Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 5249421, 15 pages
http://dx.doi.org/10.1155/2016/5249421

Research Article

Virtual Networking Performance in OpenStack Platform for Network Function Virtualization

Franco Callegati, Walter Cerroni, and Chiara Contoli

DEI, University of Bologna, Via Venezia 52, 47521 Cesena, Italy

Correspondence should be addressed to Walter Cerroni; walter.cerroni@unibo.it

Received 19 October 2015; Revised 19 January 2016; Accepted 30 March 2016

Academic Editor: Yan Luo

The emerging Network Function Virtualization (NFV) paradigm, coupled with the highly flexible and programmatic control of network devices offered by Software Defined Networking solutions, enables unprecedented levels of network virtualization that will definitely change the shape of future network architectures, where legacy telco central offices will be replaced by cloud data centers located at the edge. On the one hand, this software-centric evolution of telecommunications will allow network operators to take advantage of the increased flexibility and reduced deployment costs typical of cloud computing. On the other hand, it will pose a number of challenges in terms of virtual network performance and customer isolation. This paper intends to provide some insights on how an open-source cloud computing platform such as OpenStack implements multitenant network virtualization and how it can be used to deploy NFV, focusing in particular on packet forwarding performance issues. To this purpose, a set of experiments is presented that refer to a number of scenarios inspired by the cloud computing and NFV paradigms, considering both single tenant and multitenant scenarios. From the results of the evaluation it is possible to highlight potentials and limitations of running NFV on OpenStack.

1. Introduction

Despite the original vision of the Internet as a set of networks interconnected by distributed layer 3 routing nodes, nowadays IP datagrams are not simply forwarded to their final destination based on IP header and next-hop information. A number of so-called *middle-boxes* process IP traffic performing cross layer tasks such as address translation, packet inspection and filtering, QoS management, and load balancing. They represent a significant fraction of network operators' capital and operational expenses. Moreover, they are closed systems, and the deployment of new communication services is strongly dependent on the product capabilities, causing the so-called "vendor lock-in" and Internet "ossification" phenomena [1]. A possible solution to this problem is the adoption of *virtualized* middle-boxes based on open software and hardware solutions. Network virtualization brings great advantages in terms of flexible network management, performed at the software level, and possible coexistence of multiple customers sharing the same physical infrastructure (i.e., *multitenancy*). Network virtualization solutions are already widely deployed at different protocol layers, including Virtual Local Area Networks (VLANs), multilayer Virtual Private Network (VPN) tunnels over public wide-area interconnections, and Overlay Networks [2].

Today the combination of emerging technologies such as *Network Function Virtualization* (NFV) and *Software Defined Networking* (SDN) promises to bring innovation one step further. SDN provides a more flexible and programmatic control of network devices and fosters new forms of virtualization that will definitely change the shape of future network architectures [3], while NFV defines standards to deploy software-based building blocks implementing highly flexible network service chains capable of adapting to the rapidly changing user requirements [4].

As a consequence, it is possible to imagine a medium-term evolution of the network architectures where middle-boxes will turn into virtual machines (VMs) implementing network functions within cloud computing infrastructures, and telco central offices will be replaced by data centers located at the edge of the network [5–7]. Network operators will take advantage of the increased flexibility and reduced

deployment costs typical of the cloud-based approach, paving the way to the upcoming software-centric evolution of telecommunications [8]. However, a number of challenges must be dealt with, in terms of system integration, data center management, and packet processing performance. For instance, if VLANs are used in the physical switches and in the virtual LANs within the cloud infrastructure, a suitable integration is necessary, and the coexistence of different IP virtual networks dedicated to multiple tenants must be seamlessly guaranteed with proper isolation.

Then a few questions are naturally raised: Will cloud computing platforms be actually capable of satisfying the requirements of complex communication environments such as the operators edge networks? Will data centers be able to effectively replace the existing telco infrastructures at the edge? Will virtualized networks provide performance comparable to those achieved with current physical networks, or will they pose significant limitations? Indeed the answer to this question will be a function of the cloud management platform considered. In this work the focus is on OpenStack, which is among the state-of-the-art Linux-based virtualization and cloud management tools. Developed by the open-source software community, OpenStack implements the Infrastructure-as-a-Service (IaaS) paradigm in a multitenant context [9].

To the best of our knowledge, not much work has been reported about the actual performance limits of network virtualization in OpenStack cloud infrastructures under the NFV scenario. Some authors assessed the performance of Linux-based virtual switching [10, 11], while others investigated network performance in public cloud services [12]. Solutions for low-latency SDN implementation on high-performance cloud platforms have also been developed [13]. However, none of the above works specifically deals with NFV scenarios on OpenStack platform. Although some mechanisms for effectively placing virtual network functions within an OpenStack cloud have been presented [14], a detailed analysis of their network performance has not been provided yet.

This paper aims at providing insights on how the OpenStack platform implements multitenant network virtualization, focusing in particular on the performance issues, trying to fill a gap that is starting to get the attention also from the OpenStack developer community [15]. The paper objective is to identify performance bottlenecks in the cloud implementation of the NFV paradigms. An ad hoc set of experiments were designed to evaluate the OpenStack performance under critical load conditions, in both single tenant and multitenant scenarios. The results reported in this work extend the preliminary assessment published in [16, 17].

The paper is structured as follows: the network virtualization concept in cloud computing infrastructures is further elaborated in Section 2; the OpenStack virtual network architecture is illustrated in Section 3; the experimental testbed that we have deployed to assess its performance is presented in Section 4; the results obtained under different scenarios are discussed in Section 5; some conclusions are finally drawn in Section 6.

2. Cloud Network Virtualization

Generally speaking network virtualization is not a new concept. Virtual LANs, Virtual Private Networks, and Overlay Networks are examples of virtualization techniques already widely used in networking, mostly to achieve isolation of traffic flows and/or of whole network sections, either for security or for functional purposes such as traffic engineering and performance optimization [2].

Upon considering cloud computing infrastructures the concept of network virtualization evolves even further. It is not just that some functionalities can be configured in physical devices to obtain some additional functionality in virtual form. In cloud infrastructures whole parts of the network are virtual, implemented with software devices and/or functions running within the servers. This new "softwarized" network implementation scenario allows novel network control and management paradigms. In particular, the synergies between NFV and SDN offer programmatic capabilities that allow easily defining and flexibly managing multiple virtual network slices at levels not achievable before [1].

In cloud networking the typical scenario is a set of VMs dedicated to a given tenant, able to communicate with each other as if connected to the same Local Area Network (LAN), independently of the physical server/servers they are running on. The VMs and LAN of different tenants have to be isolated and should communicate with the outside world only through layer 3 routing and filtering devices. From such requirements stem two major issues to be addressed in cloud networking: (i) *integration* of any set of virtual networks defined in the data center physical switches with the specific virtual network technologies adopted by the hosting servers and (ii) *isolation* among virtual networks that must be logically separated because of being dedicated to different purposes or different customers. Moreover these problems should be solved with performance optimization in mind, for instance, aiming at keeping VMs with intensive exchange of data colocated in the same server, keeping local traffic inside the host and thus reducing the need for external network resources and minimizing the communication latency.

The solution to these issues is usually fully supported by the VM manager (i.e., the *Hypervisor*) running on the hosting servers. Layer 3 routing functions can be executed by taking advantage of lightweight virtualization tools, such as *Linux containers* or *network namespaces*, resulting in isolated virtual networks with dedicated network stacks (e.g., IP routing tables and netfilter flow states) [18]. Similarly layer 2 switching is typically implemented by means of kernel-level virtual bridges/switches interconnecting a VM's virtual interface to a host's physical interface. Moreover the VMs placing algorithms may be designed to take networking issues into account thus optimizing the networking in the cloud together with computation effectiveness [19]. Finally it is worth mentioning that whatever network virtualization technology is adopted within a data center, it should be compatible with SDN-based implementation of the control plane (e.g., OpenFlow) for improved manageability and programmability [20].

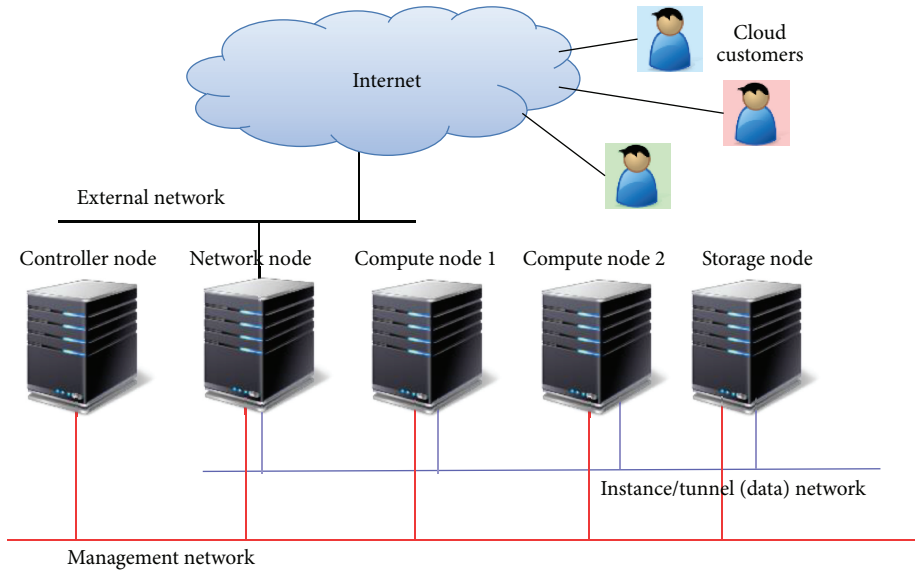

Figure 1: Main components of an OpenStack cloud setup.

