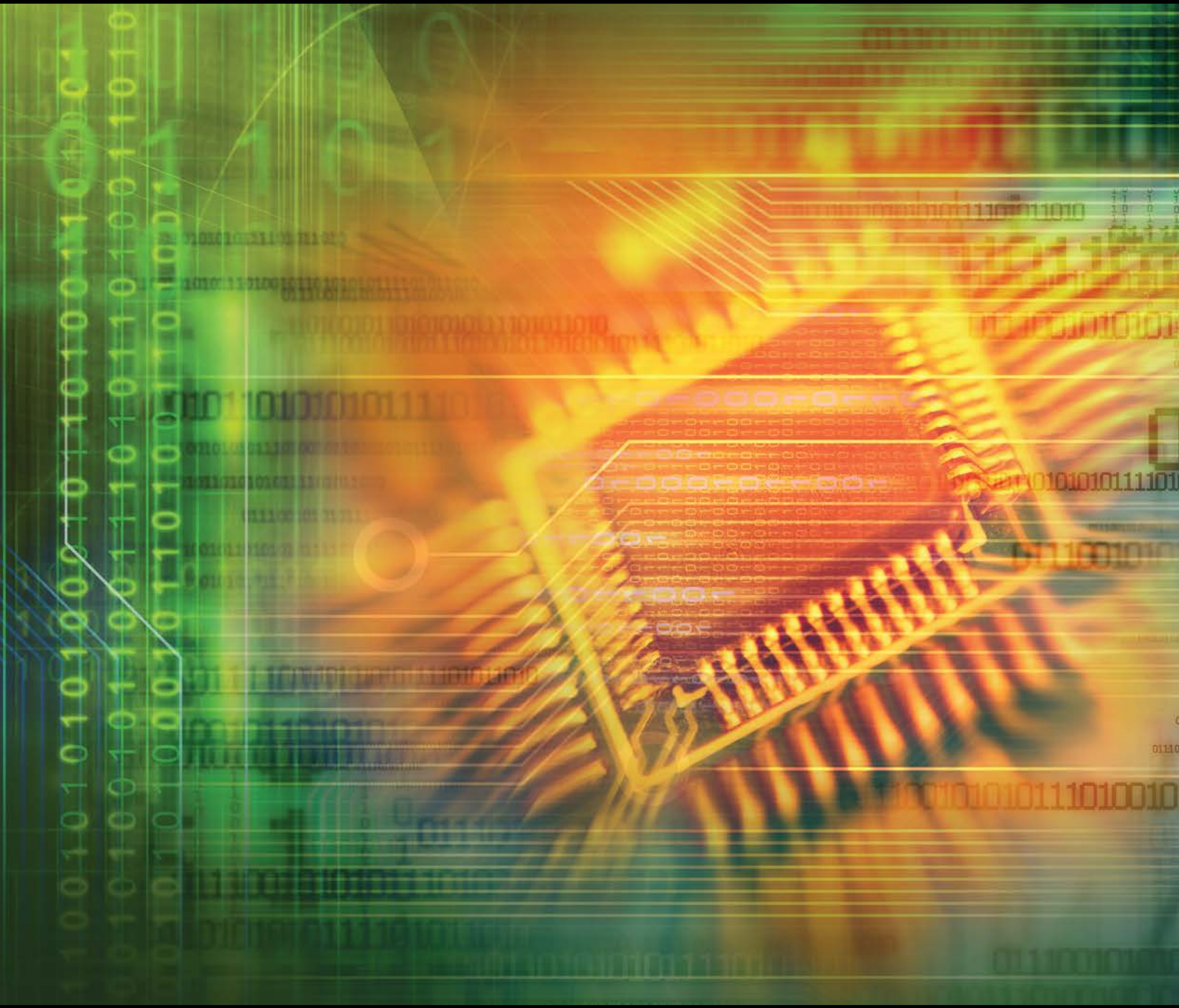
Journal of Electrical and Computer Engineering

Design of High Throughput and Cost-Efficient Data Center Networks

Guest Editors: Vincenzo Eramo, Xavier Hesselbach-Serra, Yan Luo, and Juan Felipe Botero



Design of High Throughput and Cost-Efficient Data Center Networks

Journal of Electrical and Computer Engineering

Design of High Throughput and Cost-Efficient Data Center Networks

Guest Editors: Vincenzo Eramo, Xavier Hesselbach-Serra, Yan Luo, and Juan Felipe Botero

Circuits and Systems

Communications

Signal Processing

Contents

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 4695185, 2 pages
http://dx.doi.org/10.1155/2016/4695185

Editorial

Design of High Throughput and Cost-Efficient Data Center Networks

Vincenzo Eramo,[1] Xavier Hesselbach-Serra,[2] Yan Luo,[3] and Juan Felipe Botero[4]

[1]*Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, 00184 Rome, Italy*
[2]*Network Engineering Department (ENTEL), Universitat Politecnica de Catalunya, 08034 Barcelona, Spain*
[3]*Department of Electronics and Computers Engineering (DECE), University of Massachusetts Lowell, Lowell, MA 01854, USA*
[4]*Department of Electronic and Telecomunications Engineering (DETE), Universidad de Antioquia, Oficina 19-450, Medellın, Colombia*

Correspondence should be addressed to Vincenzo Eramo; vincenzo.eramo@uniroma1.it

Received 20 April 2016; Accepted 20 April 2016

Data centers (DC) are characterized by the sharing of compute and storage resources to support Internet services. Today, many companies (Amazon, Google, Facebook, etc.) use data centers to offer storage, web search, and large-computations services with multibillion dollars business. The servers are interconnected by elements (switches, routers, interconnection systems, etc.) of a network platform that is referred to as Data Center Network (DCN).

Network Function Virtualization (NFV) technology introduced by European Telecommunications Standards Institute (ETSI) applies the cloud computing techniques in the telecommunication field allowing for a virtualization of the network functions to be executed on software modules running in data centers. Any network service is represented by a Service Function Chain (SFC) that is a set of VNFs to be executed according to a given order. The running of VNFs needs the instantiation of VNF instances (VNFI) that in general are software modules executed on virtual machines.

The support of NFV needs high performance servers due to higher requirements by the network services with respect to classical cloud applications.

The purpose of this special issue is to study and evaluate new solution for the support of NFV technology.

The special issue consists of four papers whose brief summaries are listed below.

"Server Resource Dimensioning and Routing of Service Function Chain in NFV Network Architectures" by V. Eramo et al. focuses on the resource dimensioning and SFC routing problems in NFV architecture. The objective of the problem is to minimize the number of SFCs dropped. The authors formulate the optimization problem and due to its NP-hard complexity, heuristics are proposed for both cases of offline and online traffic demand.

"A Game for Energy-Aware Allocation of Virtualized Network Functions" by R. Bruschi et al. presents and evaluates an energy-aware game theory based solution for resource allocation of Virtualized Network Functions (VNFs) within NFV environments. The authors consider each VNF as a player of the problem that competes for the physical network node capacity pool, seeking the minimization of individual cost functions. The physical network nodes dynamically adjust their processing capacity according to the incoming workload flows, by means of an Adaptive Rate strategy that aims at minimizing the product of energy consumption and processing delay.

"A Processor-Sharing Scheduling Strategy for NFV Nodes" by G. Faraci et al. focuses on the allocation strategies of processing resources to the virtual machines running the VNF. The main contribution of the paper is the definition of a processor-sharing policy, referred to as Network-Aware

Round Robin (NARR). The proposed strategy dynamically changes the slices of the CPU assigned to each VNF according to the state of the output network interface card queues. In order to not waste output link bandwidth, more processing resources are assigned to the VNF whose packets are addressed towards the least loaded output NIC.

"Virtual Networking Performance in OpenStack Platform for Network Function Virtualization" by F. Callegati et al. evaluates the performance evaluation of an Open Source Virtual Infrastructure Manager (VIM) as OpenStack focusing in particular on packet forwarding performance issues. A set of experiments are presented that refer to a number of scenarios inspired by the cloud computing and NFV paradigms, considering both single- and multitenant scenarios.

Vincenzo Eramo
Xavier Hesselbach-Serra
Yan Luo
Juan Felipe Botero

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].

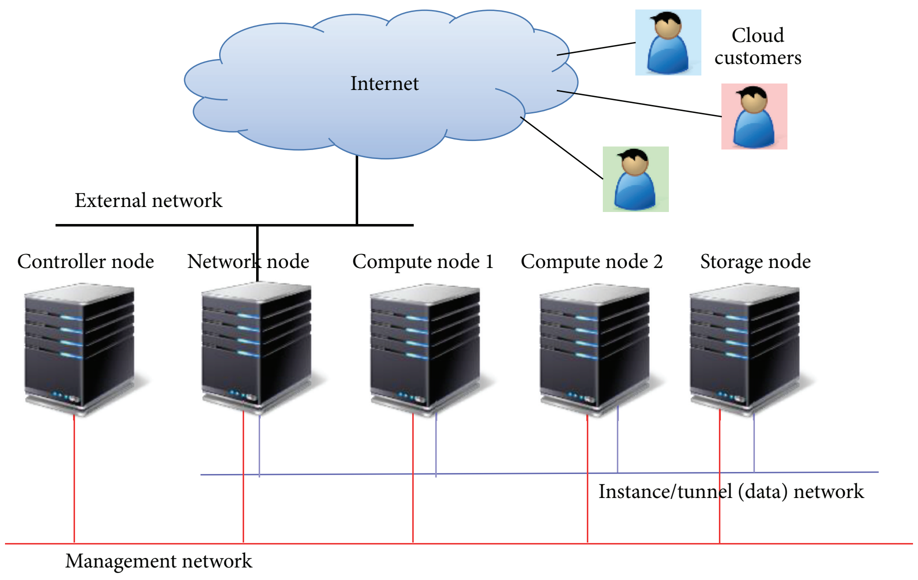


Figure 1: Main components of an OpenStack cloud setup.

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

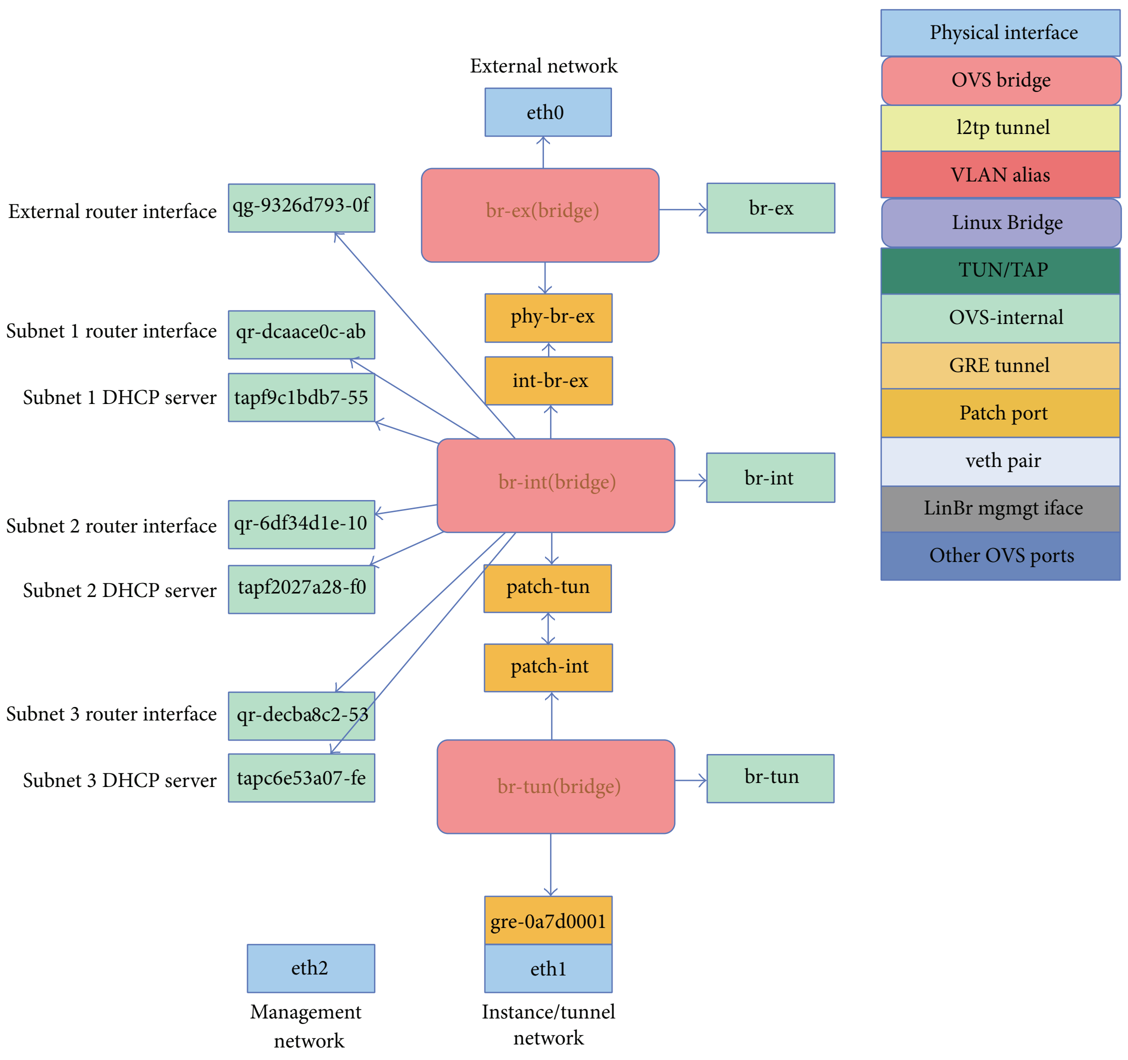


Figure 2: Network elements in an OpenStack network node connected to three virtual subnets. Three OVS bridges (red boxes) are interconnected by patch port pairs (orange boxes). *br-ex* is directly attached to the external network physical interface (*eth0*), whereas GRE tunnel is established on the instance/tunnel 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

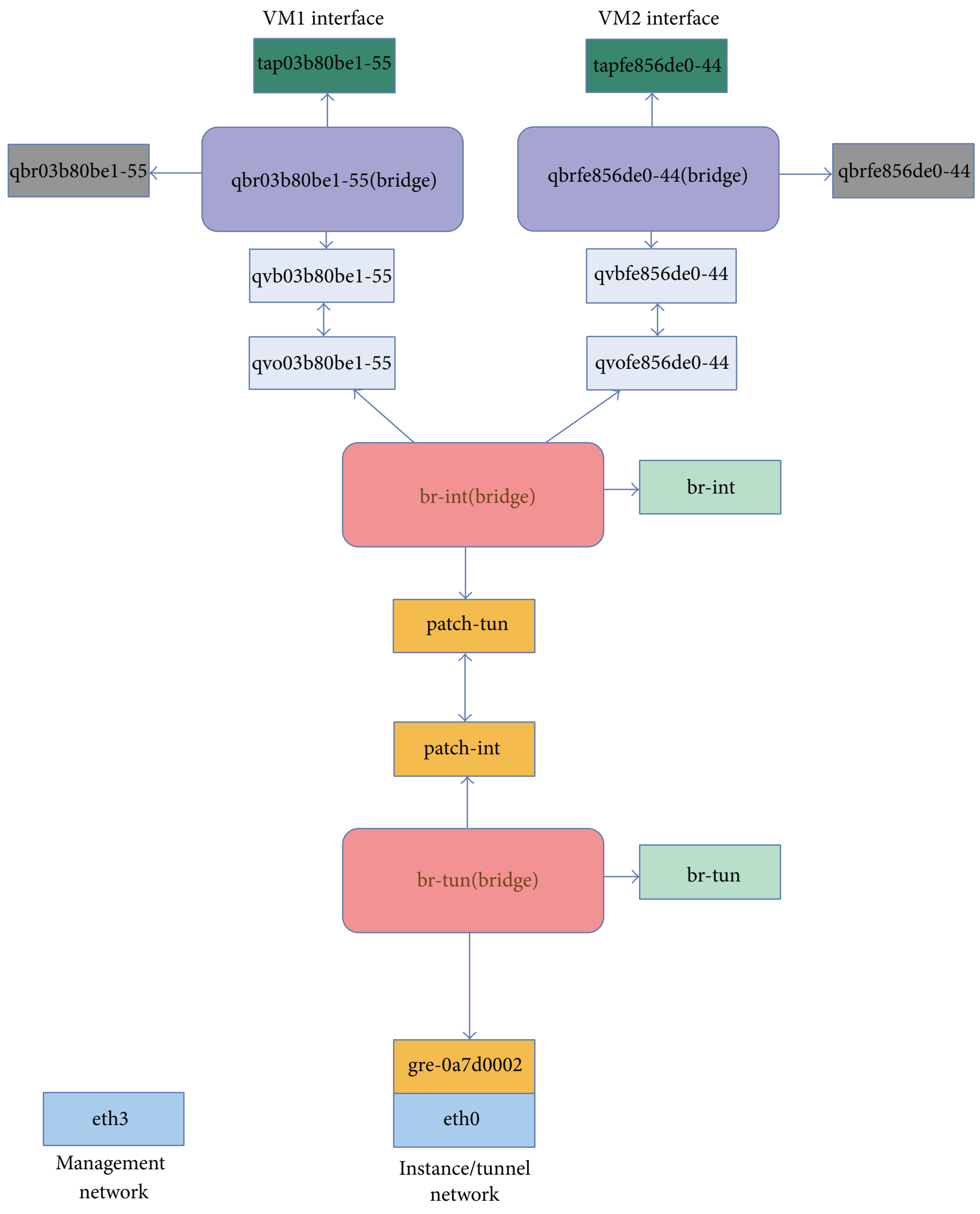


FIGURE 3: Network elements in an OpenStack compute node running two VMs. Two Linux Bridges (blue boxes) are attached to the VM tap interfaces (green boxes) and connected by virtual Ethernet pairs (light-blue boxes) to *br-int*.

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.

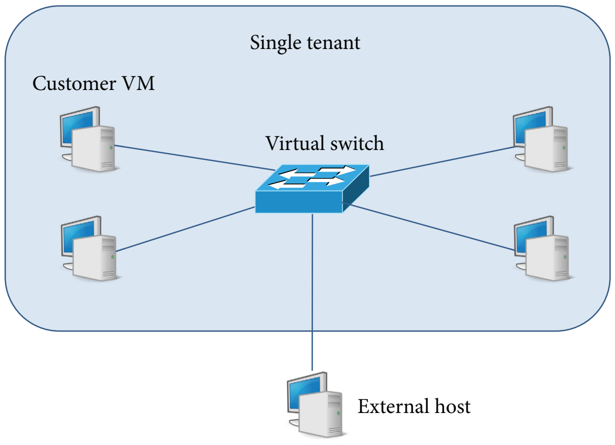


Figure 4: Reference logical architecture of a single tenant virtual infrastructure with 5 hosts: 4 hosts are implemented as VMs in the cloud and are interconnected via the OpenStack layer 2 virtual infrastructure; the 5th host is implemented by a physical machine placed outside the cloud but still connected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

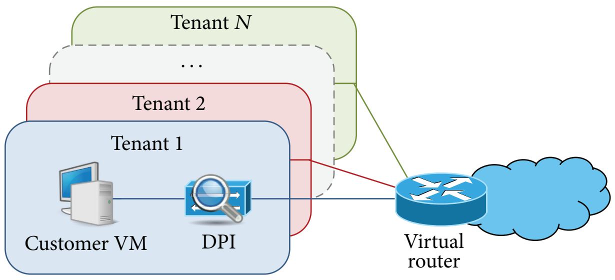


Figure 5: Multitenant NFV scenario with dedicated network functions tested on the OpenStack platform.

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

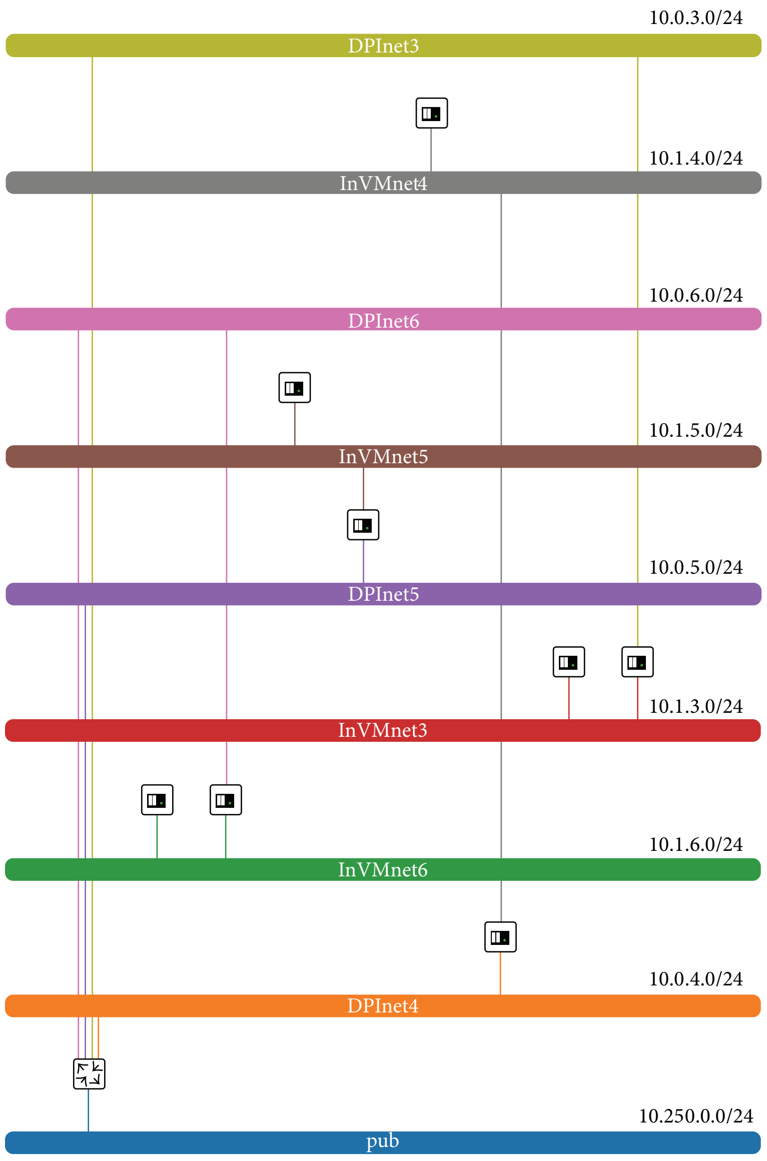


Figure 6: The OpenStack dashboard shows the tenants virtual networks (slices). Each slice includes VM connected to an internal network (InVMnet*i*) and a second VM performing DPI and packet forwarding between InVMnet*i* and DPInet*i*. Connectivity with the public Internet is provided for all by the virtual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

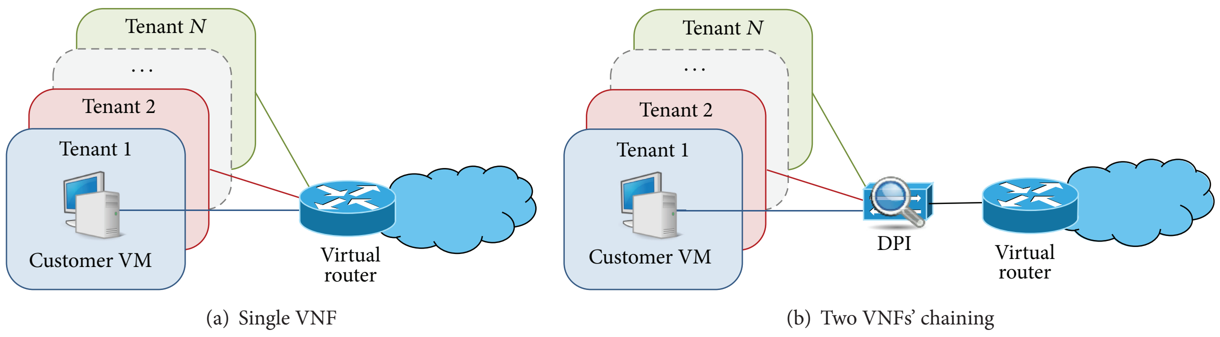
virtual network path inside the compute node for the VNF chaining of Figure 7(d) is displayed in Figure 8. The peculiar nature of NFV traffic flows is clearly shown in the figure, where packets are being forwarded multiple times across *br-int* as they enter and exit the multiple VNFs running in the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

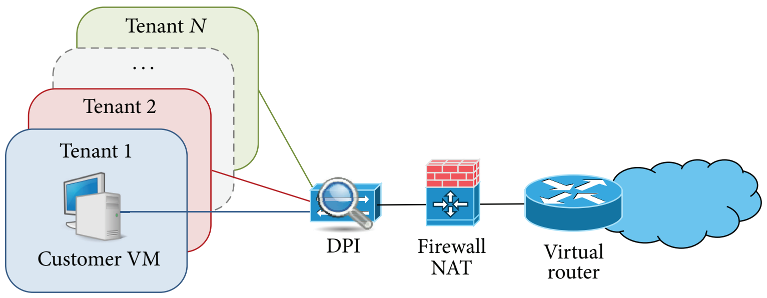
The former host acts as traffic generator while the latter acts as traffic sink. The aim is to verify and assess the maximum throughput and sustainable packet rate of the hardware platform used for the experiments. Packet flows ranging from 10^3 to 10^5 packets per second (pps), for both 64- and 1500-byte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

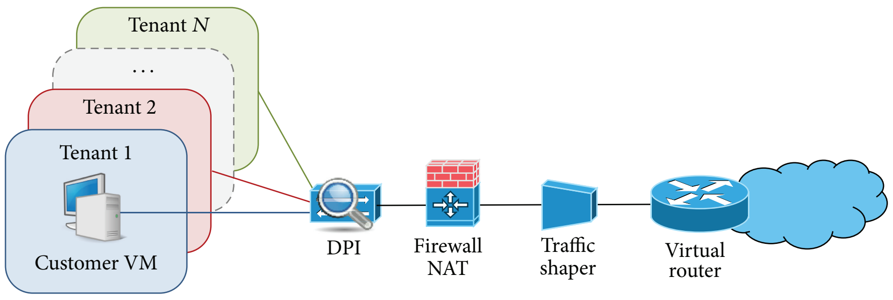


(a) Single VNF

(b) Two VNFs' chaining



(c) Three VNFs' chaining



(d) Four VNFs' chaining

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

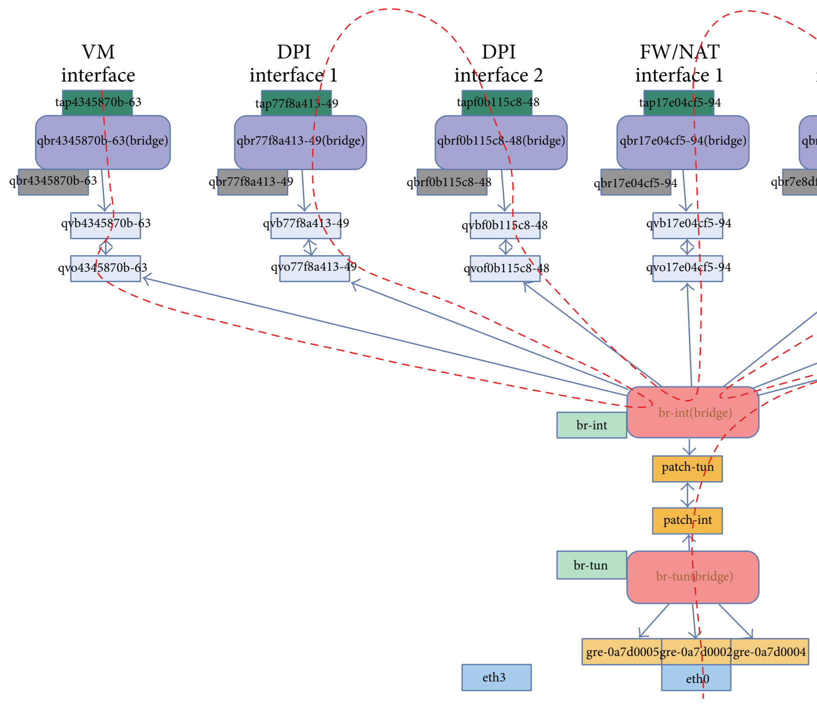


FIGURE 8: A view of the OpenStack compute node with the tenant VM and the VNFs installed including the building blocks of the virtual network infrastructure. The red dashed line shows the path followed by the packets traversing the VNF chain displayed in Figure 7(d).

