypted. The whole setup is presented in Figure 4. This setup needs minimal efforts at VNF level to exchange secure communication but requires trust on its underlying infrastructure (cloud service provider).

IV. Implementation

The implementation of the above-mentioned approaches is carried out in OpenStack based cloud infrastructure. The OpenStack release version used in this implementation is Kilo. This implementation modifies parts of Openstack networking

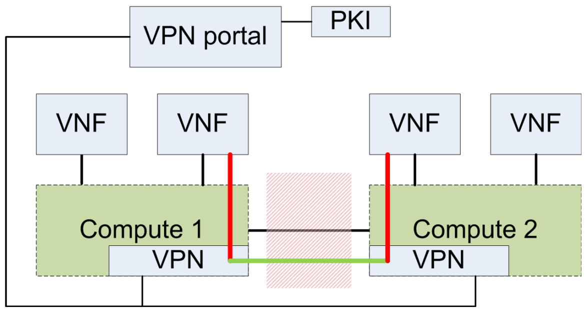

Fig. 4. NFV provided VPN service

component which is known as neutron. Neutron is an OpenStack project which provides network connectivity as a service between interface devices (e.g. virtual network interface cards (vNICs)) managed by other OpenStack services [12]. Neutron can be implemented with Linux bridge or Open vSwitch [13] technologies.

Our testbed experiment consists of four phyiscal server machines and two virtual machines, all running Debian Linux Ubuntu 14.04. This configuration setup contains two Intel Xeon Servers E5-2600 v3 @2.20 GHz with 72 GB RAM, two HP ProLiant servers DL360 G5 having Intel Xeon 5160 @3.00GHz and 24GB RAM. Both virtual machines have 2 GB RAM and 1 virtual central processing unit (vCPU) allocated. The OpenStack environment have a very light workload during this performance testing.

In implementing the VPN functionality for VNF communication, we have considered two separate approaches as described above. In the first approach, we set up two VNFs and a dedicated virtualized VPN server. Virtual VPN server is setup on Ubuntu 14.04 cloud image and runs OpenVPN package [14]. The cryptographic keys for VNFs are manually generated from VPN server and integrated in the VNF image file along with VPN server's certificate by customizing the VNF image. This was performed by using open source tool known as *libguestfs* which has the library for modifying virtual machine images [15]. VPN server is also provided with IP address and host name of the VNF instances so that it could perform the routing of the traffic. In a large and automated setup, the key generation and management could be performed efficiently using existing PKI tools.

In the second approach, we enabled the NFV infrastructure to provide the VPN functionality for its hosted VNFs. This approach requires modification in OpenStack networking component. As mentioned above, OpenStack uses multiple bridges which are inter-connected. In Figure 5, we present the diagram which shows the communication path between two VNFs running on different compute nodes in a standard OpenStack implementation. It can be seen that traffic generated by VNF1 is first passed through Linux bridge named as *qbr04d6a998-b8* and then through integration bridge *br-int* and tunnel bridge *br-eth1* and the same process follows at receiving compute node.

