ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dottorato di Ricerca in Ingegneria Elettronica, Telecomunicazioni e Tecnologie dell'Informazione

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi"

Ciclo XXIX

Settore concorsuale: 09/F2 - TELECOMUNICAZIONI
Settore scientifico disciplinare: ING-INF/03 - TELECOMUNICAZIONI

VIRTUALIZED NETWORK INFRASTRUCTURES: PERFORMANCE ANALYSIS, DESIGN AND IMPLEMENTATION

Presentata da:
CHIARA CONTOLI

Coordinatore Dottorato:
Chiar.mo Prof. Ing.
ALESSANDRO VANELLI-CORALLI

Relatore:
Chiar.mo Prof. Ing.
FRANCO CALLEGATI

Correlatore:
Chiar.mo Prof. Ing.
WALTER CERRONI

ESAME FINALE 2017

INDEX TERMS

Software Defined Networking

Network Function Virtualization

Service Function Chaining

OpenStack Performance

Intent Driven Networking

"*The only source of knowledge is experience*"

Contents

Summary

In recent decades, there has been a tremendous evolution in the traffic on the Internet and enterprises networks. Networks assisted since the beginning to two phenomena: on the one hand, the birth of a multitude of applications, each posing different requirements; on the other hand, the explosion of personal mobile networking, with an ever increasing demand of devices that require connectivity. These trends resulted in increased network complexity, leading to difficult management and high costs. At the same time, evolution in the Information Technology (IT) field led to the birth of cloud computing and growth of virtualization technologies, opening new opportunities not only for companies but for individuals (be it PC or mobile users), as well as Service and Infrastructure Providers. Emerging technologies such as Software Defined Networking (SDN) and Network Functions Virtualization (NFV) seems to be promising solutions to today's network problems. Neither standardized solutions, nor how to properly combine their usage to achieve flexible and proactive control management have been discovered yet.

This Ph.D. thesis focuses on the exploration of three plane of functionality in which software-defined (computer) networks can be divided: the data, the control and the management plane. SDN aims at introducing network programmability by separating the control from the data plane, besides simplifying network management and the development and deployment of new networking features. Whereas NFV aims at introducing network flexibility by implementing network functionality in software, leveraging IT virtualization techniques, so that can now run on general-purpose hardware. Such flexibility allows for an efficient provision of the network functionality, that can be instantiated, moved or disposed in an on-demand fashion, thus also leading to the benefit of reduced costs and reduced power consumption.

The work presented here is the outcome of part of the research activities carried out by the Network research group at the Department of Electrical, Electronics and Information Engineering "G. Marconi" (DEI), NetLAB (Network Laboratory) at the University of Bologna, Italy. In particular, the activities performed by the Network Research Group have been partially

funded by EIT ICT Labs (now EIT Digital): Activity 2013 “Smart Network at the Edge”, task T1303A - Enable Efficient and Seamless Network-Cloud Integration, Activity 2015 “SDN at the Edges” - Action Line on Future Networking Solutions and, in this latter action line, Activity “CC4BA - Certification Centre for Business Acceleration of SDN and NFV”.

In this thesis we present insights on several aspects of network virtualization, starting from virtual network performance of cloud computing infrastructures, and introducing the Service Function Chaining (SFC) mechanism, discussing its analysis, design and implementation. In particular, the original contribution of this dissertation concerns (i) performance evaluation of the OpenStack cloud platform (the *data plane*); (ii) the design and implementation of a stateful SDN controller for dynamic SFC (the *control plane*); (iii) design, implementation and performance analysis of a proposed Intent-based approach for dynamic SFC (the *management plane*).

Chapter 1

Introduction

1.1 Internet Ossification

Internet and enterprises networks in past decades have experienced two different trends: i) an ever increasing birth of different applications; ii) an ever increasing demand of connectivity by devices, be it mobile or personal. Applications range from e-mail exchange, file transfer, browsing hyper-text contents (the very early Internet applications), to buying on-line, videoconferencing, online video gaming, watching movies, downloading music and others multimedia contents, etc., which are now the dominant applications that have caused a significant evolution in the traffic of both Internet and enterprises networks. Such applications pose different requirements to the network in terms of delay, jitter and loss tolerance besides throughput demand; these are also known as Quality of Service (QoS) parameters. For example, e-mail can tolerate high value of delay, jitter and loss, and demands low throughput, while applications such as videoconferencing and Voice over IP (VoIP) can tolerate low value of delay, jitter and loss and demand medium/high and low value of throughput, respectively. Keeping an acceptable level of performance is not the only challenge a network has to deal with; other challenges, for example, are security, host mobility, (dynamic) assignment of (private) addresses that all together made the network evolve in such a way that deploying a new service in todays networks is difficult.

To cope with such challenges, networks have been enhanced with several functions, such as Intrusion Detection Systems (IDS), Deep Packet Inspector (DPI), Firewalls, Network Address Translator (NAT), packet filtering, Wide Area Network Accelerator (WANA), load balancer, traffic shaper, just to name a few. Such functions are usually implemented on closed, proprietary and expensive hardware by a plurality of vendors, and are also known as *middle-boxes.* As surveyed by Sherry et al. in 2012 [1], on a study of 57 enterprise networks, the number of middle-boxes deployed is comparable to the number of routers in a network. Moreover, large networks in window time of 5 years have spent over a million of dollars on middle-boxes, while small to medium networks have spent between \$5,000-\$50,000. Last, but not least, the multiplicity of vendors equipment requires high-skilled teams with the necessary expertise to be capable of managing all the heterogeneous devices composing the network. An additional obstacle comes from the fact that updates for such devices comes with hardware upgrade, thus leading to the so called vendor lock-in effect. This effect, so far, resulted in deploying new hardware to get new features, making todays networks expensive and difficult to manage.

1.2 A modern approach to networking

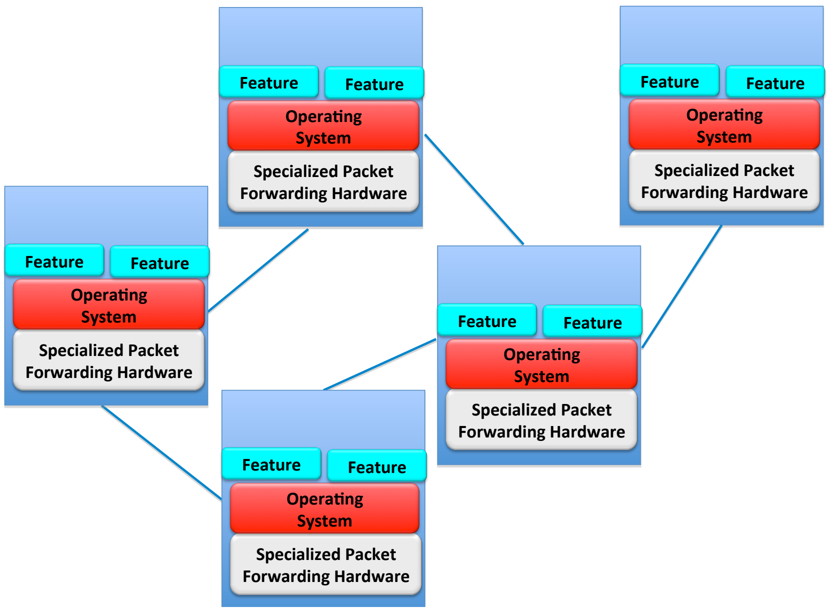
Promising solutions seem to come from emerging paradigms such as Software Defined Networking (SDN), Network Function Virtualization (NFV) and advances made in the Information Technologies (IT) field.

1.2.1 Software Defined Networking

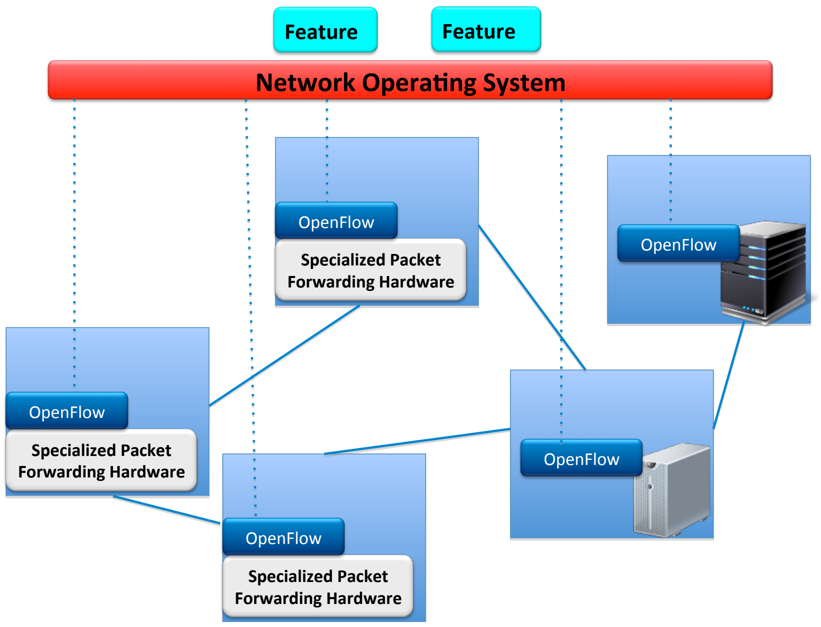
A comparison between traditional and SDN approach is given in Fig. 1.1[1]. Historically, control plane and data plane have always been co-located, together with provided features, on the same specialized hardware as shown in Fig. 1.1a. With such approach, proper control routing protocols (e.g., Open Shortest Path First, Boarder Gateway Protocol) run in a distributed fashion to discover, compute adjacency and learn the complete network topology. Instead, the SDN paradigm refers to an architecture that decouple the control plane from the data plane, thus moving the intelligence of the network to a (logically) centralized entity known as controller, as can be seen in Fig. 1.1b. With this approach, devices are just devoted to simply forwarding packets, while the control logic is moved to a Network Operating System (NOS) and applications running on top of it. To achieve this separation, two interfaces are provided: i) the North Bound Interface (NBI), ii) the South Bound Interface (SBI).

The NBI is located between the NOS and the applications and its goal is twofold: on the one hand, it provides an API (Application Programming Interface) to applications developers; on the other one, it provides abstractions to hide lower level details (e.g., how the forwarding devices are programmed).

[1]Source: S.Seetharaman, OpenFlow/SDN tutorial, OFC/NFOEC 2012



(a) Traditional network paradigm.



(b) SDN network paradigm.

Figure 1.1: The SDN network innovation.

The SBI is instead located between the NOS and the infrastructure composed of forwarding devices, with a twofold goal analogous to the one offered by the NBI: on the one hand, it provides an open interface that allow to program forwarding tables; on the other one, since southbound protocol can be seen as plug-in between the NOS and the infrastructure, it makes the NOS protocol agnostic.

The most adopted southbound interface is OpenFlow [2], which provides a protocol for the interaction between the NOS and underlying (OpenFlow enabled) devices (as can also be seen in Fig. 1.1b). OpenFlow provides an abstraction of a pipeline of flow tables; flow tables are composed of entries characterized by:

- a matching rule;
- actions to be executed on matching packets;
- counters to keep track of flows/packets statistics.

Identifying SDN with OpenFlow would not be correct, especially because SDN is not a new concept. As well surveyed by Kreutz et al. in [3], SDN has a long history that comes from programmable networks, such as active networks, programmable ATM networks and prior attempt to separate control and data plane, such as NCP (Network Control Platform) and RCP (Routing Control Platform), which find roots in the 80s and 90s. Besides OpenFlow, other more recent approaches for data plane programmability comes from ForCES (Forwarding and Control Element Separation) and POF (Protocol Oblivious Forwarding). In such SDN survey, authors also clarify the fact that also the concept of NOS is not new; in fact, traditional network devices are equipped with proprietary operating systems (e.g., Cisco IOS, JunOS, to name a few) that allow to manage and configure devices using device-specific commands. The problem is that network designers have to deal with the lack of abstractions for device details, and deal with complicated distributed algorithms. Instead, in SDN approach the NOS is usually implemented by control platforms (e.g., NOX, ONOS, OpenDaylight, OpenContrail, to name a few) in a software fashion that runs on commodity hardware. Control platforms differs in architecture and design elements, besides in services, components and interfaces (and the way those are) provided.

Therefore, main advantages introduced by the SDN approach can be summarized as follow:

- simplifying network management;
- simplifying network policy programmability;

- reducing complexity when it comes to give birth to new protocols and network applications.

To recap, such advantages are brought both by the abstractions provided by NOS to network developers and external applications willing to interact with the control platform, and by the open programmable interface that provides an abstraction for the forwarding mechanism, which is vendor independent and so do not require to know vendor-specific implementations, thus also leading to increased network programmability.

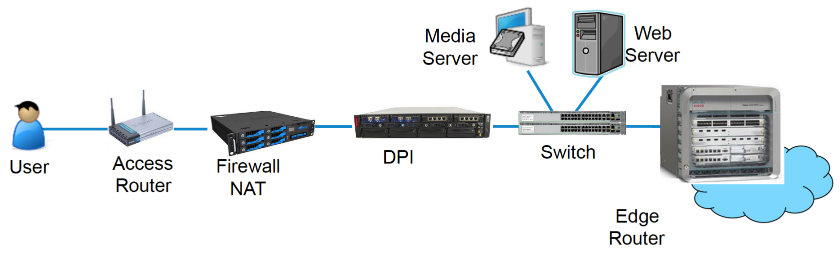
1.2.2 Network Function Virtualization

A complementary technology to SDN is NFV, and a comparison with traditional approach is given in Fig. 1.2. As can be seen in Fig. 1.2a, traditional approach consider proprietary hardware devices (middle-boxes) deployed along the path between source and destination. Operators typically apply an ordered set of network functions, i.e. a service chain, to a given traffic class or to a given user (or class of users). In the case of functions implemented by middle-boxes this requires vendor-specific configuration and management tasks, i.e., reconfiguring switches, routers, network connectivity, etc. The increased complexity and costs due to high capitol investments and network management encouraged telecommunication network operators, infrastructure providers (IPs) and enterprises to seek for alternative approaches. With the complicity of advances in virtualization technologies made in the IT fields, and the high capacity of standard server, NFV came to birth. As shown in Fig. 1.2b, network functions are now virtualized in software that can run on standard server hardware. So, NFV decouples the network functionality from underlying proprietary hardware, opening new opportunities for companies, Service and Infrastructure Providers.