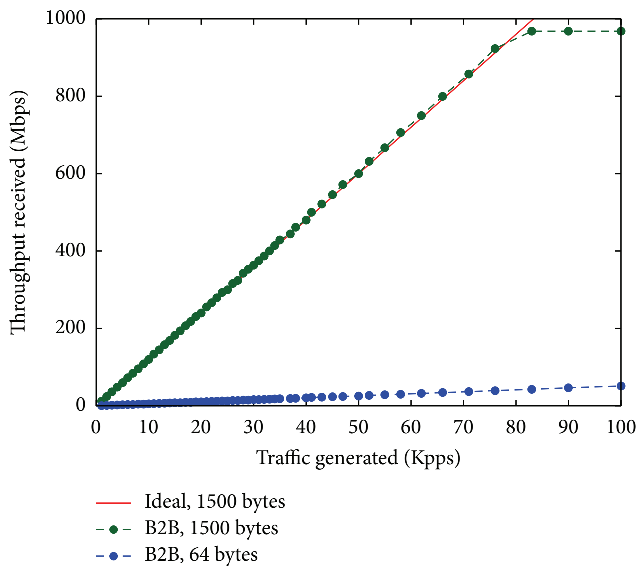


FIGURE 9: Throughput versus generated packet rate in the B2B setup for 64- and 1500-byte packets. Comparison with ideal 1500-byte packet throughput.

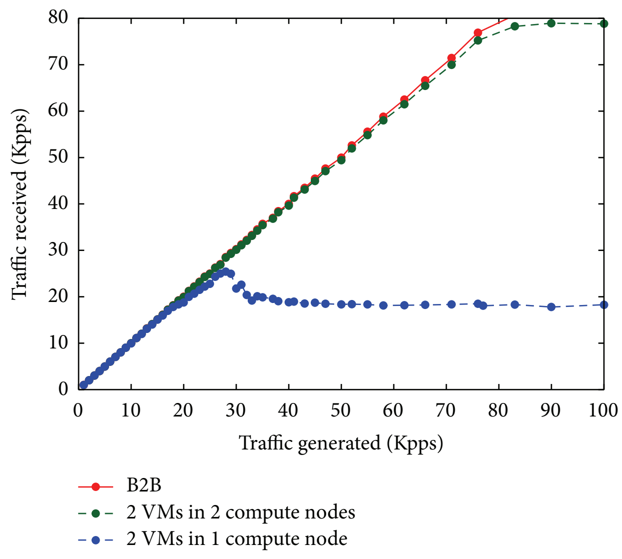


FIGURE 10: Received versus generated packet rate in the OpenStack scenario setups (1.1) and (1.2), with 1500-byte packets.

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

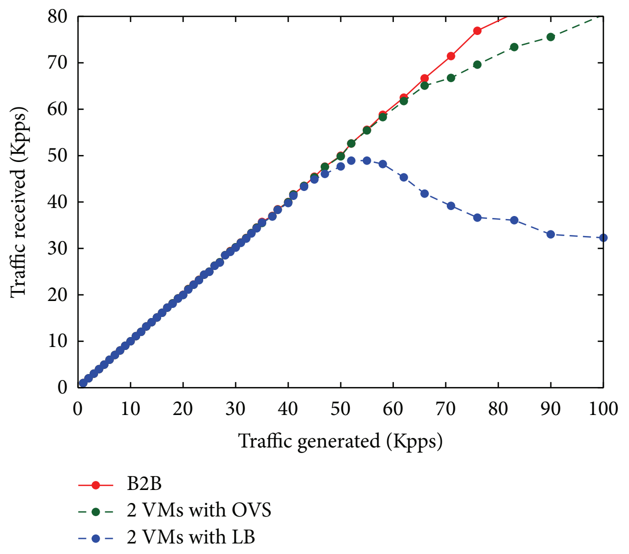


FIGURE 11: Received versus generated packet rate in the Non-OpenStack scenario setups (2.1) and (2.2), with 1500-byte packets.

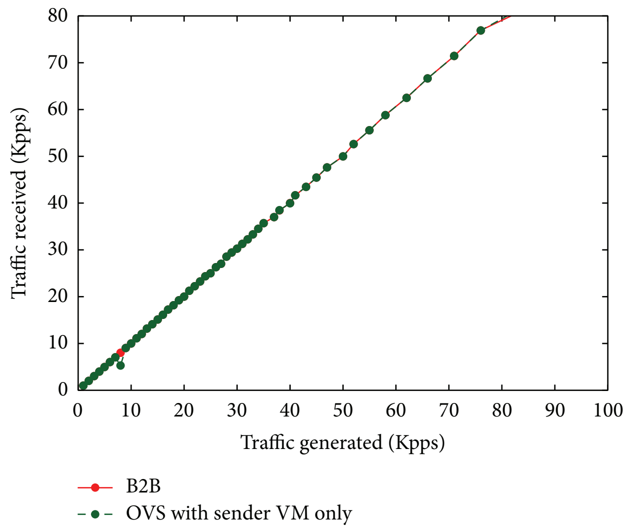


FIGURE 12: Received versus generated packet rate in the Non-OpenStack scenario setup (2.3), with 1500-byte packets.

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

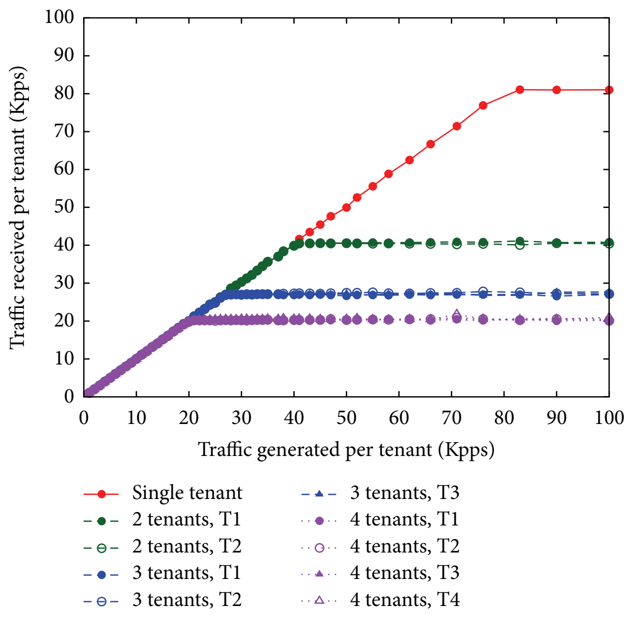


FIGURE 13: Received versus generated packet rate for each tenant (T1, T2, T3, and T4), for different numbers of active tenants, with 1500-byte IP packet size.

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

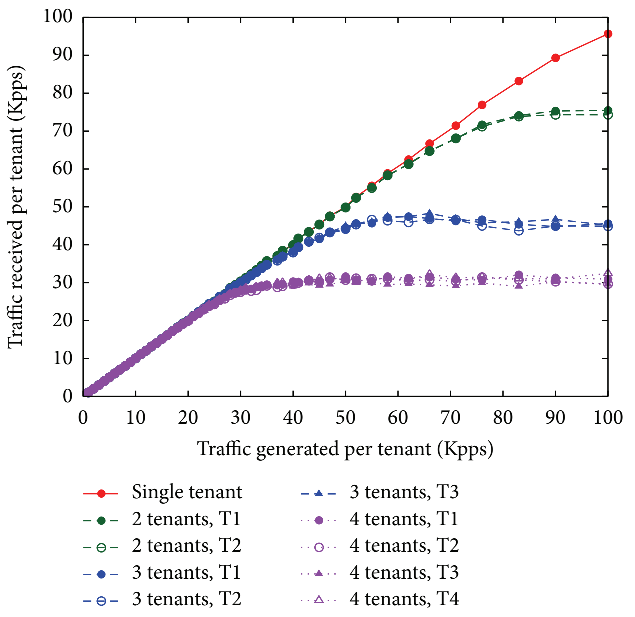


FIGURE 14: Received versus generated packet rate for each tenant (T1, T2, T3, and T4), for different numbers of active tenants, with 64-byte IP packet size.

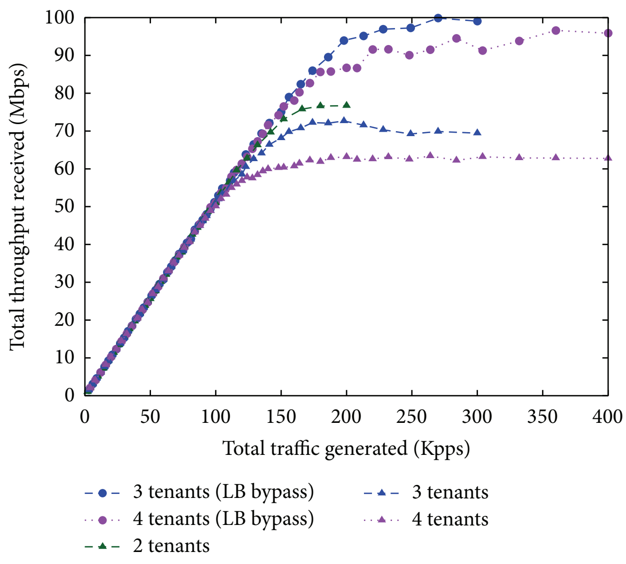


FIGURE 15: Total throughput measured versus total packet rate generated by 2 to 4 tenants for 64-byte packet size. Comparison between normal OpenStack mode and Linux Bridge bypass with 3 and 4 tenants.

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 $\times$ 2), with three tenants at a total of approximately 135 Kpps (45 $\times$ 3), and with four tenants at a total of approximately 120 Kpps (30 $\times$ 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 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)

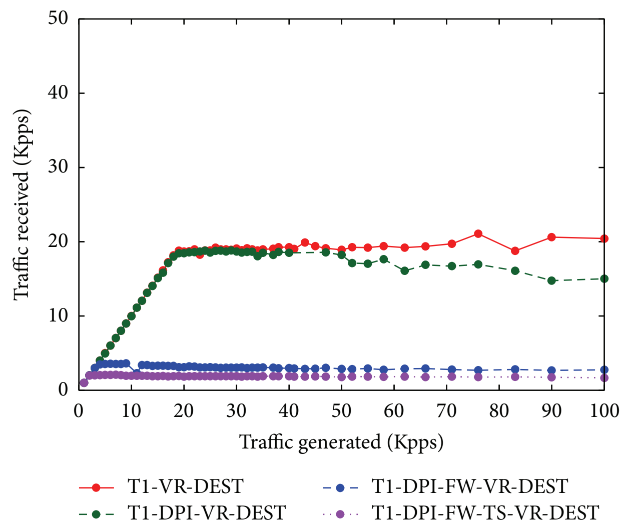


FIGURE 16: Received versus generated packet rate for one tenant (T1) when four tenants are active, with 1500-byte IP packet size and different levels of VNF chaining as per Figure 7. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

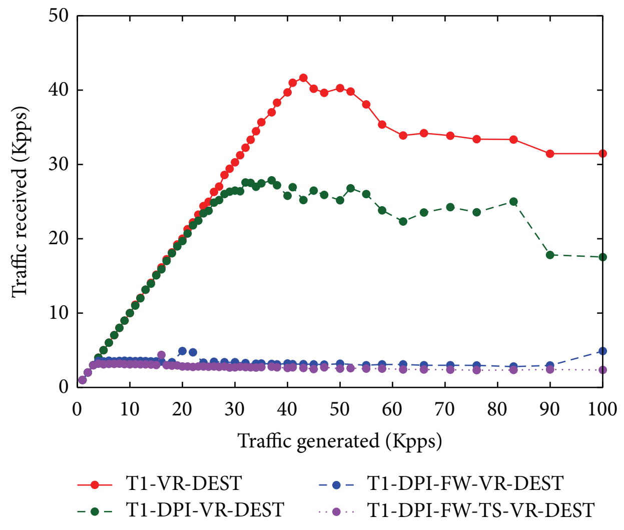


Figure 17: Received versus generated packet rate for one tenant (T1) when four tenants are active, with 64-byte IP packet size and different levels of VNF chaining as per Figure 7. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

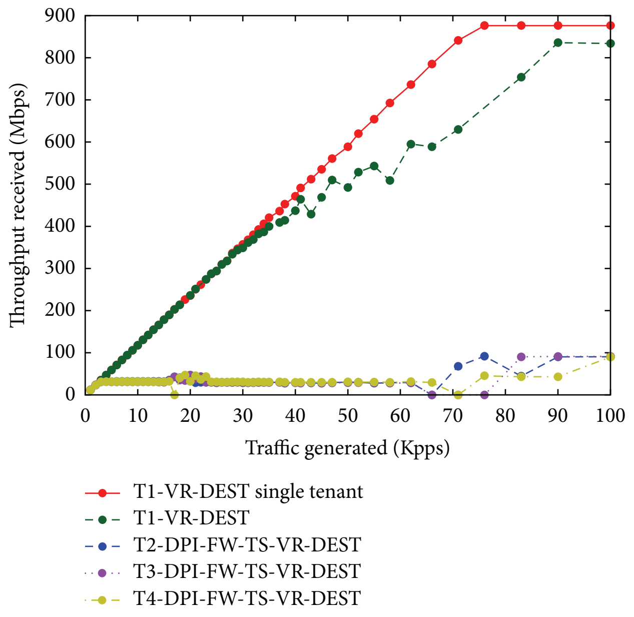


Figure 18: Received throughput versus generated packet rate for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d), with 1500-byte IP packet size. Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual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

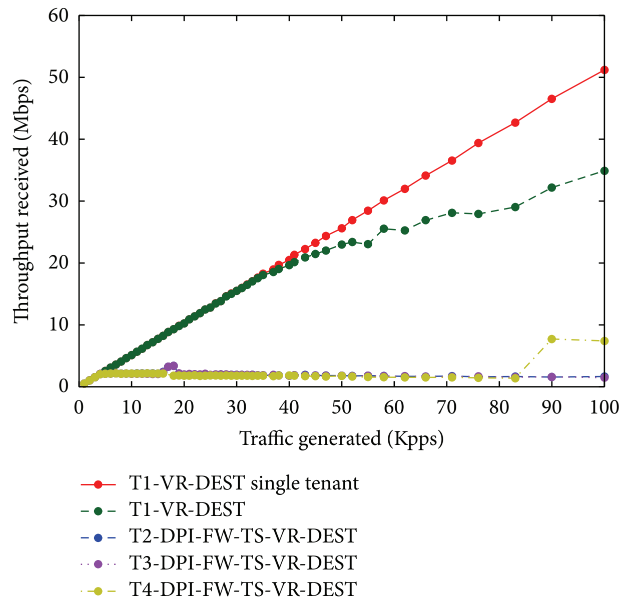


Figure 19: Received throughput versus generated packet rate for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d), with 64-byte IP packet size. Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual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

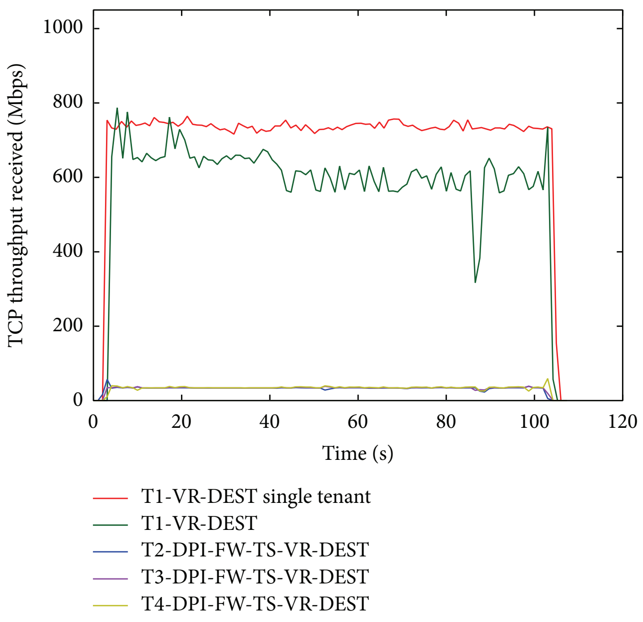


FIGURE 20: Received TCP throughput for each tenant (T1, T2, T3, and T4) when T1 does not traverse the VNF chain of Figure 7(d). Comparison with the single tenant case. DPI: deep packet inspection; FW: firewall/NAT; TS: traffic shaper; VR: virtual router; DEST: destination.

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/.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 7139852, 12 pages
http://dx.doi.org/10.1155/2016/7139852

Research Article

Server Resource Dimensioning and Routing of Service Function Chain in NFV Network Architectures

V. Eramo,[1] A. Tosti,[2] and E. Miucci[1]

[1] *DIET, "Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy*
[2] *Telecom Italia, Via di Val Cannuta 250, 00166 Roma, Italy*

Correspondence should be addressed to E. Miucci; 1kronos1@gmail.com

Received 29 September 2015; Accepted 18 February 2016

Academic Editor: Vinod Sharma

The Network Function Virtualization (NFV) technology aims at virtualizing the network service with the execution of the single service components in Virtual Machines activated on Commercial-off-the-shelf (COTS) servers. Any service is represented by the Service Function Chain (SFC) that is a set of VNFs to be executed according to a given order. The running of VNFs needs the instantiation of VNF instances (VNFI) that in general are software components executed on Virtual Machines. In this paper we cope with the routing and resource dimensioning problem in NFV architectures. We formulate the optimization problem and due to its NP-hard complexity, heuristics are proposed for both cases of offline and online traffic demand. We show how the heuristics works correctly by guaranteeing a uniform occupancy of the server processing capacity and the network link bandwidth. A consolidation algorithm for the power consumption minimization is also proposed. The application of the consolidation algorithm allows for a high power consumption saving that however is to be paid with an increase in SFC blocking probability.

1. Introduction

Today's networks are overly complex, partly due to an increasing variety of proprietary, fixed-function appliances that are unable to deliver the agility and economics needed to address constantly changing market requirements [1]. This is because network elements have traditionally been optimized for high packet throughput at the expense of flexibility, thus hampering the deployment of new services [2]. Network Function Virtualization (NFV) can provide the infrastructure flexibility and agility needed to successfully compete in today's evolving communications landscape [3]. NFV implements network functions in software running on a pool of shared commodity servers instead of using dedicated proprietary hardware. This virtualized approach decouples the network hardware from the network function and results in increased infrastructure flexibility and reduced hardware costs. Because the infrastructure is simplified and streamlined, new and expended services can be created quickly and with less expense. Implementation of the paradigm has also been proposed [4] and the performance has been investigated [5]. To support the NVF technology both ETSI [6, 7] and IETF [8, 9] are defining novel network architectures able to allocate resources for Virtualized Network Function (VNF) as well as manage and orchestrate NFV to support services. In particular the service is represented by a Service Function Chain (SFC) [8] that is a set of VNFs that have to be executed according to a given order. Any VNF is run on a VNF instance (VNFI) implemented with one Virtual Machine (VM) whose resources (Vcores, RAM memory, etc.) are allocated to [10]. Some solutions have been proposed in the literature to solve the problem of choosing the servers where to instantiate VNF and to determine the network paths interconnecting the VNFs [1]. A formulation of the optimization problem is illustrated in [11]. Three greedy algorithms and a tabu search-based heuristic are proposed in [12]. The extension of the problem in the case in which virtual routers are also considered is proposed in [13].

The innovative contribution of our paper is that differently from [12] we follow an approach that allows for both the resource dimensioning and the SFC routing. We formulate the optimization problem whose objective is the minimization of the number of dropped SFC requests with the constraint that the SFCs are routed so as to respect both the server processing capacity and network link bandwidth. With the problem being NP-hard we introduce a heuristic that allows for (i) the dimensioning of the Virtual Machines in terms of number of Vcores assigned and (ii) the SFC routing through the VNF instance implemented by the Virtual Machines. The heuristic performs simultaneously the dimensioning and routing operations. Furthermore we propose a consolidation algorithm based on Virtual Machine migrations and able to achieve power consumption savings. The proposed algorithms are evaluated in scenarios characterized by offline and online traffic demands.

The paper is organized as follows. The related work is discussed in Section 2. Section 3 is devoted to illustrating the optimization problem and the proposed heuristic for the offline traffic demand case. In Section 4 we describe an SFC planner in which the proposed heuristic is applied in the case of online traffic demand. The planner also implements a consolidation algorithm whose application allows for power consumption savings. Some numerical results are shown in Section 5 to prove the effectiveness of the proposed heuristics. Finally the main conclusions and future research items are mentioned in Section 6.

2. Related Work

The Internet Engineering Task Force (IETF) has formed the SFC Working Group [8, 9] to define Service Function Chaining related problems and to standardize the architecture and protocols. A Service Function Chain (SFC) is defined as a set of abstract service functions [15] and ordering constraints that must be applied to packets selected as a result of the classification. When virtual service functions are considered, the SFC is referred to as Virtual Network Function Forwarding Graph (VNFFG) within the ETSI [6]. To support the SFCs, Virtual Network Function instances (VNFIs) are activated and executed in COTS servers. To achieve the economics of scale expected from NFV, network link and server resources should be used efficiently. For this reason efficient algorithms have to be introduced to determine where to instantiate the VNFI and to route the SFCs by choosing the network paths and the VNFI involved. The algorithms have to take into account the limited resources of the network links and the servers and pursued objectives of load balancing, energy saving, recovery from failure, and so on [1]. The task of placing SFC is closely related to virtual network embeddings [16] and virtual data network embedding [17] and may therefore be formulated as an optimization problem, with a particular objective. The approach has been followed by [11, 18–21]. For instance, Moens and Turck [11] formulate the SFC placement problem as a linear optimization problem that has the objective of minimizing the number of active servers. Other objectives are pursued in [19] (latency, remaining data rate, number of used network nodes, etc.) and the SFC placing is formulated as a mixed integer quadratically constrained program.

It has been proved that the SFC placing problem is NP-hard. For this reason efficient heuristics have been proposed and evaluated [10, 13, 22, 23]. For example, Xia et al. [22] formulate the placement and chaining problem as binary integer programming and propose a greedy heuristic in which the SFCs are first sorted according to their resource demands and the SFCs with the highest resource demands are given priority for placement and routing.