Now, in order to encrypt the traffic flowing between VNF1

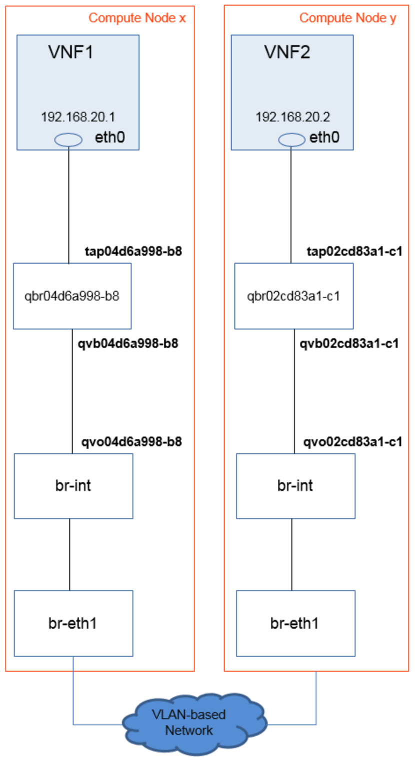

Fig. 5. Standard OpenStack network communication diagram

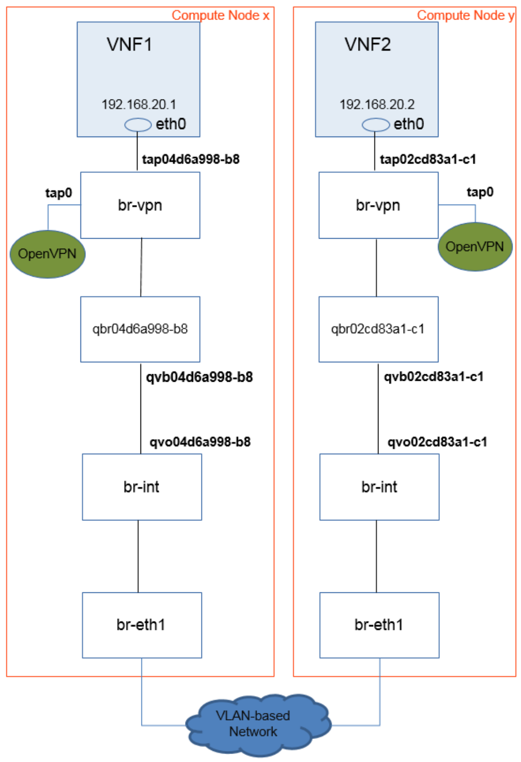

Fig. 6. Modified OpenStack network communication diagram

and VNF2, we need to add one extra bridge for providing interface to OpenVPN tool. We named this bridge as *br-vpn*. Traffic generated from VNF1 is directly feed into the *br-vpn* which then forwards the traffic to OpenVPN tool for encryption. OpenVPN uses cryptographic keys associated with VNF1 to encrypt the traffic and sends it back to *br-vpn* which then forwards the traffic to Linux bridge and the rest of the communication path remains the same as shown in Figure 6. We developed Python based guardian tool for fetching the information about the VNFs running on OpenStack. The collection of information about nodes and interfaces running on compute nodes is a relatively complex process. First, Openstack nova compute is used to check the information about all the VNFs running on each compute node. Then guardian tool checks the interfaces of all the VNFs such as their IP addresses, MAC addresses, and network ID. Using this information, port name and VLAN tag in Open vSwitch instance are resolved. The guardian agent updates itself periodically and this process is repeated for all VNFs running on each compute node. We then build graphical user interface (GUI) for the operator to select the source and destination VNF and their interfaces on which traffic encryption should be performed. Upon pressing the VPN start button in GUI, the interface names are sent to the back end to make the modifications in the existing OpenStack neutron bridges and start OpenVPN tunnel between selected VNFs. Again, the cryptographic keys used in this approach are manually created. This might be sufficient for proof of concept demonstration but in production environment existing key generation and management tools such as PyKIMP [16] could be leveraged.

A. Detecting the encrypted traffic

In order to detect the encrypted traffic between the VNFs, traffic must be captured at either the VNF end or even between VNFs. We used open source traffic capturing tool known as *tcpdump* on compute nodes. For simplicity purpose, we generated ICMP traffic from one VNF using ping command: *"ping -i 0.2 targetIP"*. We observed plain traffic generated from VNFs and flowing between compute nodes.

We then established OpenVPN tunnel between VNF1 and VNF2 from VPN portal and captured the traffic in the similar manner as mentioned earlier. We found that tcpdump did not capture any ICMP traffic. We checked all traffic flowing at *lo* interface at both compute nodes and eventually found the traffic with port number 1194. Port 1194 is the standard port for OpenVPN traffic. The ping traffic is now encrypted and is flowing inside OpenVPN packets.

B. Performance evaluation

Installing security softwares on virtual appliances, designed for standalone physical machines, could lead to performance degradation as these security tools can be CPU-intensive. In one of our proposed approaches, VPN server is virtualized so its actual performance could vary in comparison to running it in a dedicated physical machine [6] [17] [18]. Performance overhead also depends on the application and its sensitivity to the packet delay.