For the purposes of this work the implementation of layer 2 connectivity in the cloud environment is of particular relevance. Many Hypervisors running on Linux systems implement the LANs inside the servers using *Linux Bridge*, the native kernel bridging module [21]. This solution is straightforward and is natively integrated with the powerful Linux packet filtering and traffic conditioning kernel functions. The overall performance of this solution should be at a reasonable level when the system is not overloaded [22]. The Linux Bridge basically works as a transparent bridge with MAC learning, providing the same functionality as a standard Ethernet switch in terms of packet forwarding. But such standard behavior is not compatible with SDN and is not flexible enough when aspects such as multitenant traffic isolation, transparent VM mobility, and fine-grained forwarding programmability are critical. The Linux-based bridging alternative is *Open vSwitch* (OVS), a software switching facility specifically designed for virtualized environments and capable of reaching kernel-level performance [23]. OVS is also OpenFlow-enabled and therefore fully compatible and integrated with SDN solutions.

3. OpenStack Virtual Network Infrastructure

OpenStack provides cloud managers with a web-based dashboard as well as a powerful and flexible Application Programmable Interface (API) to control a set of physical hosting servers executing different kinds of Hypervisors (in general, OpenStack is designed to manage a number of computers, hosting application servers: these application servers can be executed by fully fledged VMs, lightweight containers, or bare-metal hosts; in this work we focus on the most challenging case of application servers running on VMs) and to manage the required storage facilities and virtual network infrastructures. The OpenStack dashboard also allows instantiating computing and networking resources within the data center infrastructure with a high level of transparency. As illustrated in Figure 1, a typical OpenStack cloud is composed of a number of physical nodes and networks:

(i) *Controller node*: managing the cloud platform.

(ii) *Network node*: hosting the networking services for the various tenants of the cloud and providing external connectivity.

(iii) *Compute nodes*: as many hosts as needed in the cluster to execute the VMs.

(iv) *Storage nodes*: to store data and VM images.

(v) *Management network*: the physical networking infrastructure used by the controller node to manage the OpenStack cloud services running on the other nodes.

(vi) *Instance/tunnel network* (or *data network*): the physical network infrastructure connecting the network node and the compute nodes, to deploy virtual tenant networks and allow inter-VM traffic exchange and VM connectivity to the cloud networking services running in the network node.

(vii) *External network*: the physical infrastructure enabling connectivity outside the data center.

OpenStack has a component specifically dedicated to network service management: this component, formerly known as Quantum, was renamed as *Neutron* in the Havana release. Neutron decouples the network abstractions from the actual implementation and provides administrators and users with a flexible interface for virtual network management. The Neutron server is centralized and typically runs in the controller node. It stores all network-related information and implements the virtual network infrastructure in a distributed and coordinated way. This allows Neutron to transparently manage multitenant networks across multiple

compute nodes and to provide transparent VM mobility within the data center.

Neutron's main network abstractions are

(i) *network*, a virtual layer 2 segment;

(ii) *subnet*, a layer 3 IP address space used in a network;

(iii) *port*, an attachment point to a network and to one or more subnets on that network;

(iv) *router*, a virtual appliance that performs routing between subnets and address translation;

(v) *DHCP server*, a virtual appliance in charge of IP address distribution;

(vi) *security group*, a set of filtering rules implementing a cloud-level firewall.

A cloud customer wishing to implement a virtual infrastructure in the cloud is considered an OpenStack tenant and can use the OpenStack dashboard to instantiate computing and networking resources, typically creating a new network and the necessary subnets, optionally spawning the related DHCP servers, then starting as many VM instances as required based on a given set of available images, and specifying the subnet (or subnets) to which the VM is connected. Neutron takes care of creating a port on each specified subnet (and its underlying network) and of connecting the VM to that port, while the DHCP service on that network (resident in the network node) assigns a fixed IP address to it. Other virtual appliances (e.g., routers providing global connectivity) can be implemented directly in the cloud platform, by means of containers and network namespaces typically defined in the network node. The different tenant networks are isolated by means of VLANs and network namespaces, whereas the security groups protect the VMs from external attacks or unauthorized access. When some VM instances offer services that must be reachable by external users, the cloud provider defines a pool of floating IP addresses on the external network and configures the network node with VM-specific forwarding rules based on those floating addresses.

OpenStack implements the virtual network infrastructure (VNI) exploiting multiple virtual bridges connecting virtual and/or physical interfaces that may reside in different network namespaces. To better understand such a complex system, a graphical tool was developed to display all the network elements used by OpenStack [24]. Two examples, showing the internal state of a network node connected to three virtual subnets and a compute node running two VMs, are displayed in Figures 2 and 3, respectively.

Each node runs OVS-based integration bridge named *br-int* and, connected to it, an additional OVS bridge for each data center physical network attached to the node. So the network node (Figure 2) includes *br-tun* for the instance/tunnel network and *br-ex* for the external network. A compute node (Figure 3) includes *br-tun* only.

Layer 2 virtualization and multitenant isolation on the physical network can be implemented using either VLANs or layer 2-in-layer 3/4 tunneling solutions, such as Virtual eXtensible LAN (VXLAN) or Generic Routing Encapsulation (GRE), which allow extending the local virtual networks also to remote data centers [25]. The examples shown in Figures 2 and 3 refer to the case of tenant isolation implemented with GRE tunnels on the instance/tunnel network. Whatever virtualization technology is used in the physical network, its virtual networks must be mapped into the VLANs used internally by Neutron to achieve isolation. This is performed by taking advantage of the programmable features available in OVS through the insertion of appropriate OpenFlow mapping rules in *br-int* and *br-tun*.

Virtual bridges are interconnected by means of either virtual Ethernet (*veth*) pairs or *patch port* pairs, consisting of two virtual interfaces that act as the endpoints of a pipe: anything entering one endpoint always comes out on the other side.

From the networking point of view the creation of a new VM instance involves the following steps:

(i) The OpenStack scheduler component running in the controller node chooses the compute node that will host the VM.

(ii) A *tap interface* is created for each VM network interface to connect it to the Linux kernel.

(iii) A Linux Bridge dedicated to each VM network interface is created (in Figure 3 two of them are shown) and the corresponding tap interface is attached to it.

(iv) A veth pair connecting the new Linux Bridge to the integration bridge is created.

The veth pair clearly emulates the Ethernet cable that would connect the two bridges in real life. Nonetheless, why the new Linux Bridge is needed is not intuitive, as the VM's tap interface could be directly attached to *br-int*. In short, the reason is that the antispoofing rules currently implemented by Neutron adopt the native Linux kernel filtering functions (netfilter) applied to bridged tap interfaces, which work only under Linux Bridges. Therefore, the Linux Bridge is required as an intermediate element to interconnect the VM to the integration bridge. The security rules are applied to the Linux Bridge on the tap interface that connects the kernel-level bridge to the virtual Ethernet port of the VM running in user-space.

4. Experimental Setup

The previous section makes the complexity of the OpenStack virtual network infrastructure clear. To understand optimal design strategies in terms of network performance it is of great importance to analyze it under critical traffic conditions and assess the maximum sustainable packet rate under different application scenarios. The goal is to isolate as much as possible the level of performance of the main OpenStack network components and determine where the bottlenecks are located, speculating on possible improvements. To this purpose, a test-bed including a controller node, one or two compute nodes (depending on the specific experiment), and a network node was deployed and used to obtain the results presented in the following. In the test-bed each compute node runs KVM, the native Linux VM Hypervisor, and is equipped