In order to achieve energy consumption saving, the NFV architecture should allow for migrations of VNFI, that is, the migration of the Virtual Machine implementing the VNFI. Though VNFI migrations allow for energy consumption saving, they may impact the QoS performance received by the users related to the migrated VNFs. A model has been proposed in [24] to derive some performance indicators, such as the whole service down time and the total migration time so as to make function migrations decisions.

The contribution of this paper is twofold: (i) to propose and to evaluate the performance of an algorithm that performs simultaneously resource dimensioning and SFC routing and (ii) to investigate the advantages from the point of view of the power consumption saving that the application of server consolidation techniques allow us to achieve.

3. Offline Algorithms for SFC Routing in NFV Network Architectures

We consider the case in which SFC requests are known in advance. We formulate the optimization problem whose objective is the minimization of the number of dropped SFC requests with the constraint that the SFCs are routed so as to respect both the server processing capacity and network link bandwidth. With the problem being NP-hard we introduce a heuristic that allows for (i) the dimensioning of the Virtual Machines in terms of the number of Vcores assigned and (ii) the SFC routing through the VNF instances implemented by the Virtual Machines. The heuristic performs simultaneously the dimensioning and routing operations.

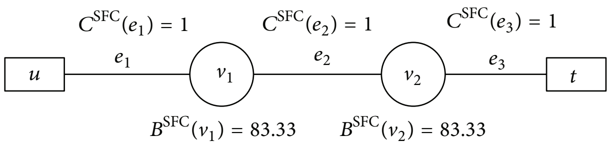
The section is organized as follows. The network and traffic model is introduced in Section 3.1. Sections 3.2 and 3.3 are devoted to illustrating the optimization problem and the proposed heuristic, respectively.

3.1. Network and Traffic Model. Next we introduce the main terminology used to represent the physical network, VNF, and the SFC traffic request [25]. We represent the physical network $\mathscr{PN}$ as a directed graph $\mathscr{G}^{\mathscr{PN}} = (\mathscr{V}^{\mathscr{PN}}, \mathscr{E}^{\mathscr{PN}})$, where $\mathscr{V}^{\mathscr{PN}}$ and $\mathscr{E}^{\mathscr{PN}}$ are the sets of physical nodes and links, respectively. The set $\mathscr{V}^{\mathscr{PN}}$ of nodes is given by the union of the three node sets $\mathscr{V}^{\mathscr{PN}}_{\mathscr{A}}$, $\mathscr{V}^{\mathscr{PN}}_{\mathscr{R}}$, and $\mathscr{V}^{\mathscr{PN}}_{\mathscr{S}}$ that are the sets of access, switching, and server nodes, respectively. The server nodes and links are characterized by the following:

(i) $N^{\mathscr{PN}}_{\text{core}}(w)$: processing capacity of the server node $w \in \mathscr{V}^{\mathscr{PN}}_{\mathscr{S}}$ in terms of the number of cores available;

(ii) $C^{\mathscr{PN}}(d)$: bandwidth of the physical link $d \in \mathscr{E}^{\mathscr{PN}}$.

We assume that F types of VNFs can be provisioned as firewall, IDS, proxy, load balancers, and so on. We denote by $\mathscr{F} = \{f_1, f_2, \ldots, f_F\}$ the set of VNFs, with f_i being the



$C^{\text{SFC}}(e_1) = 1$
$C^{\text{SFC}}(e_2) = 1$
$C^{\text{SFC}}(e_3) = 1$
u
e_1
v_1
e_2
v_2
e_3
t
$B^{\text{SFC}}(v_1) = 83.33$
$B^{\text{SFC}}(v_2) = 83.33$

Figure 1: An example of graph representing an SFC request for a flow characterized by the bandwidth of 1 Mbps and packet length of 1500 bytes.

ith VNF type. The packet processing time of the VNF f_i is denoted by t_i^{proc} $(i = 1, \ldots, F)$.

We also assume that the network operator is receiving T Service Function Chain (SFC) requests known in advance. The ith SFC request is characterized by the graph $\mathcal{G}_i^{\mathcal{SFC}} = (\mathcal{V}_i^{\mathcal{SFC}}, \mathcal{E}_i^{\mathcal{SFC}})$, where $\mathcal{V}_i^{\mathcal{SFC}}$ represents the set of access and VNF nodes and $\mathcal{E}_i^{\mathcal{SFC}}$ $(i = 1, \ldots, T)$ denotes the links between them. In particular the set $\mathcal{V}_i^{\mathcal{SFC}}$ is given by the union of $\mathcal{V}_{i,A}^{\mathcal{SFC}}$ and $\mathcal{V}_{i,F}^{\mathcal{SFC}}$ denoting the set of access nodes and VNFs, respectively. The graph is characterized by the following parameters:

(i) α_{vw}: assuming the value 1 if the access node $v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,A}^{\mathcal{SFC}}$ characterizes an SFC request starting/terminating from/to the physical access nodes $w \in \mathcal{V}_{\mathcal{A}}^{\mathcal{PN}}$; otherwise its value is 0;

(ii) β_{vk}: assuming the value 1 if the VNF node $v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}$ needs the application of the VNF f_k $(k \in [1, \ldots, F])$; otherwise its value is 0;

(iii) $B^{\mathcal{SFC}}(v)$: the processing capacity requested by the VNF node $v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}$; the parameter value depends on both the packet length and the bandwidth of the packet flow incoming to the VNF node;

(iv) $C^{\mathcal{SFC}}(e)$: bandwidth requested by the link $e \in \bigcup_{i=1}^{T} \mathcal{E}_i^{\mathcal{SFC}}$.

An example of SFC request is represented in Figure 1 where the traffic emitted by the ingress access node u has to be handled by the VNF nodes v_1 and v_2 and it is terminating to the egress access node t. The access and VNF nodes are interconnected by the links e_1, e_2, and e_3. If the flow bandwidth is 1 Mbps and the packet length is 1500 bytes we obtain the processing capacities $B^{\mathcal{SFC}}(v_1) = B^{\mathcal{SFC}}(v_2) = 83.33$ while the bandwidths $C^{\mathcal{SFC}}(e_1)$, $C^{\mathcal{SFC}}(e_2)$, and $C^{\mathcal{SFC}}(e_3)$ of all links equal 1.

3.2. Optimization Problem for the SFC Routing and VNF Instance Dimensioning. The objective of the SFC Routing and VNF Instance Dimensioning (SRVID) problem is to maximize the number of accepted SFC requests. The output of the problem is characterized by the following: (i) the servers in which the VNF nodes are executed and (ii) the network paths in which the virtual links of the SFC are routed. This arrangement has to be accomplished without violating both the server processing and the physical link capacities. We assume that all of the servers create a VNF instance for each type of VNF that will be shared by the SFCs using that server and requesting that type of VNF. For this reason each server will activate F VNF instances, one for each type, and another output of the problem is to determine the number of Vcores to be assigned to each VNF instance.

We assume that one virtual link of the SFC graphs can be routed through single physical network path. We introduce the following notations:

(i) $\mathcal{P}$: set of paths in $\mathcal{G}^{\mathcal{PN}}$,

(ii) δ_{dp}: the binary function assuming value 1 or 0 if the network link d belongs or does not to the path $p \in \mathcal{P}$, respectively,

(iii) $a^{\mathcal{PN}}(p)$ and $b^{\mathcal{PN}}(p)$: origin and destination nodes of the path $p \in \mathcal{P}$,

(iv) $a^{\mathcal{SFC}}(d)$ and $b^{\mathcal{SFC}}(d)$: origin and destination nodes of the virtual link $e \in \bigcup_{i=1}^{T} \mathcal{E}_i^{\mathcal{SFC}}$.

Next we formulate the optimal SRVID problem characterized by the following optimization variables:

(i) x_h: binary variable assuming the value 1 if the hth SFC request is accepted; otherwise its value is zero;

(ii) y_{wk}: integer variable characterizing the number of Vcores allocated to the VNF instance of type k in the server $w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}$;

(iii) z_{vw}^{k}: binary variable assuming the value 1 if the VNF node $v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}$ is served by the VNF instance of type k in the server $w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}$;

(iv) u_{dp}: binary variable assuming the value 1 if the virtual link $e \in \bigcup_{i=1}^{T} \mathcal{E}_i^{\mathcal{SFC}}$ is embedded in the physical network path $p \in \mathcal{P}$; otherwise its value is zero.

Next we report the constraints for the optimization variables:

$$\sum_{k=1}^{F} y_{wk} \le N_{\text{core}}^{\mathcal{PN}}(w), \qquad w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}, \tag{1}$$

$$\sum_{k=1}^{F} \sum_{w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}} z_{vw}^{k} \le 1, \quad v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}, \tag{2}$$

$$z_{vw}^{k} \le \beta_{vk}, \qquad k \in [1, \ldots, F], \; v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}, \; w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}, \tag{3}$$

$$\sum_{v \in \bigcup_{i=1}^{T} \mathcal{V}_{i,F}^{\mathcal{SFC}}} z_{vw}^{k} B^{\mathcal{SFC}}(v)\, t_k^{\text{proc}} \le y_{wk}, \qquad k \in [1, \ldots, F], \; w \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}, \tag{4}$$

$$u_{dp} \leq z^{k}_{a^{\mathcal{SFC}}(d)a^{\mathcal{PN}}(p)},$$
$$a^{\mathcal{SFC}}(d) \in \bigcup_{i=1}^{T} \mathcal{V}^{\mathcal{SFC}}_{i,F},\ k \in [1, \ldots, F],\ p \in \mathcal{P}, \tag{5}$$

$$u_{dp} \leq z^{k}_{b^{\mathcal{SFC}}(d)b^{\mathcal{PN}}(p)},$$
$$b^{\mathcal{SFC}}(d) \in \bigcup_{i=1}^{T} \mathcal{V}^{\mathcal{SFC}}_{i,F},\ k \in [1, \ldots, F],\ p \in \mathcal{P}, \tag{6}$$

$$u_{dp} \leq \alpha^{k}_{a^{\mathcal{SFC}}(d)a^{\mathcal{PN}}(p)},$$
$$a^{\mathcal{SFC}}(d) \in \bigcup_{i=1}^{T} \mathcal{V}^{\mathcal{SFC}}_{i,A},\ k \in [1, \ldots, F],\ p \in \mathcal{P}, \tag{7}$$

$$u_{dp} \leq \alpha^{k}_{b^{\mathcal{SFC}}(d)b^{\mathcal{PN}}(p)},$$
$$b^{\mathcal{SFC}}(d) \in \bigcup_{i=1}^{T} \mathcal{V}^{\mathcal{SFC}}_{i,A},\ k \in [1, \ldots, F],\ p \in \mathcal{P}, \tag{8}$$

$$\sum_{p \in \mathcal{P}} u_{dp} \leq 1, \quad d \in \bigcup_{i=1}^{T} \mathcal{E}^{\mathcal{SFC}}_{i}, \tag{9}$$

$$\sum_{d \in \bigcup_{i=1}^{T} \mathcal{E}^{\mathcal{SFC}}_{i}} C^{\mathcal{SFC}}(d) \sum_{p \in \mathcal{P}} \delta_{ep} u_{dp} \leq C^{\mathcal{PN}}(e), \tag{10}$$

$$x_h \leq \sum_{k=1}^{F} \sum_{w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}} z^{k}_{vw},$$
$$h \in [1, \ldots, T],\ v \in \mathcal{V}^{\mathcal{SFC}}_{h,F}, \tag{11}$$

$$x_h \leq \sum_{p \in \mathcal{P}} u_{dp},$$
$$h \in [1, \ldots, T],\ d \in \mathcal{E}^{\mathcal{SFC}}_{h}. \tag{12}$$

Constraint (1) establishes the fact that at most a number of Vcores equal to the available ones are used for each server node $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}$. Constraint (2) expresses the condition that a VNF can be served by one only VNF instance. To guarantee that a node $v \in \mathcal{V}^{\mathcal{SFC}}_{h,F}$ needing the application of a VNF type is mapped to a correct VNF instance we introduce constraint (3). Constraint (4) limits the number of VNF nodes assigned to one VNF instance by taking into account both the number of Vcores assigned to the VNF instance and the required VNF node processing capacities. Constraints (5)–(8) establish the fact that when the virtual link d is supported by the physical network path p then $a^{\mathcal{PN}}(p)$ and $b^{\mathcal{PN}}(p)$ must be the physical network nodes that the nodes $a^{\mathcal{SFC}}(d)$ and $b^{\mathcal{SFC}}(d)$ of the virtual graph are assigned to. The choice of mapping of any virtual link on a single physical network path is represented by constraint (9). Constraint (10) avoids any overloading on any physical network link. Finally constraints (11) and (12) establish the fact that an SFC request can be accepted when the nodes and the links of the SFC graph are assigned to VNF instances and physical network paths.

The problem objective is to maximize the number of SFCs accepted given by

$$\max \sum_{h=1}^{T} x_h. \tag{13}$$

The introduced optimization problem is intractable because it requires solving a NP-hard bin packing problem [26]. Due to its high complexity, it is not possible to solve the problem directly in a timely manner given the large number of servers and network nodes. For this reason we propose an efficient heuristic in Section 3.3.

3.3. Maximizing the Accepted SFC Requests Number (MASRN) Heuristic. The Maximizing the Accepted SFC Requests Number (MASRN) heuristic is based on the embedding algorithm proposed in [14] and has the objective of maximizing the number of accepted SFC requests. The MASRN algorithm tries routing the SFCs one by one by using the least loaded link and server resources so as to balance their use. Algorithm 1 illustrates the operation mode of the MASRN algorithm. The routing of a target SFC, the k_{tar}th, is illustrated. The main inputs of the algorithm are (i) the set T_{pre} containing the indexes of the SFC requests routed successfully before the k_{tar}th SFC request; (ii) the k_{tar}th SFC request graph $\mathcal{G}^{\mathcal{SFC}}_{k_{\text{tar}}} = (\mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}}}, \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}})$; (iii) the values of the parameters α_{vw} for the k_{tar}th SFC request; (iv) the values of the variables z^{k}_{vw} and u_{dp} for the SFC requests successfully routed before the k_{tar}th SFC request and defined in Section 3.2.

The operation mode of the heuristic is based on the following variables:

(i) $A^{k}(w)$: the load of the cores allocated for the VNF instance of type $k \in [1, \ldots, F]$ in the server $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}$; the value of $A^{k}(w)$ is initialized by taking into account the core resource amount used by the SFCs successfully routed before the k_{tar}th SFC request; hence its initial value is

$$A^{k}(w) = \sum_{v \in \bigcup_{i \in T_{k_{\text{tar}}}} \mathcal{V}^{\mathcal{SFC}}_{i,F}} z^{k}_{vw} B^{\mathcal{SFC}}(v)\, t^{\text{proc}}_{k}. \tag{14}$$

(ii) $S_N(w)$: defined as the stress of the node $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}$ [14] and characterizing the server load; its value is initialized to

$$S_N(w) = \sum_{k=1}^{F} A^{k}(w). \tag{15}$$

(iii) $S_L(e)$: defined as the stress of the physical network link $e \in \mathcal{E}^{\mathcal{PN}}$ [14] and characterizing the link load; its value is initialized to

$$S_L(e) = \sum_{d \in \bigcup_{i \in T_{k_{\text{tar}}}} \mathcal{E}^{\mathcal{SFC}}_{i}} C^{\mathcal{SFC}}(d) \sum_{p \in \mathcal{P}} \delta_{ep} u_{dp}. \tag{16}$$

```
(1) Input: Physical Network Graph $\mathcal{G}^{\mathcal{PN}} = (\mathcal{V}^{\mathcal{PN}}, \mathcal{E}^{\mathcal{PN}})$; $k_{\text{tar}}$th SFC request; $k_{\text{tar}}$th SFC request Graph
      $\mathcal{G}^{\mathcal{SFC}}_{k_{\text{tar}}} = (\mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}}}, \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}})$; $T_{\text{pre}}$;
      $\{\alpha_{vw},\ v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},A}\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{A}}\}$;
      $\{z^k_{vw},\ k \in [1,\dots,F]\ v \in \bigcup_{i \in T_{\text{pre}}} \mathcal{V}^{\mathcal{SFC}}_{i,F}\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      $\{u_{dp},\ d \in \bigcup_{i \in T_{\text{pre}}} \mathcal{E}^{\mathcal{SFC}}_{i}\ p \in \mathcal{P}\}$;
(2) Variables: $\{A^k(w),\ k \in [1,\dots,F]\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      $\{S_N(w),\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      $\{S_L(e),\ e \in \mathcal{E}^{\mathcal{PN}}\}$;
      $\{y_{wk},\ k \in [1,\dots,F]\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      /* Evaluation of the candidate mapping of $\mathcal{G}^{\mathcal{SFC}}_{k_{\text{tar}}} = (\mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}}}, \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}})$ in $\mathcal{G}^{\mathcal{PN}} = (\mathcal{V}^{\mathcal{PN}}, \mathcal{E}^{\mathcal{PN}})$*/
(3) assign the node $v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},A}$ to the physical network node $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}$ according to the value of the parameter $\alpha_{vw}$;
(4) select the nodes $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}$ and the physical network paths $p \in \mathcal{P}$ to be assigned to the server nodes $v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},F}$ and virtual links
      $d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}},S}$ by applying the algorithm proposed in [14] and based on the values of the node and link stresses $S_N(w)$ and $S_L(e)$;
      determine the variable values:
      $\{z^k_{vw},\ k \in [1,\dots,F]\ v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},F}\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      $\{u_{dp},\ d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}}\ p \in \mathcal{P}\}$;
      /* Server Resource Availability Check Phase*/
(5) for $v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},F}, w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}, k \in [1,\dots,F]$ do
(6)     if $\sum_{s=1(s\neq k)}^{F} y_{ws} + \lceil A^k(w) + z^k_{vw} B^{\mathcal{SFC}}(v) t^{\text{proc}}_k \rceil \le N^{\mathcal{PN}}_{\text{core}}(w)$ then
(7)         $A^k(w) = A^k(w) + z^k_{vw} B^{\mathcal{SFC}}(v) t^{\text{proc}}_k$;
(8)         $S_N(w) = S_N(w) + z^k_{vw} B^{\mathcal{SFC}}(v) t^{\text{proc}}_k$;
(9)         $y_{wk} = \lceil A^k(w) + z^k_{vw} B^{\mathcal{SFC}}(v) t^{\text{proc}}_k \rceil$;
(10)    else
(11)        REJECT the $k_{\text{tar}}$th SFC request
(12)    end if
(13) end for
      /* Link Resource Availability Check Phase*/
(14) for $e \in \mathcal{E}^{\mathcal{PN}}$ do
(15)    if $S_L(e) + \sum_{d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}}} C^{\mathcal{SFC}}(d) \sum_{p \in \mathcal{P}} \delta_{ep} u_{dp} \le C^{\mathcal{PN}}(e)$ then
(16)        $S_L(e) = S_L(e) + \sum_{d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}}} C^{\mathcal{SFC}}(d) \sum_{p \in \mathcal{P}} \delta_{ep} u_{dp}$;
(17)    else
(18)        REJECT the $k_{\text{tar}}$th SFC request
(19)    end if
(20) end for
(21) Output: $\{z^k_{vw},\ k \in [1,\dots,F]\ v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},F}\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$;
      $\{u_{dp},\ d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}}\ p \in \mathcal{P}\}$
```

Algorithm 1: MASRN algorithm.

The candidate physical network links and nodes in which to embed the target SFC are evaluated. First of all the access nodes are evaluated (line 3) according to the values of the parameters α_{vw}, $v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},A}$, $w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{A}}$. Next the MASRN algorithm selects a cluster of server nodes (line 4) that are not lightly loaded but also likely to result in low substrate link stresses when they are connected. Details on the procedure are reported in [14].

In the next phases the resource availability in the selected cluster is checked. The resource availability in the server nodes and physical network links is verified in the Server (lines 5–13) and Link (lines 14–20) Resource Availability Check Phases, respectively. If resources are not available in either links or nodes, the target SFC request is rejected. In particular, in the Server Resource Availability Check Phase, the algorithm verifies whether the allocation of new Vcores is needed and in this case their availability is checked (line 6). The variable y_{wk} is also updated in this phase (line 9).

Finally the outputs (line 21) of the MASRN algorithm are the selected values for the variables $\{z^k_{vw}, k \in [1,\dots,F]\ v \in \mathcal{V}^{\mathcal{SFC}}_{k_{\text{tar}},F}\ w \in \mathcal{V}^{\mathcal{PN}}_{\mathcal{S}}\}$ and $\{u_{dp}, d \in \mathcal{E}^{\mathcal{SFC}}_{k_{\text{tar}}}\ p \in \mathcal{P}\}$.

The computational complexity of the MASRN algorithm depends on the procedure for the evaluation of the potential nodes [14]. Let N_s be the total number of servers. The complexity of the phase in which the potential nodes are evaluated can be carried out according to the following remarks: (i) as many servers as the number of the VNFs of the SFC are determined and (ii) the hth server is selected by evaluating the $(N_s - h)$ shortest paths and by performing a minimum operation of a list with $(N_s - h)$ elements. According to these remarks the computational complexity is given by $O(V(V + N_s)N_l \log(N_s + N_n))$, where V is the

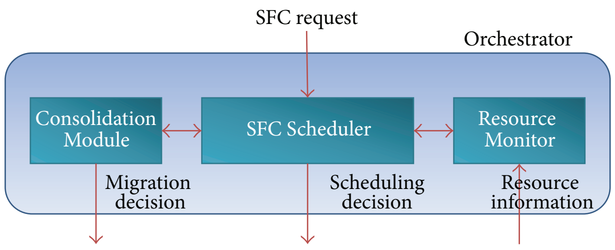


FIGURE 2: SFC planner architecture.

maximum number of VNFs in an SFC and N_l and N_n are the numbers of nodes and links of the network.

4. Online Algorithms for SFC Routing in NFV Network Architectures

In the online approach the SFC requests arrive at the system over the time and each of them is characterized by a certain duration. In order to reduce the complexity of the online SFC resource allocation algorithm we designed an SFC planner whose general architecture is described in Section 4.1. The operation mode of the SFC planner is based on some algorithms that we describe in Section 4.2.

4.1. SFC Planner Architecture. The architecture of the SFC planner is shown in Figure 2. It could make up the main component of an orchestrator in the NFV architecture [6]. It is composed of three components: the *SFC Scheduler*, the *Resource Monitor*, and the *Consolidation Module*.

Once the SFC Scheduler receives an SFC request, it executes an algorithm to verify if resources are available and to decide which resources to use.

The physical resource, as well as the Virtual Machines running on the servers, is monitored by the *Resource Monitor*. If a failure of any physical/virtual node or link occurs it notifies the event to the SFC Scheduler.

The Consolidation Module allows for the server resource consolidation in low traffic periods. When the consolidation technique is applied, VMs are migrated to as fewer servers as possible; the remaining ones are turned off and power consumption saving can be achieved.

4.2. SFC Scheduling and Consolidation Algorithms. In this section we describe the algorithms executed by the SFC Scheduler and the Consolidation Module.

Whenever a new SFC request arrives at the SFC planner, the SFC scheduling algorithm is executed in order to find the best embedding in the system, maintaining a uniform usage of the network resource. The main steps of this procedure are reported in the flow chart in Figure 3 and are based on MASRN heuristic described in Section 3.3. The algorithm starts selecting the set of servers on which the VNFs requested by the SFC will be executed. In the second step the physical paths on which the virtual links will be mapped are selected. If there are not enough available

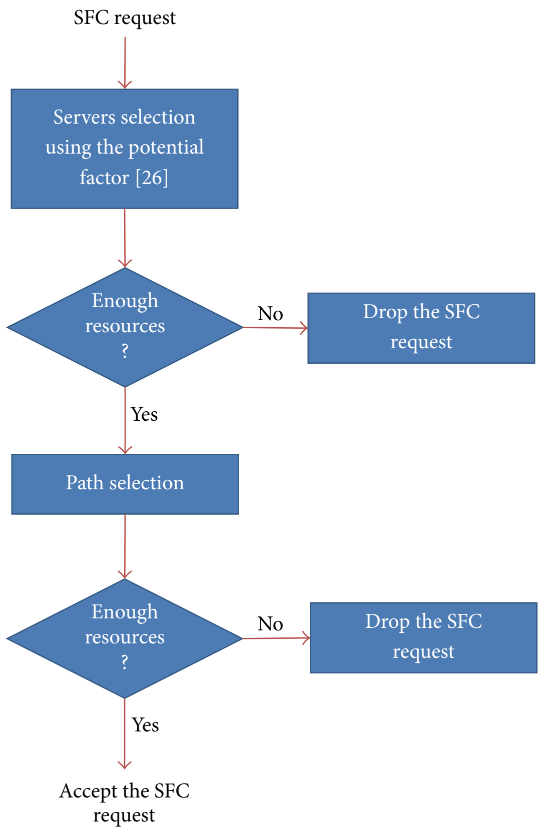


FIGURE 3: Flow chart of the SFC scheduling algorithm.

resources in the first or in the second step the request is dropped.