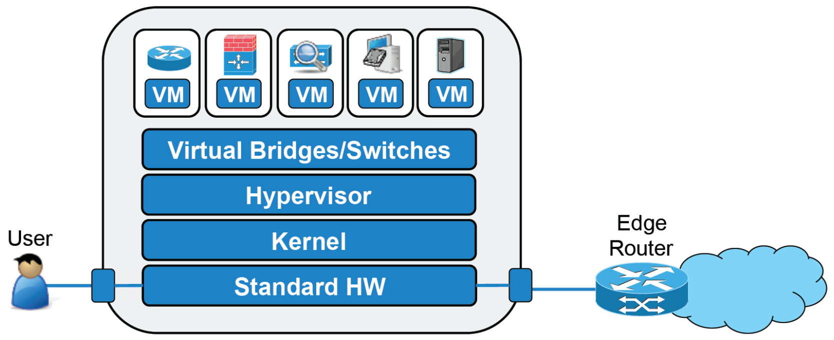
As the European Telecommunications Standard Institute (ETSI) says in its first white paper [4], NFV takes advantage of two enabling technologies:

- cloud computing;
- industry standard high volume servers.

In particular, the cloud computing provides matured hardware virtualization mechanisms through hypervisors, as well as software Ethernet switches that allow traffic to flow between Virtual Machines (VMs) and physical interfaces. The adoption of such mechanism for Network Functions (NFs) leverage the cloud computing paradigm, that is, the service on-demand model; with this model, NFs could be instantiated, removed or migrated to any location in the



(a) Traditional network devices deployment: hardware middle-boxes.



(b) NFV network devices deployment: commodity hardware.

Figure 1.2: The NFV innovation.

network, without the need of deploying new hardware. Such model is also fostered by cloud computing infrastructure, that allow to virtualize compute, storage as well as network resources thus creating an unprecedented degree of flexibility. Moreover, cloud computing infrastructure should be capable of providing orchestration and management mechanisms that allow to automate VMs instantiation and deal with, for example, VMs allocation and recovery due to failure.
High volume servers adoption allow to take advantage of the economies of scale and of general purpose hardware that could replace middle-boxes to implement virtual network appliances. In fact, if we look again at Fig. 1.1b, according to the SDN vision the underlying infrastructure can be composed both of traditional proprietary solution running on specialized hardware and commodity white box solution[2].

Therefore, main advantages introduced by NFV approach can be summarized as follow:

- increasing flexibility and efficient provisioning of network functions;

[2]http://www.opencompute.org/projects/networking/

- reducing equipment costs and power consuption;
- reducing time-to-market by allowing software and hardware to evolve independently;
- supporting hardware resource sharing among multiple (concurrent) software instances, providing proper isolation mechanisms.

1.3 Motivation

SDN and NFV are independent of each other; NFV can be implemented without SDN being in place and vice-versa, but can leverage mutual benefits. Combining their usage can bring advantages to the telecommunications industry in terms of Operational Costs (Opex) and Capital Expenditure (Capex) reduction; nevertheless, this evolution does not come without challenges. In this thesis, we focus in particular on performance trade-off and management concerns, considering scenario of future telecommunication network infrastructures that are of interest for network operators, service and infrastructure providers.

Such evolution will likely take place at network edges, where most of network functionalities are located. The vision is that future edge networks will take the shape of a data centers (interconnected by a stateless optical core network) where NFs will be provisioned. Several service providers (e.g., AT&T, SK Telecom to cite a few) are leading both research and development in this direction, aiming at providing to their customers solutions that take full advantage of combination among SDN, NFV and elasticity of commodity clouds so that they are able to bring data centers economies and cloud agility to Telco Central Office[3].
If this will be the case, in spite of improved standard hardware performance, it is worth investigating if virtualized networks will provide performance comparable to those achieved with current physical networks. Therefore, performance of underlying platform should be known in order to achieve a better planning about how properly dimension networks and virtual appliances placement.

Both industry and academia are encouraged to contribute to help the process of convergence between IT and network standardization. Therefore, several effort is put in the definition of reference SDN-NFV architectures that combines several functional blocks that need to be properly managed and orchestrated. For example, Verizon service provider published the first

[3]http://opencord.org/

version of its SDN-NFV architecture as a network infrastructure planning in February 2016 [5]. From an operational efficiency point of view, dynamic traffic steering and Service Function Chaining (SFC) are one of the main goals of SDN-NFV architectures.
Without doubt, dynamic traffic steering and SFC are correlated; SFC is about chaining NFs that can be deployed on the same physical node or across multiple ones; in fact, being capable of dynamically steering traffic towards required NFs is a key factor. Therefore, combining SDN and NFV aspects requires investigation from i) control plane and ii) management & orchestration perspective. Orchestration can be seen as a subset of the management in the context of combining NFV and SDN.

Control plane investigation is needed to understand how control plane can achieve proper traffic steering taking into account the fact that it has to deal with a multiplicity of users, and the fact that service chain can change over time. Management & orchestration, considering as a reference architecture the one proposed by ETSI [6], is a broad topic, but one of the most critical interface is the one between the cloud orchestration systems and SDN controllers[4]. The orchestrator component in the ETSI architecture is defined as a functional block that has several responsibilities. According to the Open Networking Foundation (ONF), such critical interface could benefit of the so called Intent-based approach; recently, ONF published a technical report in which definitions and principles of Intent NBI approach is given [7] but how to implement such interface is left open.

1.4 Thesis Contributions

How to properly combine SDN and NFV usage to achieve flexible and proactive control management, nor standardized solutions have been discovered yet. In order to help answer to this question and to help addresses performance, and management & orchestration concerns described in the previous section, we performed a study that cross all the three planes of functionality a SDN-NFV network can be divided in: i) data plane; ii) control plane; iii) management plane. In this thesis we make the following contributions:

- **Performance Analysis:** such contribution is given at all planes, and can be summarized as follow:
    - *data plane:* in order to explore performance concerns of underlying platform, we lead a deep investigation of performance evaluation

[4]http://www.opennetworking.org/?p=1633&option=com_wordpress&Itemid=155

of one of the most widely adopted cloud platform: OpenStack. We focus in particular on performance of its virtual network infrastructure, as well as on performance of simple NFV use cases scenario deployment (Chapter 2).

  - *control plane:* we investigate the performance of a stateful approach to dynamic traffic steering aimed at achieving fully dynamic SFC; such evaluation is led both on an OpenStack deployment at the University of Bologna, and on a real production environment (Chapter 3).
  - *management:* we investigate the performance of our proposed Intent NBI approach aimed at providing a vendor-independent, technology agnostic technique for controlling dynamic SFC on top of a SDN infrastructure (Chapter 3).

- **Design:**
  - *control plane:* we propose the design of a stateful SDN control plane as a general approach to service chain reconfiguration; in particular, we leverage the abstraction provided by a Mealy machine, that allow us to model the controller behavior to achieve fully dynamic and adaptive SFC (Chapter 3).
  - *management:* we propose the design of vendor-independent, technology agnostic Intent-based NBI approach that could be put in place between an orchestrator and a SDN controller platform; our approach allow to dynamically handle a SFC on top of SDN infrastructure (Chapter 4).

- **Implementation:**
  - *control plane:* we implemented the stateful finite state machine on the Ryu framework; in order to prove the feasibility of our approach, we developed a proof-of-concept (PoC) both on an OpenStack deployment at the University of Bologna and on a real like production environment (Chapter 3).
  - *management:* to show the feasibility of our approach, we implemented such Intent NBI as an application running on top of the ONOS control platform; whereas the first validation of our PoC is performed on top of an OpenFlow network infrastructure emulated with Mininet (Chapter 4).

Chapter 2

Performance of Network Virtualization in Cloud Infrastructures

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 [8]. 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 [9].

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 [10], 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 [11].

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 [12, 13, 14]. 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 [15]. 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 [16].

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 [17, 18], while others investigated network performance in public cloud services [19]. Solutions for low-latency SDN implementation on high-performance cloud platforms have also been developed [20]. 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 [21], a detailed analysis of their network performance has not been provided yet.

This chapter 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 [22]. The 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 [23, 24].

In the following, we briefly introduce in section 2.1 the main concepts of network virtualization, discussing examples of virtualization techniques; we dive into the OpenStack virtual network architecture explanation (section 2.2), focusing our attention on the main components of the infrastructure and its network elements; then we present the experimental test-bed setup that we have deployed to assess OpenStack performance, investigating several use case scenarios (section 2.3). Finally, we discuss the numerical results and draw some conclusions (sections 2.4 and 3.3, respectively).

2.1 Main concepts of 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 [9].

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 [8].

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) [25]. 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 placement algorithms may be designed to take networking issues into account thus optimizing the networking in the cloud together with computation effectiveness [26]. 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 [27].

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 [28]. 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 [29]. 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 [30]. OVS is also OpenFlow-enabled and therefore fully compatible and integrated with SDN solutions.

2.2 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 [1] 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 Fig. 2.1, a typical OpenStack cloud is composed of a number of physical nodes and networks:

- *controller node*: manages the cloud platform;
- *network node*: hosts the networking services for the various tenants of the cloud and provides external connectivity;
- *compute nodes*: as many hosts as needed in the cluster to execute the VMs;
- *storage nodes*: to store data and VM images;
- *management network*: the physical networking infrastructure used by the controller node to manage the OpenStack cloud services running on the other nodes;
- *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;
- *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

[1]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

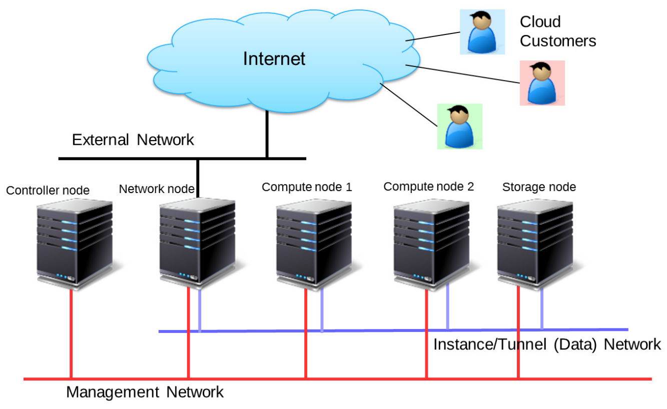


Figure 2.1: Main components of an OpenStack cloud setup.

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:

- *network*, a virtual layer 2 segment;
- *subnet*, a layer 3 IP address space used in a network;
- *port*, an attachment point to a network and to one or more subnets on that network;
- *router*, a virtual appliance that performs routing between subnets and address translation;
- *DHCP server*, a virtual appliance in charge of IP address distribution;
- *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 [31]. 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 Figs. 2.2 and 2.3, respectively.

Each node runs an 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 (Fig. 2.2) includes *br-tun* for the instance/tunnel network and *br-ex* for the external network. 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 a 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.
A compute node (Fig. 2.3) includes *br-tun* only. 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*.

Layer 2 virtualization and multi-tenant 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), that allow to extend the local virtual networks also to remote data centers [30]. The examples shown in Figs. 2.2 and 2.3 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

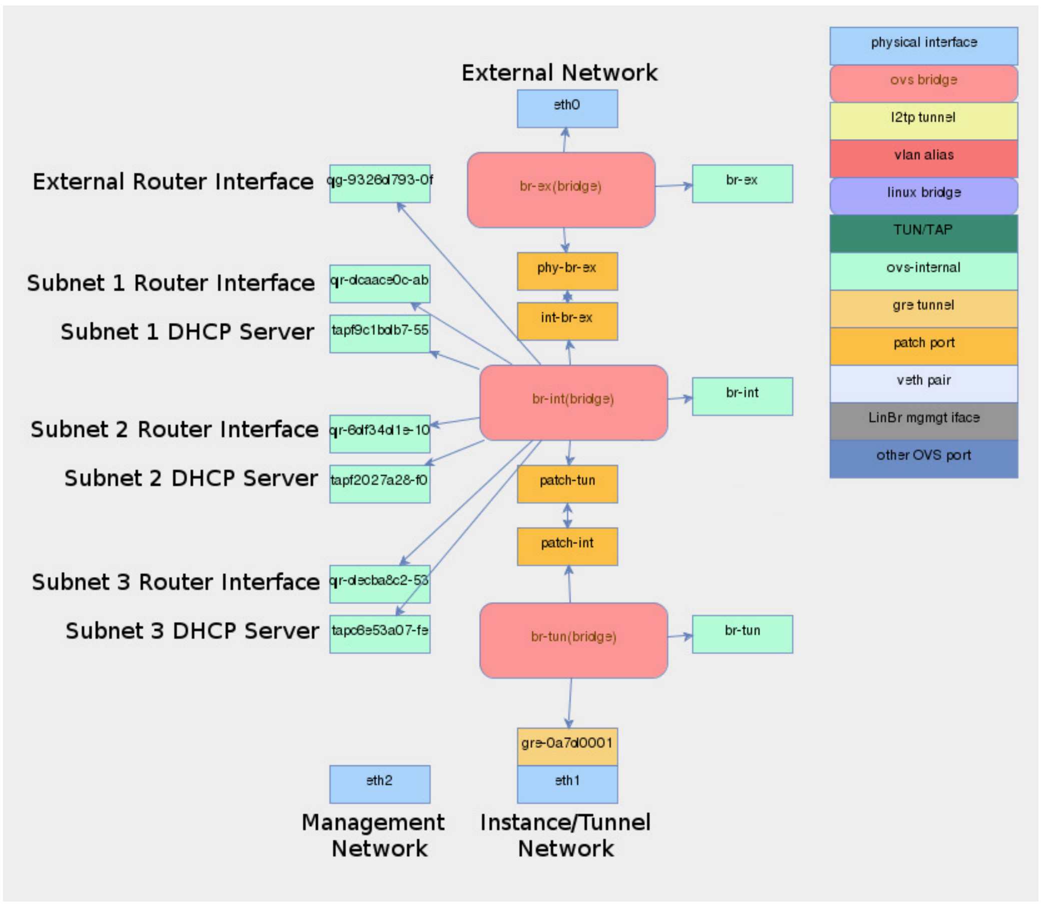


Figure 2.2: Network elements in an OpenStack network node connected to three virtual subnets.