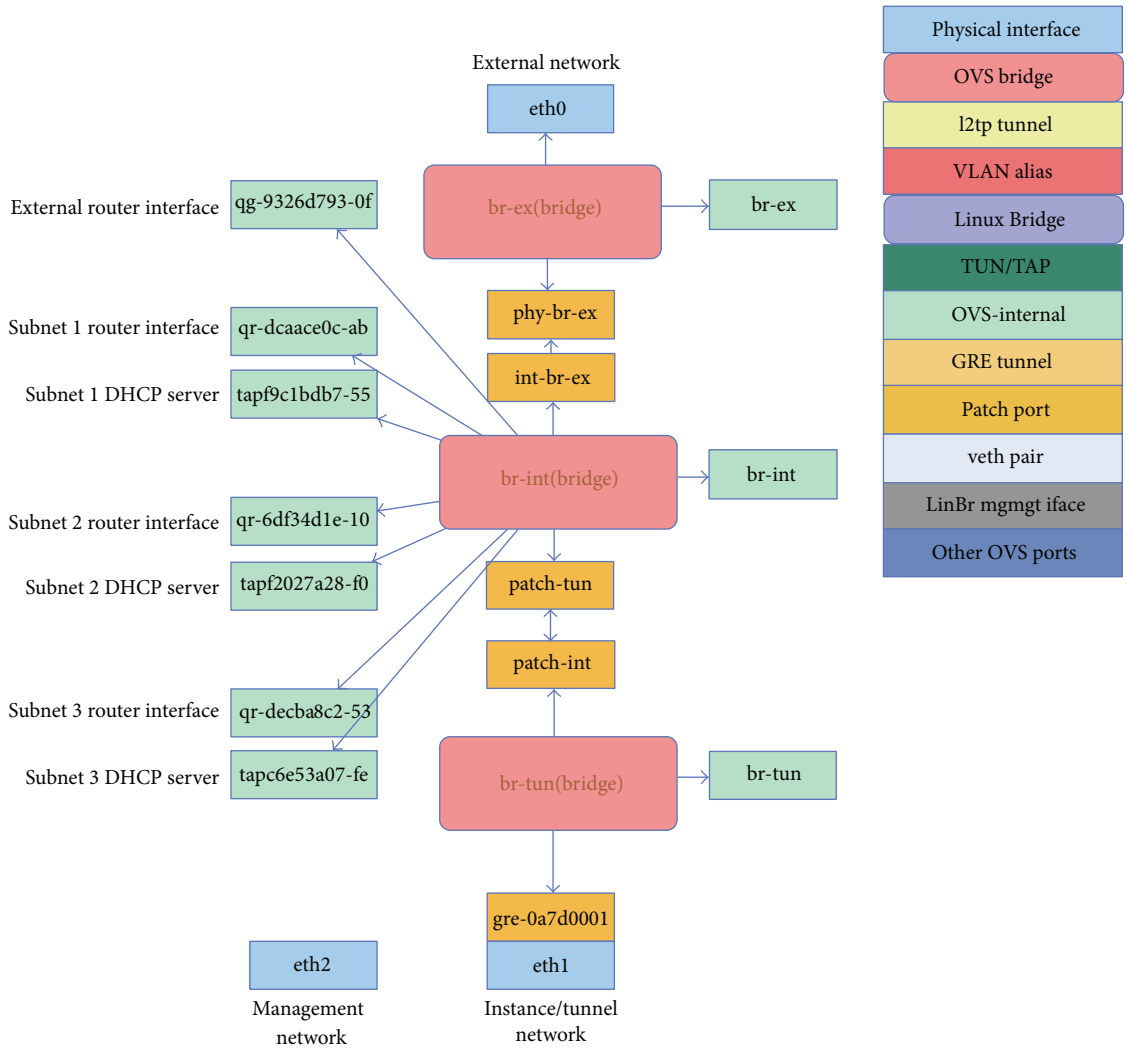

Figure 2: Network elements in an OpenStack network node connected to three virtual subnets. Three OVS bridges (red boxes) are interconnected by patch port pairs (orange boxes). *br-ex* is directly attached to the external network physical interface (*eth0*), whereas GRE tunnel is established on the instance/tunnel network physical interface (*eth1*) to connect *br-tun* with its counterpart in the compute node. A number of *br-int* ports (light-green boxes) are connected to four virtual router interfaces and three DHCP servers. An additional physical interface (*eth2*) connects the network node to the management network.

with 8 GB of RAM and a quad-core processor enabled to hyperthreading, resulting in 8 virtual CPUs.

The test-bed was configured to implement three possible use cases:

(1) A typical single tenant cloud computing scenario.

(2) A multitenant NFV scenario with dedicated network functions.

(3) A multitenant NFV scenario with shared network functions.

For each use case multiple experiments were executed as reported in the following. In the various experiments typically a traffic source sends packets at increasing rate to a destination that measures the received packet rate and throughput. To this purpose the *RUDE & CRUDE* tool was used, for both traffic generation and measurement [26]. In some cases, the *Iperf3* tool was also added to generate background traffic at a fixed data rate [27]. All physical interfaces involved in the experiments were Gigabit Ethernet network cards.

4.1. Single Tenant Cloud Computing Scenario. This is the typical configuration where a single tenant runs one or multiple VMs that exchange traffic with one another in the cloud or with an external host, as shown in Figure 4. This is a rather trivial case of limited general interest but is useful to assess some basic concepts and pave the way to the deeper analysis developed in the second part of this section. In the experiments reported, as mentioned above, the virtualization Hypervisor was always KVM. A scenario with OpenStack running the cloud environment and a scenario without OpenStack were considered to assess some general

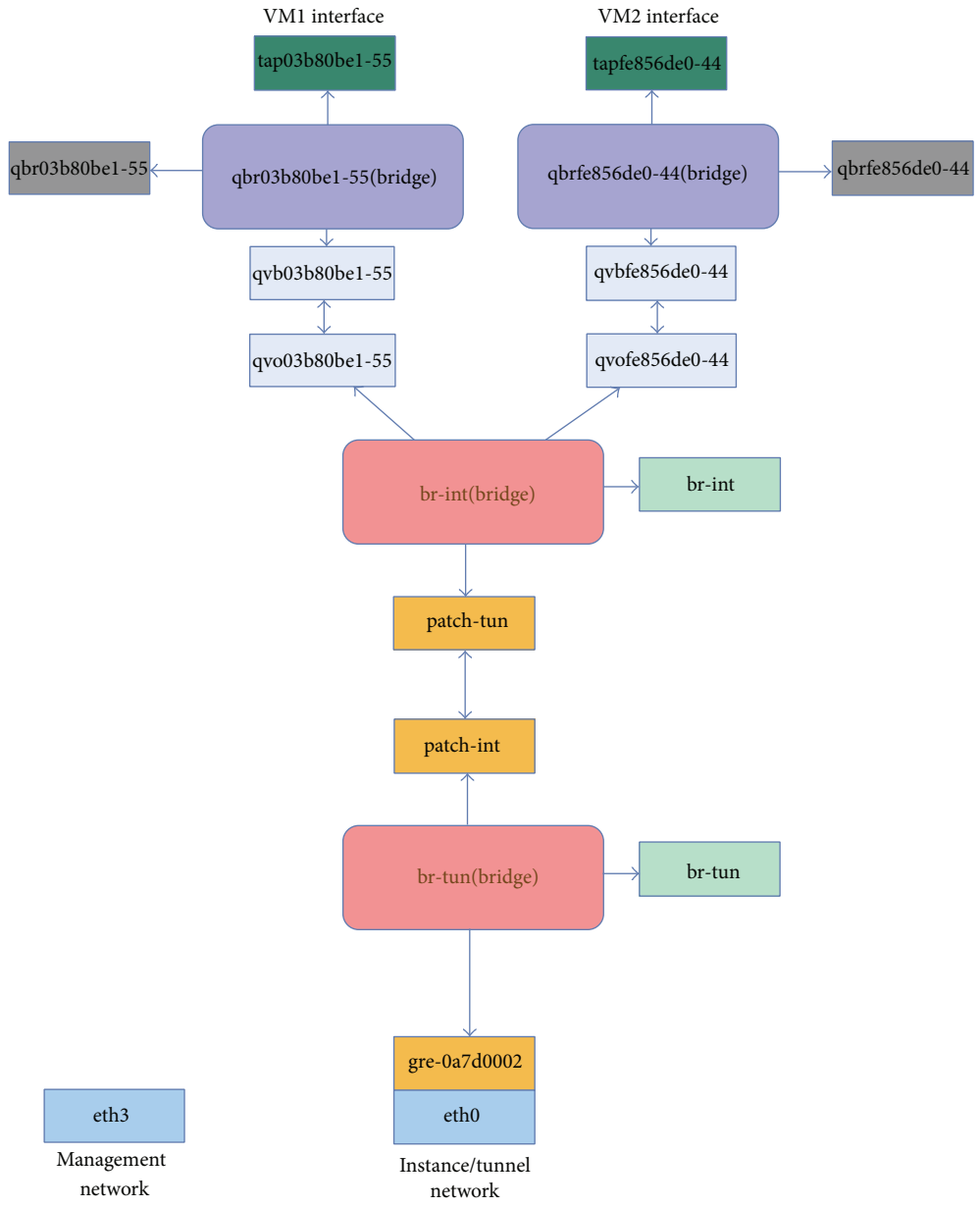

Figure 3: Network elements in an OpenStack compute node running two VMs. Two Linux Bridges (blue boxes) are attached to the VM tap interfaces (green boxes) and connected by virtual Ethernet pairs (light-blue boxes) to *br-int*.

comparison and allow a first isolation of the performance degradation due to the individual building blocks, in particular Linux Bridge and OVS. The experiments report the following cases:

(1) *OpenStack scenario*: it adopts the standard OpenStack cloud platform, as described in the previous section, with two VMs, respectively, acting as sender and receiver. In particular, the following setups were tested:

(1.1) A single compute node executing two colocated VMs.

(1.2) Two distinct compute nodes, each executing a VM.

(2) *Non-OpenStack scenario*: it adopts physical hosts running Linux-Ubuntu server and KVM Hypervisor, using either OVS or Linux Bridge as a virtual switch. The following setups were tested:

(2.1) One physical host executing two colocated VMs, acting as sender and receiver and directly connected to the same Linux Bridge.

(2.2) The same setup as the previous one, but with OVS bridge instead of a Linux Bridge.

(2.3) Two physical hosts: one executing the sender VM connected to an internal OVS and the other natively acting as the receiver.