The flow chart of the consolidation algorithm is reported in Figure 4. Each server $s \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}$ is characterized by a parameter η_s defined as the ratio of the total amount of incoming/outgoing traffic Λ_s that it is handling to the power P_s it is consuming; that is, $\eta_s = \Lambda_s / P_s$. The power consumption model assumes a constant power contribution and a variable one that linearly increases versus the handled traffic. The server power consumption is expressed by

$$P_s = P_{\text{idle}} + (P_{\max} - P_{\text{idle}}) \frac{L_s}{C_s}, \quad \forall s \in \mathcal{V}_{\mathcal{S}}^{\mathcal{PN}}, \qquad (17)$$

where P_{idle} is the power consumed by the server when no traffic is handled, $P_{\max}$ is the maximum server power consumption, L_s is the total load handled by the server s, and C_s is its maximum capacity. The maximum power that the server can consume is expressed as $P_{\max} = P_{\text{idle}}/a$, where a is a parameter characterizing how much the power consumption is depending on the handled traffic. The power consumption is as much more rate adaptive as a is lower. For $a = 1$ the rate adaptive power consumption is absent and the consumed power is constant.

The proposed algorithm tries switching off the server with minimum power consumption per bit carried. For this reason when the consolidation algorithm is executed it starts selecting the server $s_{\min}$ with the minimum value of η_s as

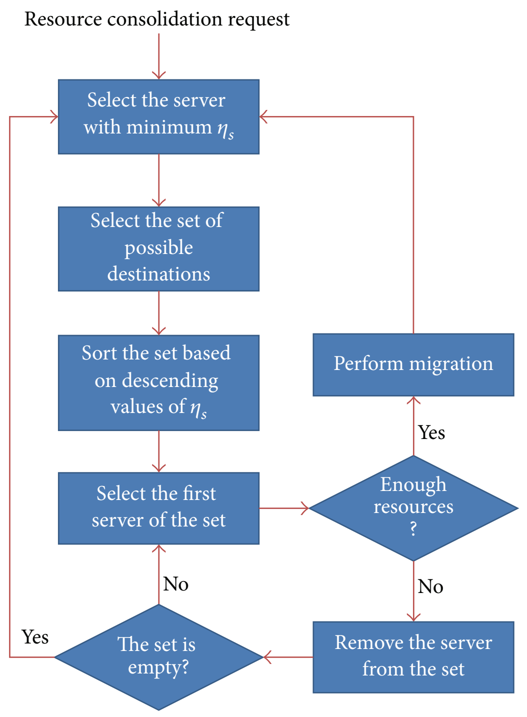

Resource consolidation request
Select the server with minimum η_s
Select the set of possible destinations
Sort the set based on descending values of η_s
Select the first server of the set
Enough resources ?
Yes
Perform migration
No
Remove the server from the set
The set is empty?
No
Yes

Figure 4: Flow chart of the SFC consolidation algorithm.

the one to turn off. A set of possible destination servers able to host the VMs executed on $s_{\min}$ is then evaluated and sorted in descending order based again on the value of η_s. The algorithm proceeds selecting the first element of the ordered set and evaluating if there are enough resources in the site to reroute all the flows generated from or terminated to $s_{\min}$. If it succeeds the migration is performed and the server $s_{\min}$ is turned off. If it fails the destination server is removed and the next server of the list is considered. When the ordered set of possible destinations becomes empty another server to turn off is selected.

The SFC Consolidation Module also manages the resource deconsolidation procedure when the reactivation of physical servers/links is needed. Basically the deconsolidation algorithm selects the most loaded VMs already migrated to a new server and moves them to their original servers. The flows handled by these VMs are accordingly rerouted.

5. Numerical Results

We evaluate the effectiveness of the proposed algorithms in the case of the network scenario reported in Figure 5. The network is composed of five core nodes, five edge nodes, and six access nodes in which the SFC requests are randomly generated and terminated. A Network Function Virtualization (NFV) site is connected to edge nodes 3 and 4 through two access routers 1 and 2. The NFV site is also composed of two switches and eight servers. In the basic configuration, 40 Gbps links are considered except the links connecting the server to the switches in the NFV sites whose rate is equal to 10 Gbps. The sixteen servers are equipped with 48 Vcores each.

Each SFC request is composed of three Virtual Network Functions (VNFs), that is, a firewall, a load balancer, and VPN encryption whose packet processing times are assumed to be equal to 7,08 μs, 0,65 μs, and 1,64 μs [27], respectively. We also assume that the load balancer splits the input traffic towards a number of output links chosen uniformly from 1 to 3.

An example of graph for a generated SFC is reported in Figure 6 in which u_t is an ingress access node and v_t^1, v_t^2, and v_t^3 are egress access nodes. The first and second VNFs are a firewall and VPN encryption; the third VNF is a load balancer splitting the traffic towards three output links. In the considered case study we also assume that the SFC handles traffic flows of packet length equal to 1500 bytes and required bandwidth uniformly distributed from 500 Mbps to 1 Gbps.

5.1. Offline SFC Scheduling. In the offline case we assume that a given number T of SFC requests are a priori known. For the case study previously described we apply the MASRN heuristic proposed in Section 3.3. We report the percentage of the dropped SFCs in Figure 7 as a function of the offered number of SFCs. We report the curve for the basic scenario and the ones in which the link capacities are doubled, quadrupled, and increased per ten times, respectively. From Figure 7 we can see how, in order to have a dropped SFC percentage lower than 10%, the number of SFCs requests has to be smaller than 60 in the case of basic scenario. This remarkable blocking is due to the shortage of network capacity. In fact we can notice from Figure 7 how the dropped SFC percentage significantly decreases when the link bandwidth increases. For instance, when the capacity is quadrupled, the network infrastructure is able to accommodate up to 180 SFCs without any loss.

This is confirmed in Figure 8 where we report the number of Vcores occupied in the servers in the case of $T = 360$. Each of the servers is represented on the x-axis with the identification (ID) of Figure 5. We also show in Figure 8 the number of Vcores allocated to each firewall, load balancer, and VPN encryption VNF with the blue, green, and red colors, respectively. The basic scenario case and the ones in which the link capacity is doubled, quadrupled, and increased by 10 times are reported in Figures 8(a), 8(b), 8(c), and 8(d), respectively. From Figure 8 we can notice how the MASRN heuristic works correctly by guaranteeing a uniform occupancy of the servers in terms of total number of Vcores used. We can also notice how the firewall VNF is the one needing the higher number of Vcores allocated. That is a consequence of the higher processing time that the firewall VNF requires with respect to the load balancer and VPN encryption VNFs.

5.2. Online SFC Scheduling. The numerical results are achieved in the following traffic scenario. The traffic pattern we consider is supposed to be sinusoidal with a peak value during daytime and minimum value during nighttime. We also assume that SFC requests follow a Poisson process and the SFC duration time is distributed according to an

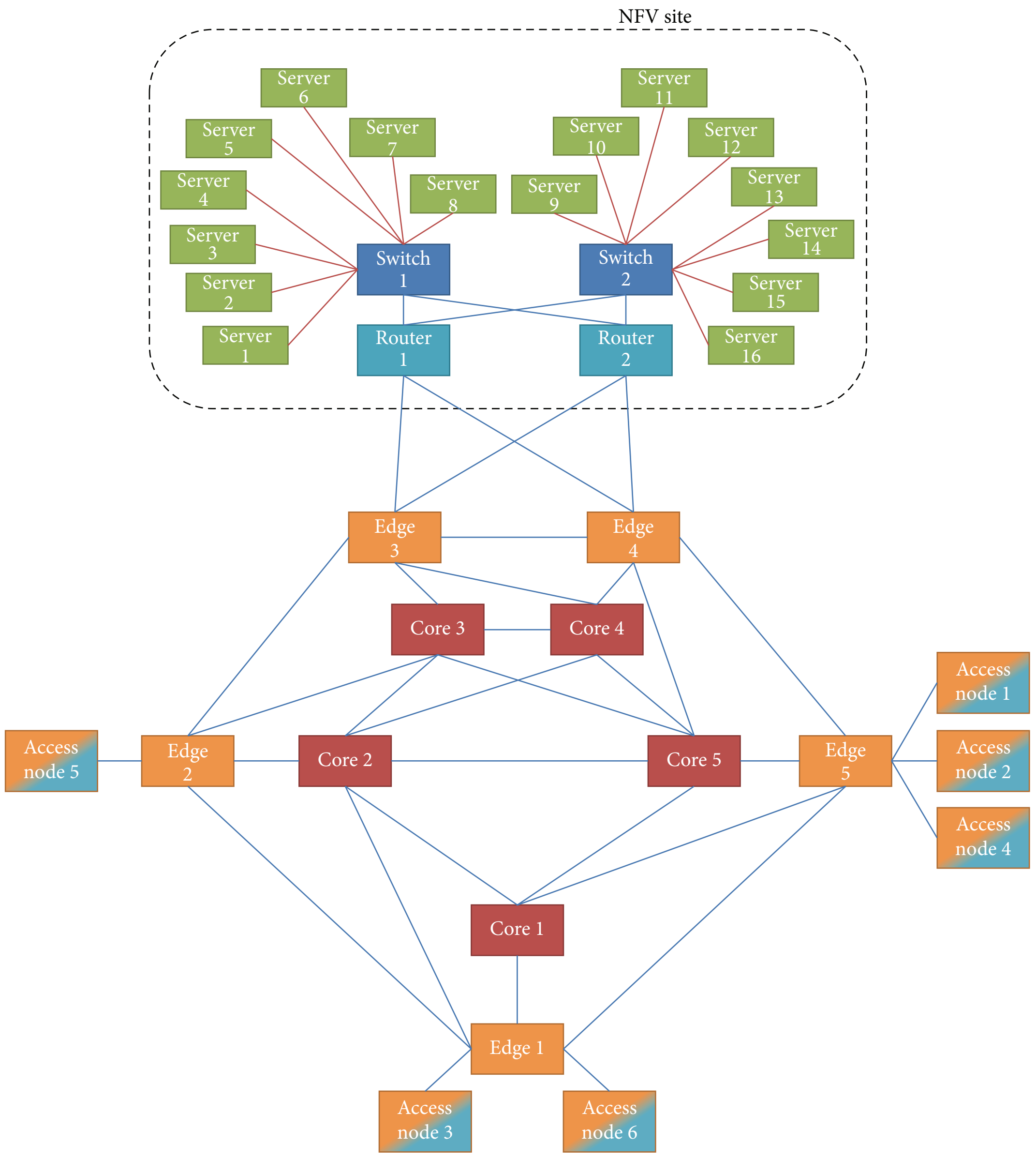


Figure 5: The network topology is composed of six access nodes, five edge nodes, and five core nodes. The NFV site is composed of two routers, two switches, and sixteen servers.

exponential distribution. The following parameters define the SFC requests in the Peak Hour Interval (PHI):

(i) λ is the arrival rate of SFC requests;

(ii) μ is the termination rate of an embedded SFC;

(iii) $[\beta^{\min}, \beta^{\max}]$ is the range in which the bandwidth request for an SFC is randomly selected.

We consider K time intervals in a day and each of them is characterized by a scaling factor α_h with $h = 0, \dots, K-1$. In order to capture the sinusoidal shape of the traffic in a day, α_h is evaluated with the following expression:

$$\alpha_h = \begin{cases} 1, & \text{if } h = 0, \\ 1 - 2\dfrac{h}{K}\left(1 - \alpha_{\min}\right), & h = 1, \dots, \dfrac{K}{2}, \\ 1 - 2\dfrac{K-h}{K}\left(1 - \alpha_{\min}\right), & h = \dfrac{K}{2} + 1, \dots, K-1, \end{cases} \quad (18)$$

where α_0 and $\alpha_{\min}$ refer to the peak traffic and the minimum traffic scenario, respectively.

The parameter α_h affects the SFC arrival rate and the bandwidth requested. In particular we have the following

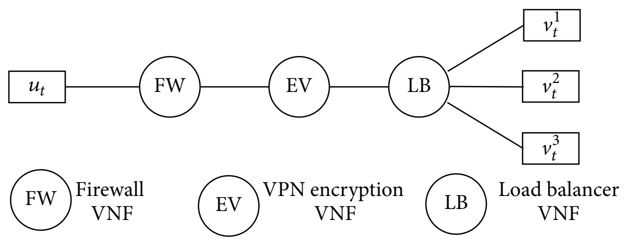


Figure 6: An example of SFC composed of one firewall VNF, one VPN encryption VNF, and one load balancer VNF that splits the input traffic towards three output logical links. u_t denotes the ingress access node while v_t^1, v_t^2, and v_t^3 denote the egress access nodes.

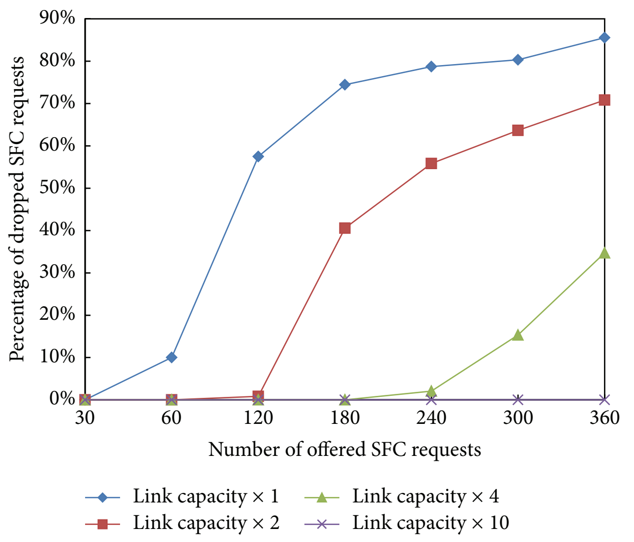


Figure 7: Dropped SFC percentage as a function of the number of SFCs offered. The four curves are reported for the basic scenario case and the ones in which the link capacities are doubled, quadrupled, and increased by 10 times.

expression for the SFC request rate λ_h and the minimum and the maximum requested bandwidths $\beta_h^{\min}$ and $\beta_h^{\max}$, respectively, in the interval h ($h = 0, \ldots, K - 1$):

$$\lambda_h = \alpha_h \lambda,$$
$$\beta_h^{\min} = \alpha_h \beta^{\min}, \tag{19}$$
$$\beta_h^{\max} = \alpha_h \beta^{\max}.$$

Next we evaluate the performance of SFC planner proposed in Section 4 in dynamic traffic scenario and for parameter values $\beta^{\min} = 500$ Mbps, $\beta^{\max} = 1$ Gbps, $\lambda = 1$ rich/min, and $\mu = 1/15\ \text{min}^{-1}$. We also assume $K = 24$ with traffic change and consequently application of the consolidation algorithm every hour.

Finally we consider servers whose power consumption is characterized by the expression (17) with parameters $P_{\max} = 1000$ W and $C_s = 10$ Gbps.

We report the SFC blocking probability as a function of the average number of offered SFC requests during the PHI (λ/μ) in Figure 9. We show the results in the case of basic link capacity scenario and in the ones obtained considering a capacity of the links that is twice and four times higher than the basic one. We consider the case of server with no rate adaptive power consumption ($a = 1$). As we can observe, when the offered traffic is low, the degradation in terms of SFC blocking probability introduced by the consolidation technique increases. In fact in this low traffic scenario, more resource consolidation is possible with the consequence of higher SFC blocking probabilities. When the offered traffic increases the blocking probability with or without applying the consolidation technique does not change much because the network is heavily loaded and only few servers can be turned off. Furthermore, as we can expect, the blocking probability decreases when the links capacity increases. The performance loss due to the application of the consolidation technique is what we have to pay in order to obtain benefits in terms of power saved by turning off servers. The curves in Figure 10 compare the power consumption percentage savings that we can achieve in the cases $a = 0.1$ and $a = 1$. The results show that when the offered traffic decreases and the link capacity increases, higher power consumption saving can be achieved. The reason is that we have more resources on the links and less loaded VMs that allow for a higher number of VM migrations and resource consolidation. The other result to notice is that the use of rate adaptive servers reduces the power consumption saving when we perform resource consolidation. This is due to the lower value P_{idle} of the constant power consumption of the server that leads to lower power consumption saving when a server is switched off.

6. Conclusions

The aim of this paper has been to propose heuristics for the resource dimensioning and the routing of Service Function Chain in network architectures employing the Network Function Virtualization technology. We have introduced the optimization problem that has the objective of minimizing the number of SFCs offered and the compliance of server processing and link bandwidth capacity. With the problem being NP-hard, we have proposed the Maximizing the Accepted SFC Requests Number (MASRN) heuristic that is based on the uniform occupancy of the server and link resources. The heuristic is also able to dimension the Vcores of the servers by evaluating the number of Vcores to be allocated to each VNF instance.

An SFC planner has been already proposed for the dynamic traffic scenario. The planner is based on a consolidation algorithm that allows for the switching off of servers during the low traffic periods. A case study has been analyzed in which we have shown that the heuristics works correctly by uniformly allocating the server resources and reserving a higher number of Vcores for the VNFs characterized by higher packet processing time. Furthermore the proposed SFC planner allows for a power consumption saving dependent on the offered traffic and varying from 10% to 70%. Such a saving is to be paid with an increase in SFC blocking probability. As future research we will propose and investigate Virtual Machine migration policies

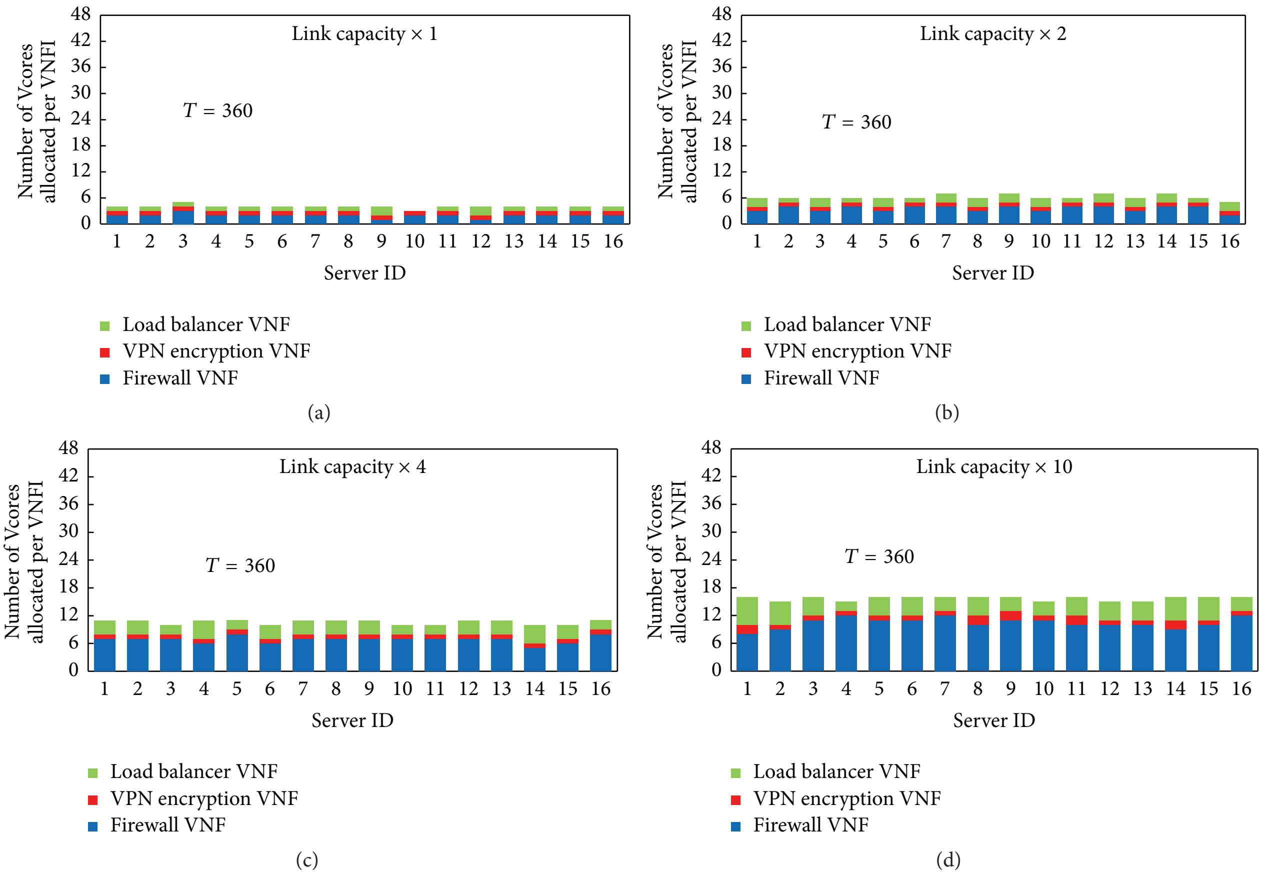


FIGURE 8: Number of allocated Vcores in each server. The number of Vcores for firewall, load balancer, and encryption VPN VNFs are represented with blue, green, and red colors.

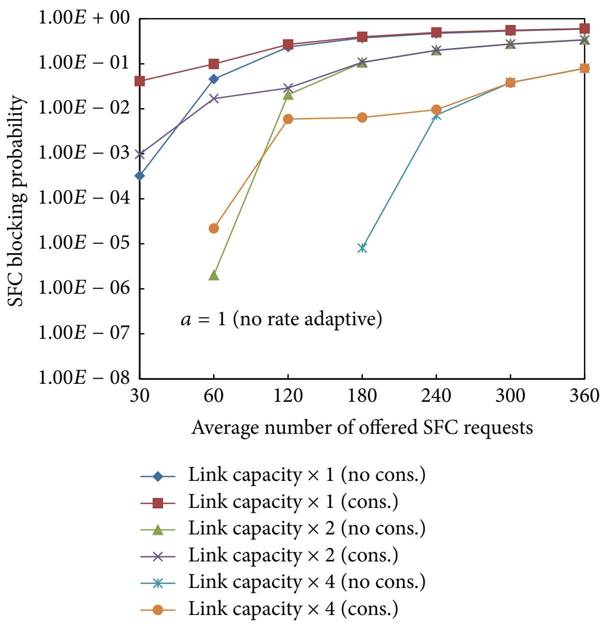


FIGURE 9: SFC blocking probability as a function of the average number of SFCs offered in the PHI for the cases in which the consolidation technique is applied and it is not. The server power consumption is constant ($a = 1$).

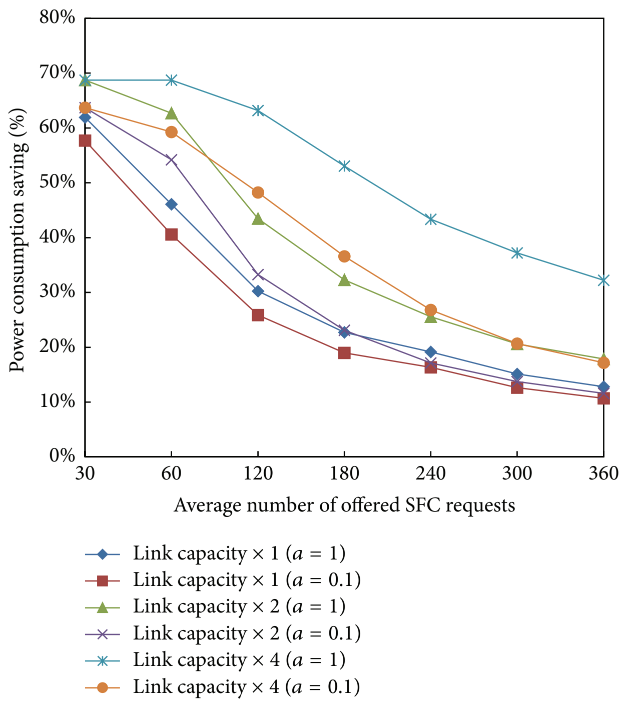


FIGURE 10: The power consumption percentage saving achieved with the application of the consolidation technique versus the average number of SFCs offered in the PHI. The cases $a = 1$ and $a = 0.1$ are considered.

allowing for the achievement of a right trade-off between power consumption saving and SFC blocking probability degradation.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The work described in this paper has been funded by Telecom Italia within the research agreement titled "Definition and Evaluation of Protocols and Architectures of a Network Automation Platform for TI-NVF Infrastructure."

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[2] P. Veitch, M. J. McGrath, and V. Bayon, "An instrumentation and analytics framework for optimal and robust NFV deployment," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 126–133, 2015.

[3] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Network function virtualization in the multi-tenant cloud," *IEEE Network*, vol. 29, no. 3, pp. 42–47, 2015.

[4] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform handling and abstraction of NFV hardware accelerators," *IEEE Network*, vol. 29, no. 3, pp. 22–29, 2015.

[5] 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.

[6] ETSI Industry Specification Group (ISG) NFV, "ETSI Group Specifications on Network Function Virtualization. 1st Phase Documents," January 2015, http://docbox.etsi.org/ISG/NFV/Open/.

[7] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[8] The Internet Engineering Task Force (IETF), *Service Function Chaining (SFC) Working Group (WG)*, 2015, https://datatracker.ietf.org/wg/sfc/charter/.

[9] The Internet Engineering Task Force (IETF), Service Function Chaining (SFC) Working Group (WG), Documents, 2015, https://datatracker.ietf.org/wg/sfc/documents/.