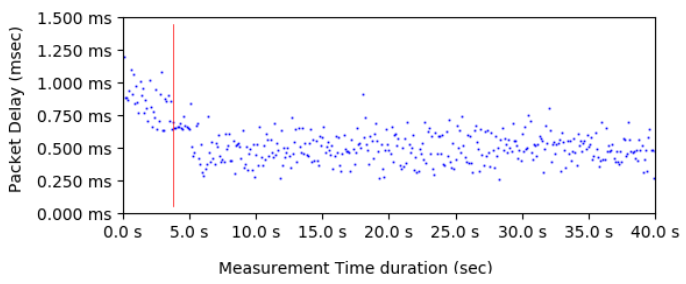

Fig. 7. Packet delay with VPN tunnel insertion

Our solution proposes integrating OpenVPN software tool in OpenStack's compute nodes. When encrypting the data generated by the VNFs using OpenVPN, there are a number of factors which affect the throughputs:

1) number of OpenVPN clients connecting simultaneously
2) bandwidth per client
3) encryption algorithms or ciphers
4) compression
5) packet size
6) transport protocol (TCP/UDP)
7) network errors

In order to measure delay of using encrypted traffic through VPN tunnel, we ran 100 VPN start events and summed the delay time to get the average delay which was about 1 seconds. This is the time taken to setup the br-vpn bridge, mapping the cryptographic keys in OpenVPN tool and also performing the path modifications in OpenStack neutron. The same delay time is observed in tearing down the VPN connection between the VNFs. This 1 second delay seems to be reasonable keeping in mind the number of components that need to be reconfigured in the process. The performance evaluation of OpenVPN solution is already benchmarked and available in [19].

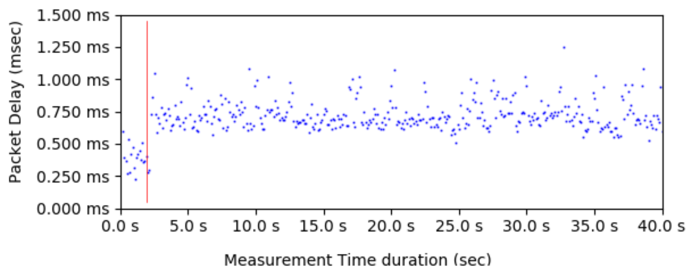

Fig. 8. Packet delay with VPN tunnel removal

Next, we measured the round trip time (RTT) and network throughout between the VNFs before and after establishing the VPN tunnel. The results are shown in Figure 7 and 8 and also in Table I and II. In both figures, the y-axis represents the packet delay and the x-axis shows the measurement time duration and the red line shows the time interval when a VPN tunnel is initiated or teared down. Figure 7 shows the increase of the average packet delay to 0.756 milliseconds when VPN tunnel is established. The average packet delay decreases to 0.429 milliseconds when VPN tunnel is teared down as portrayed in Figure 8. These figures demonstrate that encrypted network traffic comes at the cost of a slight drop in performance. Next, we measured the network throughput impact using the *iperf* tool [20]. One VNF was running iperf as a server and and the other was serving as a client. It can be observed from Table I that traffic latency is higher for the encrypted traffic in comparison to the unencrypted traffic. The same is the case with network throughput in Table II. In our experiment, we noted that while traffic encryption and decryption was taking place, all the computing resources were fully utilized. The openVPN tool was observed in utilizing 100% of the CPU resources when taking the throughput measurements. One way to improve the network throughput is to use crypto accelerators, which perform computationally intensive cryptographic operations.

TABLE I
ROUND TRIP TIME IMPACT

VPN Tunnel	min(ms)	avg(ms)	max(ms)	mdev(ms)
No	0.228ms	0.429ms	1.539ms	0.167ms
Yes	0.510ms	0.756ms	22.905ms	2.813ms

TABLE II
THROUGHPUT IMPACT

VPN Tunnel	Proto	Packet/datagram size	Jitter	Throughput
No	TCP	1470 Bytes	-	15.5 Gbits/sec
No	UDP	1470 Bytes	0.005ms	806 Mbits/sec
Yes	TCP	1470 Bytes	-	585 Mbits/sec
Yes	UDP	1470 Bytes	0.022ms	72.05 Mbits/sec

V. CONCLUSION AND FUTURE WORK

In this paper, we discussed the need for securing the communication among VNFs in telco cloud environment. In this vein, we devised mechanisms to address these problems. We implemented a testbed setup on OpenStack based cloud. In order to integrate our solutions, we modified OpenStack networking component known as neutron in one of our approaches and incorporated open source OpenVPN tool to perform traffic encryption between VNFs.