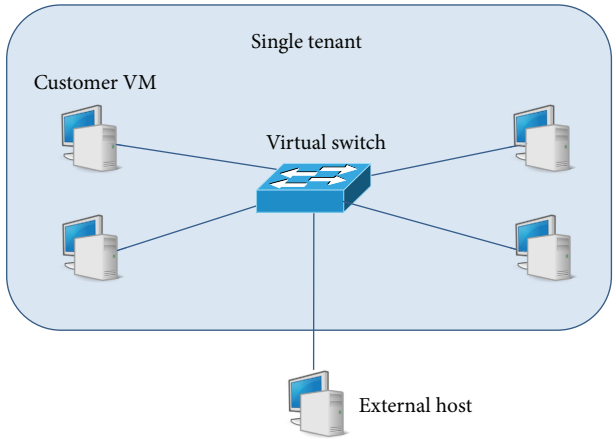

Figure 4: Reference logical architecture of a single tenant virtual infrastructure with 5 hosts: 4 hosts are implemented as VMs in the cloud and are interconnected via the OpenStack layer 2 virtual infrastructure; the 5th host is implemented by a physical machine placed outside the cloud but still connected to the same logical LAN.

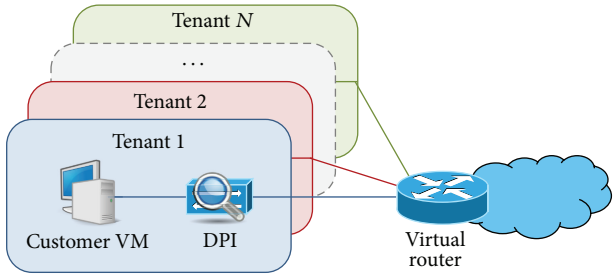

Figure 5: Multitenant NFV scenario with dedicated network functions tested on the OpenStack platform.

4.2. Multitenant NFV Scenario with Dedicated Network Functions. The multitenant scenario we want to analyze is inspired by a simple NFV case study, as illustrated in Figure 5: each tenant's service chain consists of a customer-controlled VM followed by a dedicated deep packet inspection (DPI) virtual appliance and a conventional gateway (router) connecting the customer LAN to the public Internet. The DPI is deployed by the service operator as a separate VM with two network interfaces, running a traffic monitoring application based on the nDPI library [28]. It is assumed that the DPI analyzes the traffic profile of the customers (source and destination IP addresses and ports, application protocol, etc.) to guarantee the matching with the customer service level agreement (SLA), a practice that is rather common among Internet service providers to enforce network security and traffic policing. The virtualization approach executing the DPI in a VM makes it possible to easily configure and adapt the inspection function to the specific tenant characteristics. For this reason every tenant has its own DPI with dedicated configuration. On the other hand the gateway has to implement a standard functionality and is shared among customers. It is implemented as a virtual router for packet forwarding and NAT operations.

The implementation of the test scenarios has been done following the OpenStack architecture. The compute nodes of the cluster run the VMs, while the network node runs the virtual router within a dedicated network namespace. All layer 2 connections are implemented by a virtual switch (with proper VLAN isolation) distributed in both the compute and network nodes. Figure 6 shows the view provided by the OpenStack dashboard, in the case of 4 tenants simultaneously active, which is the one considered for the numerical results presented in the following. The choice of 4 tenants was made to provide meaningful results with an acceptable degree of complexity, without lack of generality. As results show this is enough to put the hardware resources of the compute node under stress and therefore evaluate performance limits and critical issues.

It is very important to outline that the VM setup shown in Figure 5 is not commonly seen in a traditional cloud computing environment. The VMs usually behave as single hosts connected as endpoints to one or more virtual networks, with one single network interface and no pass-through forwarding duties. In NFV the virtual network functions (VNFs) often perform actions that require packet forwarding. Network Address Translators (NATs), Deep Packet Inspectors (DPIs), and so forth all belong to this category. If such VNFs are hosted in VMs the result is that VMs in the OpenStack infrastructure must be allowed to perform packet forwarding which goes against the typical rules implemented for security reasons in OpenStack. For instance, when a new VM is instantiated it is attached to a Linux Bridge to which filtering rules are applied with the goal of avoiding that the VM sends packet with MAC and IP addresses that are not the ones allocated to the VM itself. Clearly this is an antispoofing rule that makes perfect sense in a normal networking environment but impairs the forwarding of packets originated by another VM as is the case of the NFV scenario. In the scenario considered here, it was therefore necessary to permanently modify the filtering rules in the Linux Bridges, by allowing, within each tenant slice, packets coming from or directed to the customer VM's IP address to pass through the Linux Bridges attached to the DPI virtual appliance. Similarly the virtual router is usually connected just to one LAN. Therefore its NAT function is configured for a single pool of addresses. This was also modified and adapted to serve the whole set of internal networks used in the multitenant setup.

4.3. Multitenant NFV Scenario with Shared Network Functions. We finally extend our analysis to a set of multitenant scenarios assuming different levels of shared VNFs, as illustrated in Figure 7. We start with a single VNF, that is, the virtual router connecting all tenants to the external network (Figure 7(a)). Then we progressively add a shared DPI (Figure 7(b)), a shared firewall/NAT function (Figure 7(c)), and a shared traffic shaper (Figure 7(d)). The rationale behind this last group of setups is to evaluate how NFV deployment on top of an OpenStack compute node performs under a realistic multitenant scenario where traffic flows must be processed by a chain of multiple VNFs. The complexity of the

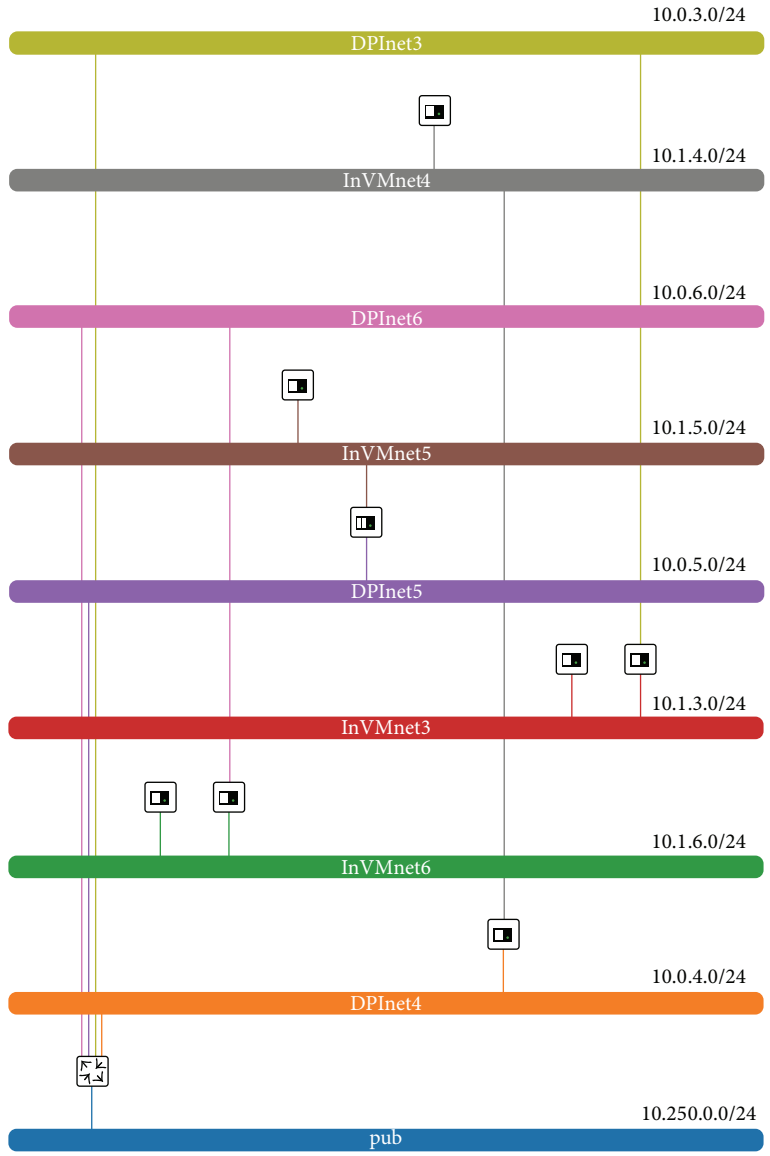

Figure 6: The OpenStack dashboard shows the tenants virtual networks (slices). Each slice includes VM connected to an internal network (InVMnet*i*) and a second VM performing DPI and packet forwarding between InVMnet*i* and DPInet*i*. Connectivity with the public Internet is provided for all by the virtual router in the bottom-left corner.

virtual network path inside the compute node for the VNF chaining of Figure 7(d) is displayed in Figure 8. The peculiar nature of NFV traffic flows is clearly shown in the figure, where packets are being forwarded multiple times across *br-int* as they enter and exit the multiple VNFs running in the compute node.

5. Numerical Results

5.1. Benchmark Performance. Before presenting and discussing the performance of the study scenarios described above, it is important to set some benchmark as a reference for comparison. This was done by considering a back-to-back (B2B) connection between two physical hosts, with the same hardware configuration used in the cluster of the cloud platform.

The former host acts as traffic generator while the latter acts as traffic sink. The aim is to verify and assess the maximum throughput and sustainable packet rate of the hardware platform used for the experiments. Packet flows ranging from 10^3 to 10^5 packets per second (pps), for both 64- and 1500-byte IP packet sizes, were generated.