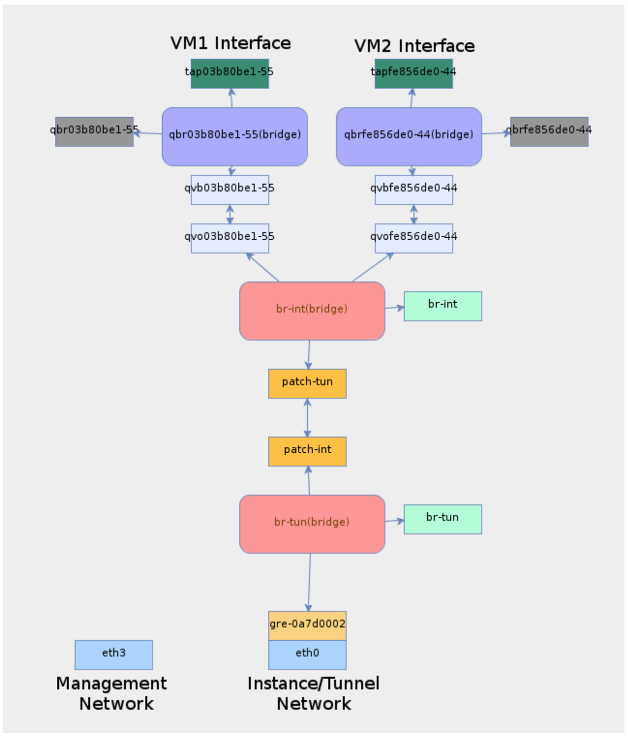


Figure 2.3: Network elements in an OpenStack compute node running two VMs.

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:

- the OpenStack scheduler component running in the controller node chooses the compute node that will host the VM;
- a *tap interface* is created for each VM network interface to connect it to the Linux kernel;
- a Linux Bridge dedicated to each VM network interface is created (in Fig. 2.3 two of them are shown) and the corresponding tap interface is attached to it;
- 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 anti-spoofing 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 in 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.

2.3 Experimental Testbed

The previous section makes clear the complexity of the OpenStack virtual network infrastructure. To understand optimal design strategies in terms on 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 with 8 GB of RAM and a quad-core processor enabled to hyper-threading, 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 multi-tenant NFV scenario with dedicated network functions;

3. a multi-tenant 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, both for traffic generation and measurement [32]. In some cases, the *Iperf3* tool was also added to generate background traffic at a fixed data rate [33]. All physical interfaces involved in the experiments were Gigabit Ethernet network cards.

2.3.1 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 Fig. 2.4. 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 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*, which 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:

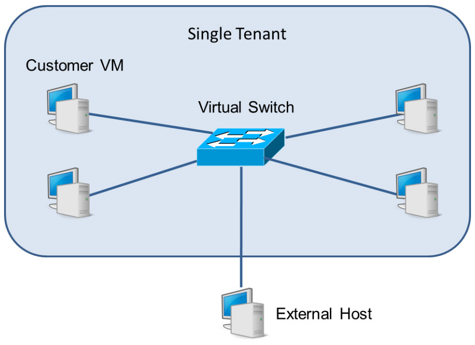


Figure 2.4: Reference logical architecture of a single-tenant virtual infrastructure with 5 hosts: 4 hosts are implemented as VMs in the cloud and are interconnected via the OpenStack layer-2 virtual infrastructure; the 5th host is implemented by a physical machine placed outside the cloud, but still connected to the same logical LAN.

(1.1) a single compute node executing two co-located VMs;

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

2. *Non-OpenStack scenario*, which 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 co-located VMs, acting as sender and receiver and directly connected to the same Linux Bridge;

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

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

2.3.2 Multi-tenant NFV scenario with dedicated network functions

The multi-tenant scenario we want to analyze is inspired by a simple NFV case-study, as illustrated in Fig. 2.5: 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 [34]. It is assumed that

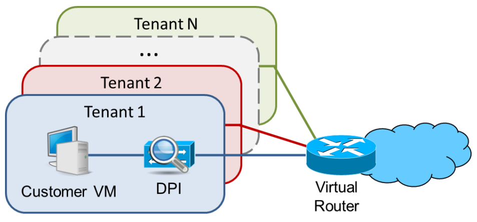


Figure 2.5: Multi-tenant NFV scenario with dedicated network functions tested on the OpenStack platform.

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 2.6 shows the view of the cluster setup 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. Each slice includes a 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 of the figure. The choice of 4 tenants was made to provide meaningful results with an acceptable degree of complexity, without lack of generality. As results shows 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 Fig. 2.5 is not commonly seen in a traditional cloud computing environment. The VMs usually behave as single hosts connected as end-points 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

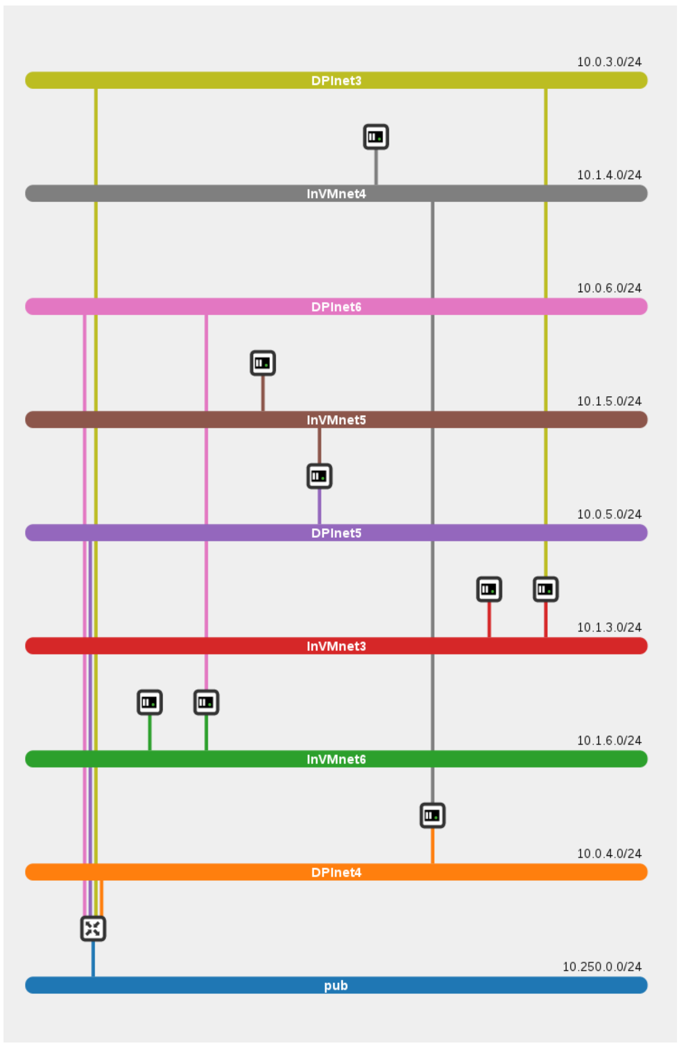
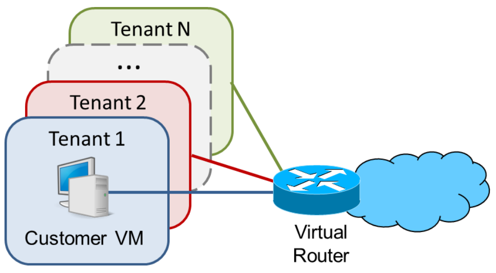


Figure 2.6: The OpenStack dashboard shows the tenants virtual networks (slices).

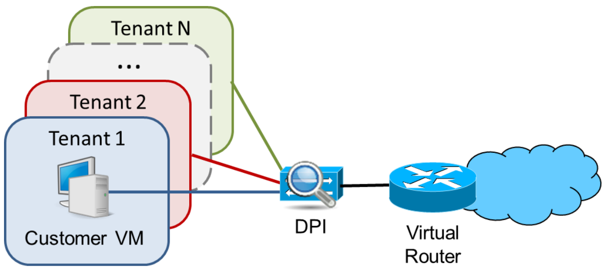
that require packet forwarding. Network Address Translators (NATs), Deep Packet Inspectors (DPIs), etc. all belong to this category. If such VNFs are hosted in VMs the result is that VMs in the OpenStack infrastructure must be allowed to perform packet forwarding which goes against the typical rules implemented for security reasons in OpenStack. For instance when a new VM is instantiated it is attached to a Linux bridge to which are applied filtering rules with the goal to avoid that the VM sends packet with MAC and IP addresses that are not the ones allocated to the VM itself. Clearly this is an anti-spoofing 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 multi-tenant setup.

2.3.3 Multi-tenant NFV scenario with shared network functions

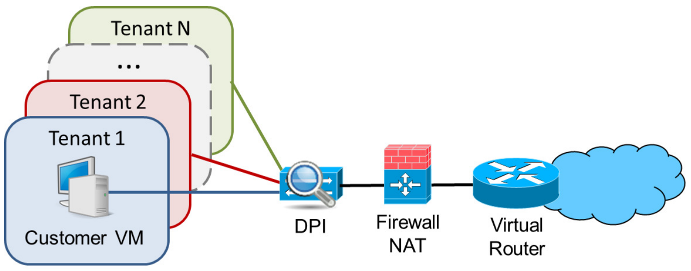
We finally extend our analysis to a set of multi-tenant scenarios assuming different levels of shared VNFs, as illustrated in Fig. 2.7. We start with a single VNF, i.e. the virtual router connecting all tenants to the external network (Fig. 2.7a). Then we progressively add a shared DPI (Fig. 2.7b), a shared firewall/NAT function (Fig. 2.7c) and a shared traffic shaper (Fig. 2.7d). The rationale behind this last group of setups is to evaluate how a NFV deployment on top of an OpenStack compute node performs under a realistic multi-tenant scenario where traffic flows must be processed by a chain of multiple VNFs. The complexity of the virtual network path inside the compute node for the VNF chaining of Fig. 2.7d is displayed in Fig. 2.8. 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. The red dashed line shows the path followed by the packets traversing the VNF chain displayed in Fig. 2.7d.



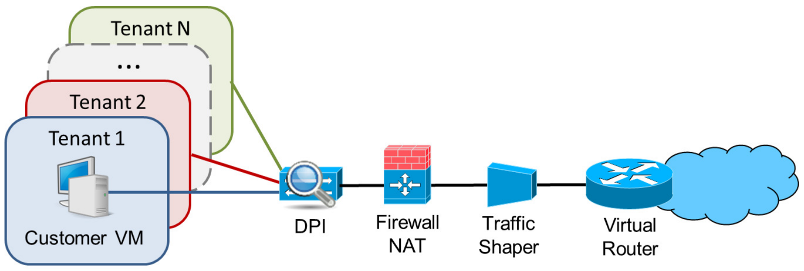
(a) Single VNF.



(b) Two VNFs chaining.



(c) Three VNFs chaining.



(d) Four VNFs chaining.

Figure 2.7: Multi-tenant NFV scenario with shared network functions tested on the OpenStack platform.

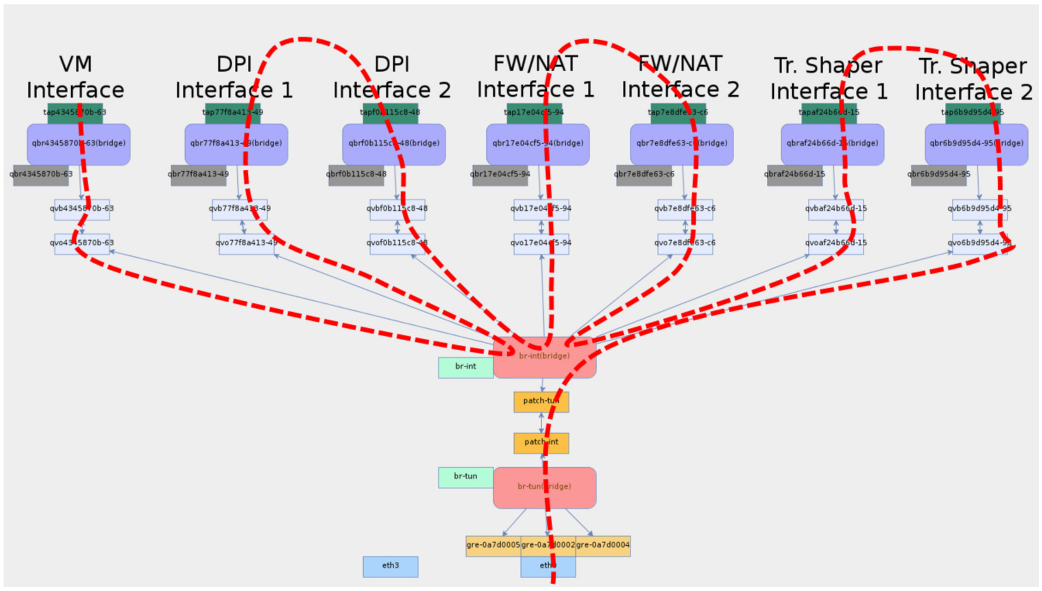


Figure 2.8: A view of the OpenStack compute node with the tenant VM and the VNFs installed including the building blocks of the Virtual Network Infrastructure.

2.4 OpenStack performance: numerical results

2.4.1 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), both for 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

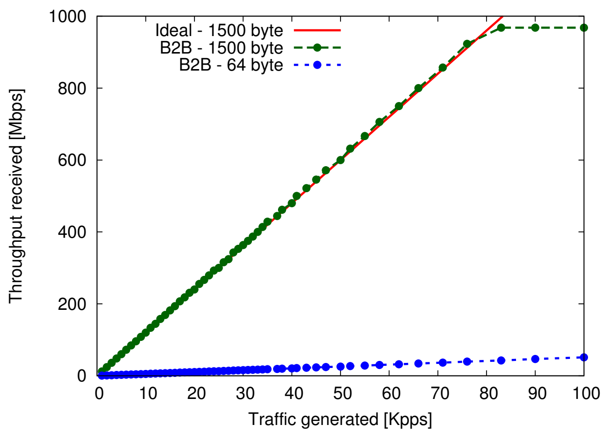


Figure 2.9: Throughput vs. generated packet rate in the B2B setup for 64 and 1500-byte packets. Comparison with ideal 1500-byte packet throughput.

factor is not the Ethernet bandwidth, but the maximum sustainable packet processing rate of the computer node. These results are shown in Fig. 2.9.

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.