[10] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "MORSA: a multi-objective resource scheduling algorithm for NFV infrastructure," in *Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium (APNOMS '14)*, pp. 1–6, Hsinchum, Taiwan, September 2014.

[11] H. Moens and F. D. Turck, "Vnf-p: a model for efficient placement of virtualized network functions," in *Proceedings of the 10th International Conference on Network and Service Management (CNSM '14)*, pp. 418–423, November 2014.

[12] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft '15)*, pp. 1–9, University College London, London, UK, April 2015.

[13] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, Krakow, Poland, May 2014.

[14] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pp. 1–12, IEEE, Barcelona, Spain, April 2006.

[15] G. Lee, M. Kim, S. Choo, S. Pack, and Y. Kim, "Optimal flow distribution in service function chaining," in *Proceedings of the 10th International Conference on Future Internet (CFI '15)*, pp. 17–20, ACM, Seoul, Republic of Korea, June 2015.

[16] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: a survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[17] M. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 177–184, Ghent, Belgium, May 2013.

[18] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann, "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem," in *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications and Challenges (AllThingsCellular '14)*, pp. 33–38, ACM, Chicago, Ill, USA, August 2014.

[19] M. Bagaa, T. Taleb, and A. Ksentini, "Service-aware network function placement for efficient traffic handling in carrier cloud," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '14)*, pp. 2402–2407, IEEE, Istanbul, Turkey, April 2014.

[20] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proceedings of the IEEE 3rd International Conference on Cloud Networking (CloudNet '14)*, pp. 7–13, October 2014.

[21] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft '15)*, pp. 1–9, London, UK, April 2015.

[22] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for NFV chaining in packet/optical datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, Article ID 7005460, pp. 1565–1570, 2015.

[23] O. Hyeonseok, Y. Daeun, C. Yoon-Ho, and K. Namgi, "Design of an efficient method for identifying virtual machines compatible with service chain in a virtual network environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 11, no. 9, Article ID 197208, 2014.

[24] W. Cerroni and F. Callegati, "Live migration of virtual network functions in cloud-based edge networks," in *Proceedings of the IEEE International Conference on Communications (ICC '14)*, pp. 2963–2968, IEEE, Sydney, Australia, June 2014.

[25] P. Quinn and J. Guichard, "Service function chaining: creating a service plane via network service headers," *Computer*, vol. 47, no. 11, pp. 38–44, 2014.

[26] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba, "VDC Planner: dynamic migration-aware Virtual Data Center embedding for clouds," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 18–25, May 2013.

[27] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward predictable performance in software packet-processing platforms," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, pp. 1–14, April 2012.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 4067186, 10 pages
http://dx.doi.org/10.1155/2016/4067186

Research Article

A Game for Energy-Aware Allocation of Virtualized Network Functions

Roberto Bruschi,[1] Alessandro Carrega,[1,2] and Franco Davoli[1,2]

[1] *CNIT-University of Genoa Research Unit, 16145 Genoa, Italy*
[2] *DITEN, University of Genoa, 16145 Genoa, Italy*

Correspondence should be addressed to Alessandro Carrega; alessandro.carrega@unige.it

Received 2 October 2015; Revised 22 December 2015; Accepted 11 January 2016

Academic Editor: Xavier Hesselbach

Network Functions Virtualization (NFV) is a network architecture concept where network functionality is virtualized and separated into multiple building blocks that may connect or be chained together to implement the required services. The main advantages consist of an increase in network flexibility and scalability. Indeed, each part of the service chain can be allocated and reallocated at runtime depending on demand. In this paper, we present and evaluate an energy-aware Game-Theory-based solution for resource allocation of Virtualized Network Functions (VNFs) within NFV environments. We consider each VNF as a player of the problem that competes for the physical network node capacity pool, seeking the minimization of individual cost functions. The physical network nodes dynamically adjust their processing capacity according to the incoming workload, by means of an Adaptive Rate (AR) strategy that aims at minimizing the product of energy consumption and processing delay. On the basis of the result of the nodes' AR strategy, the VNFs' resource sharing costs assume a polynomial form in the workflows, which admits a unique Nash Equilibrium (NE). We examine the effect of different (unconstrained and constrained) forms of the nodes' optimization problem on the equilibrium and compare the power consumption and delay achieved with energy-aware and non-energy-aware strategy profiles.

1. Introduction

In the last few years, power consumption has shown a growing and alarming trend in all industrial sectors, particularly in Information and Communication Technology (ICT). Public organizations, Internet Service Providers (ISPs), and telecom operators started reporting alarming statistics of network energy requirements and of the related carbon footprint since the first decade of the 2000s [1]. The Global e-Sustainability Initiative (GeSI) estimated a growth of ICT greenhouse gas emissions (in $GtCO_2e$, Gt of CO_2 equivalent gases) to 2.3% of global emissions (from 1.3% in 2002) in 2020, if no Green Network Technologies (GNTs) would be adopted [2]. On the other hand, the abatement potential of ICT in other industrial sectors is seven times the size of the ICT sector's own carbon footprint.

Only recently, due to the rise in energy price, the continuous growth of customer population, the increase in broadband access demand, and the expanding number of services being offered by telecoms and providers, has energy efficiency become a high-priority objective also for wired networks and service infrastructures (after having started to be addressed for datacenters and wireless networks).

The increasing network energy consumption essentially depends on new services offered, which follow Moore's law, by doubling every two years, and on the need to sustain an ever-growing population of users and user devices. In order to support new generation network infrastructures and related services, telecoms and ISPs need a larger equipment base, with sophisticated architecture able to perform more and more complex operations in a scalable way. Notwithstanding these efforts, it is well known that most networks and networking equipment are currently still provisioned for busy or rush hour load, which typically exceeds their average utilization by a wide margin. While this margin is generally reached in rare and short time periods, the overall power consumption in today's networks remains more or less constant with respect to different traffic utilization levels.

The growing trend toward implementing networking functionalities by means of software [3] on general-purpose machines and of making more aggressive use of virtualization—as represented by the paradigms of Software Defined Networking (SDN) [4] and Network Functions Virtualization (NFV) [5]—would also not be sufficient in itself to reduce power consumption, unless accompanied by "green" optimization and consolidation strategies acting as energy-aware traffic engineering policies at the network level [6]. At the same time, processing devices inside the network need to be capable of adapting their performance to the changing traffic conditions, by trading off power and Quality of Service (QoS) requirements. Among the various techniques that can be adopted to this purpose to implement Control Policies (CPs) in network processing devices, Dynamic Adaptation ones consist of adapting the processing rate (AR) or of exploiting low power consumption states in idle periods (LPI) [7].

In this paper, we introduce a Game-Theory-based solution for energy-aware allocation of Virtualized Network Functions (VNFs) within NFV environments. In more detail, in NFV networks, a collection of service chains must be allocated on physical network nodes. A service chain is a set of one or more VNFs grouped together to provide specific service functionality and can be represented by an oriented graph, where each node corresponds to a particular VNF and each edge describes the operational flow exchanged between a pair of VNFs.

A service request can be allocated on dedicated hardware or by using resources deployed by the Service Provider (SP) that processes the request through virtualized instances. Because of this, two types of service deployments are possible in an NFV network: (i) on physical nodes and (ii) on virtualized instances.

In this paper, we focus on the second type of service deployment. We refer to this paradigm as pure NFV. As already outlined above, the SP processes the service request by means of VNFs. In particular, we developed an energy-aware solution to the problem of VNFs' allocation on physical network nodes. This solution is based on the concept of Game Theory (GT). GT is used to model interactions among self-interested players and predict their choice of strategies to optimize cost or utility functions, until a Nash Equilibrium (NE) is reached, where no player can further increase its corresponding utility through individual action (see, e.g., [8] for specific applications in networking).

More specifically, we consider a bank of physical network nodes (in this paper, we also use the terms node and resource to refer to the physical network node) performing tasks on requests submitted by players' population. Hence, in this game, the role of the players is represented by VNFs that compete for the processing capacity pool, each by seeking the minimization of an individual cost function. The nodes can dynamically adjust their processing capacity according to the incoming workload (the processing power required by incoming VNF requests) by means of an AR strategy that aims at minimizing the product of energy consumption and processing delay. On the basis of the result of the nodes' AR strategy, the VNFs' resource sharing costs assume a polynomial form in the workloads, which admits a unique NE. We examine the effect of different (unconstrained and constrained) forms of the nodes' optimization problem on the equilibrium and compare the power consumption and delay achieved with energy-aware and non-energy-aware strategy profiles.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 briefly summarizes the power model and AR optimization that was introduced in [9]. Although the scenario in [9] is different, it is reasonable to assume that similar considerations are valid here, too. On the basis of this result, we derive a number of competitive strategies for the VFNs' allocation in Section 4. Section 5 presents various numerical evaluations, and Section 6 contains the conclusions.

2. Related Work

We now discuss the most relevant works that deal with the resource allocation of VNFs within NFV environments. We decided not only to describe solutions based on GT, but also to provide an overview of the current state of the art in this area. As introduced in the previous section, the key enabling paradigms that will considerably affect the dynamics of ICT networks are SDN and NFV, which are discussed in recent surveys [10–12]. Indeed, SPs and Network Operators (NOs) are facing increasing problems to design and implement novel network functionalities, following rapid changes that characterize the current ISPs and telecom operators (TOs) [13].

Virtualization represents an efficient and cost-effective strategy to exploit and share physical network resources. In this context, the Network Embedding Problem (NEP) has been considered in several recent works [14–19]. In particular, the Virtual Network Embedding Problem (VNEP) consists of finding the mapping between a set of requests for virtual network resources and the available underlying physical infrastructure (the so-called *substrate*), ensuring that some given performance requirements (on nodes and links) are guaranteed. Typical node requirements are computational resources (i.e., CPU) or storage space, whereas links have a limited bandwidth and introduce a delay. It has been shown that this problem is NP-hard (it includes as subproblem the multiway separator problem). For this reason, heuristic approaches have been devised [20].

The consolidation of virtual resources is considered in [18], by taking into account energy efficiency. The problem is formulated as a mixed integer linear programming (MILP) model, to understand the potential benefits that can be achieved by packing many different virtual tasks on the same physical infrastructure. The observed energy saving is up to 30% for nodes and up to 25% for link energy consumption. Reference [21] presents a solution for the resilient deployment of network functions, using OpenStack for the design and implementation of the proposed service orchestrator mechanism.

An allocation mechanism, based on auction theory, is proposed in [22]. In particular, the scheme selects the most remunerative virtual network requests, while satisfying QoS requirements and physical network constraints. The system is split into two network substrates modeling physical and virtual resources, with the final goal of finding the best mapping of virtual nodes and links onto physical ones according to the QoS requirements (i.e., bandwidth, delay, and CPU bounds).

A novel network architecture is proposed in [23], to provide efficient coordinated control of both internal network function state and network forwarding state, in order to help operators achieve the following goals: (i) offering and satisfying tight service level agreements (SLAs); (ii) accurately monitoring and manipulating network traffic; and (iii) minimizing operating expenses.

Various engineering problems, where the action of one component has some impacts on the other components, have been modeled by GT. Indeed, GT, which has been applied at the beginning in economics and related domains, is gaining much interest today as a powerful tool to analyze and design communication networks [24]. Therefore, the problems can be formulated in the GT framework, and a stable solution for the components is obtained using the concept of equilibrium [25]. In this regard, GT has been used extensively to develop understanding of stable operating points for autonomous networks. The nodes are considered as the players. Payoff functions are often defined according to achieved connection bandwidth or similar technical metrics.

To construct algorithms with provable convergence to equilibrium points, many approaches consider network models that can be mapped to specially constructed games. Among this type of games, potential games use a real-valued function that represents the entire player set to optimize some performance metric [8, 26]. We mention Altman et al. [27], who provide an extensive survey on networking games. The models and papers discussed in this reference mostly deal with noncooperative GT, the only exception being a short section focused on bargaining games. Finally, Seddiki et al. [25] presented an approach based on two-stage noncooperative games for bandwidth allocation, which aims at reducing the complexity of network management and avoiding bandwidth performance problems in a virtualized network environment. The first stage of the game is the bandwidth negotiation, where the SP requests bandwidth from multiple Infrastructure Providers (InPs). Each InP decides whether to accept or deny the request when the SP would cause link congestion. The second stage is the bandwidth provisioning game, where different SPs compete for the bandwidth capacity of a physical link managed by a single InP.

3. Modeling Power Management Techniques

Dynamic Adaptation techniques in physical network nodes reduce the energy usage, by exploiting the fact that systems do not need to run at peak performance all the time.

Rate adaptation is obtained by tuning the clock frequency and/or voltage of processors (DVFS, Dynamic Voltage and Frequency Scaling) or by throttling the CPU clock (i.e., the clock signal is disabled for some number of cycles at regular intervals). Decreasing the operating frequency and the voltage of the node, or throttling its clock, obviously allows the reduction of power consumption and of heat dissipation, at the price of slower performance.

Advantage can be taken of these adaptation capabilities to devise control techniques based on feedback on the incoming load. One of the first proposals is that examined in a seminal paper by Nedevschi et al. [28]. Among other possibilities, we consider here the simple optimization strategy developed in [9] aimed at minimizing the product of power consumption and processing delay with respect to the processing load. The application of this control technique gives rise to quadratic dependence of the power-delay product on the load, which will be exploited in our subsequent game theoretic resource allocation problem. Unlike the cited works, our solution considers as load the requests rate of services that the SP must process. However, apart from this difference, the general considerations described are valid here, too.

Specifically, the dynamic frequency-dependent part of the processor power consumption is proportional to the clock frequency ν and to the square of the supply voltage V [29]. However, if DVFS is used, there is proportionality between the frequency and the voltage raised to some power γ, $0 < \gamma \leq 1$ [30]. Moreover, the power scaling effect induced by alternating active-idle periods in the queueing system served by the physical network node can be accounted for by multiplying the dynamic power consumption by an increasing concave function of the resource utilization of the form $(f/\mu)^{1/\alpha}$, where $\alpha \geq 1$ is a parameter and f and μ are the workload (processing requests per unit time) and the task processing speed, respectively. By taking $\gamma = 1$ (which implies a cubic relationship in the operating frequency), the overall dependence of the power consumption $\mathbf{\Phi}$ on the processing speed (which is proportional to the operating frequency) can be expressed as [9]

$$\mathbf{\Phi}(\mu) = K\mu^3 \left(\frac{f}{\mu}\right)^{1/\alpha}, \tag{1}$$

where $K > 0$ is a proportionality constant. This expression will be exploited in the optimization problems to be defined and studied in the next section.

4. System Model of Coordinated Network Management Optimization and Players' Game

We consider the situation shown in Figure 1. There are S VNFs that are represented by the incoming workload rates $\lambda_1, \ldots, \lambda_S$, competing for N physical network nodes. The workload to the jth node is given by

$$f_j = \sum_{i=1}^{S} f_j^i, \tag{2}$$

where f_j^i is VNF i's contribution. The total workload vector of player i is $\mathbf{f}^i = \mathrm{col}[f_1^i, \ldots, f_N^i]$ and, obviously,

$$\sum_{j=1}^{N} f_j^i = \lambda_i. \tag{3}$$

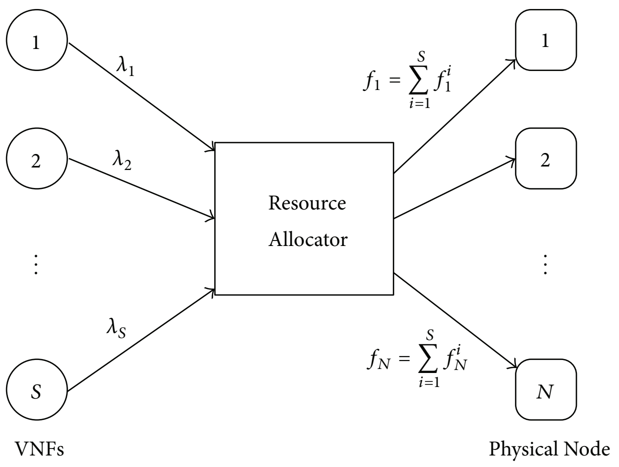


Figure 1: System model for VNFs resource allocation.

The computational resources have processing rates μ_j, $j = 1, \dots, N$, and we associate a cost function with each one; namely,

$$c_j(\mu_j) = \frac{K_j (f_j)^{1/\alpha_j} (\mu_j)^{3-(1/\alpha_j)}}{\mu_j - f_j}. \tag{4}$$

Cost functions of the form shown in (4) have been introduced in [9] and represent the product of power and processing delay, under $M/M/1$ assumption on each node's queue. They actually correspond to the average energy consumption that can be attributed to functions' handling (considering that the node will be active whenever there are queued functions). Despite the possible inaccuracy in the representation of the queueing behavior, the denominator of (4) reflects the penalty paid for approaching the resource capacity, which is a major aspect in a model to be used for control purposes.

We now consider two specific control problems, whose resulting resource allocations are interconnected. Specifically, we assume the presence of Control Policies (CPs) acting in the network node, whose aim is to dynamically adjust the processing rate, in order to minimize cost functions of the form of (4). On the other hand, the players representing the VNFs, knowing the policy adopted by the CPs and the resulting form of their optimal costs, implement their own strategies to distribute their requests among the different resources.

The simple control structure that we envisage actually represents a situation that may be of interest in a number of different operational settings. We only mention two different cases of current high relevance: (i) a multitenant environment, where a number of ISPs are offered services by a datacenter operator (or by a telecom operator in the access network) on a number of virtual machines, and (ii) a collection of virtual environments created by a SP on behalf of its customers, where services are activated to represent customers' functionalities on their own private virtual LANs. It is worth noting that in both cases the nodes' customers may be unwilling to disclose their loads to one another, which justifies the decentralized game optimization.

4.1. Nodes' Control Policies.

We consider two possible variants.

4.1.1. CP1 (Unconstrained).

The direct minimization of (4) with respect to μ_j yields immediately:

$$\begin{aligned} \mu_j^*(f_j) &= \frac{3\alpha_j - 1}{2\alpha_j - 1} f_j = \vartheta_j f_j, \\ c_j^*(f_j) &= K_j \frac{(\vartheta_j)^{3-(1/\alpha_j)}}{\vartheta_j - 1} (f_j)^2 = h_j \cdot (f_j)^2. \end{aligned} \tag{5}$$

We must note that we are considering a continuous solution to the node capacity adjustment. In practice, the physical resources allow a discrete set of working frequencies, with corresponding processing capacities. This would also ensure that the processing speed does not decrease below a lower threshold, avoiding excessive delay in the case of low load. The unconstrained problem is an idealized variant that would make sense only when the load on the node is not too small.

4.1.2. CP2 (Individual Constraints).

Each node has a maximum and a minimum processing capacity, which are taken explicitly into account ($\mu_j^{\min} \le \mu_j \le \mu_j^{\max}$, $j = 1, \dots, N$). Then,

$$\mu_j^*(f_j) = \begin{cases} \vartheta_j f_j, & \underline{f}_j = \dfrac{\mu_j^{\min}}{\vartheta_j} \le f_j \le \dfrac{\mu_j^{\max}}{\vartheta_j} = \overline{f}_j, \\ \mu_j^{\min}, & 0 < f_j < \underline{f}_j, \\ \mu_j^{\max}, & \mu_j^{\max} > f_j > \overline{f}_j. \end{cases} \tag{6}$$

Therefore,

$$c_j^*(f_j) = \begin{cases} h_j \cdot (f_j)^2, & \underline{f}_j \le f_j \le \overline{f}_j, \\ K_j \dfrac{(f_j)^{1/\alpha_j} (\mu_j^{\max})^{3-(1/\alpha_j)}}{\mu_j^{\max} - f_j}, & \mu_j^{\max} > f_j > \overline{f}_j, \\ K_j \dfrac{(f_j)^{1/\alpha_j} (\mu_j^{\min})^{3-(1/\alpha_j)}}{\mu_j^{\min} - f_j}, & 0 < f_j < \underline{f}_j. \end{cases} \tag{7}$$

In this case, we assume explicitly that $\sum_{i=1}^{S} \lambda_i < \sum_{j=1}^{N} \mu_j^{\max}$.

4.2. Noncooperative Players' Optimization.

Given the optimal CPs, which can be found in functional form, we can then state the following.

4.2.1. Players' Problem.

Each VNF i wants to minimize, with respect to its workload vector $\mathbf{f}^i = \mathrm{col}[f_1^i, \dots, f_N^i]$, a weighted (over its workload distribution) sum of the resources' costs, given the flow distribution of the others:

$$
\begin{aligned}
f^{i^*} &= \underset{f_1^i,\ldots,f_N^i:\sum_{j=1}^N f_j^i=\lambda_i}{\operatorname{argmin}} J^i \\
&= \underset{f_1^i,\ldots,f_N^i:\sum_{j=1}^N f_j^i=\lambda_i}{\operatorname{argmin}} \frac{1}{\lambda_i}\sum_{j=1}^N f_j^i c_j^*\left(f_j^i\right) \quad i = 1,\ldots,S.
\end{aligned}
\tag{8}
$$

In this formulation, the players' problem is a noncooperative game, of which we can seek a NE [31].

We examine the case of the application of CP1 first. Then, the cost of VNF i is given by

$$J^i = \frac{1}{\lambda_i}\sum_{j=1}^N f_j^i h_j \cdot \left(f_j\right)^2 = \frac{1}{\lambda_i}\sum_{j=1}^N f_j^i h_j \cdot \left(f_j^i + f_j^{-i}\right)^2, \tag{9}$$

where $f_j^{-i} = \sum_{k\neq i} f_j^k$ is the total flow from the other VNFs to node j.

The cost in (9) is convex in f_j^i and belongs to a category of cost functions previously investigated in the networking literature, without specific reference to energy efficiency [32, 33]. In particular, it is in the family of functions considered in [33], for which the NE Point (NEP) existence and uniqueness conditions of Rosen [34] have been shown to hold. Therefore, our players' problem admits a unique NEP. The latter can be found by considering the Karush-Kuhn-Tucker stationarity conditions with Lagrange multiplier ξ_i:

$$
\begin{aligned}
\xi_i &= \frac{\partial J^i}{\partial f_k^i}, \quad f_k^i > 0, \\
\xi_i &\le \frac{\partial J^i}{\partial f_k^i}, \quad f_k^i = 0, \\
&\qquad k = 1,\ldots,N,
\end{aligned}
\tag{10}
$$

from which, for $f_k^i > 0$,

$$
\begin{aligned}
\lambda_i\xi_i &= h_k\left(f_k^i + f_k^{-i}\right)^2 + 2h_k f_k^i\left(f_k^i + f_k^{-i}\right), \\
f_k^i &= -\frac{2}{3}f_k^{-i} \pm \frac{1}{3h_k}\sqrt{\left(h_k f_k^{-i}\right)^2 + 3h_k\lambda_i\xi_i}.
\end{aligned}
\tag{11}
$$

Excluding the negative solution, the nonzero components are those with the smallest and equal partial derivatives that, in a subset $\mathcal{M} \subseteq \{1,\ldots,N\}$, yield $\sum_{j\in\mathcal{M}} f_j^i = \lambda_i$; that is,

$$\sum_{j\in\mathcal{M}}\left[-\frac{2}{3}f_j^{-i} + \sqrt{4\left(h_j f_j^{-i}\right)^2 + 3h_j\lambda_i\xi_i}\right] = \lambda_i \tag{12}$$

from which ξ_i can be determined.

As regards form (7) of the optimal node cost, we can note that if we are restricted to $\underline{f}_j \le f_j \le \overline{f}_j$, $j = 1,\ldots,N$ (a coupled convex constraint set), the conditions of Rosen still hold. If we allow the larger constraint set $0 < f_j < \mu_j^{\max}$, $j = 1,\ldots,N$, the nodes' optimal cost functions are no longer of polynomial type. However, the composite function is still continuously differentiable (see the Appendix). Then, it would (i) satisfy the conditions for a convex game (the overall function composed of the three parts is convex), which guarantee the existence of a NEP [34, Th. 1], and (ii) possess equivalent properties to functions of Type A as defined in [32], which lead to uniqueness of the NEP.

5. Numerical Results

In order to evaluate our criterion, we present numerical results deriving from the application of the simplest Control Policy (CP1), which implies the solution of a completely quadratic problem (corresponding to costs as in (9) for the NEP). We compare the resulting allocations, power consumption, and average delays with those stemming from the application (on non-energy-aware nodes) of the algorithm for the minimization of the pure delay function that was developed in [35, Proposition 1]. This algorithm is considered a valid reference point in order to provide good validation of our criterion. The corresponding cost of VNF i is

$$J_T^i = \sum_{j=1}^N f_j^i T_j\left(f_j\right), \quad i = 1,\ldots,S, \tag{13}$$

where

$$T_j\left(f_j\right) = \begin{cases} \dfrac{1}{\mu_j^{\max} - f_j}, & f_j < \mu_j^{\max}, \\ \infty, & f_j \ge \mu_j^{\max}. \end{cases} \tag{14}$$

The total demand is assumed to be less than the total operating capacity, as we have done with our CP2 in (7).