For 1500-byte packets, the throughput saturates to about 970 Mbps at 80 Kpps. Given that the measurement does not consider the Ethernet overhead, this limit is clearly very close to the 1 Gbps which is the physical limit of the Ethernet interface. For 64-byte packets, the results are different since the maximum measured throughput is about 150 Mbps. Therefore the limiting factor is not the Ethernet bandwidth

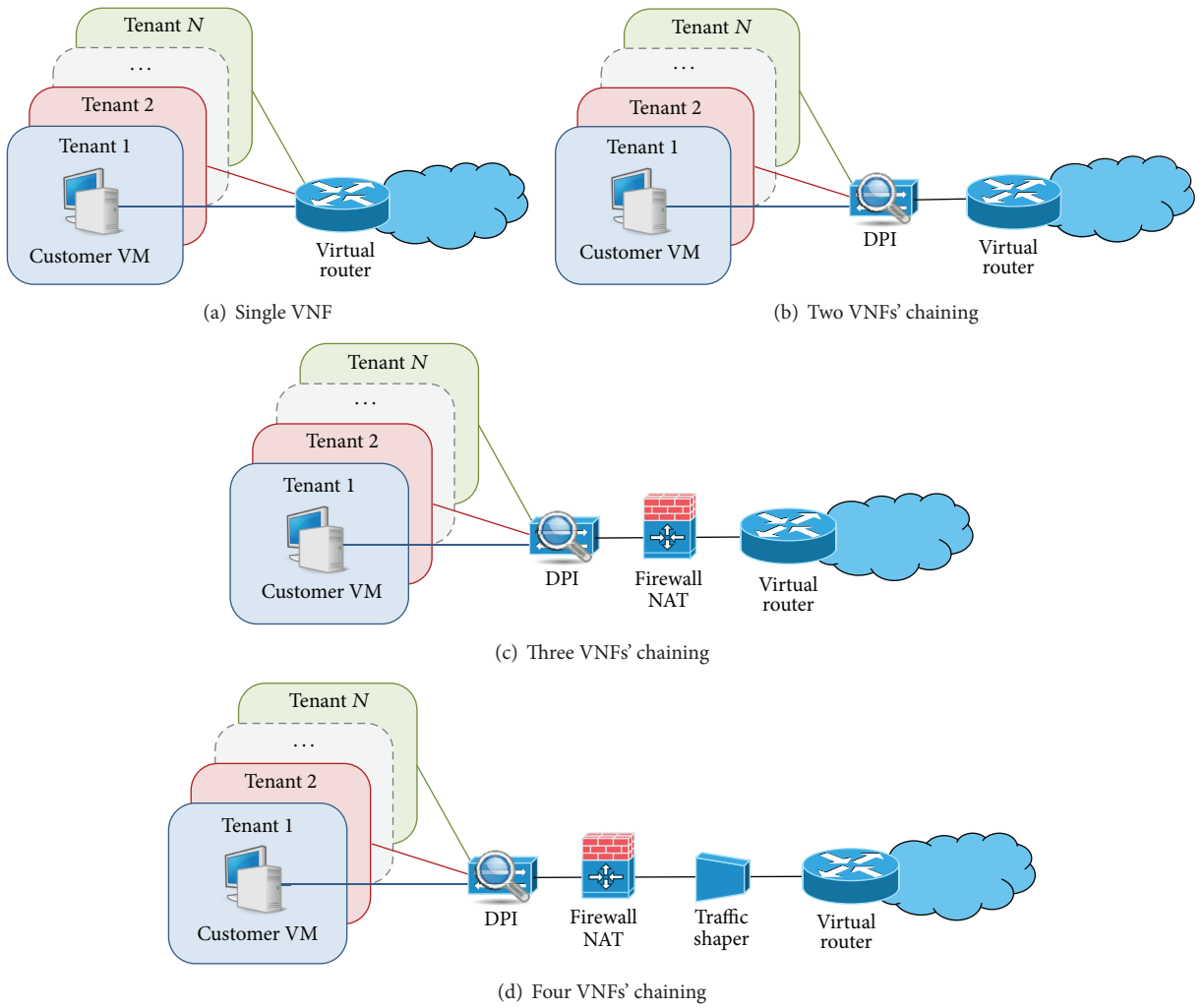

FIGURE 7: Multitenant NFV scenario with shared network functions tested on the OpenStack platform.

but the maximum sustainable packet processing rate of the computer node. These results are shown in Figure 9.

This latter limitation, related to the processing capabilities of the hosts, is not very relevant to the scopes of this work. Indeed it is always possible, in a real operation environment, to deploy more powerful and better dimensioned hardware. This was not possible in this set of experiments where the cloud cluster was an existing research infrastructure which could not be modified at will. Nonetheless the objective here is to understand the limitations that emerge as a consequence of the networking architecture, resulting from the deployment of the VNFs in the cloud, and not of the specific hardware configuration. For these reasons as well as for the sake of brevity, the numerical results presented in the following mostly focus on the case of 1500-byte packet length, which will stress the network more than the hosts in terms of performance.

5.2. Single Tenant Cloud Computing Scenario. The first series of results is related to the single tenant scenario described in Section 4.1. Figure 10 shows the comparison of OpenStack setups (1.1) and (1.2) with the B2B case. The figure shows that the different networking configurations play a crucial role in performance. Setup (1.1) with the two VMs colocated in the same compute node clearly is more demanding since the compute node has to process the workload of all the components shown in Figure 3, that is, packet generation and reception in two VMs and layer 2 switching in two Linux Bridges and two OVS bridges (as a matter of fact the packets are both outgoing and incoming at the same time within the same physical machine). The performance starts deviating from the B2B case at around 20 Kpps, with a saturating effect starting at 30 Kpps. This is the maximum packet processing capability of the compute node, regardless of the physical networking capacity, which is not fully exploited in this particular scenario where the traffic flow does not leave the physical host. Setup (1.2) splits the workload over two physical machines and the benefit is evident. The performance is almost ideal, with a very little penalty due to the virtualization overhead.

These very simple experiments lead to an important conclusion that motivates the more complex experiments

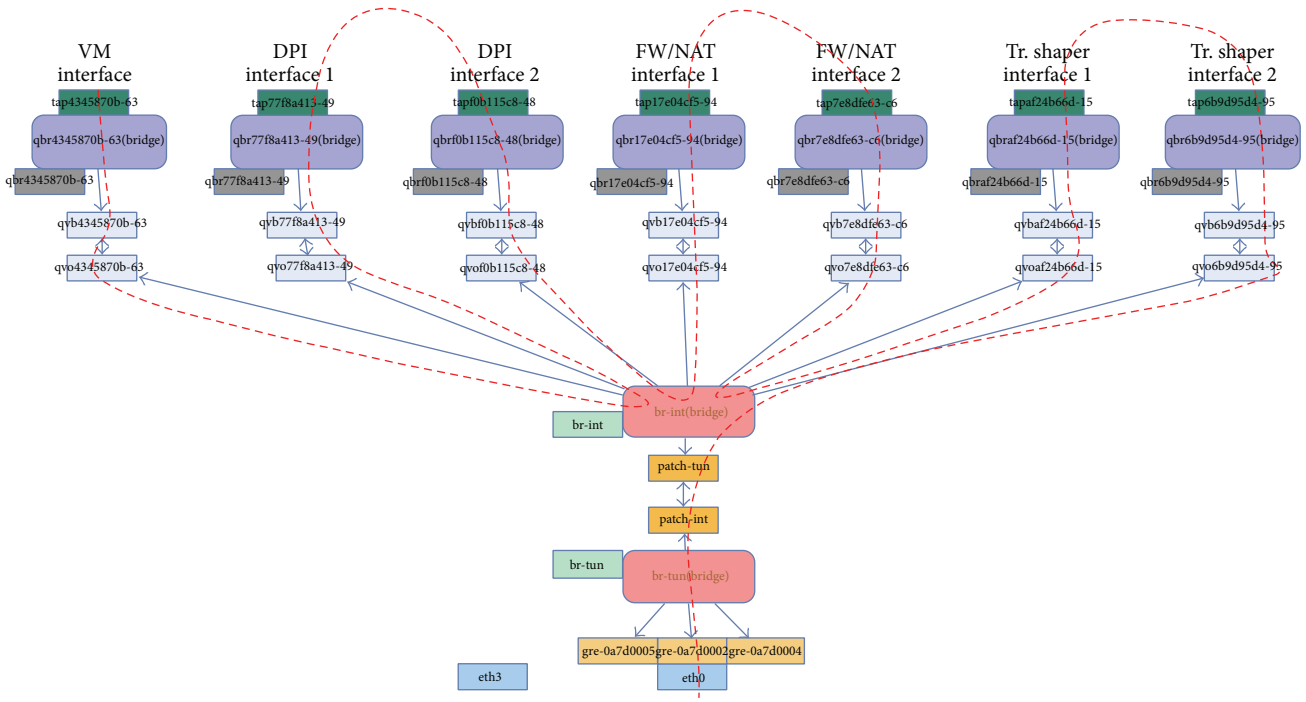

Figure 8: A view of the OpenStack compute node with the tenant VM and the VNFs installed including the building blocks of the virtual network infrastructure. The red dashed line shows the path followed by the packets traversing the VNF chain displayed in Figure 7(d).