2.4.2 Single-tenant cloud computing scenario

The first series of results is related to the single-tenant scenario described in section 2.3.1. Figure 2.10 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 on performance. Setup (1.1) with the two VMs co-located in the same compute node clearly is more demanding since the compute node has to process the workload of all the components shown in Fig. 2.3, i.e. packet generation and reception in two VMs and layer 2 switching in two Linux Bridges and two OVS bridges (as a matter of fact

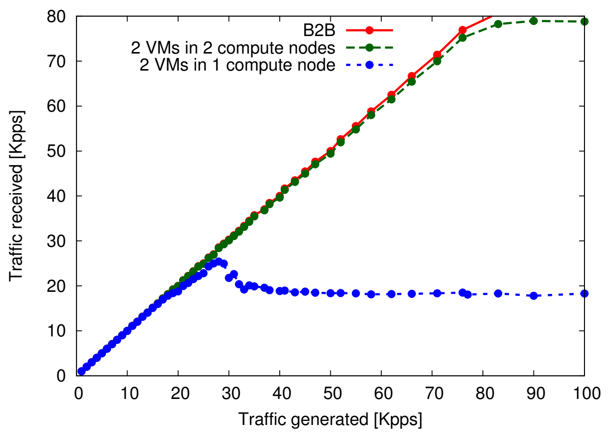


Figure 2.10: Received vs. generated packet rate in the OpenStack scenario setups (1.1) and (1.2), with 1500-bytes packets.

the packets are either 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 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 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 2.11. The graphs show interesting and important results that can be summarized as follows:

- the introduction of some virtual network component (thus introducing the processing load of the physical hosts in the equation) is always a cause of performance degradation but with very different degrees of magnitude depending on the virtual network component;

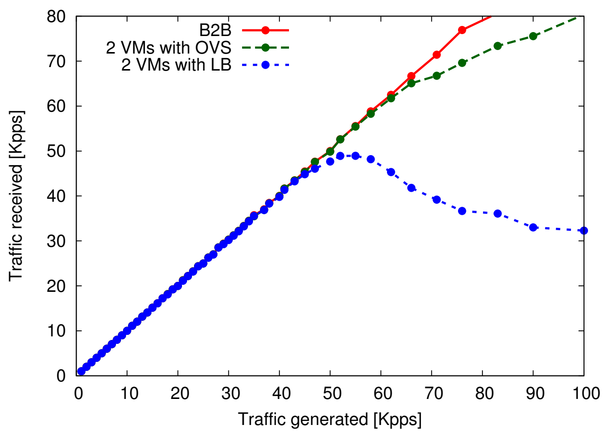


Figure 2.11: Received vs. generated packet rate in the Non-OpenStack scenario setups (2.1) and (2.2), with 1500-bytes packets.

- OVS introduces a rather limited performance degradation at very high packet rate with a loss of some percent;
- 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 of the OpenStack performance degradation. Results obtained from testing setup (2.3) are displayed in Fig. 2.12 confirming that with OVS it is possible to reach performance comparable with the baseline.

2.4.3 Multi-tenant NFV scenario with dedicated network functions

The second series of experiments was performed with reference to the multi-tenant NFV scenario with dedicated network functions described in section 2.3.2. The case study considers that different numbers of tenants are hosted in the same compute node, sending data to a destination outside the local LAN, therefore beyond the virtual gateway. Figure 2.13 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

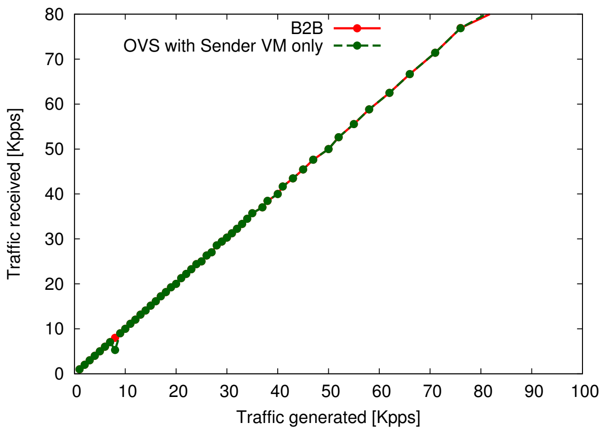


Figure 2.12: Received vs. generated packet rate in the Non-OpenStack scenario setup (2.3), with 1500-bytes packets.

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, i.e. when the total packet rate is slightly more than 80 Kpps, corresponding to 1 Gbps.

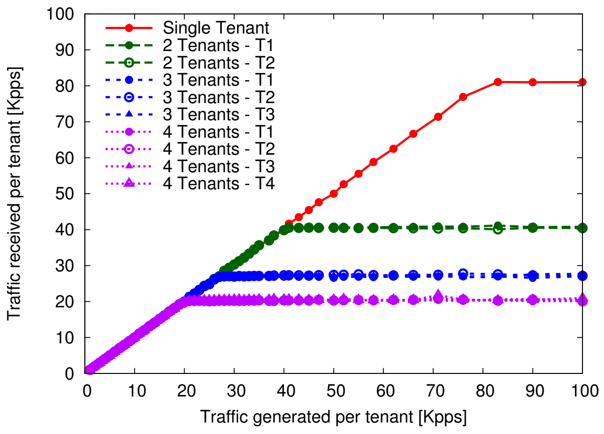


Figure 2.13: Received vs. generated packet rate for each tenant (T1, T2, T3 and T4), for different numbers of active tenants, with 1500-byte IP packet size.

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 2.14 reports the 64-byte packet size case. As discussed

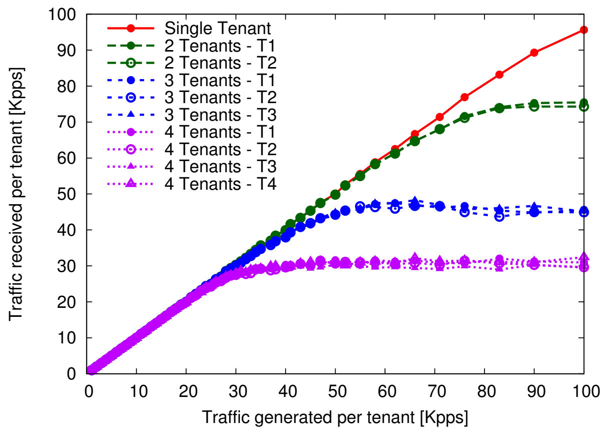


Figure 2.14: Received vs. generated packet rate for each tenant (T1, T2, T3 and T4), for different numbers of active tenants, with 64-byte IP packet size.

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.*[2] As could be easily expected from the results presented in Fig. 2.9, the virtual network is not able to use the whole physical capacity. Even in the case of just one tenant, a total bit rate of about 77 Mbps, well below 1 Gbps, is measured. Moreover *this penalty increases with the number of tenants* (i.e., with the complexity of the virtual system). With two tenants the curve saturates at a total of approximately 150 Kpps (75×2), with three tenants at a total of approximately 135 Kpps (45×3), and with four tenants at a total of approximately 120 Kpps (30×4). This is to say that an increase of one unit in the number of tenants results in a decrease of about 10% in the usable overall network capacity and in a similar penalty per tenant.

Given the results of the previous section, it is likely that the Linux bridges are responsible for most of this performance degradation. In Fig. 2.15 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

[2]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 processing and switching in three Linux Bridges per tenant and two OVS bridges.

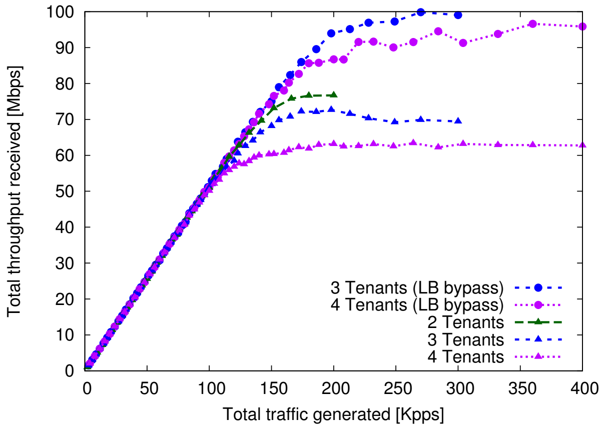


Figure 2.15: Total throughput measured vs. total packet rate generated by 2 to 4 tenants for 64-byte packet size. Comparison between normal OpenStack mode and Linux Bridge bypass with 3 and 4 tenants.

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.

2.4.4 Multi-tenant NFV scenario with shared network functions

The third series of experiments was performed with reference to the multi-tenant NFV scenario with shared network functions described in section 2.3.3. In each experiment, four tenants are equally generating increasing amounts of traffic, ranging from 1 to 100 Kpps. Figures 2.16 and 2.17 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 VNFs and sink involved in the chain are: DPI (deep packet inspection), FW (firewall/NAT), TS (traffic shaper), VR (virtual router) and DEST (destination/sink). 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) does not significantly impact the forwarding performance of the OpenStack compute node for a packet rate below 50 Kpps.[3] Then the throughput slowly degrades. In contrast, when 64-byte packets are generated, even a single VNF can cause

[3]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 Fig. 2.13.

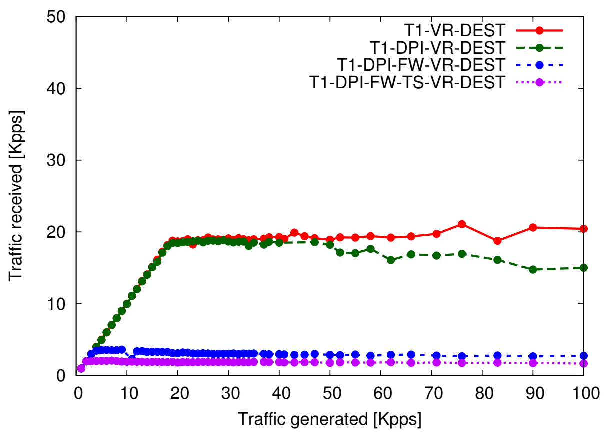


Figure 2.16: Received vs. generated packet rate for one tenant (T1) when four tenants are active, with 1500-byte IP packet size and different levels of VNF chaining as per Fig. 2.7.

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, were 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 Fig. 2.7a), whereas flows originated from T2, T3 and T4 are processed by four VNFs (as in Fig. 2.7d). The results presented in Figs. 2.18 and 2.19 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 Fig. 2.20, we can reach the same conclusions as in the UDP case.

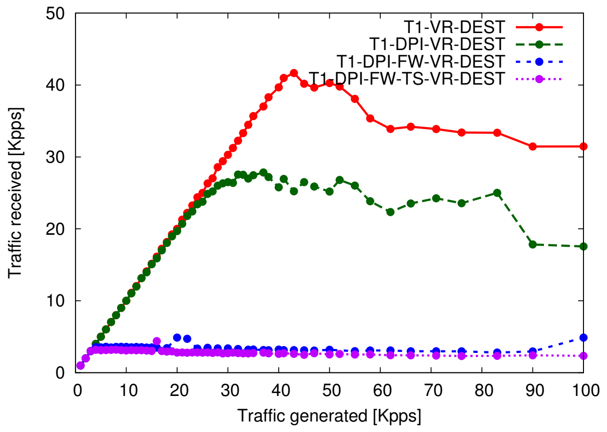


Figure 2.17: Received vs. generated packet rate for one tenant (T1) when four tenants are active, with 64-byte IP packet size and different levels of VNF chaining as per Fig. 2.7.

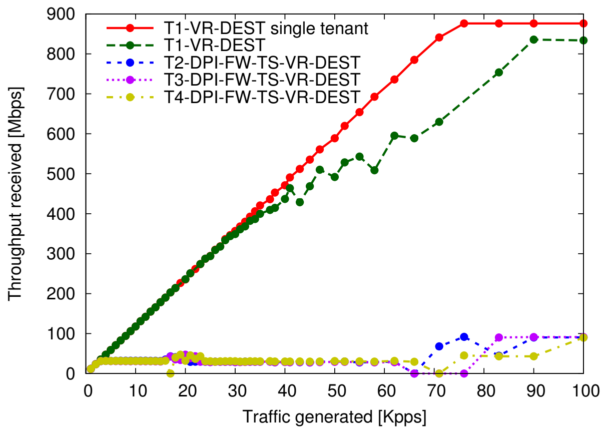


Figure 2.18: Received throughput vs. generated packet rate for each tenant (T1, T2, T3 and T4) when T1 does not traverse the VNF chain of Fig. 2.7d, with 1500-byte IP packet size. Comparison with the single tenant case.

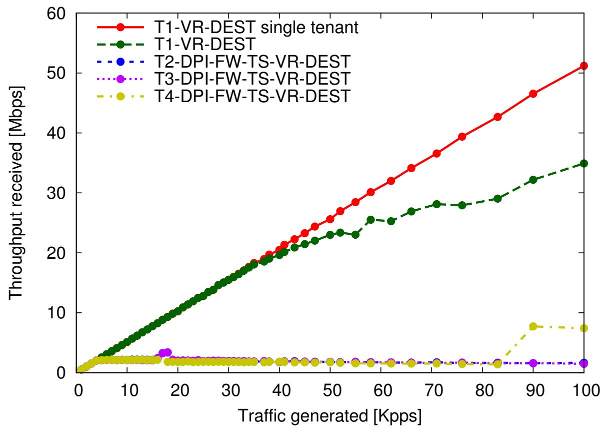


Figure 2.19: Received throughput vs. generated packet rate for each tenant (T1, T2, T3 and T4) when T1 does not traverse the VNF chain of Fig. 2.7d, with 64-byte IP packet size. Comparison with the single tenant case.

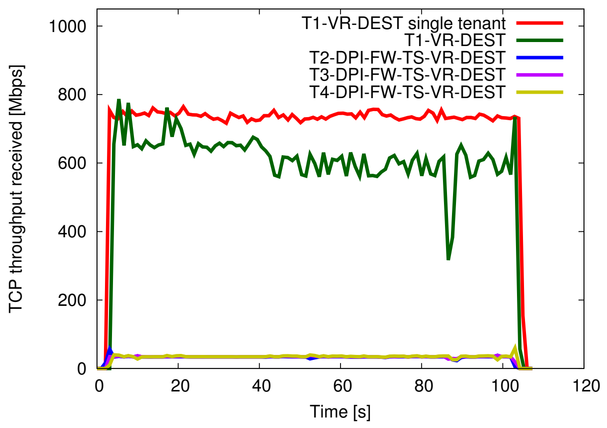


Figure 2.20: Received TCP throughput for each tenant (T1, T2, T3 and T4) when T1 does not traverse the VNF chain of Fig. 2.7d. Comparison with the single tenant case.

2.5 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 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.

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 to observe 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 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 re-design 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.

After this deep investigation of the data plane performance, and a former study of how to implement a NFV use case on the OpenStack platform, we moved our attention on the control plane of the network. In particular,we focused on the Service Function Chaining mechanism, which is considered one of the crucial network capability from an operational efficiency point of view for a telecommunication operator. Design and evaluation aspects on implementing dynamic NFV are investigated and discussed in chapter 3.