To make the comparison fair, we have to make sure that the maximum operating capacities $\mu_j^{\max}$, $j = 1,\ldots,N$ (that are constant in (14)), are compatible with the quadratic behavior stemming from the application of CP1. To this aim, let ${}^{(\mathrm{LCP1})}\overline{f}_j$ denote the total input workload to node j corresponding to the NEP obtained by our problem; we then choose

$$\mu_j^{\max} = \vartheta_j\,{}^{(\mathrm{LCP1})}\overline{f}_j, \quad j = 1,\ldots,N. \tag{15}$$

We indicate with ${}^{(T)}f_j, {}^{(T)}f_j^i$, $j = 1,\ldots,N$, $i = 1,\ldots,S$, the total node input workloads and those corresponding to VNF i, respectively, produced by the NEP deriving from the minimization of costs in (13). The average delays for the flow of player i under the two strategy profiles are obtained as

$$\begin{aligned} D_T^i &= \frac{1}{\lambda_i}\sum_{j=1}^{N}\frac{{}^{(T)}f_j^i}{\mu_j^{\max} - {}^{(T)}f_j} \\ &= \frac{1}{\lambda_i}\sum_{j=1}^{N}\frac{{}^{(T)}f_j^i}{\vartheta_j{}^{(\mathrm{LCP1})}\overline{f}_j - {}^{(T)}f_j}, \\ D_{\mathrm{LCP1}}^i &= \frac{1}{\lambda_i}\sum_{j=1}^{N}\frac{{}^{(\mathrm{LCP1})}\overline{f}_j^i}{\vartheta_j{}^{(\mathrm{LCP1})}\overline{f} - {}^{(\mathrm{LCP1})}\overline{f}_j} \\ &= \frac{1}{\lambda_i}\sum_{j=1}^{N}\frac{{}^{(\mathrm{LCP1})}\overline{f}_j^i}{\left(\vartheta_j - 1\right){}^{(\mathrm{LCP1})}\overline{f}_j}, \end{aligned} \tag{16}$$

and the power consumption values are given by

$$\begin{aligned} {}^{(T)}P_j &= K_j\left(\mu_j^{\max}\right)^3 = K_j\left(\vartheta_j{}^{(\mathrm{LCP1})}\overline{f}_j\right)^3, \\ {}^{(\mathrm{LCP1})}P_j &= K_j\left(\vartheta_j{}^{(\mathrm{LCP1})}\overline{f}_j\right)^3\left(\frac{\overline{f}_j}{\vartheta_j{}^{(\mathrm{LCP1})}\overline{f}_j}\right)^{1/\alpha_j} \\ &= K_j\left(\vartheta_j{}^{(\mathrm{LCP1})}\overline{f}_j\right)^3\vartheta_j^{-1/\alpha_j}. \end{aligned} \tag{17}$$

Our data for the comparison are organized as follows. We consider normalized input rates, so that $\lambda_i \le 1,\ i = 1,\dots,S$. We generate randomly S values corresponding to the VNFs' input rates from independent uniform distributions; then,

(1) we find the NEP of our quadratic game, by choosing the parameters $K_j, \alpha_j,\ j = 1,\dots,N$, also from independent uniform distributions (with $1 \le K_j \le 10,\ 2 \le \alpha_j \le 3$, resp.);

(2) by using the workload profiles obtained, we compute the values $\mu_j^{\max},\ j = 1,\dots,N$, from (15) and find the NEP corresponding to costs in (13) and (14), by using the same input rates;

(3) we compute the corresponding average delays and power consumption values for the two cases, by using (16) and (17), respectively.

Steps (1)–(3) are repeated with a new configuration of random values of the input rates for a certain number of times; then, for both games, we average the values of the delays and power consumption per VNF, and of the total delay and power consumption averaged over the VNFs, over all trials. We compare the results obtained in this way for four different settings of VNFs and nodes; namely, $S = N = 3$; $S = 3,\ N = 5$; $S = 5,\ N = 3$; $S = 5,\ N = 5$. The rationale of repeating the experiments with random configurations is to explore a number of different possible settings and collect the results in a synthetic form.

For each VNFs-nodes setting, 30 random input configurations are generated to produce the final average values. In Figures 2–10, the labels EE and NEE refer to the energy-efficient case (quadratic cost) and to the non-energy-efficient

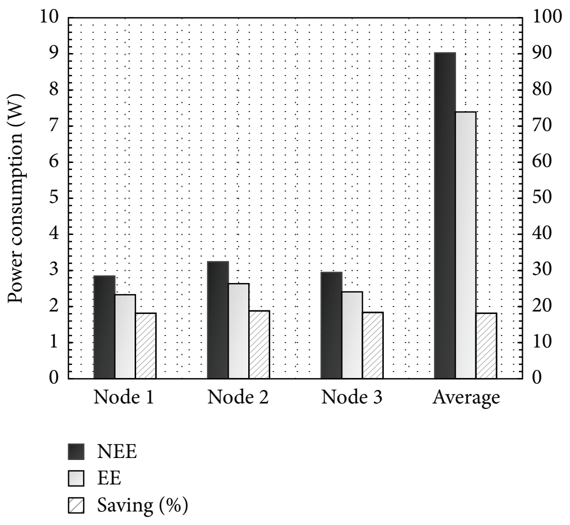


Figure 2: Power consumption with $S = N = 3$.

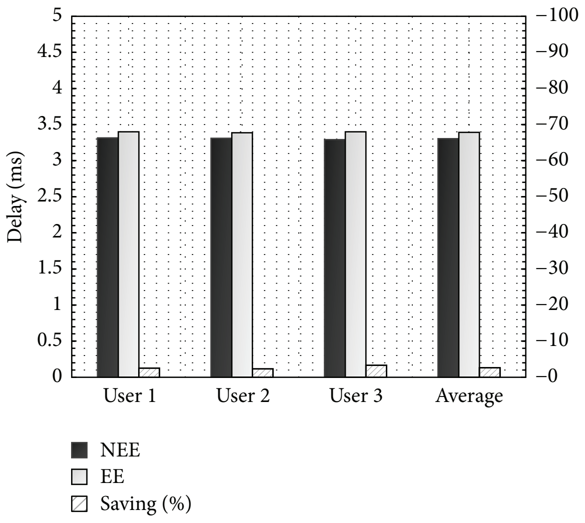


Figure 3: User (VNF) delays with $S = N = 3$.

one (see (13) and (14)), respectively. Instead, the label "Saving (%)" refers to the saving (in percentage) of the EE case compared to the NEE one. When it is negative, the EE case presents higher values than the NEE one. Finally, the label "User" refers to the VNF.

The results are reported in Figures 2–10. The model implementation and the relative results have been developed with MATLAB® [36].

In general, the energy-efficient optimization tends to save energy at the expense of a relatively small increase in delay. Indeed, as shown in the figures, the saving is positive for the energy consumption and negative for delay. The energy saving is always higher than 18% in every case, while the delay increase is always lower than 4%.

However, it can be noted (by comparing, e.g., Figures 5 and 7) that configurations with lower load are penalized in the delay. This effect is caused by the behavior of the simple linear adaptation strategy, which maintains constant utilization, and

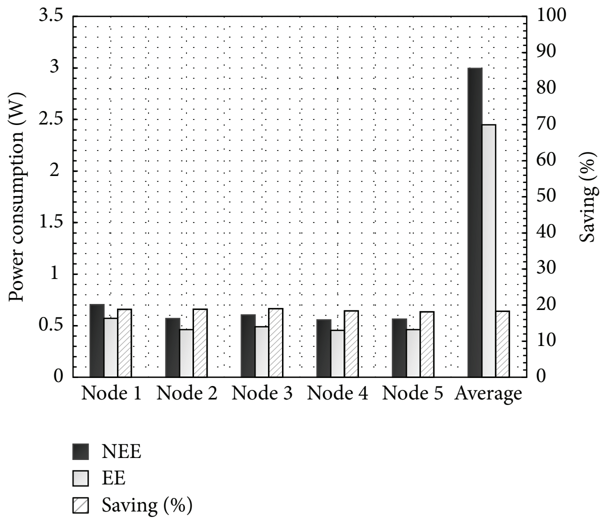


Figure 4: Power consumption with $S = 3$, $N = 5$.

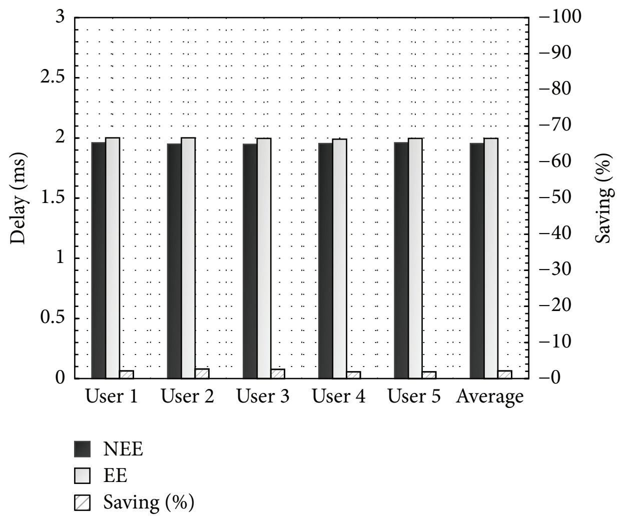


Figure 7: User (VNF) delays with $S = 5$, $N = 3$.

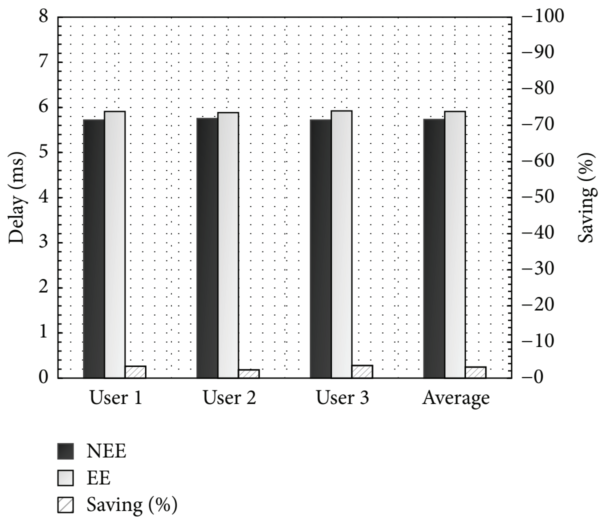


Figure 5: User (VNF) delays with $S = 3$, $N = 5$.

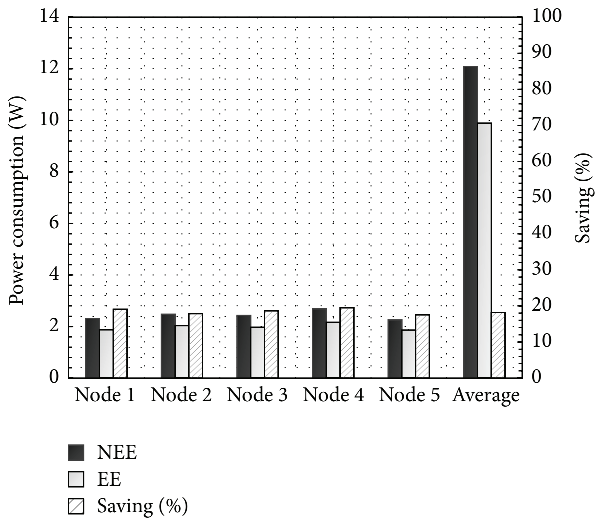


Figure 8: Power consumption with $S = 5$, $N = 5$.

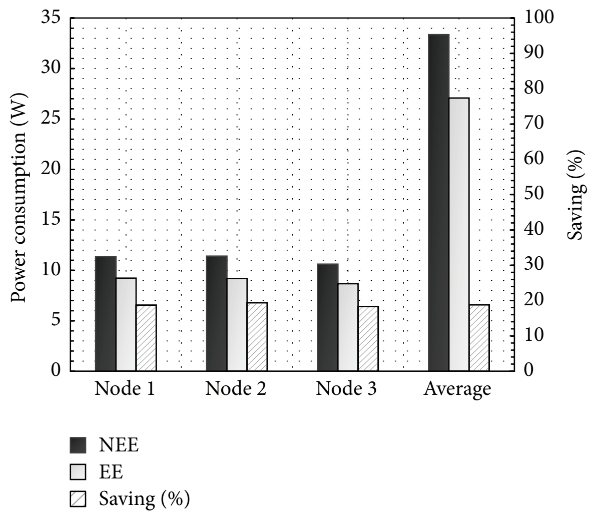


Figure 6: Power consumption with $S = 5$, $N = 3$.

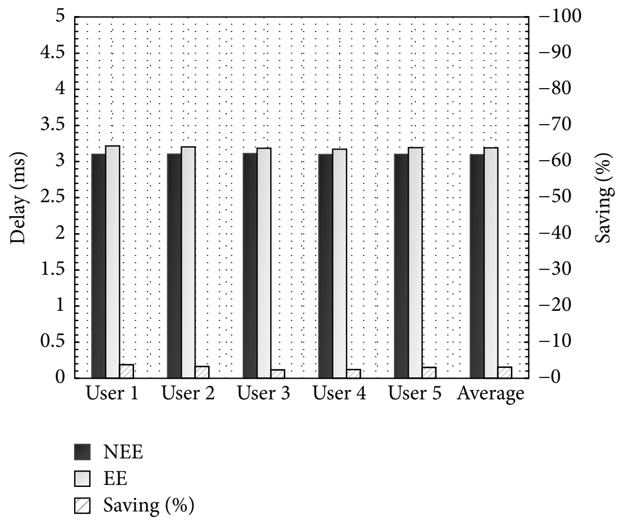


Figure 9: User (VNF) delays with $S = 5$, $N = 5$.

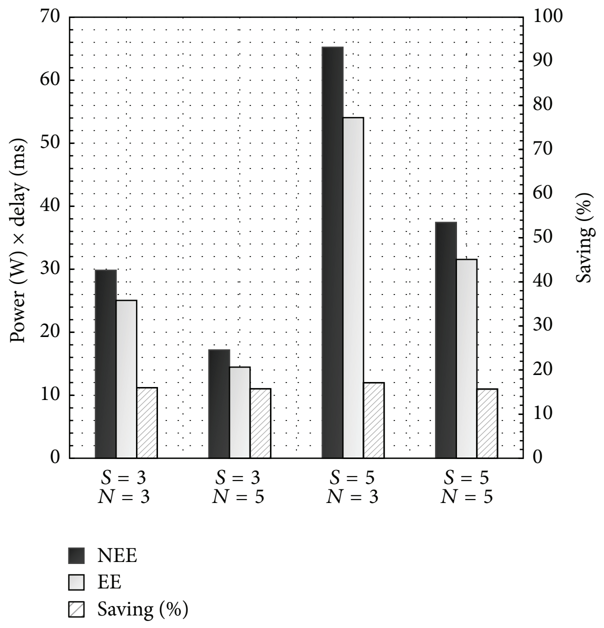


Figure 10: Power consumption and delay product of the various cases.

highlights the relevance of setting not only maximum, but also minimum values for the processing capacity (or for the node input workloads).

Another consideration arising from the results concerns the variation of the energy consumption, by changing the number of players and/or the nodes. For example, Figure 6 (case $S = 5$, $N = 3$) shows total power consumption significantly higher than the one shown in Figure 8 (case $S = 5$, $N = 5$). The reasons for this difference are due to the lower workload managed by the nodes of the case $S = 5$, $N = 5$ (greater number of nodes with an equal number of users) making it possible to exploit the AR mechanisms with significant energy saving.

Finally, Figure 10 shows the product of the power consumption and the delay for each studied case. As described in the previous sections, our criterion aims at minimizing this product. As can be easily seen, the saving resulting from the application of our policy is always above 10%.

6. Conclusions

We have examined the possible effect of introducing simple energy optimization strategies in physical network nodes performing services for a set of VNFs that request their computational power within an NFV environment. The VNFs' strategy profiles for resource sharing have been obtained from the solution of a noncooperative game, where the players are aware of the adaptive behavior of the resources, and aim at minimizing costs that reflect this behavior. We have compared the energy consumption and processing delay with those stemming from a game played with non-energy-aware nodes and VNFs. The results show that the effect on delay is almost negligible, whereas significant power saving can be obtained. In particular, the energy saving is always higher than 18% in every case, with a delay increase always lower than 4%.

From another point of view, it would also be of interest to investigate socially optimal solutions stemming from a Nash Bargaining Problem (NBP), along the lines of [37], by defining suitable utility functions for the players. Another interesting evolution of this work could be the application of additional constraints to the proposed solution such as, for example, the maximum delay that a flow can support.

Appendix

We check the maintenance of continuous differentiability at the point $\overline{f}_j = \mu_j^{\max}/\vartheta_j$. By noting that the derivative of the cost function of player i with respect to f_j^i has the form

$$\frac{\partial J^i}{\partial f_j^i} = c_j^*\left(f_j\right) + f_j^i c_j^{*'}\left(f_j\right), \tag{A.1}$$

it is immediate to note that we need to compare

$$\frac{\partial}{\partial f_j^i} K_j \frac{\left(f_j^i + f_j^{-i}\right)^{1/\alpha_j}\left(\mu_j^{\max}\right)^{3-1/\alpha_j}}{\mu_j^{\max} - f_j^i - f_j^{-i}}\Bigg|_{f_j^i=\mu_j^{\max}/\vartheta_j - f_j^{-i}},$$
$$\frac{\partial}{\partial f_j^i} h_j \cdot \left(f_j\right)^2\Bigg|_{f_j^i=\mu_j^{\max}/\vartheta_j - f_j^{-i}}. \tag{A.2}$$

We have

$$K_j \frac{\left(1/\alpha_j\right)\left(f_j\right)^{1/\alpha_j-1}\left(\mu_j^{\max}\right)^{3-1/\alpha_j}\left(\mu_j^{\max} - f_j\right) + \left(f_j\right)^{1/\alpha_j}\left(\mu_j^{\max}\right)^{3-1/\alpha_j}}{\left(\mu_j^{\max} - f_j\right)^2}\Bigg|_{f_j=\mu_j^{\max}/\vartheta_j}$$
$$= \frac{K_j\left(\mu_j^{\max}\right)^{3-1/\alpha_j}\left(\mu_j^{\max}/\vartheta_j\right)^{1/\alpha_j}\left[\left(1/\alpha_j\right)\left(\mu_j^{\max}/\vartheta_j\right)^{-1}\left(\mu_j^{\max} - \mu_j^{\max}/\vartheta_j\right) + 1\right]}{\left(\mu_j^{\max} - \mu_j^{\max}/\vartheta_j\right)^2}$$

$$= \frac{K_j \left(\mu_j^{\max}\right)^3 \left(\vartheta_j\right)^{-1/\alpha_j} \left[\left(1/\alpha_j\right) \vartheta_j \left(1 - 1/\vartheta_j\right) + 1\right]}{\left(\mu_j^{\max}\right)^2 \left(1 - 1/\vartheta_j\right)^2} = \frac{K_j \mu_j^{\max} \left(\vartheta_j\right)^{-1/\alpha_j} \left(\vartheta_j/\alpha_j - 1/\alpha_j + 1\right)}{\left(\left(\vartheta_j - 1\right)/\vartheta_j\right)^2},$$

$$2h_j f_j\Big|_{f_j=\mu_j^{\max}/\vartheta_j} = 2K_j \frac{\left(\vartheta_j\right)^{3-1/\alpha_j}}{\vartheta_j - 1} \cdot \frac{\mu_j^{\max}}{\vartheta_j}. \tag{A.3}$$

Then, let us check when

$$\begin{aligned}
&\frac{K_j \mu_j^{\max} \left(\vartheta_j\right)^{-1/\alpha_j} \left(\vartheta_j/\alpha_j - 1/\alpha_j + 1\right)}{\left(\left(\vartheta_j - 1\right)/\vartheta_j\right)^2} \\
&\quad = 2K_j \frac{\left(\vartheta_j\right)^{3-1/\alpha_j}}{\vartheta_j - 1} \cdot \frac{\mu_j^{\max}}{\vartheta_j}; \\
&\frac{\left(\vartheta_j/\alpha_j - 1/\alpha_j + 1\right) \cdot \left(\vartheta_j\right)^2}{\vartheta_j - 1} = 2\left(\vartheta_j\right)^2; \\
&\frac{\vartheta_j}{\alpha_j} - \frac{1}{\alpha_j} + 1 = 2\vartheta_j - 2; \\
&\left(\frac{1}{\alpha_j} - 2\right) \frac{3\alpha_j - 1}{2\alpha_j - 1} - \frac{1}{\alpha_j} + 3 = 0; \\
&\left(\frac{1 - 2\alpha_j}{\alpha_j}\right) \frac{3\alpha_j - 1}{2\alpha_j - 1} - \frac{1}{\alpha_j} + 3 = 0; \\
&\frac{1 - 3\alpha_j}{\alpha_j} - \frac{1}{\alpha_j} + 3 = 0; \\
&1 - 3\alpha_j - 1 + 3\alpha_j = 0.
\end{aligned} \tag{A.4}$$

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was partially supported by the H2020 European Projects ARCADIA (http://www.arcadia-framework.eu/) and INPUT (http://www.input-project.eu/), funded by the European Commission.

References

[1] C. Bianco, F. Cucchietti, and G. Griffa, "Energy consumption trends in the next generation access network—a telco perspective," in *Proceedings of the 29th International Telecommunication Energy Conference (INTELEC '07)*, pp. 737–742, Rome, Italy, September-October 2007.

[2] GeSI SMARTer 2020: The Role of ICT in Driving a Sustainable Future, December 2012, http://gesi.org/portfolio/report/72.

[3] A. Manzalini, "Future edge ICT networks," *IEEE COMSOC MMTC E-Letter*, vol. 7, no. 7, pp. 1–4, 2012.

[4] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[5] "Network functions virtualisation—introductory white paper," in *Proceedings of the SDN and OpenFlow World Congress*, Darmstadt, Germany, October 2012.

[6] R. Bolla, R. Bruschi, F. Davoli, and C. Lombardo, "Fine-grained energy-efficient consolidation in SDN networks and devices," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 132–145, 2015.

[7] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 223–244, 2011.

[8] I. Menache and A. Ozdaglar, *Network Games: Theory, Models, and Dynamics*, Morgan & Claypool Publishers, San Rafael, Calif, USA, 2011.

[9] R. Bolla, R. Bruschi, F. Davoli, and P. Lago, "Optimizing the power-delay product in energy-aware packet forwarding engines," in *Proceedings of the 24th Tyrrhenian International Workshop on Digital Communications—Green ICT (TIWDC '13)*, pp. 1–5, IEEE, Genoa, Italy, September 2013.

[10] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a hypervisor for the internet?" *IEEE Communications Magazine*, vol. 50, no. 1, pp. 136–143, 2012.

[11] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012.

[12] G. M. Roy, S. K. Saurabh, N. M. Upadhyay, and P. K. Gupta, "Creation of virtual node, virtual link and managing them in network virtualization," in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, pp. 738–742, IEEE, Mumbai, India, December 2011.

[13] A. Manzalini, R. Saracco, C. Buyukkoc et al., "Software-defined networks or future networks and services—main technical challenges and business implications," in *Proceedings of the IEEE Workshop SDN4FNS*, Trento, Italy, November 2013.

[14] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–19, 2008.

[15] A. Jarray and A. Karmouch, "Decomposition approaches for virtual network embedding with one-shot node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 1012–1025, 2015.

[16] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proceedings of the 28th Conference on Computer Communications (IEEE INFOCOM '09)*, pp. 783–791, Rio de Janeiro, Brazil, April 2009.

[17] X. Cheng, S. Su, Z. Zhang et al., "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, 2011.

[18] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, 2012.

[19] A. Leivadeas, C. Papagianni, and S. Papavassiliou, "Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1077–1086, 2013.

[20] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[21] M. Schöller, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions," in *Proceedings of the 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT '13)*, pp. 208–214, Almaty, Kazakhstan, September 2013.

[22] A. Jarray and A. Karmouch, "VCG auction-based approach for efficient Virtual Network embedding," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 609–615, Ghent, Belbium, May 2013.

[23] A. Gember-Jacobson, R. Viswanathan, C. Prakash et al., "OpenNF: enabling innovation in network function control," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '14)*, pp. 163–174, ACM, Chicago, Ill, USA, August 2014.

[24] J. Antoniou and A. Pitsillides, *Game Theory in Communication Networks: Cooperative Resolution of Interactive Networking Scenarios*, CRC Press, Boca Raton, Fla, USA, 2013.

[25] M. S. Seddiki, M. Frikha, and Y.-Q. Song, "A non-cooperative game-theoretic framework for resource allocation in network virtualization," *Telecommunication Systems*, vol. 61, no. 2, pp. 209–219, 2016.

[26] L. Pavel, *Game Theory for Control of Optical Networks*, Springer, New York, NY, USA, 2012.

[27] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter, "A survey on networking games in telecommunications," *Computers & Operations Research*, vol. 33, no. 2, pp. 286–311, 2006.

[28] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*, pp. 323–336, San Francisco, Calif, USA, April 2008.

[29] S. Henzler, *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies*, vol. 25 of *Springer Series in Advanced Microelectronics*, Springer, Dordrecht, The Netherlands, 2007.

[30] K. Li, "Scheduling parallel tasks on multiprocessor computers with efficient power management," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW '10)*, pp. 1–8, IEEE, Atlanta, Ga, USA, April 2010.