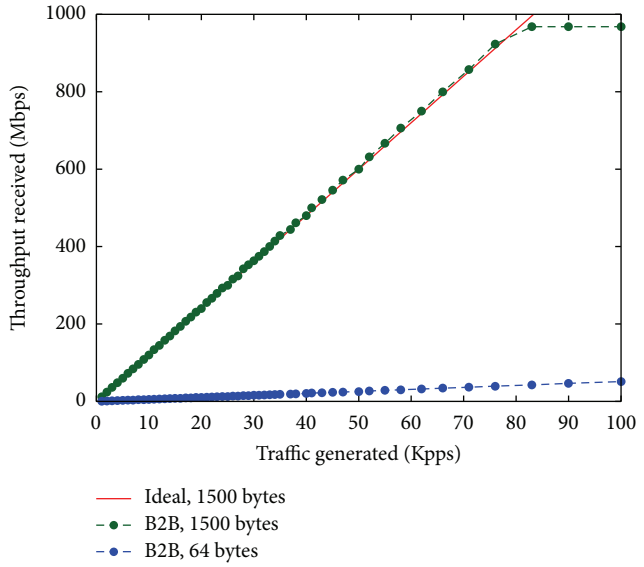

Figure 9: Throughput versus generated packet rate in the B2B setup for 64- and 1500-byte packets. Comparison with ideal 1500-byte packet throughput.

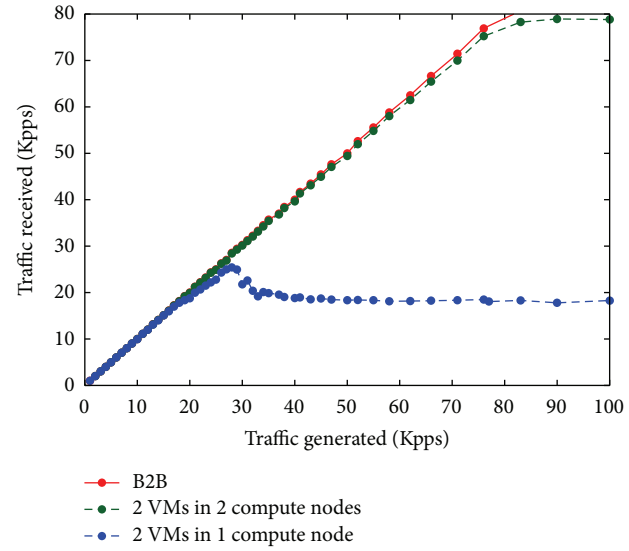

Figure 10: Received versus generated packet rate in the OpenStack scenario setups (1.1) and (1.2), with 1500-byte packets.

that follow: the standard OpenStack virtual network implementation can show significant performance limitations. For this reason the first objective was to investigate where the possible bottleneck is, by evaluating the performance of the virtual network components in isolation. This cannot be done with OpenStack in action; therefore ad hoc virtual networking scenarios were implemented deploying just parts of the typical OpenStack infrastructure. These are called Non-OpenStack scenarios in the following.

Setups (2.1) and (2.2) compare Linux Bridge, OVS, and B2B, as shown in Figure 11. The graphs show interesting and important results that can be summarized as follows:

(i) The introduction of some virtual network component (thus introducing the processing load of the physical

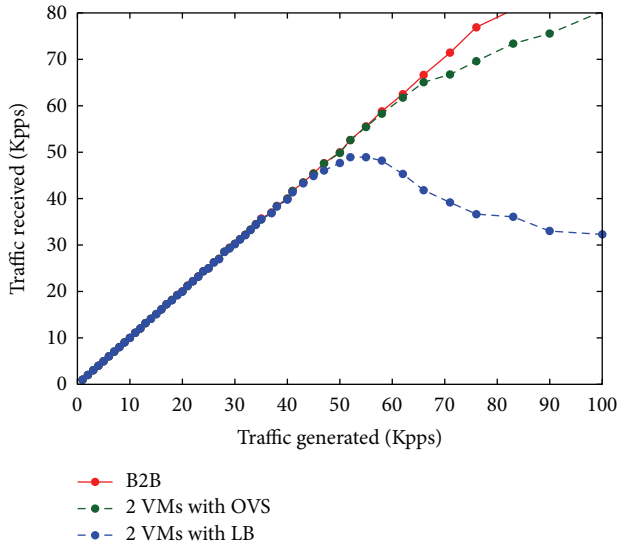

Figure 11: Received versus generated packet rate in the Non-OpenStack scenario setups (2.1) and (2.2), with 1500-byte packets.

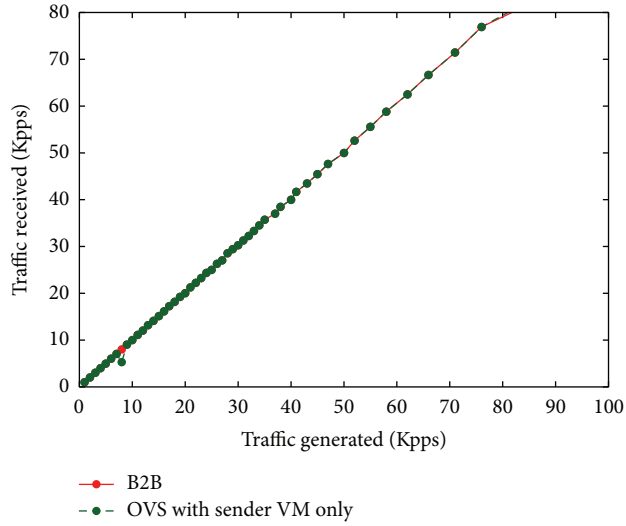

Figure 12: Received versus generated packet rate in the Non-OpenStack scenario setup (2.3), with 1500-byte packets.

hosts in the equation) is always a cause of performance degradation but with very different degrees of magnitude depending on the virtual network component.

(ii) OVS introduces a rather limited performance degradation at very high packet rate with a loss of some percent.

(iii) Linux Bridge introduces a significant performance degradation starting well before the OVS case and leading to a loss in throughput as high as 50%.

The conclusion of these experiments is that the presence of additional Linux Bridges in the compute nodes is one of the main reasons for the OpenStack performance degradation. Results obtained from testing setup (2.3) are displayed in Figure 12 confirming that with OVS it is possible to reach performance comparable with the baseline.

5.3. Multitenant NFV Scenario with Dedicated Network Functions. The second series of experiments was performed with reference to the multitenant NFV scenario with dedicated network functions described in Section 4.2. The case study considers that different numbers of tenants are hosted in the same compute node, sending data to a destination outside the LAN, therefore beyond the virtual gateway. Figure 13 shows the packet rate actually received at the destination for each tenant, for different numbers of simultaneously active tenants with 1500-byte IP packet size. In all cases the tenants generate the same amount of traffic, resulting in as many overlapping curves as the number of active tenants. All curves grow linearly as long as the generated traffic is sustainable, and then they saturate. The saturation is caused by the physical bandwidth limit imposed by the Gigabit Ethernet interfaces involved in the data transfer. In fact, the curves become flat as soon as the packet rate reaches about 80 Kpps for 1 tenant, about 40 Kpps for 2 tenants, about 27 Kpps for 3 tenants, and

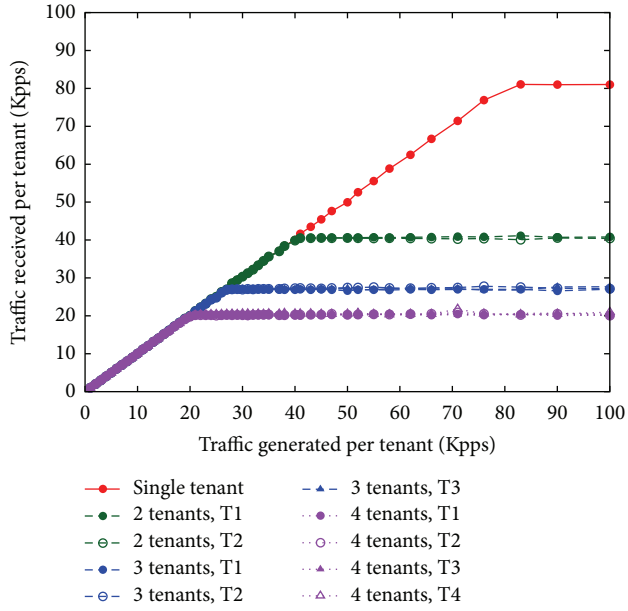

Figure 13: Received versus generated packet rate for each tenant (T1, T2, T3, and T4), for different numbers of active tenants, with 1500-byte IP packet size.

about 20 Kpps for 4 tenants, that is, when the total packet rate is slightly more than 80 Kpps, corresponding to 1 Gbps.