Chapter 3

SDN Control plane design for Dynamic NFV

The Service Function Chaining is defined as an ordered set of Service Functions (SFs) that are chained together and that are in charge of analyze, classify, condition, and secure data traffic. Such functions have been traditionally implemented by means of vendor-dependent middle-boxes; but since NFV is gaining a lot of interest as a flexible and cost-effective solution for replacing such hardware-based middle-boxes with software appliances running on general purpose hardware in the cloud [8], a shift of paradigm is expected, and will most likely take place at the network edges, where most of the these functions are located [12].

This chapter will first investigate the issues of implementing chains of network functions in a "softwarized" environment and, in particular, the complexity of the SDN control plane within a cloud-based edge network implementing NFV (section 3.1). Then, we investigate design methodology for implementing the SDN control plane capable of steering specific data flows toward the required VNF locations and achieving fully dynamic service chaining section (section 3.2). Such methodology was formerly implemented on the OpenStack platform to provide a practical example of the feasibility and degree of complexity of the proposed approach. Then, the work was further elaborated in a joint collaboration with *Telecom Italia - Strategy and Innovation, Italy*, and *Ericsson Telecomunicazioni S.p.A, Italy*, whose goal was to provide a proof of concept (PoC) demonstrating the added value that dynamic software-defined networking control, coordinated with a flexible cloud management approach, brings to telco operators and service providers.

3.1 Dynamic chaining of Virtual Network Functions in edge-networks

Future computing, storage, and connectivity services will be provided by software-defined infrastructures built according to the cloud paradigm, where network functions can be virtualized and executed on top of general-purpose hardware [35]. One of the biggest advantages brought by NFV will be the possibility for network operators to dynamically and flexibly select and apply the specific edge network functions needed for a given class of user data traffic at a given time. Also, owing to the software nature of these functions, infrastructure providers will be able to place specific instances where they are actually needed, and to dynamically migrate, duplicate, or delete them according to the emerging requirements. This approach will allow Telco operators to successfully combine a flexible enforcement of user's Quality of Service (QoS) with an efficient communication resource utilization.

Such flexibility in deploying virtual edge network functions independently of the underlying hardware must be coupled with as much elasticity in controlling the user data traffic independently of the specific forwarding plane adopted. Indeed, the capability of steering specific data flows towards the required virtual appliance locations is a key factor to achieve fully dynamic service chaining. Even before the advent of NFV, Software Defined Networking (SDN) solutions, such as OpenFlow [2], have been considered as a viable option to flexibly steer traffic towards the required middle-boxes, posing design challenges and integration issues with existing infrastructures [36], [37]. Other key aspects include a correct data-plane design to achieve full network programmability [38], as well as high-performance implementation of the Virtual Network Functions (VNFs) [39]. Previous work also discussed the challenges behind a control plane that is able to dynamically manage the VNF state taking into account the related SDN forwarding state [40].

In the following subsections, we intend to describe the actions to be programmed into OpenFlow switches in order to achieve dynamic service chaining for two representative case studies, namely Layer 2 (L2) and Layer 3 (L3) edge network function implementations (subsection 3.1.3). The reference scenario and both case studies are first described in subsections 3.1.1 and 3.1.2. The section concludes with a proof-of-concept implementation with the Mininet emulation platform (which was used to provide a practical example of the feasibility and degree of complexity of such approaches, subsection 3.1.4).

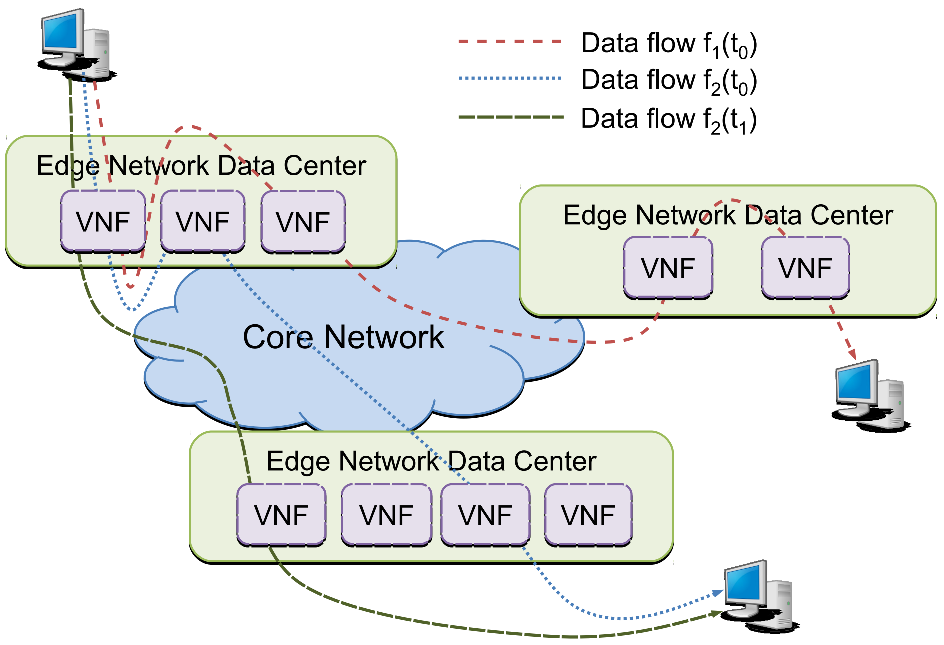


Figure 3.1: Reference network scenario with NFV chaining at the edge and dynamic traffic steering.

3.1.1 Reference network scenario

The reference network scenario assumed here is illustrated in Fig. 3.1. Borrowing the main idea from [12], a future Telco infrastructure is envisioned such that the complexity is mostly shifted towards the edge. A high-performance, flat, optical core network provides stateless broadband connectivity to a number of edge networks, where most of the functions required to compose the offered service are provisioned in form of software appliances capable of multi-tenancy isolation (e.g., virtual machines, lightweight containers, etc.) executed on top of standard, general purpose computing hardware. Therefore, future edge networks are assumed to take the shape of data centers, where VNFs can be easily deployed, cloned, migrated, destroyed as a sort of on-demand utility service, i.e. according to the general cloud computing paradigm.

However, differently from the typical cloud computing or storage services, these cloud-based edge networks have the peculiar aspect that many VNFs must be applied to crossing traffic originating and terminating outside of the data center. Furthermore, in order to adapt to emerging network conditions and owing to the flexibility of software appliances, type, number, and location of VNFs traversed by a given user data flow may change in time.

In principle, the choice of the most appropriate VNFs could be the result of the execution of other VNFs. For instance, a traffic classification function could be used to determine what kind of traffic conditioning function (e.g., shaping or priority scheduling) should be applied to a given user data flow to improve the QoS. Although this example recalls traditional solutions for QoS enforcing, the added value of implementing it through NFV is given by the much higher flexibility of managing software-based virtual appliances rather than deploying hardware-based middle-boxes.

Therefore, the edge network data center infrastructure must be flexible enough to allow dynamic and conditional VNF chaining. This also means that there are two aspects that matter: space chaining and time chaining diversity. Considering the example in Fig. 3.1, two different user data flows f_1 and f_2 at time t_0, in this case coming from the same user end-point, must first cross the same VNF, then must be forwarded to two different local functions, and finally sent to some remote edge networks where, after crossing additional VNFs, they eventually reach their respective destinations. In this case, since f_1 and f_2 are simultaneously active, the programmable network must provide *space chaining diversity.* Similarly, assuming that flow f_2 is subject to different service chains at two different times t_0 and t_1, the programmable network must also implement *time chaining diversity.*

A viable approach to enable the required flexibility in both space and time chaining diversity is to use OpenFlow to properly steer traffic flows. A logically centralized SDN controller is in charge of programming and coordinating the edge networks and make sure that each user data flow crosses the required VNFs in the required order. Furthermore, any VNF chain provisioning and change should be compliant with standard user access protocols. The complexity of achieving this goal with an OpenFlow implementation is discussed in the following sections, with reference to two practical case studies of VNF chain deploying.

3.1.2 Case studies: Layer 2 and Layer 3 topologies

The example chosen is that of an operator providing connectivity services to two users exchanging traffic with a remote host via an edge router. The user end-points are assumed to be implemented as virtual machines (VMU1 and VMU2) running applications inside a cloud computing platform, which is directly attached to, or even deployed within, the edge network data center. This approach follows the idea of a shared data center infrastructure integrating both cloud and NFV services for multi-tenant customers, as envisioned in the OpenStack implementation described in [24]. However, the study does not lose generality if other scenarios with physical connectivity to user premises are considered.

The two end users have two different types of Service Level Agreement (SLA):

1. VMU1 is a priority user and requires a WAN acceleration service when needed, but in case of adequate bandwidth available, the traffic of this user follows traditional routing rules;

2. VMU2 is provided with a best effort service, so it follows traditional routing, unless higher priority users enter (or are already in) the network; in this case, traffic belonging to VMU2 goes through a shaping function, thus limiting its bandwidth usage.

According to the NFV paradigm, the edge node forwarding packets to the remote host (H1) is implemented as a Virtual Router (VR). Other VNFs provisioned as dedicated virtual machines include: a traffic analyzer based on Deep Packet Inspection (DPI), a Wide Area Network Accelerator (WANA), and Linux kernel's Traffic Control (TC) tools to implement traffic shaping. Given that the various VNFs are available in the edge network data center, the issue now is to properly steer the traffic flows to make them traverse the chain of VNFs that implement the correct SLAs for each user. Basically, traffic flows originated at VMU2 should be forwarded according to normal IP routing when there is no bandwidth contention, and redirected to the TC when traffic flows from VMU1 are contending for the output bandwidth. Similarly, traffic flows from VMU1 should be forwarded according to normal IP routing when there is no bandwidth contention, and redirected to the WANA when other best effort traffic is contending for the output bandwidth.

Therefore traffic steering policies must be implemented in a dynamic way, adjusted to the ongoing traffic flows. This is managed by means of the SDN-based control plane that will program the forwarding plane according to the required VNF chaining. Given the flexibility and cross-layer operations allowed by the SDN approach, a first question to answer is at which logical network layer the steering actions should be implemented. The following two alternatives are considered:

1. L2 topology, shown in Fig. 3.2, in which network functions are implemented as L2 virtual appliances connected to a single broadcast domain;

2. L3 topology, shown in Fig. 3.3, in which network functions are implemented as L3 virtual appliances connected to two different broadcast domains, which requires the Gateway (GW) as an additional function.

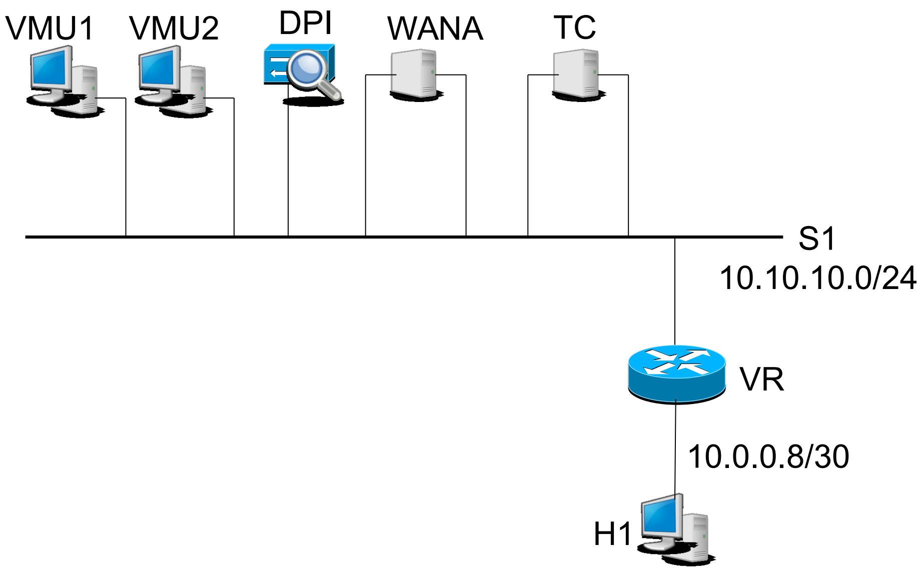


Figure 3.2: Layer 2 edge network topology.

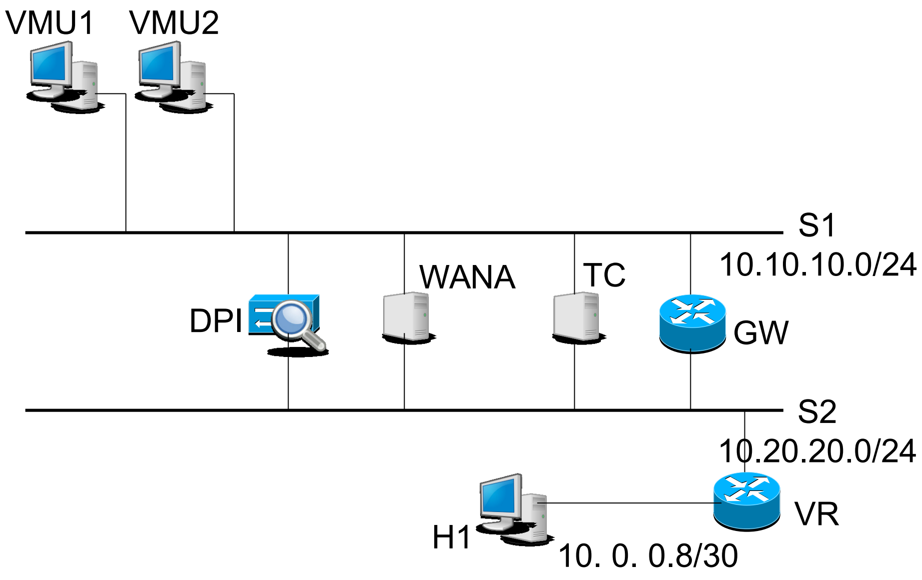


Figure 3.3: Layer 3 edge network topology.

Owing to the features of the SDN control plane, the two approaches provide identical results but have different pros and cons. Generally speaking, the L2 topology is completely transparent to the end user, who assumes to be directly connected to the edge router: in this case, the provider must deploy the required VNF chaining while keeping the same L2 connectivity on the user side. On the other hand, in the L3 topology the user always sees an intermediate gateway (GW): the provider must then maintain transparent L3 connectivity when deploying the required VNFs. Therefore, the dynamic traffic steering and the SDN controller must be designed taking into account these transparency requirements.