[31] T. Başar, *Lecture Notes on Non-Cooperative Game Theory*, Hamilton Institute, Kildare, Ireland, 2010, http://www.hamilton.ie/ollie/Downloads/Game.pdf.

[32] A. Orda, R. Rom, and N. Shimkin, "Competitive routing in multiuser communication networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 5, pp. 510–521, 1993.

[33] E. Altman, T. Başar, T. Jiménez, and N. Shimkin, "Competitive routing in networks with polynomial costs," *IEEE Transactions on Automatic Control*, vol. 47, no. 1, pp. 92–96, 2002.

[34] J. B. Rosen, "Existence and uniqueness of equilibrium points for concave N-person games," *Econometrica*, vol. 33, no. 3, pp. 520–534, 1965.

[35] Y. A. Korilis, A. A. Lazar, and A. Orda, "Architecting noncooperative networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1241–1251, 1995.

[36] MATLAB®, The Language of Technical Computing, http://www.mathworks.com/products/matlab.

[37] H. Yaïche, R. R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 667–678, 2000.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 3583962, 10 pages
http://dx.doi.org/10.1155/2016/3583962

Research Article

A Processor-Sharing Scheduling Strategy for NFV Nodes

Giuseppe Faraci, Alfio Lombardo, and Giovanni Schembra

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica (DIEEI), University of Catania, 95123 Catania, Italy

Correspondence should be addressed to Giovanni Schembra; schembra@dieei.unict.it

Received 2 November 2015; Accepted 12 January 2016

Academic Editor: Xavier Hesselbach

The introduction of the two paradigms SDN and NFV to "softwarize" the current Internet is making management and resource allocation two key challenges in the evolution towards the Future Internet. In this context, this paper proposes Network-Aware Round Robin (NARR), a processor-sharing strategy, to reduce delays in traversing SDN/NFV nodes. The application of NARR alleviates the job of the Orchestrator by automatically working at the intranode level, dynamically assigning the processor slices to the virtual network functions (VNFs) according to the state of the queues associated with the output links of the network interface cards (NICs). An extensive simulation set is presented to show the improvements achieved with respect to two more processor-sharing strategies chosen as reference.

1. Introduction

In the last few years the diffusion of new complex and efficient distributed services in the Internet is becoming increasingly difficult because of the ossification of the Internet protocols, the proprietary nature of existing hardware appliances, the costs, and the lack of skilled professionals for maintaining and upgrading them.

In order to alleviate these problems, two new network paradigms, Software Defined Networks (SDN) [1–5] and Network Functions Virtualization (NFV) [6, 7], have been recently proposed with the specific target of improving the flexibility of network service provisioning and reducing the time to market of new services.

SDN is an emerging architecture that aims at making the network dynamic, manageable, and cost-effective, by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying system that forwards traffic to the selected destination (the data plane). In this way the network control becomes directly programmable and the underlying infrastructure is abstracted for applications and network services.

NFV is a core structural change in the way telecommunication infrastructure is deployed. The NFV initiative started in late 2012 by some of the biggest telecommunications service providers, which formed an Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI). The interest has grown, involving today more than 28 network operators and over 150 technology providers from across the telecommunications industry [7]. The NFV paradigm leverages on virtualization technologies and commercial off-the-shelf programmable hardware, such as general-purpose servers, storage, and switches, with the final target of decoupling the software implementation of network functions from the underlying hardware.

The coexistence and the interaction of both NFV and SDN paradigms is giving to the network operators the possibility of achieving greater agility and acceleration in new service deployments, with a consequent considerable reduction of both Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [8].

One of the main challenging problems in deploying an SDN/NFV network is an efficient design of resource allocation and management, functions that are in charge of the network Orchestrator. Although this task is well covered in data center and cloud scenarios [9, 10], it is currently a challenging problem in a geographic network where transmission delays cannot be neglected, and transmission capacities of the interconnection links are not comparable with the case of above scenarios. This is the reason why the problem of orchestrating an SDN/NFV network is still open and attracts a lot of research interest from both academia and industry.

More specifically, the whole problem of orchestrating an SDN/NFV network is very complex because it involves a design work at both internode and intranode levels [11, 12]. At the internode level, and in all the cases where the time between each execution is significantly greater than the time to collect, compute, and disseminate results, the application of a centralized approach is practicable [13, 14]. In these cases, in fact, taking into consideration both traffic characterization and required level of quality of service (QoS), the Orchestrator is able to decide in a centralized manner how many instances of the same function have to be run simultaneously, the network nodes that have to execute them, and the routing strategies that allow traffic flows to cross the nodes where the requested network functions are running.

Instead, centralizing a strategy dealing with operations that require dynamic reconfiguration of network resources is actually unfeasible to be executed by the Orchestrator for problems of resilience and scalability. Therefore, adaptive management operations with short timescales require a distributed approach. The authors of [15] propose a framework to support adaptive resource management operations, which involve short timescale reconfiguration of network resources, showing how requirements in terms of load-balancing and energy management [16–18] can be satisfied. Another work in the same direction is [19] that proposes a solution for the consolidation of VMs on local computing resources, exclusively based on local information. The work in [20] aims at alleviating the inter-VM network latency defining a hypervisor scheduler algorithm that is able to take into consideration the allocation of the resources within a consolidated environment, scheduling VMs to reduce their waiting latency in the run queue. Another approach is applied in [21], which introduces a policy to manage the internal on/off switching of virtual network functions (VNFs) in NFV-compliant Customer Premises Equipment (CPE) devices.

The focus of this paper is on another fundamental problem that is inherent to resource allocation within an SDN/NFV node, that is, the decision of the percentage of CPU to be assigned to each VNF. If at a first glance this could show a classical problem of processor sharing that has been widely explored in the past literature [15, 22, 23], actually it is much more complex because performance can be strongly improved by leveraging on the correlation with the output queues associated with the network interface cards (NICs).

With all this in mind, the main contribution of this paper is the definition of a processor-sharing policy, in the following referred to as *Network-Aware Round Robin* (NARR), which is specific for SDN/NFV nodes. Starting from the consideration that packets that have received the service of a network function from a virtual machine (VM) running on a given node are enqueued to wait for transmission through a given NIC, the proposed strategy dynamically changes the slices of the CPU assigned to each VNF according to the state of the output NIC queues. More specifically, the NARR strategy gives a larger CPU slice to serve packets that will leave the node through the NIC that is currently less loaded, in such a way as to minimize wastes of the NIC output link capacities, also minimizing the overall delay experienced by packets traversing nodes that implement NARR.

As a side contribution, the paper calculates an on-off model for the traffic leaving the SDN/NFV node on each NIC output link. This model can be used as a building block for the design and performance evaluation of an entire network.

The paper is structured as follows. Section 2 describes the node architecture. Section 3 introduces the NARR strategy. Section 4 presents a case study and shows some numerical results. Finally, Section 5 draws some conclusions and discusses some future work.

2. System Description

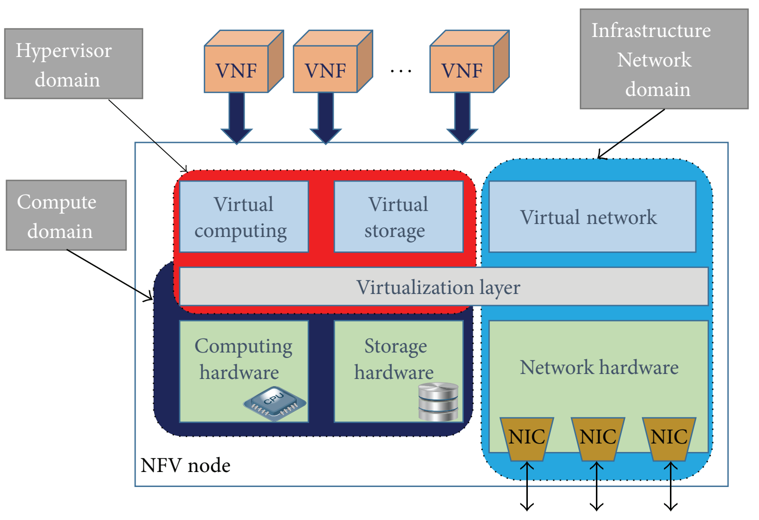
The target of this section is the description of the system we consider in the rest of the paper. It is an SDN/NFV node as the one considered in [11, 12], where we will apply the NARR strategy. Its architecture, shown in Figure 1(a), is compliant with the ETSI Specifications [24]. It is composed of three different domains, namely, the *Compute* domain, the *Hypervisor* domain, and the *Infrastructure Network* domain. The *Compute* domain provides the computational and storage hardware resources that allow the node to host the VNFs. Thanks to the computing and storage virtualization provided by the *Hypervisor* domain, a VM can be created, migrated from one node to another one, and halted, in order to optimize the deployment according to specific performance parameters. Communications among the VMs, and between the VMs and the external environment, are provided by the *Infrastructure Network* domain.

The SDN/NFV node is remotely controlled by the *Orchestrator*, whose architecture is shown in Figure 1(b). It is constituted by three main blocks. The *Orchestration Engine* executes all the algorithms and the strategies to manage and orchestrate the whole network. After each decision, the Orchestration Engine requests that the *NFV Coordinator* instantiates, migrates, or halts VMs and consequently requests that the SDN Controller modifies the flow tables of the SDN switches in the network in such a way that traffic flows can traverse VMs hosting the requested VNFs.

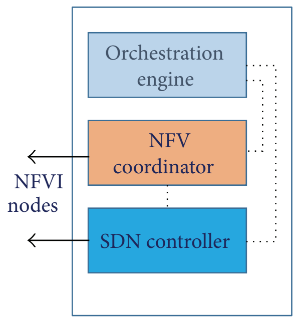
With this in mind, a functional architecture of the NFV node is represented in Figure 2. Its main components are the *Processor*, which manages the Compute domain and hosts the Hypervisor domain, and the Network Card Interfaces (NICs) with their queues, which constitute the "Network Hardware" block in Figure 1.

Let M be the number of virtual network functions (VNFs) that are running in the node, and let L be the number of output NICs. In order to simplify notation, in the following we will assume that all the NICs have the same characteristics in terms of buffer capacity and output rate. So, let $K^{(\mathrm{NIC})}$ be the size of the queue associated with each NIC, that is, the maximum number of packets that each queue can contain, and let $\mu^{(\mathrm{NIC})}$ be the transmission rate of the output link associated with each NIC, expressed in bit/s.

The *Flow Distributor* block has the task of routing each entering flow towards the function required by the flow. It is a software SDN switch that can be implemented, for example, with OpenvSwitch [25]. It routes the flows to the VMs running the requested VNFs according to the control messages received by the SDN Controller residing in the



(a) NFV node



(b) Orchestrator

Figure 1: Network function virtualization infrastructure.

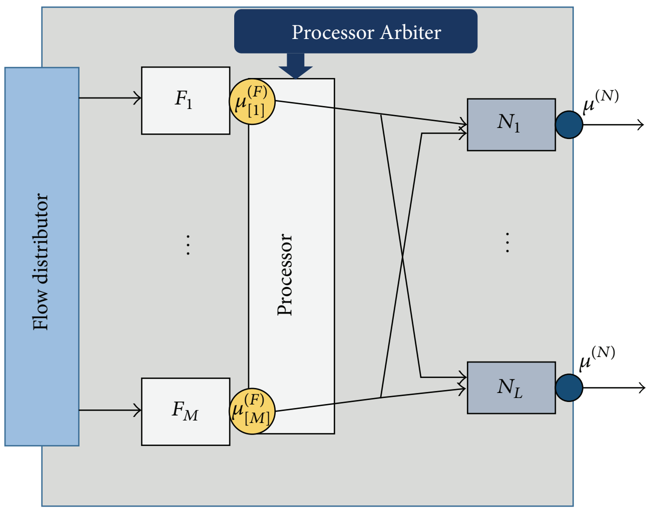


Figure 2: NFV node functional architecture.

Orchestrator. The most common protocol that can be used for the communications between the SDN Controller and the Orchestrator is OpenFlow [26].

Let $\mu^{(P)}$ be the total processing rate of the processor, expressed in packets/s. This rate is shared among all the active functions according to a processor rate scheduling strategy. Let $\underline{\mu}^{(F)}$ be the array whose generic element, $\mu^{(F)}_{[m]}$, with $m \in \{1, \ldots, M\}$, is the portion of the processor rate assigned to the VM implementing the function F_m. Of course we have

$$\sum_{m=1}^{M} \mu^{(F)}_{[m]} = \mu^{(P)}. \tag{1}$$

Once a packet has been served by the required function, it is sent to one of the NICs to exit from the node. If the NIC is transmitting another packet, the arriving packets are enqueued in the NIC queue. We will indicate the queue associated with the generic NIC l as $Q^{(\mathrm{NIC})}_l$.

In order to implement the NARR strategy proposed in this paper, we realize the block relative to each function with a set of L parallel queues, in the following referred to as *inside-function queues*, as shown in Figure 3 for the generic function F_m. The generic lth inside-function queue of the function F_m, indicated as $Q_{m,l}$ in Figure 3, is used to enqueue packets that, after receiving the service of the function F_m, will leave the node through the NIC l. Let $K^{(F)}_{\mathrm{Ins}}$ be the size of each inside-function queue. Each inside-function queue of the generic function F_m receives a portion of the processor rate assigned to that function. Let $\mu^{(F_{\mathrm{Ins}})}_{[m,l]}$ be the portion of the processor rate assigned to the queue $Q_{m,j}$ of the function F_m. Of course, we have

$$\sum_{l=1}^{L} \mu^{(F_{\mathrm{Ins}})}_{[m,l]} = \mu^{(F)}_{[m]}. \tag{2}$$

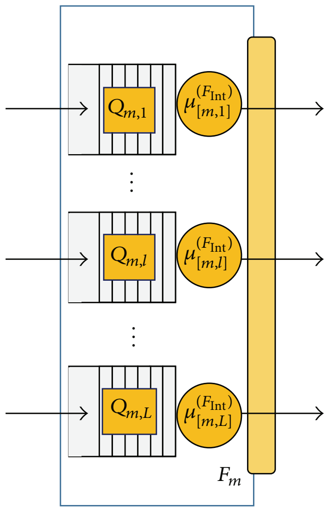

$Q_{m,1}$
$\mu_{[m,1]}^{(F_{\text{Int}})}$
$Q_{m,l}$
$\mu_{[m,l]}^{(F_{\text{Int}})}$
$Q_{m,L}$
$\mu_{[m,L]}^{(F_{\text{Int}})}$
F_m

Figure 3: Block diagram of the generic function F_m.

The portion of processor rate associated with each inside-function queue is dynamically changed by the *Processor Arbiter* according to the NARR strategy described in Section 3.

3. NARR Processor-Sharing Strategy

The NARR (Network-Aware Round Robin) processor-sharing strategy observes the state of both the inside-function queues and the NIC queues, with the goal of reducing as far as possible the inactivity periods of the NIC output links. As already introduced in the Introduction, its definition starts from the fact that packets that have received the service of a VNF are enqueued to wait for transmission through a given NIC. So, in order to avoid output link capacity waste, NARR dynamically changes the slices of the CPU assigned to each VNF, and in particular to each inside-function queue, according to the state of the output NIC queues, assigning larger CPU slices to serve packets that will leave the node through less-loaded NICs.

More specifically, the Processor Arbiter decides the processor rate portions according to two different steps.

Step 1 (assignment of the processor rate portion to the aggregation of queues whose output is a specific NIC). This step meets the target of the proposed strategy that is to reduce, as much as possible, underutilization of the NIC output links and, as a consequence, delays in the relative queues. To this purpose, let us consider a virtual queue that contains all the packets that are stored in all the inside-function queues $Q_{m,l}$, for each $m \in [1, M]$, that is, all the packets that will leave the node through the NIC l. Let us indicate this virtual queue as $Q_{\text{Aggr}}^{(\to \text{NIC}_l)}$, and its service rate as $\mu_{\text{Aggr}}^{(\to \text{NIC}_l)}$. Of course, we have

$$\mu_{\text{Aggr}}^{(\to \text{NIC}_l)} = \sum_{m=1}^{M} \mu_{[m,l]}^{(F_{\text{Ins}})}. \qquad (3)$$

With this in mind, the idea is to give a higher processor slice to the inside-function queues whose flows are directed to the NICs that are emptying.

Taking into account the goal of privileging the flows that will leave the node through underloaded NICs, the Processor Arbiter calculates $\mu_{\text{Aggr}}^{(\to \text{NIC}_l)}$ as follows:

$$\mu_{\text{Aggr}}^{(\to \text{NIC}_l)} = \frac{q_{\text{ref}} - S_l^{(\text{NIC})}}{\sum_{j=1}^{L} \left\{ q_{\text{ref}} - S_j^{(\text{NIC})} \right\}} \mu^{(P)}, \qquad (4)$$

where $S_l^{(\text{NIC})}$ represents the state of the queue associated with the NIC l, while q_{ref} is defined as follows:

$$q_{\text{ref}} = \min \left\{ \alpha \cdot \max_j \left(S_j^{(\text{NIC})} \right), K^{(\text{NIC})} \right\}. \qquad (5)$$

The term q_{ref} is a reference target value calculated from the state of the NIC queue that has the highest length, amplified with a coefficient α, and truncated to the maximum queue size $K^{(\text{NIC})}$. It is determined in such a way that, if we consider $\alpha = 1$, the NIC queue that has the highest length does not receive packets from the inside-function queues because the service rate of them is set to zero; the other queues receive packets with a rate that is proportional to the distance between their length and the length of the most overloaded NIC queue. However, through an extensive set of simulations, we deduced that setting $\alpha = 1$ causes bad performance because there is always a group of inside-function queues that are not served. Instead, all the α values in the interval $]1, 2]$ give almost equivalent performance. For this reason, in the numerical analysis presented in Section 4, we have set $\alpha = 1.2$.

Step 2 (assignment of the processor rate portion to each inside-function queue). Let us consider the generic lth inside-function queue of the function F_m, that is, the queue $Q_{m,l}$. Its service rate is calculated as being proportional to the current state of this queue in comparison with the other lth queues of the other functions. To this purpose, let us indicate the state of the virtual queue $Q_{\text{Aggr}}^{(\to \text{NIC}_l)}$ as $S_{\text{Aggr}}^{(\to \text{NIC}_l)}$. Of course, it can be calculated as the sum of the states of all the inside-function queues $Q_{m,l}$, for each $m \in [1, M]$: that is,

$$S_{\text{Aggr}}^{(\to \text{NIC}_l)} = \sum_{k=1}^{M} S_{k,l}^{(F_{\text{Ins}})}. \qquad (6)$$

So, the service rate of the inside-function queue $Q_{m,l}$ is determined as a fraction of the service rate assigned at the first step to the virtual queue $Q_{\text{Aggr}}^{(\to \text{NIC}_l)}$, $\mu_{\text{Aggr}}^{(\to \text{NIC}_l)}$, as follows:

$$\mu_{[m,l]}^{(F_{\text{Ins}})} = \frac{S_{m,l}^{(F_{\text{Ins}})}}{S_{\text{Aggr}}^{(\to \text{NIC}_l)}} \mu_{\text{Aggr}}^{(\to \text{NIC}_l)}. \qquad (7)$$

Of course, if at any time an inside-function queue remains empty, the processor rate portion assigned to it will be shared among the other queues proportionally to the processor portions previously assigned. Likewise, if at some instant an empty queue receives a new packet, the previous processor rate portion is reassigned to that queue.

4. Case Study

In this section we present a numerical analysis of the behavior of an SDN/NFV node that applies the proposed NARR processor-sharing strategy, with the target of evaluating the achieved performance. To this purpose, we will consider two other processor-sharing strategies as reference, in the following referred to as *round robin* (RR) and *queue-length weighted round robin* (QLWRR). In both the reference cases, the node has the same L NIC queues, but it has only M processor queues, one for each function. Each of the M queues has a size of $K^{(F)} = L \cdot K_{\text{Ins}}^{(F)}$, where $K_{\text{Ins}}^{(F)}$ represents the size of each internal function queue, already defined so far for the proposed strategy.

The RR strategy applies the classical round robin scheduling policy to serve the M function queues; that is, it serves each function queue with a rate $\mu_{\text{RR}}^{(F)} = \mu^{(P)}/M$.

The QLWRR strategy, on the other hand, serves each function queue with a rate that is proportional to the queue length; that is,

$$\mu_{\text{QLWRR}}^{(F_m)} = \frac{S^{(F_m)}}{\sum_{k=1}^{M} S^{(F_k)}} \mu^{(P)}, \tag{8}$$

where $S^{(F_m)}$ is the state of the queue associated with the function F_m.

4.1. Parameter Settings. In this numerical analysis, we consider a node with $M = 4$ VNFs, and $L = 3$ output NICs. We loaded the node with a balanced traffic constituted by $N_F = 12$ flows, each characterized by a different 2-uple $\{$*requested NFV*, *output NIC*$\}$. Each flow has been generated with an on-off model characterized by exponentially distributed on and off periods. When a flow is in off state, no packets arrive to the node from it; instead, when it is in on state, packets arrive with an average rate λ_{ON}. Each packet is assumed with an exponential distributed size with a mean of 1 kbyte. In all the simulations we have considered the same on-off cycle duration, $\delta = \overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}} = 5$ msec, while we have varied the burstiness. Burstiness of on-off sources is defined as

$$b = \frac{\lambda_{\text{ON}}}{\lambda_{\text{Mean}}}, \tag{9}$$

where λ_{Mean} is the mean emission rate. Now, indicating the probability of the ON state as π_{ON}, and taking into account that $\lambda_{\text{Mean}} = \lambda_{\text{ON}} \cdot \pi_{\text{ON}}$ and $\pi_{\text{ON}} = \overline{T}_{\text{ON}}/(\overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}})$, we have

$$b = \frac{\overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}}}{\overline{T}_{\text{ON}}}. \tag{10}$$

In our analysis, the burstiness has been varied in the range $[2, 22]$. Consequently, we have derived the mean durations of the off and on periods, as follows:

$$\begin{aligned} \overline{T}_{\text{ON}} &= \frac{\delta}{b}, \\ \overline{T}_{\text{OFF}} &= \delta - \overline{T}_{\text{ON}}. \end{aligned} \tag{11}$$

Finally, in order to maintain the same mean packet rate for different values of $\overline{T}_{\text{OFF}}$ and $\overline{T}_{\text{ON}}$, we have assumed that 75 packets are transmitted, on average: that is, $\lambda_{\text{ON}} = 75/\overline{T}_{\text{ON}}$. The resulting mean emission rate is $\lambda_{\text{Mean}} = 122.9$ Mbit/s.

As far as the NICs are concerned, we have considered an output rate of $\mu^{(\text{NIC})} = 980$ Mbit/s, so having a utilization coefficient on each NIC of $\rho^{(\text{NIC})} = 0.5$, and a queue size $K^{(\text{NIC})} = 3000$ packets. Instead, regarding the processor, we considered a size of each inside-function queue of $K_{\text{Ins}}^{(F)} = 3000$ packets. Finally, we have analyzed two different processor cases. In the first case we considered a processor that is able to process $\mu^{(P)} = 306$ kpackets/s, while in the second case we assumed a processor rate of $\mu^{(P)} = 204$ kpackets/s. Therefore, defining the processor utilization coefficient as follows:

$$\rho^{(P)} = \frac{N_F \cdot \lambda_{\text{Mean}}}{\mu^{(P)}} \tag{12}$$

with $N_F \cdot \lambda_{\text{Mean}}$ being the total mean arrival rate to the node, the two considered cases are characterized by a processor utilization coefficient of $\rho_{\text{Low}}^{(P)} = 0.6$ and $\rho_{\text{High}}^{(P)} = 0.9$, respectively.

4.2. Numerical Results. In this section we present some results achieved by discrete-event simulations. The simulation tool used in the paper is publicly available in [27]. We first present a temporal analysis of the main variables characterizing the SDN/NFV node, and then we show a performance comparison of NARR with the two strategies RR and QLWRR, taken as a reference.

For the temporal analysis we focus on a short time interval of 720 μsec, in order to be able to clearly highlight the evolution of the considered processes. We have loaded the node with an on-off traffic like the one described in Section 4.1, with a burstiness $b = 7$.

Figures 4, 5, and 6 show the time evolution of the length of the queues associated with the NICs, the processor slice assigned to each virtual queue loading each NIC, and the length of the same virtual queues. We can subdivide the considered time interval into three different periods.

In the first period, ranging from the instants 0.1063 and 0.1067, from Figure 4 we can notice that the NIC queue $Q_1^{(\text{NIC})}$ has a greater length than the queue $Q_3^{(\text{NIC})}$, while $Q_2^{(\text{NIC})}$ is empty. For this reason in this period, as shown in Figure 5, the processor is shared between $Q_{\text{Aggr}}^{(\to \text{NIC}_1)}$ and $Q_{\text{Aggr}}^{(\to \text{NIC}_3)}$, and the slice assigned to serve $Q_{\text{Aggr}}^{(\to \text{NIC}_3)}$ is higher, in such a way that the two queue lengths $Q_1^{(\text{NIC})}$ and $Q_3^{(\text{NIC})}$ reach the same value, situation that happens at the end of the first period, around the instant 0.1067. During this period, the behavior of both the virtual queues $Q_{\text{Aggr}}^{(\to \text{NIC}_1)}$ and $Q_{\text{Aggr}}^{(\to \text{NIC}_3)}$ in Figure 6 remains flat, showing the fact that the received processor rates are able to balance the arrival rates.