In this case it is worth investigating what happens for small packets, therefore putting more pressure on the processing capabilities of the compute node. Figure 14 reports the 64-byte packet size case. As discussed previously in this case the performance saturation *is not caused by the physical bandwidth limit, but by the inability of the hardware platform to cope with the packet processing workload* (in fact the single compute node has to process the workload of all the components involved, including packet generation and DPI in the VMs of each tenant, as well as layer 2 packet

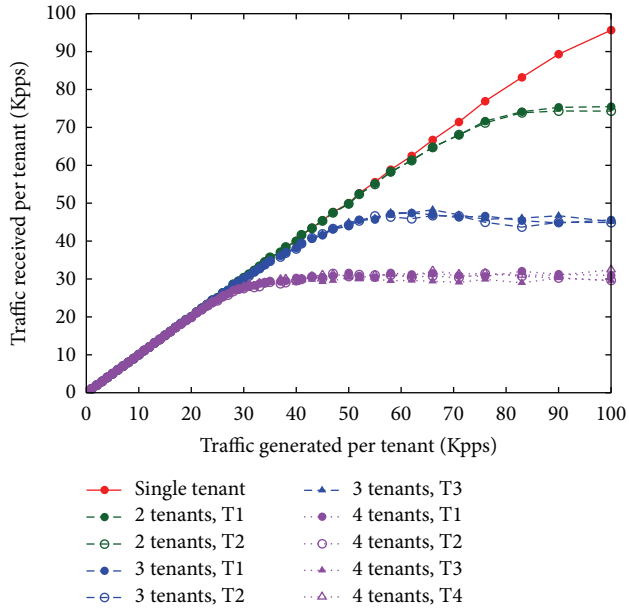

FIGURE 14: Received versus generated packet rate for each tenant (T1, T2, T3, and T4), for different numbers of active tenants, with 64-byte IP packet size.

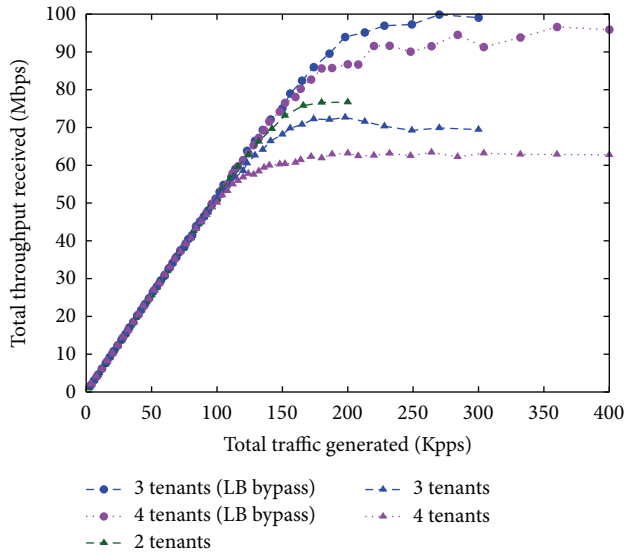

FIGURE 15: Total throughput measured versus total packet rate generated by 2 to 4 tenants for 64-byte packet size. Comparison between normal OpenStack mode and Linux Bridge bypass with 3 and 4 tenants.

processing and switching in three Linux Bridges per tenant and two OVS bridges). As could be easily expected from the results presented in Figure 9, the virtual network is not able to use the whole physical capacity. Even in the case of just one tenant, a total bit rate of about 77 Mbps, well below 1 Gbps, is measured. Moreover this penalty increases with the number of tenants (i.e., with the complexity of the virtual system). With two tenants the curve saturates at a total of approximately 150 Kpps (75 × 2), with three tenants at a total of approximately 135 Kpps (45 × 3), and with four tenants at a total of approximately 120 Kpps (30 × 4). This is to say that an increase of one unit in the number of tenants results in a decrease of about 10% in the usable overall network capacity and in a similar penalty per tenant.

Given the results of the previous section, it is likely that the Linux Bridges are responsible for most of this performance degradation. In Figure 15 a comparison is presented between the total throughput obtained under normal OpenStack operations and the corresponding total throughput measured in a custom configuration where the Linux Bridges attached to each VM are bypassed. To implement the latter scenario, the OpenStack virtual network configuration running in the compute node was modified by connecting each VM's tap interface directly to the OVS integration bridge. The curves show that the presence of Linux Bridges in normal OpenStack mode is indeed causing performance degradation, especially when the workload is high (i.e., with 4 tenants). It is interesting to note also that the penalty related to the number of tenants is mitigated by the bypass, but not fully solved.

5.4. Multitenant NFV Scenario with Shared Network Functions. The third series of experiments was performed with

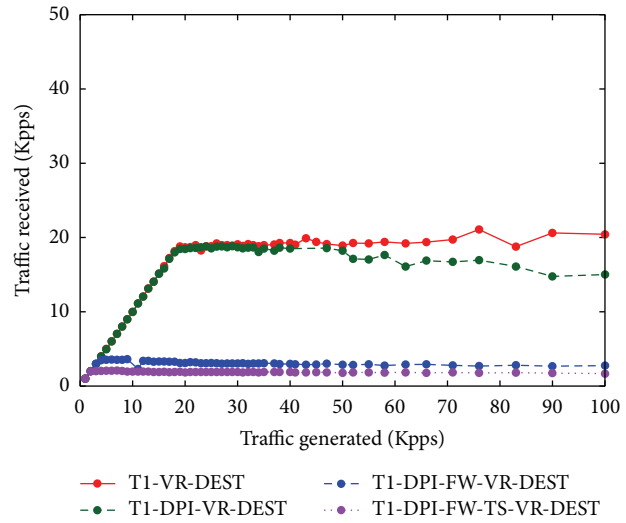

FIGURE 16: Received versus generated packet rate for one tenant (T1) when four tenants are active, with 1500-byte IP packet size and different levels of VNF chaining as per Figure 7. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

reference to the multitenant NFV scenario with shared network functions described in Section 4.3. In each experiment, four tenants are equally generating increasing amounts of traffic, ranging from 1 to 100 Kpps. Figures 16 and 17 show the packet rate actually received at the destination from tenant T1 as a function of the packet rate generated by T1, for different levels of VNF chaining, with 1500- and 64-byte IP packet size, respectively. The measurements demonstrate that, for the 1500-byte case, adding a single shared VNF (even one that executes heavy packet processing, such as the DPI)

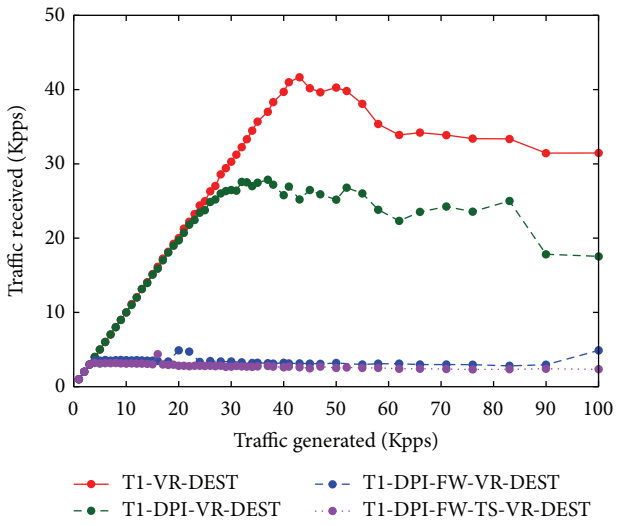

Figure 17: Received versus generated packet rate for one tenant (T1) when four tenants are active, with 64-byte IP packet size and different levels of VNF chaining as per Figure 7. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

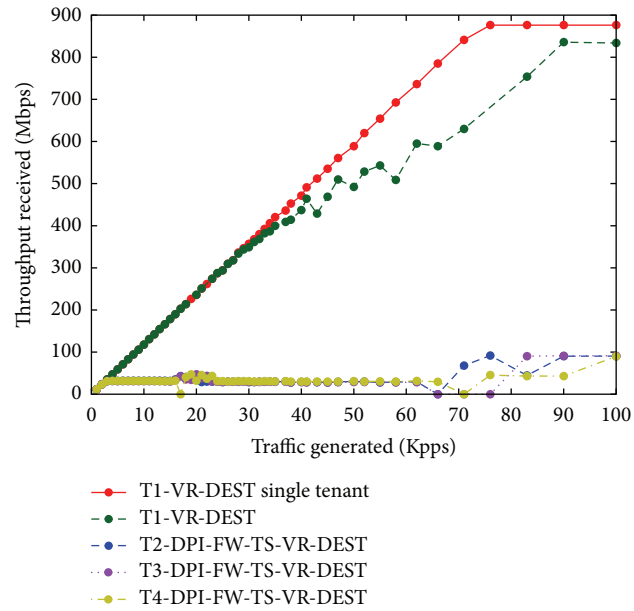

Figure 18: Received throughput versus generated packet rate for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d), with 1500-byte IP packet size. Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

does not significantly impact the forwarding performance of the OpenStack compute node for a packet rate below 50 Kpps (note that the physical capacity is saturated by the flows simultaneously generated from four tenants at around 20 Kpps, similarly to what happens in the dedicated VNF case of Figure 13). Then the throughput slowly degrades. In contrast, when 64-byte packets are generated, even a single VNF can cause heavy performance losses above 25 Kpps, when the packet rate reaches the sustainability limit of the forwarding capacity of our compute node. Independently of the packet size, adding another VNF with heavy packet processing (the firewall/NAT is configured with 40,000 matching rules) causes the performance to rapidly degrade. This is confirmed when a fourth VNF is added to the chain, although for the 1500-byte case the measured packet rate is the one that saturates the maximum bandwidth made available by the traffic shaper. Very similar performance, which we do not show here, was measured also for the other three tenants.

To further investigate the effect of VNF chaining, we considered the case when traffic generated by tenant T1 is not subject to VNF chaining (as in Figure 7(a)), whereas flows originated from T2, T3, and T4 are processed by four VNFs (as in Figure 7(d)). The results presented in Figures 18 and 19 demonstrate that, owing to the traffic shaping function applied to the other tenants, the throughput of T1 can reach values not very far from the case when it is the only active tenant, especially for packet rates below 35 Kpps. Therefore, a smart choice of the VNF chaining and a careful planning of the cloud platform resources could improve the performance of a given class of priority customers. In the same situation, we measured the TCP throughput achievable by the four tenants. As shown in Figure 20, we can reach the same conclusions as in the UDP case.