3.1.3 Design logic for Layer 2 & Layer 3 topology

In this subsection we discuss how and under which logic we designed the SDN controller for each of the two topologies previously introduced, which have been emulated using Mininet 2.1 [41], with POX 0.2.0 as an OpenFlow (OF) 1.0 controller. We adopt a hybrid approach by using switches capable of handling traffic as in legacy network devices, but also capable of taking advantage of OF features, as proposed in recent works [42]. Each scenario requires its own custom topology and custom controller, because they face different issues and different *actions* when dealing with traffic steering.

When planning the controller logic, we identified some rules which do not require flow by flow processing: in particular, whenever a packet must be forwarded from one of the VNFs to either VMUs or VR, it can follow legacy forwarding rules. For this reason, we decided to install these rules during the handshake phase between the switch and the controller. In particular, on switch S1 all IP packets destined to network 10.10.10.0/24 will be processed according the *NORMAL* action, which means the standard L2/L3 processing. Same thing on switch S2 for all IP packets coming from network 10.10.10.0/24.

So initially, both switches start with the above mentioned rules, while the others are added incrementally after *PacketIn* events, based on the type of flow generated by VMUs and according to the following logic:

- ARP and ICMP messages are processed according to the NORMAL action, in order to enforce and check basic connectivity;

- TCP and UDP messages are processed dynamically, i.e. they are treated according to specific OF rules that change based on network traffic conditions, as well as the SLA of each VMU.

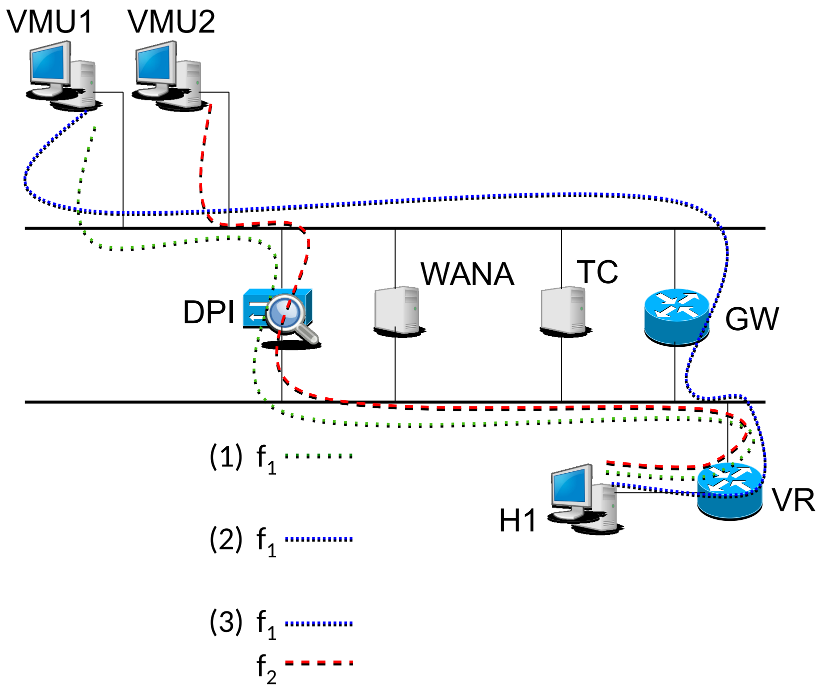
An example of L3 dynamic traffic steering and VNF chaining is shown in Fig. 3.4, where flows f_1 and f_2 are generated by VMU1 and VMU2 respectively. In order to apply traffic steering, we cannot simply play with the *output(port)* action. During phase (1), identified by the green line in Fig. 3.4a, as soon as the first packet of f_1 reaches S1, the packet is sent to the controller generating a PacketIn event. The controller then installs a rule specific for the two end-points (VMU1 and H1). Not only it tells the switch to forward packets out of the port to which the DPI is connected to, but it also tells the switch to change datalink layer destination address with the MAC address assigned to DPI's interface. At the same time, the controller installs the following rule on S2 switch: packets originated from H1 and destined to VMU1, coming from VR's port, not only will be forwarded out of the port to which DPI is connected to, but also their destination MAC address will be replaced with the one assigned to DPI's interface. In this way, we are able to steer bidirectional f_1 traffic towards the DPI, and we also reduce the amount of messages sent to the controller, thus decreasing the controller load.

After the DPI classification completes, confirming the compliance of f_1 to VMU1's SLA, according to our scenario VMU1 enters phase (2), identified by the blue line in Fig. 3.4a. By using flow_mod() messages with higher priority compared to rules installed during phase (1), the controller installs the following rules:

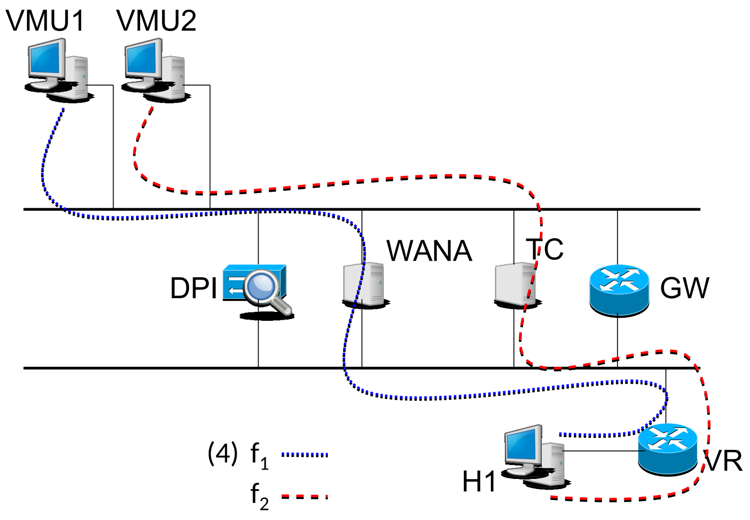
- on switch S1, traffic originated from VMU1 and destined to H1 is processed according to NORMAL action, so that it follows traditional routing;
- on switch S2, traffic going in the opposite direction coming from VR's port is also processed according to NORMAL action.

In this way, traffic stops going to DPI and goes directly to the gateway thanks to the higher rule priority. This configuration holds until best effort f_2 traffic, originated from VMU2, enters the network; as soon as VMU2's traffic reaches switch S1, the controller installs equivalent rules as those described for VMU1 during phase (1). Phase (3) starts and VMU2's traffic is forwarded to the DPI as depicted in Fig. 3.4a by the dashed red line (flow f_2), while the other flow (f_1) continues going through GW. After DPI classifies f_2 as best-effort traffic, phase (4) begins and new rules (with higher priority) are installed on both switches according to the following logic:

- on switch S1, traffic originated from VMU1 and destined to H1 will be forwarded to the port connected to WANA, changing the destination MAC address accordingly; a similar policy applies for traffic originated from VMU2 and destined to H1, that will be forwarded to TC instead.

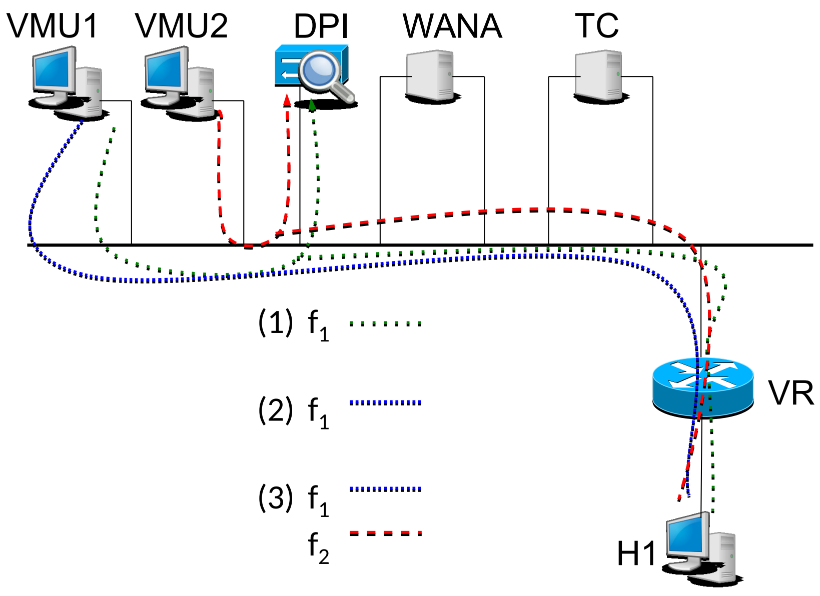


(a) Transitional phases.

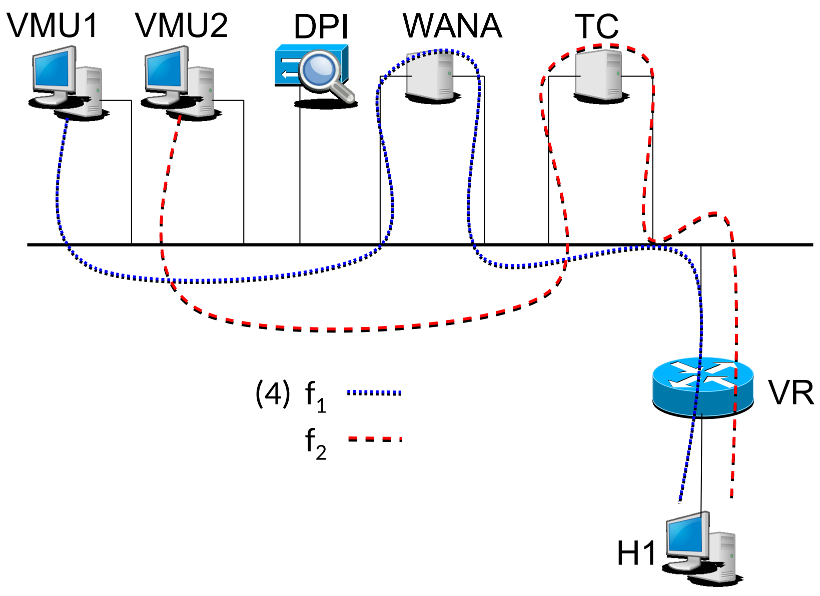


(b) Final phase.

Figure 3.4: Traffic steering in Layer 3 topology: (a) transitional VNF chaining in phases (1), (2) and (3); (b) final VNF chaining in phase (4).



(a) Transitional phases.



(b) Final phase.

Figure 3.5: Traffic steering in Layer 2 topology: (a) transitional VNF chaining in phases (1), (2) and (3); (b) final VNF chaining in phase (4).

- on switch S2, traffic going in the opposite direction coming from VR's port will be forwarded to WANA, changing the destination MAC address accordingly; a similar policy applies for traffic directed to VMU2, that will be forwarded to TC instead.

In this way, the final service chain is the one shown in Fig. 3.4b phase (4) with the blue dashed line and red dashed line, representing flows f_1 and f_2, respectively.

In the L2 topology case, some of the VNFs use an internal bridge to connect the input port to the output port (e.g., WANA and TC). This configuration creates topology loops and, if proper countermeasures are not taken, the network could end up flooded by broadcast storms. This issue cannot be addressed by means of the legacy Spanning Tree Protocol (STP), since it would disable one of the ports of the bridged VNFs preventing the packets to correctly flow through them. Therefore, this task is delegated to the controller by installing appropriate rules during the handshake phase. ARP, ICMP, TCP and UDP messages are treated using the same logic as for L3 topology, but this time the controller does not have to deal with MAC address change actions, given the single L2 broadcast domain. The four phases of the L2 dynamic service chaining are depicted in Fig. 3.5. It is worth to note that, whenever a new flow enters the network, the controller instructs the switch to forward its packets to the VR and to simultaneously mirror them to the DPI interface, as this is the typical L2 DPI configuration.

3.1.4 Proof of concept

As a proof of concept, we run several Mininet emulations for both topologies in order to prove the feasibility of the dynamic service chaining. The emulations have been implemented by measuring the throughput received at the destination H1, and on relevant nodes, such as DPI, TC and WANA (the throughput was measured also on GW for the L3 topology). On a time window of about 600 seconds, we generated two distinct TCP flows at different time instants, including a transitional period when the flows contend for the bandwidth, thus producing the series of traffic steering phases already described in the previous section.

Figure 3.6 shows the measured throughput at DPI, GW, TC and WANA nodes for the L3 scenario during the emulation, highlighting all the phases described above. In fact, at the beginning VMU1's traffic (f_1) is captured by the DPI entering phase (1); after correct classification, flow f_1 enters phase (2), and is captured by GW, taking advantage of all the bandwidth available (100 Mbps, as configured in the Mininet emulator). After about 150s from

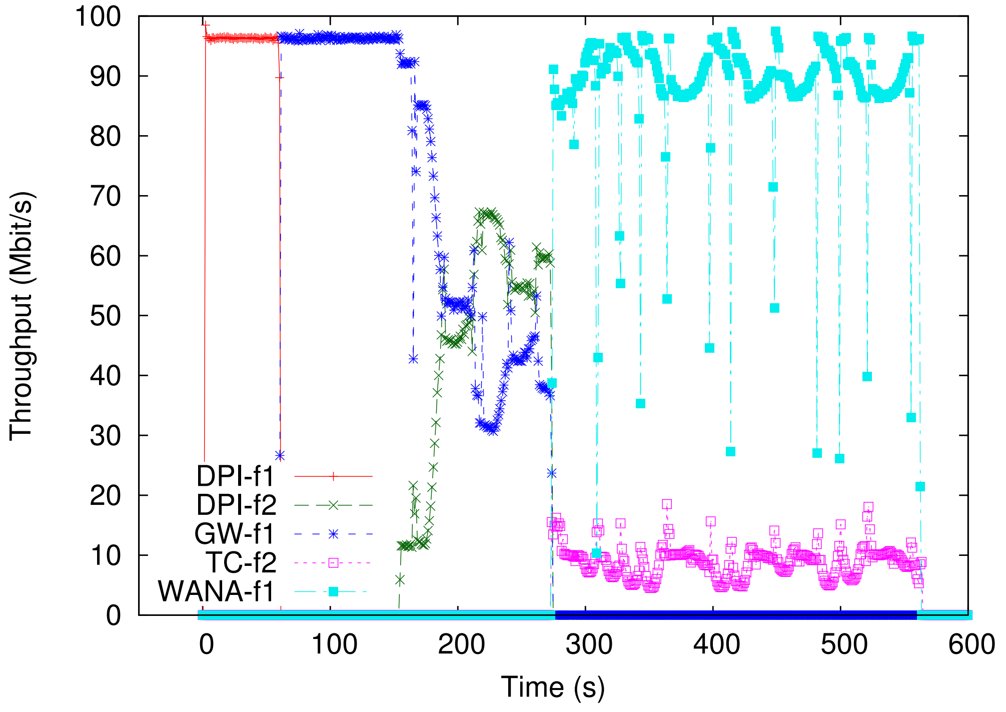


Figure 3.6: L3 topology: throughput measured at DPI, GW, WANA and TC.