Furthermore, we evaluated the performance of our testbed setup using metrics such as time delay to setup VPN tunnel between two VNFs and impact of encryption overhead on RTT and throughput. In the performance evaluation, we found that it takes about 1 second on average to setup and also tear down a single VPN tunnel. The RTT and throughput are also affected as encryption comes with cost of performance overhead. We believe that the results provided in this paper could be improved by using crypto accelerators which can speed up the encryption and decryption process.

In the future, we would like to build an efficient cryptographic key generation and distribution mechanism along with optimizing the existing solution. This would help in reducing the delay in setting up VPN tunnel between VNFs and improving the network latency and throughput by using dedicated crypto accelerator devices.

Acknowledgment

This work was funded by Finnish Dimecc Cyber Trust Program and also partially supported by the ANASTACIA project, that has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement N 731558 and from the Swiss State Secretariat for Education, Research and Innovation.

References

[1] T. Taleb, A. Ksentini, and R. Jantti, "Anything as a Service for 5G Mobile Systems", in IEEE Network Magazine, Vol. 30, No. 6, Dec. 2016.
[2] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a Service to Ease Mobile Core Network," in IEEE Network Magazine, Vol. 29, No. 2, Mar. 2015. pp.78 - 88.
[3] T. Taleb, "Towards Carrier Cloud: Potential, Challenges, & Solutions" in IEEE Wireless Communications Magazine, Vol. 21, No. 3, Jun. 2014. pp. 80-91.
[4] S. Ravidas, S. Lal, I. Oliver and L. Hippelainen. "Incorporating trust in NFV: Addressing the challenges." in IEEE conference on Innovations in Clouds, Internet and Networks (ICIN), Mar. 2017.
[5] S. Lal, I. Oliver, S. Ravidas, T. Taleb, "Assuring Virtual Network Function Image Integrity and Host Sealing in Telco Cloud," in Proc. IEEE ICC 2017, Paris, France, May 2017.
[6] CSA Security Guidance Version 3: Virtualization https://cloudsecurityalliance.org/wp-content/uploads/2011/09/Domain-13.docx Accessed: 28-07-2017
[7] F.Z. Yousaf and T. Taleb, "Fine Granular Resource-Aware Virtual Network Function Management for 5G Carrier Cloud," in IEEE Network Magazine, Vol. 30, No. 2, Mar. 2016. pp. 110 - 115.
[8] F.Z. Yousaf, P. Loreiro, F. Zdarsky, T. Taleb, and M. Leibsch, "Cost Analysis of initial deployment strategies of a Virtual Network Infrastructure in a Datacenter," in IEEE Communications Magazine, Vol. 53, No. 12, Dec. 2015, pp. 60 - 66.
[9] S. Lal, T. Taleb, A. Dutta, "NFV: Security Threats and Best Practices", IEEE Communications Magazine. May 2017.
[10] S. Peter, V. Fusenig, V. Souza, M. Melo, P. Murray, H. Debar, H. Medhioub, and D. Zeghlache, "Challenges for Cloud Networking Security." In MONAMI, 2010, pp. 298-313.
[11] OpenStack Docs: Scenario: Classic with Open vSwitch https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html Accessed: 28-07-2017
[12] OpenStack Neutron. http://specs.openstack.org/openstack/neutron-specs/ Accessed: 28-07-2017
[13] Linux Foundation: Open vSwitch http://openvswitch.org Accessed: 28-07-2017
[14] OpenVPN open source VPN https://openvpn.net/ Accessed: 28-07-2017
[15] Libguestfs tools for accessing and modifying virtual machine disk images http://libguestfs.org/ Accessed: 28-07-2017
[16] Open source Python implementation of the KMIP Specification https://github.com/OpenKMIP/PyKMIP Accessed: 28-07-2017
[17] B. Ahmad, T. Taleb, A. Vajda, and M. Bagaa, "Dynamic Cloud Resource Scheduling in Virtualized 5G Mobile Systems," in Proc. IEEE Globecom 2016, Washington, USA, Dec. 2016.
[18] B. Ahmad, A. Vajda, and T. Taleb, "Impact of Network Function Virtualization: A Study based on Real-Life Mobile Network Data," in Proc. IEEE IWCMC 2016, Paphos, Cyprus, Sep. 2016.
[19] OpenVPN Performance Benchmarking https://community.openvpn.net/openvpn/wiki/PerformanceTesting Accessed: 28-07-2017
[20] iPerf. https://iperf.fr/. Accessed: 28-07-2017