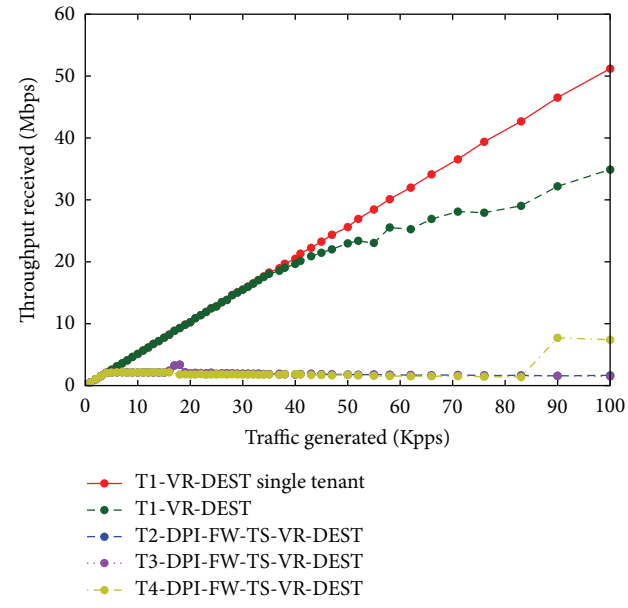

Figure 19: Received throughput versus generated packet rate for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d), with 64-byte IP packet size. Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

6. Conclusion

Network Function Virtualization will completely reshape the approach of telco operators to provide existing as well as novel network services, taking advantage of the increased

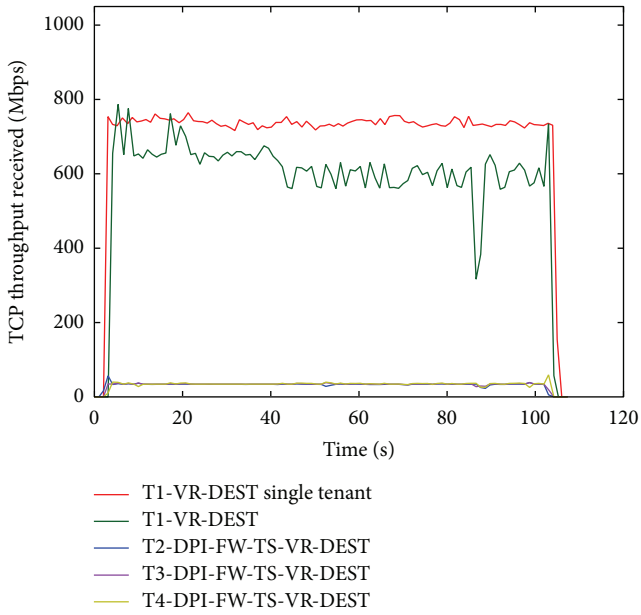

Figure 20: Received TCP throughput for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d). Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

flexibility and reduced deployment costs of the cloud computing paradigm. In this work, the problem of evaluating complexity and performance, in terms of sustainable packet rate, of virtual networking in cloud computing infrastructures dedicated to NFV deployment was addressed. An OpenStack-based cloud platform was considered and deeply analyzed to fully understand the architecture of its virtual network infrastructure. To this end, an ad hoc visual tool was also developed that graphically plots the different functional blocks (and related interconnections) put in place by Neutron, the OpenStack networking service. Some examples were provided in the paper.

The analysis brought the focus of the performance investigation on the two basic software switching elements natively adopted by OpenStack, namely, Linux Bridge and Open vSwitch. Their performance was first analyzed in a single tenant cloud computing scenario, by running experiments on a standard OpenStack setup as well as in ad hoc stand-alone configurations built with the specific purpose of observing them in isolation. The results prove that the Linux Bridge is the critical bottleneck of the architecture, while Open vSwitch shows an almost optimal behavior.

The analysis was then extended to more complex scenarios, assuming a data center hosting multiple tenants deploying NFV environments. The case studies considered first a simple dedicated deep packet inspection function, followed by conventional address translation and routing, and then a more realistic virtual network function chaining shared among a set of customers with increased levels of complexity. Results about sustainable packet rate and throughput performance of the virtual network infrastructure were presented and discussed.

The main outcome of this work is that an open-source cloud computing platform such as OpenStack can be effectively adopted to deploy NFV in network edge data centers replacing legacy telco central offices. However, this solution poses some limitations to the network performance which are not simply related to the hosting hardware maximum capacity but also to the virtual network architecture implemented by OpenStack. Nevertheless, our study demonstrates that some of these limitations can be mitigated with a careful redesign of the virtual network infrastructure and an optimal planning of the virtual network functions. In any case, such limitations must be carefully taken into account for any engineering activity in the virtual networking arena.

Obviously, scaling up the system and distributing the virtual network functions among several compute nodes will definitely improve the overall performance. However, in this case the role of the physical network infrastructure becomes critical, and an accurate analysis is required in order to isolate the contributions of virtual and physical components. We plan to extend our study in this direction in our future work, after properly upgrading our experimental test-bed.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was partially funded by EIT ICT Labs, Action Line on Future Networking Solutions, Activity no. 15270/2015: "SDN at the Edges." The authors would like to thank Mr. Giuliano Santandrea for his contributions to the experimental setup.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[2] N. M. Mosharaf Kabir Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[3] The Open Networking Foundation, *Software-Defined Networking: The New Norm for Networks*, ONF White Paper, The Open Networking Foundation, 2012.

[4] The European Telecommunications Standards Institute, "Network functions virtualisation (NFV); architectural framework," ETSI GS NFV 002, V1.2.1, The European Telecommunications Standards Institute, 2014.

[5] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 63–70, 2013.

[6] J. Soares, C. Goncalves, B. Parreira et al., "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, 2015.

[7] Open Networking Lab, *Central Office Re-Architected as Datacenter (CORD)*, ON.Lab White Paper, Open Networking Lab, 2015.

[8] K. Pretz, "Software already defines our lives—but the impact of SDN will go beyond networking alone," *IEEE. The Institute*, vol. 38, no. 4, p. 8, 2014.

[9] OpenStack Project, http://www.openstack.org.

[10] F. Sans and E. Gamess, "Analytical performance evaluation of different switch solutions," *Journal of Computer Networks and Communications*, vol. 2013, Article ID 953797, 11 pages, 2013.

[11] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *Proceedings of the 3rd International Conference on Cloud Networking (CloudNet '13)*, pp. 120–125, IEEE, Luxembourg City, Luxembourg, October 2014.

[12] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," in *Proceedings of the 33rd IEEE Conference on Computer Communications (INFOCOM '14)*, pp. 1285–1293, IEEE, Ontario, Canada, May 2014.

[13] P. Rad, R. V. Boppana, P. Lama, G. Berman, and M. Jamshidi, "Low-latency software defined network for high performance clouds," in *Proceedings of the 10th System of Systems Engineering Conference (SoSE '15)*, pp. 486–491, San Antonio, Tex , USA, May 2015.

[14] S. Oechsner and A. Ripke, "Flexible support of VNF placement functions in OpenStack," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NETSOFT '15)*, pp. 1–6, London, UK, April 2015.

[15] G. Almasi, M. Banikazemi, B. Karacali, M. Silva, and J. Tracey, "Openstack networking: it's time to talk performance," in *Proceedings of the OpenStack Summit*, Vancouver, Canada, May 2015.

[16] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of network virtualization in cloud computing infrastructures: the Openstack case," in *Proceedings of the 3rd IEEE International Conference on Cloud Networking (CloudNet '14)*, pp. 132–137, Luxemburg City, Luxemburg, October 2014.

[17] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of multi-tenant virtual networks in OpenStack-based cloud infrastructures," in *Proceedings of the 2nd IEEE Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA '14), in Conjunction with IEEE Globecom 2014*, pp. 81–85, Austin, Tex, USA, December 2014.

[18] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pp. 253–264, ACM, December 2012.

[19] P. Bellavista, F. Callegati, W. Cerroni et al., "Virtual network function embedding in real cloud environments," *Computer Networks*, vol. 93, part 3, pp. 506–517, 2015.

[20] M. F. Bari, R. Boutaba, R. Esteves et al., "Data center network virtualization: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[21] The Linux Foundation, *Linux Bridge*, The Linux Foundation, 2009, http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge.

[22] J. T. Yu, "Performance evaluation of Linux bridge," in *Proceedings of the Telecommunications System Management Conference*, Louisville, Ky, USA, April 2004.

[23] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets '09)*, New York, NY, USA, October 2009.

[24] G. Santandrea, *Show My Network State*, 2014, https://sites.google.com/site/showmynetworkstate.

[25] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[26] "RUDE & CRUDE: Real-Time UDP Data Emitter & Collector for RUDE," http://sourceforge.net/projects/rude/.

[27] iperf3: a TCP, UDP, and SCTP network bandwidth measurement tool, https://github.com/esnet/iperf.

[28] nDPI: Open and Extensible LGPLv3 Deep Packet Inspection Library, http://www.ntop.org/products/ndpi/.