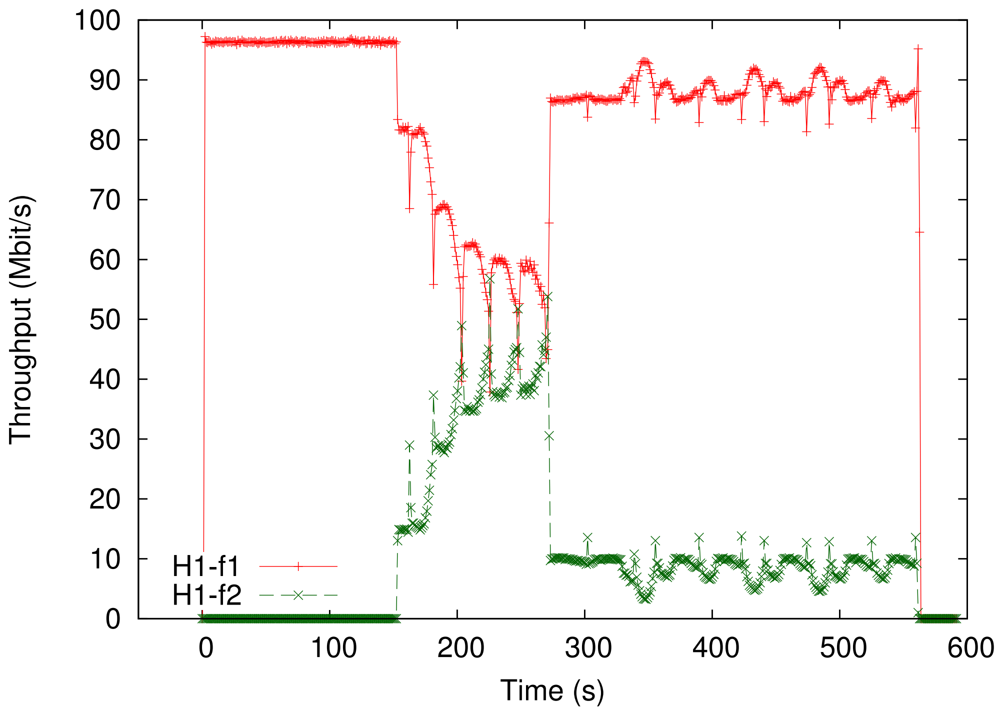


Figure 3.7: L2 topology: throughput measured at H1.

the beginning, VMU2's traffic (f_2) enters the network, generating bandwidth contention, phase (3), until the second classification occurs; this determines a new change in the service chain: from now on, f_1 is sent to WANA, while f_2 is sent to TC.

As far as L2 scenario is concerned, we decided to show the throughput measured at the destination H1, which is equivalent (in terms of highlighted phases) to the one shown for L3 scenario. As reported in Fig. 3.7, at the beginning f_1 experiences all the bandwidth available during phases (1) and (2); contention occurs during phase (3) when f_2 enters the network, ending in phase (4) when both flows follow different chains according to their SLAs.

The main outcome of this empirical study is that, with a careful analysis of the practical issues posed by a given NFV deployment and a smart definition of the set of actions to be programmed in the network, considering both legacy forwarding and flexible traffic steering, it is possible to simplify the SDN controller design and successfully achieve dynamic service chaining.

3.2 Coordinating dynamic SDN control

As we said, virtualization brings the immediate advantage of making deployment independent of the underlying hardware, simplifying service planning and implementation, given that to date the most typical approach to service chaining is based on static forwarding rules unaware of user needs. The service chain dedicated to a given customer (or set of customers) may be dynamic, changing over a rather small time scale, driven by external commands and/or by the changing network conditions.

Our goal is to provide a general conceptual framework to implement such concept. The VNFs are deployed in a cloud environment and the service chain is created, modified, and maintained by properly adapting the underlining virtual network by means of a Software Defined Networking (SDN) approach. Even before the advent of NFV, SDN has been considered as a viable option to flexibly steer traffic across physical middleboxes, posing significant challenges for deployment and integration with existing infrastructures [37, 43]. However, the proposed solutions did not take into account the additional requirements of virtualized middleboxes in terms of mobility and flexibility. Different attempts have been made to formalize SDN control plane design. Some authors proposed an SDN programming model aimed at defining network-wide forwarding behaviors by applying high-level algorithms targeted to performance challenges [44]. Another solution consisted in taking advantage of operating system mechanisms and principles, showing how they could be applied in the SDN context [45]. Further works were more focused on modeling the SDN data plane, e.g., by means of multiple level pipeline and finite state machines [46], or through a stateful flow processing inside the forwarding device itself [47]. However, none of the aforementioned works directly tackles the issue of an SDN control plane design for dynamic VNF service chaining in a cloud network platform.

In the following subsections, the controller design methodology will be discussed (subsection3.2.1) together with a case study example; then a former implementation of such example is provided on the OpenStack cloud platform (subsection 3.2.2). Some preliminary results about methodology adoption is given in subsection 3.2.3.

3.2.1 The Finite State Machine approach

To devise a general approach to service chain reconfiguration, the behavior of the SDN control plane is modeled here by means of a Mealy Machine [48] abstraction, in particular a *Finite State Machine* (FSM). The controller sets up and manages the forwarding rules for data flows. Flow f represents a traffic stream at some logical level. Therefore service chains may be applied to application-related flows, user-related flows, and/or aggregated flows. The SDN controller runs a thread for each f, which installs in the network a set of forwarding rules that will lead to a specific behavior to implement a specific service chain.

An example of FSM state diagram is shown in Fig. 3.8. State transitions depend on current state and input received, whereas each transition generates an action on f. We apply the following formal definitions:

- state s is from a finite set (in the example, $s \in \{Init, C, E, N, D\}$, where $s_0 = Init$ is the initial state);
- input i is from a finite set (e.g., $i \in \{$PKT_IN, SLA_C, SLA_NC, CONG, NO_CONG$\}$);
- actions $A(f, s, i)$ are defined according to the SDN technology used for traffic steering;
- state transition T is a function that maps a pair (s, i) to a pair $(s', A(f, s, i))$.

The set of input i can be composed of events (e.g., PKT_IN represents a packet arrived at the controller) and output coming from other entities, such as DPI or traffic monitor (e.g., SLA_C and SLA_NC represent output coming from DPI, meaning that a traffic is compliant or not to the SLA, respectively; while CONG and NO_CONG represent output coming from a traffic monitor, meaning that the network is congested or not, respectively).

To make the remainder of the discussion easier to follow and more concrete, a specific example will be considered, i.e. the situation of a network operator willing to apply the NFV principles in a multi-tenant scenario to dynamically enforce QoS and satisfy users' Service Level Agreement (SLA) in case of potential network congestion. However, the modeling and operational approach is general and can be applied to other situations.

In this specific scenario the set of states is defined as in Fig. 3.8:

1. *Init*, *Initial*, in which the controller can install general, flow-independent forwarding rules in the network nodes; this is the state from which a new thread departs whenever a new flow f arrives;
2. *C*, *Classification*, in which the new flow f is analyzed and classified to determine its SLA;
3. *E*, *Enforcement*, in which QoS is strictly enforced on flow f, according to its SLA;

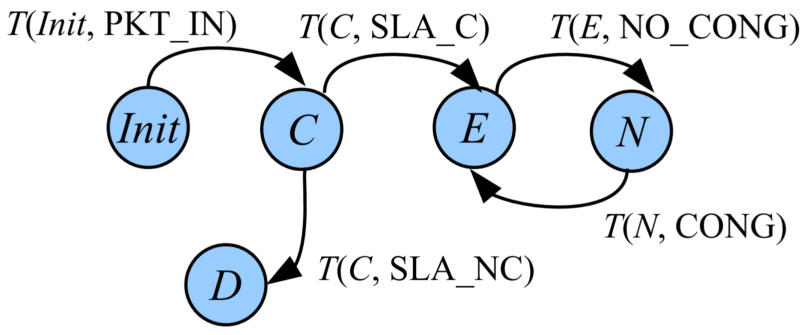


Figure 3.8: Example of a state diagram of the FSM representing the controller thread processing a flow.

4. N, *Non-enforcement*, in which QoS is not strictly enforced on f, because the network is not in a potential congested state and more resources can be given to f;

5. D, *Drop*, in which flow f is subject to policing actions (e.g., packets are dropped) because it is not compliant with the SLA.

When a new flow f arrives, i.e., a PKT_IN input is received, the controller spawns a new thread that enters state C, while action $A(f, \mathit{Init}, \text{PKT\_IN})$ enables the steering of f to a VNF chain that implements flow classification. Depending on the result (an input to the FSM), f can be either policed, if not compliant (SLA_NC), or treated according to QoS requirements (SLA_C). The idea in the latter case is to follow a conservative approach at first instance, which means enforcing the SLA with a proper steering action and moving to state E. However, if the network is not congested and more than enough resources are available, flow f can be processed by a simpler service chain that does not strictly enforce QoS, and the thread can move to state N. We assume that a separate entity, in charge of monitoring the state of the network, interacts with the controller through a northbound interface to notify whether (CONG) or not (NO_CONG) a potential congestion arises, allowing the thread to adaptively switch between states N and E.

As already said, any state transition implies a steering action $A(f, s, i)$ to be performed on flow f. Although the specific operations depend on the particular SDN technology adopted, we can define an additional set of general parameters as follows:

- $NT = \{SW_1, SW_2, \ldots, SW_{N_{\text{SW}}}\}$, the set of switches in the network topology;

- $SW_j = \{p_1, p_2, \ldots, p_{N_{\mathrm{p},j}}\}$, the set of ports on the j-th switch;
- $U = \{u_1, u_2, \ldots, u_{N_\mathrm{u}}\}$, the set of users;
- $NF = \{F_1, F_2, \ldots, F_{N_\mathrm{F}}\}$, the set of VNFs;
- $Ch(f, s) = \{F_{l_1}, F_{l_2}, \ldots, F_{l_{n(f,s)}}\}$, the service chain of $n(f, s)$ VNFs to be applied to flow f in state s.

Then each action can be expressed as a composition of atomic tasks. We can define a few general ones:

$$\begin{aligned}(SW_j, p_m) &= \mathit{get\_port}(u_k)\\ (SW_j, p_m) &= \mathit{get\_in\_port}(F_l, d)\\ (SW_j, p_m) &= \mathit{get\_out\_port}(F_l, d)\end{aligned} \tag{3.1}$$

where $SW_j \in NT$, $p_m \in SW_j$, $u_k \in U$, $F_l \in NF$, and $d \in \{\mathrm{inbound}, \mathrm{outbound}\}$.

These tasks provide an abstraction to obtain topology information, since they are used to discover the switch SW_j and port p_m a given user u_k or VNF F_l is connected to. For VNFs, it is assumed in general that traffic flows can enter and exit through different ports, and that in many cases (e.g., NAT) this depends on the direction of the packet, either inbound or outbound.

The result of tasks (3.1) is essential to perform another atomic task:

$$\mathit{flow\_mod}(SW_j, \mathit{cmd}, \mathit{opts}, \mathit{match}, \mathit{fwdlist}) \tag{3.2}$$

which represents the act of executing a command *cmd* (e.g., add or delete a forwarding rule) on switch SW_j, possibly specifying optional parameters *opts* (e.g., priority or timers), providing the flow matching rule *match* and the specific forwarding operation list *fwdlist* to be applied to the matching flow.

With the help of the aforementioned abstractions, the proposed FSM is able to capture the sequence of operations that each controller thread must execute on the respective flow, regardless of the underlying network infrastructure and chosen SDN technology. Therefore, the SDN controller can be designed according to the proposed general FSM model, whereas the specific actions to be applied to the forwarding nodes and the specific southbound interface protocol to be used can be programmed as separate, technology-dependent "drivers". This includes, for instance, also the possibility to tag packets forwarded across multiple switches in order to keep the network aware of the current VNF context and chain segment [37, 43].

3.2.2 Applying stateful SDN control to a Layer-2 NFV Topology

In this subsection we illustrate how we applied the previously described design methodology to a practical case of NFV deployment on OpenStack[1] platform. To keep VNFs transparent to the users, we configured the topology depicted in Fig. 3.9, consisting of two layer-2 SDN edge networks interconnected by a legacy core network. Following the proposed approach, we define:

$$\begin{aligned} NT &= \{SW_1, SW_2\} \\ U &= \{\mathrm{BU}, \mathrm{RU}, \mathrm{DEST}\} \\ NF &= \{\mathrm{DPI}, \mathrm{TC}, \mathrm{WANA}_1, \mathrm{WANA}_2, \mathrm{VR}_1, \mathrm{VR}_2\} \end{aligned}$$

The two switches are compliant with OpenFlow and programmed by a suitable SDN controller. The two users generate traffic flows f_{BU} and f_{RU} towards the same destination server with different SLAs: when QoS is enforced, BU and RU are subject to WAN acceleration and traffic shaping, respectively. We describe in detail which steering actions and service chaining are applied to f_{BU} in SW_1, proving that our approach is general enough to easily derive the corresponding actions and chaining to be applied to f_{RU} and in SW_2.

We first define to which service chain the flow is subject given the current state. $Ch(f_{\mathrm{BU}}, Init) = Ch(f_{\mathrm{BU}}, D) = \mathrm{nil}$: no chain is applied; $Ch(f_{\mathrm{BU}}, C) = \{\mathrm{DPI}\ \&\ \mathrm{VR}_1\}$: traffic is sent to VR_1 and mirrored to DPI for classification (the chaining order does not matter); $Ch(f_{\mathrm{BU}}, E) = \{\mathrm{WANA}_1, \mathrm{VR}_1\}$: traffic is sent first to WANA_1 and then to VR_1 (the order matters); $Ch(f_{\mathrm{BU}}, N) = \{\mathrm{VR}_1\}$: traffic is sent directly to VR_1. A pseudo-code description (using POX-like[2] notation) of the steps that need to be taken, during state transitions or in a given state, is provided in the following.

Figure 3.10 shows an example of flow-independent rules to be installed during the startup phase (*Init* state). The controller sends *flow_mod* messages to each switch attached to VNFs with multiple ports to install ARP broadcast dropping rules: this is needed to avoid that VNFs working in layer-2 mode generate broadcast storms by replicating ARP requests.