At the beginning of the second period that ranges between the instants 0.1067 and 0.10693, the processor rate assigned to the virtual queue $Q_{\text{Aggr}}^{(\to \text{NIC}_3)}$ has become not sufficient to serve the amount of arriving traffic, and so the virtual queue $Q_{\text{Aggr}}^{(\to \text{NIC}_3)}$ increases its length, as shown in Figure 6.

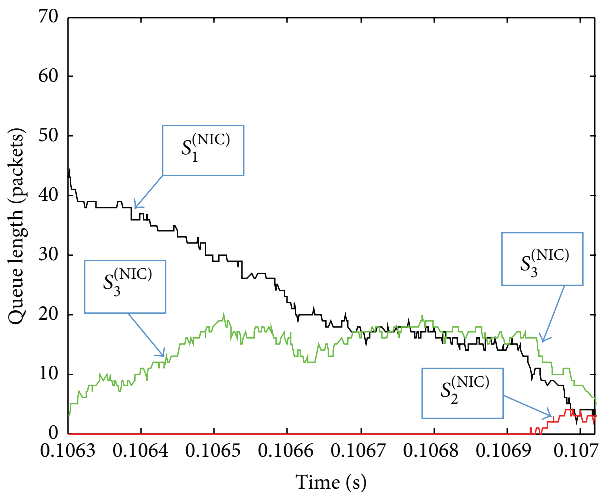


FIGURE 4: Lenght evolution of the NIC queues.

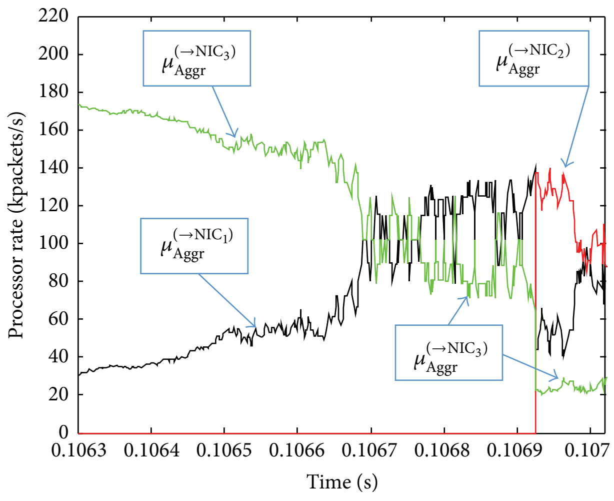


FIGURE 5: Packet rate assigned to the virtual queues.

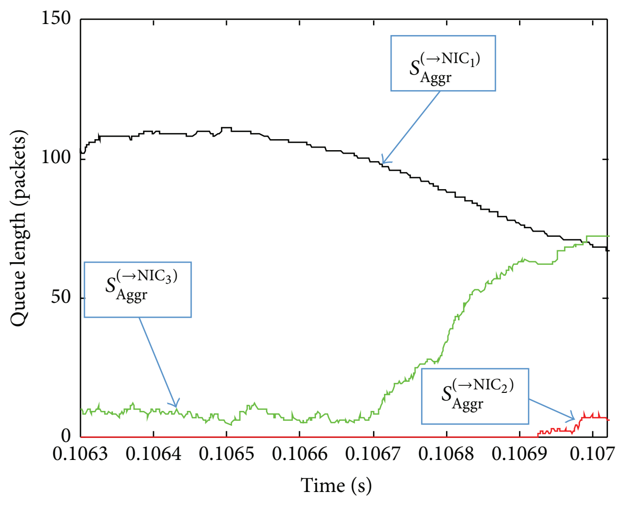


FIGURE 6: Lenght evolution of the virtual queues.

During this second period, the processor slices assigned to the two queues $Q_{Aggr}^{(\rightarrow NIC_1)}$ and $Q_{Aggr}^{(\rightarrow NIC_3)}$ are adjusted in such a way that $Q_1^{(NIC)}$ and $Q_3^{(NIC)}$ remain with comparable lengths.

The last period starts at the instant 0.10693, characterized by the fact that the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_2)}$ leaves the empty state, and therefore participates in the processor-sharing process. Since the NIC queue $Q_2^{(NIC)}$ is low-loaded, as shown in Figure 4, the largest slice is assigned to $Q_{Aggr}^{(\rightarrow NIC_2)}$ in such a way that $Q_2^{(NIC)}$ can reach the same length of the other two NIC queues as soon as possible.

Now, in order to show how the second step of the proposed strategy works, we present the behavior of the inside-function queues whose output is sent to the NIC queue $Q_3^{(NIC)}$ during the same short time interval considered so far. More specifically, Figures 7 and 8 show the length of the considered inside-function queues, and the processor slices assigned to them, respectively. The behavior of the queue $Q_{2,3}$ is not shown because it is empty in the considered period. As we can observe from the above figures, we can subdivide the interval into four periods:

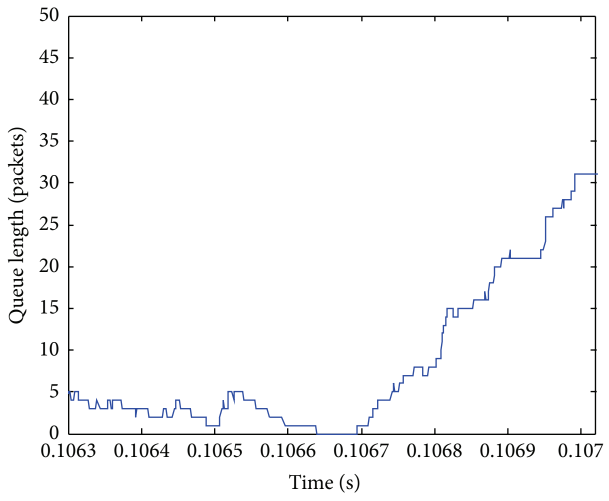
(i) The first period, ranging in the interval [0.1063, 0.10657] is characterized by an empty state of the queue $Q_{3,3}$. Thus, in this period, the processor slice assigned to the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_3)}$ is shared by $Q_{1,3}$ and $Q_{4,3}$, only.

(ii) During the second period, ranging in the interval [0.10657, 0.1067], $Q_{1,3}$ is scarcely loaded (in particular it is empty in the second part of this period), and so the processor slice assigned to $Q_{3,3}$ is increased.

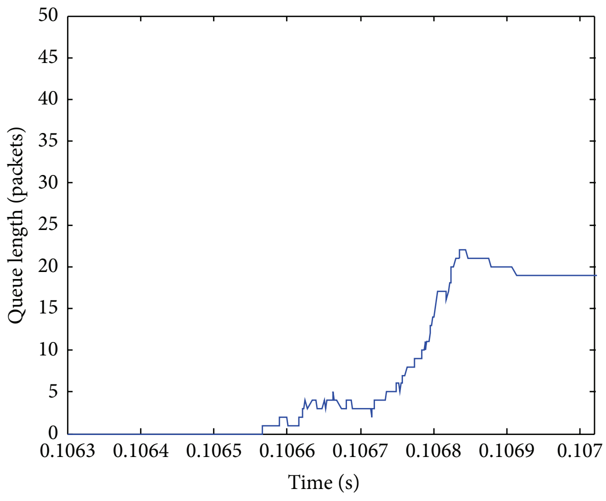
(iii) In the third period, ranging between 0.1067 and 0.10693, all the queues increase and equally share the processor.

(iv) Finally, in the last period, starting at the instant 0.10693, as already observed in Figure 5, the processor slice assigned to the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_3)}$ is suddenly decreased, and consequently the slices assigned to the queues $Q_{1,3}$, $Q_{3,3}$, and $Q_{4,3}$ are decreased as well.

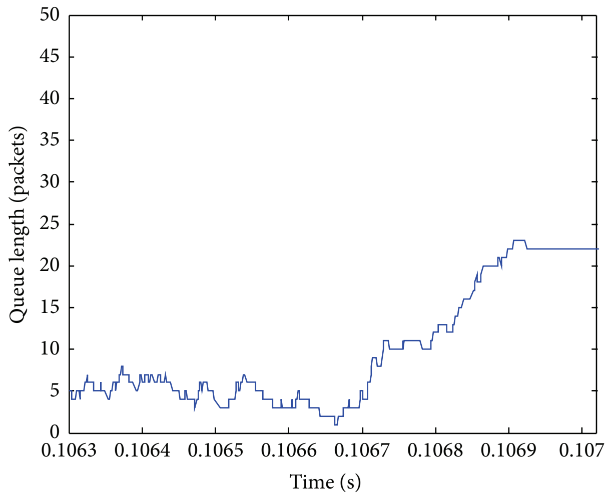
The steady-state analysis is presented in Figures 9, 10, and 11, which show the mean delay in the inside-function queues, in the NIC queues, and the durations of the off- and on-states on the output links. The values reported in all the figures have been derived as the mean values of the results of many simulation experiments, using Student's t-distribution and with a 95% confidence interval. The number of experiments carried out to evaluate each numerical result has been automatically decided by the simulation tool with the requirement of achieving a confidence interval less than 0.001 of the estimated mean value. To this purpose, the confidence interval is calculated at the end of each run, and simulation is stopped only when the confidence interval matches the maximum error requirement. The duration of each run has been chosen in such a way that the sample standard deviation is so low that less than 30 runs are enough to match the requirement.



(a) Queue $Q_{1,3}$



(b) Queue $Q_{3,3}$



(c) Queue $Q_{4,3}$

FIGURE 7: Length evolution of the inside-function queues $Q_{m,3}$, for each $m \in [1, M]$.

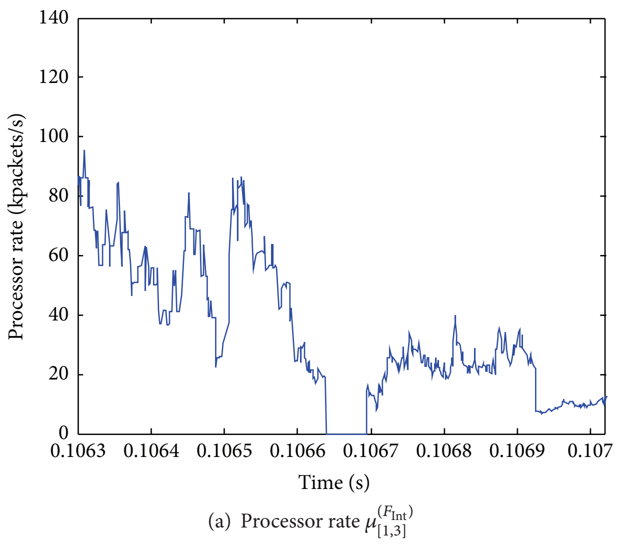
The figures compare the results achieved with the proposed strategy with the ones obtained with the two strategies RR and QLWRR. As said so far, results have been obtained against the burstiness, and for two different values of the utilization coefficient: that is, $\rho_{\text{Low}}^{(P)} = 0.6$ and $\rho_{\text{High}}^{(P)} = 0.9$.

As expected, the mean lengths of the processor queues and the NIC queues increase with both the burstiness and the utilization coefficient.

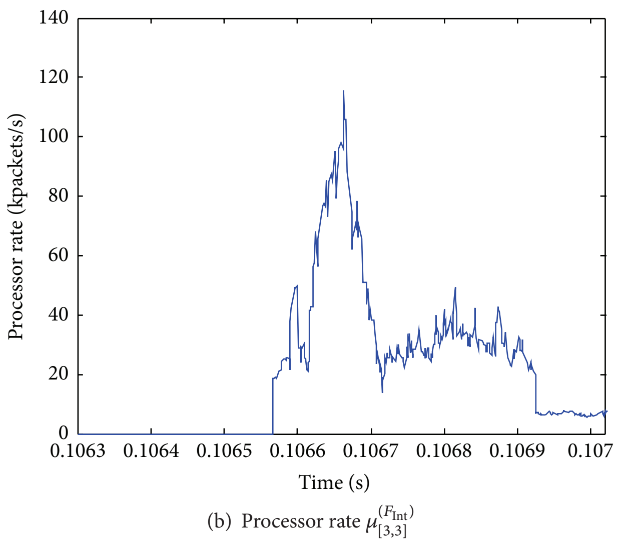
Instead, we can note that the mean length of the processor queues is not affected by the applied policy. In fact, packets requiring the same function are enqueued in a unique queue and served with a rate $\mu^{(P)}/4$, when RR or QLWRR strategies are applied, while, when the NARR strategy is used, they are split into 12 different queues and served with a rate $\mu^{(P)}/12$. If in a classical queueing theory [28] the second case is worse than the first one because of the presence of service rate wastes during the more probable periods of empty queues, this is not the case here because the processor capacity of an empty queue is dynamically reassigned to the other queues.

The advantage of the NARR strategy is evident in Figure 10, where the mean delay in the NIC queues is represented. In fact, we can observe that, with only giving more processor rate to the most loaded processor queues (with the QLWRR strategy), performance improvements are negligible, while applying the NARR strategy we are able to obtain a delay reduction of about 12% in the case of a more performant processor ($\rho_{\text{Low}}^{(P)} = 0.6$), reaching the 50% when the processor works with a $\rho_{\text{High}}^{(P)} = 0.9$. The performance gain achieved with the NARR strategy increases with burstiness and the processor load, conditions that are both likely. In fact, the first condition is due to the high burstiness of the Internet traffic; the second one is true as well because the processor should be not overdimensioned for economic purposes; otherwise, if overdimensioned, it can be controlled with a processor rate management policy like the one presented in [29] in order to save energy.

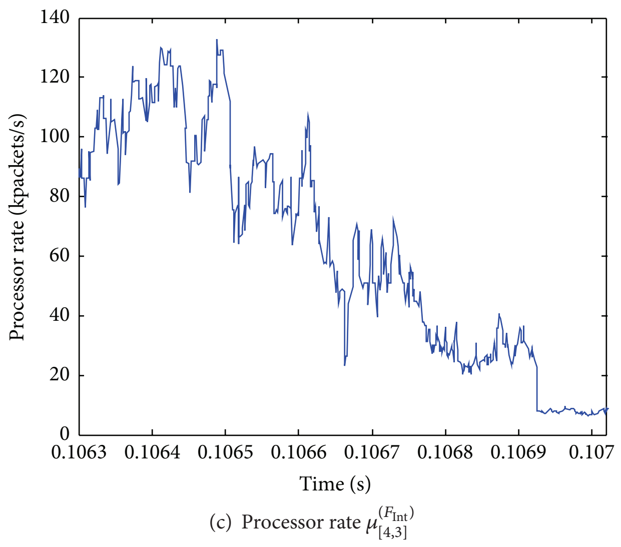
Finally, Figure 11 shows the mean durations of the on- and off-periods on the node output links. Only one curve is



(a) Processor rate $\mu_{[1,3]}^{(F_{\mathrm{Int}})}$



(b) Processor rate $\mu_{[3,3]}^{(F_{\mathrm{Int}})}$



(c) Processor rate $\mu_{[4,3]}^{(F_{\mathrm{Int}})}$

FIGURE 8: Processor rate assigned to the inside-function queues $Q_{m,3}$, for each $m \in [1, M]$.

shown for each case because we have considered a utilization coefficient $\rho^{(\mathrm{NIC})} = 0.5$ on each NIC queue, and therefore in the considered case we have $\overline{T}_{\mathrm{ON}} = \overline{T}_{\mathrm{OFF}}$. As expected, the mean on-off durations are higher when the processor rate is higher (i.e., lower utilization coefficient). This is because, in this case, the output processor rate is lower, and therefore batches of packets in the NIC queues are served quickly. These results can be used to model the output traffic of each SDN/NFV node as an Interrupted Poisson Process (IPP) [30]. This model can be iteratively used to represent the input traffic of other nodes, with the final target of realizing the model of a whole SDN/NFV network.

5. Conclusions and Future Work

This paper addresses the problem of intranode resource allocation, by introducing NARR, a processor-sharing strategy that leverages on the consideration that, in any SDN/NFV node, packets that have received the service of a VNF are enqueued to wait for transmission through one of the output NICs. Therefore, the idea at the base of NARR is to dynamically change the slices of the CPU assigned to each VNF according to the state of the output NIC queues, giving more CPU to serve packets that will leave the node through the less-loaded NICs. In this way, wastes of the NIC output link capacities are minimized, and consequently the overall delay experienced by packets traversing the nodes that implement NARR is reduced.

SDN/NFV nodes that implement NARR can coexist in the same network with nodes that use other strategies, so facilitating a gradual introduction in the Future Internet.

As a side contribution, the simulator tool, which is public available on the web, gives the on-off model of the output links associated with each NIC as one of the results. This model can be used as a building block to realize a model for

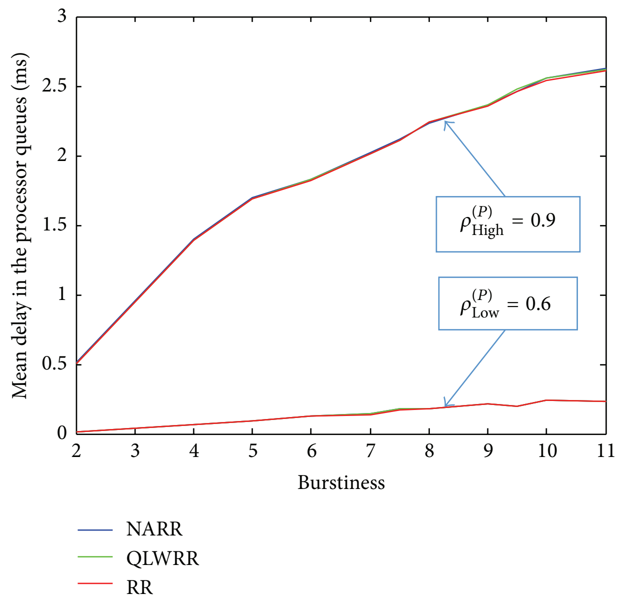


FIGURE 9: Mean delay in the function internal queues.

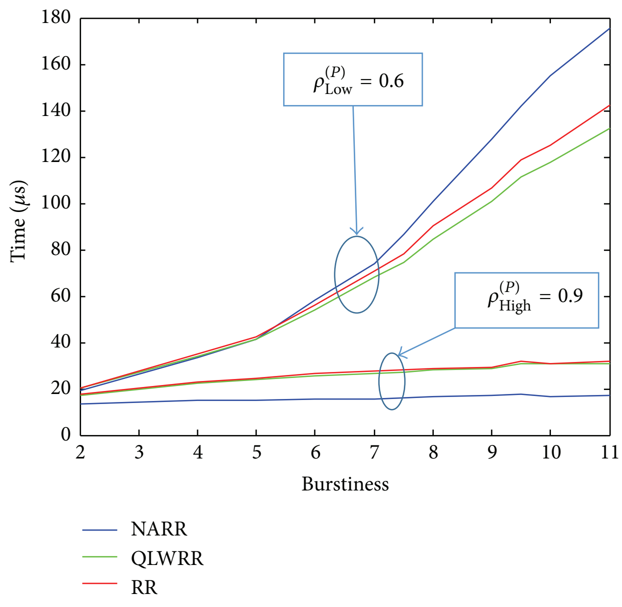


FIGURE 11: Mean off- and on-states duration of the traffic on the output links.

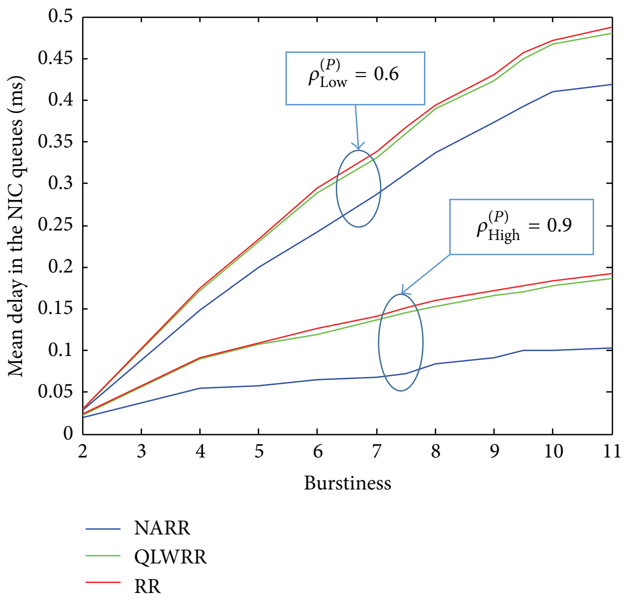


FIGURE 10: Mean delay in the NIC queues.

the design and performance evaluation of a whole SDN/NFV network.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work has been partially supported by the INPUT (In-Network Programmability for Next-Generation Personal cloUd Service Support) project funded by the European Commission under the Horizon 2020 Programme (Call H2020-ICT-2014-1, Grant no. 644672).

References

[1] White paper on "Software-Defined Networking: The New Norm for Networks", https://www.opennetworking.org/.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proceedings of the Symposium on Network Systems Design and Implementation (NSDI '13)*, vol. 13, pp. 29–42, Lombard, Ill, USA, April 2013.

[3] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[4] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] White paper on "Network Functions Virtualisation", http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[7] Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress, ETSI, October 2013, http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[8] A. Manzalini, R. Saracco, E. Zerbini et al., "Software-Defined Networks for Future Networks and Services," White Paper based on the IEEE Workshop SDN4FNS, http://sites.ieee.org/sdn4fns/whitepaper/.

[9] R. Z. Frantz, R. Corchuelo, and J. L. Arjona, "An efficient orchestration engine for the cloud," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology*

and Science (CloudCom '11), vol. 2, pp. 711–716, Athens, Greece, December 2011.

[10] K. Bousselmi, Z. Brahmi, and M. M. Gammoudi, "Cloud services orchestration: a comparative study of existing approaches," in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '14)*, pp. 410–416, Victoria, Canada, May 2014.

[11] A. Lombardo, A. Manzalini, V. Riccobene, and G. Schembra, "An analytical tool for performance evaluation of software defined networking services," in *Proceedings of the IEEE Network Operations and Management Symposium (NMOS '14)*, pp. 1–7, Krakow, Poland, May 2014.

[12] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene, "An open framework to enable NetFATE (network functions at the edge)," in *Proceedings of the IEEE Workshop on Management Issues in SDN, SDI and NFV (Mission '15)*, pp. 1–6, London, UK, April 2015.

[13] 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.

[14] H. Moens and F. De Turck, "VNF-P: a model for efficient placement of virtualized network functions," in *Proceedings of the 10th International Conference on Network and Service Management (CNSM '14)*, pp. 418–423, Rio de Janeiro, Brazil, November 2014.

[15] K. Katsalis, G. S. Paschos, Y. Viniotis, and L. Tassiulas, "CPU provisioning algorithms for service differentiation in cloud-based environments," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 61–74, 2015.

[16] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, "Towards decentralized and adaptive network resource management," in *Proceedings of the 7th International Conference on Network and Service Management (CNSM '11)*, pp. 1–6, Paris, France, October 2011.

[17] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, "DACoRM: a coordinated, decentralized and adaptive network resource management scheme," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '12)*, pp. 417–425, IEEE, Maui, Hawaii, USA, April 2012.

[18] M. Charalambides, D. Tuncer, L. Mamatas, and G. Pavlou, "Energy-aware adaptive network resource management," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 369–377, Ghent, Belgium, May 2013.

[19] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, 2013.

[20] B. Guan, J. Wu, Y. Wang, and S. U. Khan, "CIVSched: a communication-aware inter-VM scheduling technique for decreased network latency between co-located VMs," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 320–332, 2014.

[21] G. Faraci and G. Schembra, "An analytical model to design and manage a green SDN/NFV CPE node," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 435–450, 2015.

[22] L. Kleinrock, "Time-shared systems: a theoretical treatment," *Journal of the ACM*, vol. 14, no. 2, pp. 242–261, 1967.

[23] S. F. Yashkov and A. S. Yashkova, "Processor sharing: a survey of the mathematical theory," *Automation and Remote Control*, vol. 68, no. 9, pp. 1662–1731, 2007.

[24] ETSI GS NFV—INF 001, v1.1.1, "Network Functions Virtualization (NFV): Infrastructure Overview", 2015, https://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf.

[25] T. N. Subedi, K. K. Nguyen, and M. Cheriet, "OpenFlow-based in-network Layer-2 adaptive multipath aggregation in data centers," *Computer Communications*, vol. 61, pp. 58–69, 2015.

[26] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[27] NARR simulator, v.0.1, http://www.diit.unict.it/arti/Tools/NARR_Simulator_v01.zip.

[28] L. Kleinrock, *Queueing Systems. Volume 1: Theory*, Wiley-Interscience, 1st edition, 1975.

[29] R. Bruschi, P. Lago, A. Lombardo, and G. Schembra, "Modeling power management in networked devices," *Computer Communications*, vol. 50, pp. 95–109, 2014.

[30] W. Fischer and K. S. Meier-Hellstern, "The Markov-Modulated Poisson process (MMPP) cookbook," *Performance Evaluation*, vol. 18, no. 2, pp. 149–171, 1993.