Figure 3.11 describes the actions taken on f_{BU} at the source edge network during the transition from *Init* to *C*. With *get_match*(f) (line 8) we refer to a function that returns the exact matching rule for outbound packets of flow f, while *get_match*(f') (line 16) returns the dual matching rule for inbound packets. After getting port information (lines 1-3), the OpenFlow actions to

[1]`https://www.openstack.org/`

[2]`http://www.noxrepo.org/pox/about-pox/`

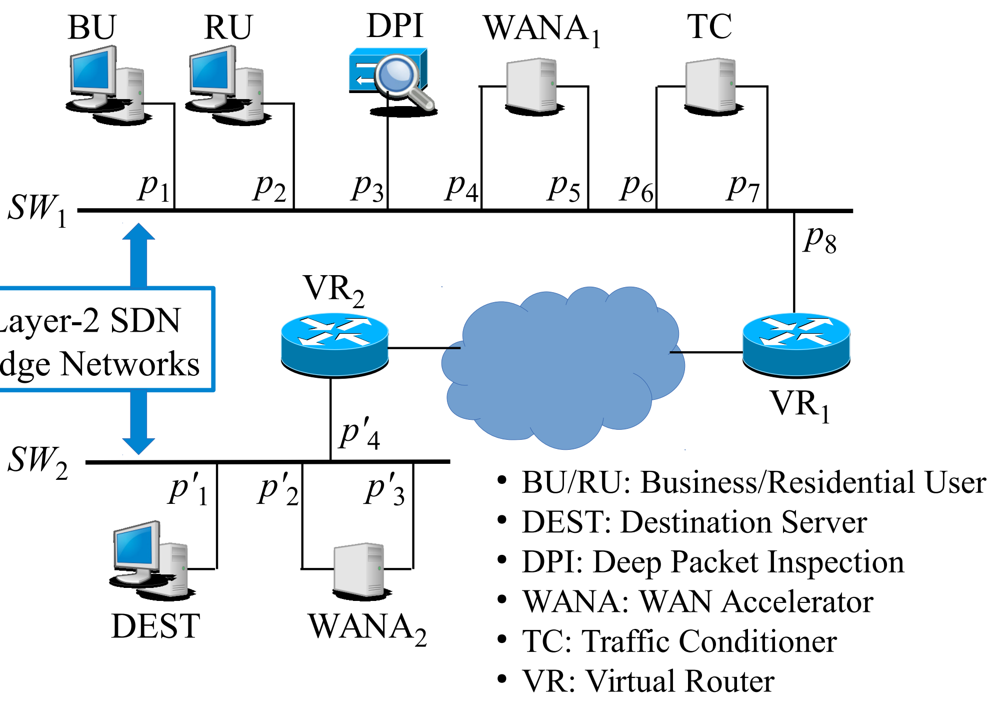


Figure 3.9: Case study NFV topology. Each layer-2 SDN edge network is implemented by an OpenFlow switch, whose numbered ports are connected to users and VNFs as displayed.

be performed on outbound packets are specified, i.e., sending a copy of each packet to the DPI (line 4), inserting the VLAN tag used by OpenStack on the physical network for tenant isolation (line 5), and sending the packet to VR$_1$ (line 6). The matching rule and its expiration timer t_{out} and priority h are specified (lines 7-8). Then the *flow_mod* message for outbound packets is sent to the switch (line 9). Finally, a dual flow matching rule for inbound packets is computed and installed (lines 10-17). Note that the procedure has been simplified considering the setup in Fig. 3.9, where the source edge network consists of a single switch (SW_1), but it can be extended to a general multi-switch case. It can be also easily extended to express the actions to perform at the destination edge network switch (SW_2).

When the controller thread is in state C, the flow is being analyzed by the DPI to verify SLA compliance. If the classification reveals that f_{BU} is not compliant with its SLA, the controller is notified through the northbound interface and the relevant thread moves to state D, forcing the switch to drop BU's packets, as specified in Fig. 3.12. In this case, the rule priority is higher

Initialization

```
1: for all $F_l$ in $\{\text{TC}, \text{WANA}_1, \text{WANA}_2\}$ do
2:     $(sw_{\text{in}}, p_{\text{in}}) = get\_in\_port(F_l, \text{outbound})$
3:     $(sw_{\text{out}}, p_{\text{out}}) = get\_out\_port(F_l, \text{outbound})$
4:     $fwdlist$ = append(“drop”)
5:     for all $(sw, p)$ in $\{(sw_{\text{in}}, p_{\text{in}}), (sw_{\text{out}}, p_{\text{out}})\}$ do
6:         $match$ = “ofp_match(in_port = $p$, dl_type = ARP_TYPE, dl_dst =
           ETHER_BCAST)”
7:         $flow\_mod(sw, \text{ADD}, \text{nil}, match, fwdlist)$
8:     end for
9: end for
```

Figure 3.10: Initialization of flow-independent rules.

(line 3) than that of the rule inserted by action $A(f_{\text{BU}}, Init, \text{PKT\_IN})$: as a consequence, the packet dropping action is immediately applied, while the previous temporary rule expires after t_{out} without any further intervention from the controller.

Otherwise, if f_{BU} is found compliant by the DPI, the controller thread moves to state E. Figure 3.13 describes the steps that need to be taken to perform the required traffic steering: outbound packets must be first sent to WANA_1 (lines 1-6) and then to VR_1, after inserting the required VLAN tag (lines 7-12); inbound packets must be processed in the reverse order (lines 13-23).

In case the network is not congested, it is possible to avoid the strict enforcement of QoS functions and send packets directly to the edge router VR_1, thus reducing the processing burden on VNFs and potential latency on user data traffic due to service chaining. Transition $T(E, \text{NO\_CONG})$ brings the controller thread to state N, and the related steering action $A(f_{\text{BU}}, E, \text{NO\_CONG})$ can be expressed similarly to Fig. 3.11. Indeed, after removing lines 2, 4, 12, and 15 (not needed because packets must not be sent to the DPI), only line 7 must be changed, setting the rule options to $opts$ = “priority=$h + 2$” for immediate deployment of the new steering action. Not specifying an expiration time makes the new rules permanent. However, as soon as a potential network congestion is reported, the thread moves back to state E with transition $T(N, \text{CONG})$. The corresponding steering action $A(f_{\text{BU}}, N, \text{CONG})$ is similar to $A(f_{\text{BU}}, E, \text{NO\_CONG})$, except for the fact that the *flow_mod* command that must be specified is DELETE instead of ADD. This way, the QoS-enforcing service chaining is immediately restored. According to the notifications received from a network resource monitoring system, the con-

$T(\mathit{Init}, \mathrm{PKT\_IN}) = (C, A(f_{\mathrm{BU}}, \mathit{Init}, \mathrm{PKT\_IN}))$

```
1: $(sw, p_{\mathrm{in}}) = \mathit{get\_port}(\mathrm{BU})$
2: $(sw, p_{\mathrm{out},0}) = \mathit{get\_in\_port}(\mathrm{DPI}, \mathrm{outbound})$
3: $(sw, p_{\mathrm{out},1}) = \mathit{get\_in\_port}(\mathrm{VR}_1, \mathrm{outbound})$
4: $\mathit{fwdlist}$ = append("ofp_action_output(port=$p_{\mathrm{out},0}$)")
5: $\mathit{fwdlist}$ = append("ofp_action_vlan_vid(vlan_id = internal_vid)")
6: $\mathit{fwdlist}$ = append("ofp_action_output(port=$p_{\mathrm{out},1}$)")
7: $\mathit{opts}$ = "hto=$t_{\mathrm{out}}$, priority=$h$"
8: $\mathit{match}$ = "ofp_match(in_port=$p_{\mathrm{in}}$,$\mathit{get\_match}(f_{\mathrm{BU}})$)"
9: $\mathit{flow\_mod}(sw, \mathrm{ADD}, \mathit{opts}, \mathit{match}, \mathit{fwdlist})$
10: $(sw, p_{\mathrm{in}}) = \mathit{get\_out\_port}(\mathrm{VR}_1, \mathrm{inbound})$
11: $(sw, p_{\mathrm{out},0}) = \mathit{get\_port}(\mathrm{BU})$
12: $(sw, p_{\mathrm{out},1}) = \mathit{get\_in\_port}(\mathrm{DPI}, \mathrm{inbound})$
13: $\mathit{fwdlist}$ = append("ofp_action_strip_vlan_vid()")
14: $\mathit{fwdlist}$ = append("ofp_action_output(port=$p_{\mathrm{out},0}$)")
15: $\mathit{fwdlist}$ = append("ofp_action_output(port=$p_{\mathrm{out},1}$)")
16: $\mathit{match}$ = "ofp_match(in_port=$p_{\mathrm{in}}$,$\mathit{get\_match}(f'_{\mathrm{BU}})$)"
17: $\mathit{flow\_mod}(sw, \mathrm{ADD}, \mathit{opts}, \mathit{match}, \mathit{fwdlist})$
```

Figure 3.11: Actions taken on f_{BU} during state transition from *Init* to C.

troller thread can switch back and forth between states E and N adapting the VNF chaining to the current network conditions.

The steering actions presented above for BU traffic can be replicated also for RU flows. In this case, the service chain in state E is different, i.e., $Ch(f_{\mathrm{RU}}, E) = \{\mathrm{TC}, \mathrm{VR}_1\}$, and this is reflected in $A(f_{\mathrm{RU}}, C, \mathrm{SLA\_C})$, which is exactly the same as Fig. 3.13 after replacing BU with RU and WANA_1 with TC.

3.2.3 Preliminary experimental results

As a proof of concept of our controller design approach, we deployed the topology shown in Fig. 3.9 on a real cloud computing platform based on OpenStack. BU, RU, DPI, WANA1 and TC are all implemented as virtual machines (VMs) running inside a Compute Node and connected to an internal OpenFlow-compliant Open vSwitch bridge, whereas VR1 is a container-based virtual router running in the Network Node, as in a typical OpenStack configuration [18]. The destination edge network is outside the OpenStack cluster, although DEST, WANA_2 and VR_2 are also implemented as VMs con-

```
$T(C, \text{SLA\_NC}) = (D, A(f_{\text{BU}}, C, \text{SLA\_NC}))$

1: $(sw, p_{\text{in}}) = get\_port(\text{BU})$
2: $fwdlist$ = append("drop")
3: $opts$ = "priority=$h+1$"
4: $match$ = "ofp_match(in_port=$p_{\text{in}}$,$get\_match(f_{\text{BU}})$)"
5: $flow\_mod(sw, \text{ADD}, opts, match, fwdlist)$
```

Figure 3.12: Actions taken on f_{BU} during state transition from C to D.

Table 3.1: State transition times of the two flows measured in Fig. 3.14 .

Flow	**State transition**	**Time (s)**
f_{BU}	*Init* $\rightarrow C$	10.62
	$C \rightarrow E$	71.36
	$E \rightarrow N$	73.46
	$N \rightarrow E$	168.43
	flow terminated	404.11
f_{RU}	*Init* $\rightarrow C$	106.45
	$C \rightarrow E$	167.36
	flow terminated	416.74

nected to an Open vSwitch bridge. As a concrete implementation of VNFs, we use Traffic Squeezer[3] as a WAN accelerator, and the nDPI library[4] as DPI.

We designed a POX-based OpenFlow controller according to our methodology, and measured the throughput of two flows generated by BU and RU on relevant ports of SW_1, as shown in Fig. 3.14a and 3.14b. f_{BU} and f_{RU} were distinct UDP flows generated with iperf at 100 Mbit/s each. The figures highlight how the flow throughput is affected by the dynamic chaining, whereas the corresponding state transition times are reported in Table 3.1. At the beginning, f_{BU} is processed in state C (traffic sent to DPI and VR). After SLA compliance is determined, the state moves to E for a very short time (traffic peak sent to WANA$_1$ at around 70 s), and then quickly switches to N (traffic sent to VR$_1$ only) due to almost immediate NO_CONG notification. As soon as f_{RU} enters the network, it experiences state C chaining. Then the corresponding state moves to E, but network congestion is imme-

[3]`http://sourceforge.net/projects/trafficsqueezer/`

[4]`http://www.ntop.org/products/ndpi/`

$T(C, \text{SLA\_C}) = (E, A(f_{\text{BU}}, C, \text{SLA\_C}))$

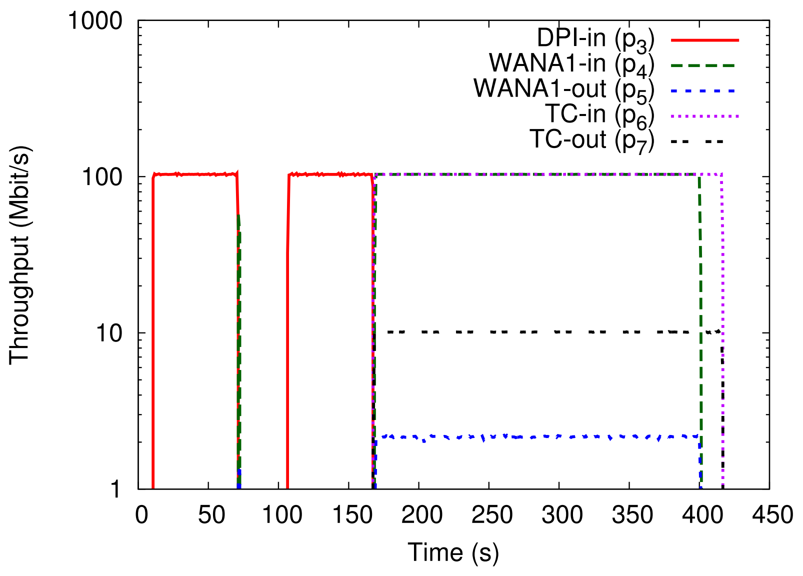
```
1: $(sw, p_{\text{in}}) = \textit{get\_port}(\text{BU})$
2: $(sw, p_{\text{out}}) = \textit{get\_in\_port}(\text{WANA}_1, \text{outbound})$
3: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=}p_{\text{out}}\text{)”})$
4: $\textit{opts} = \text{“priority=}h + 1\text{”}$
5: $\textit{match} = \text{“ofp\_match(in\_port=}p_{\text{in}}\text{,}\textit{get\_match}(f_{\text{BU}})\text{)”}$
6: $\textit{flow\_mod}(sw, \text{ADD}, \textit{opts}, \textit{match}, \textit{fwdlist})$
7: $(sw, p_{\text{in}}) = \textit{get\_out\_port}(\text{WANA}_1, \text{outbound})$
8: $(sw, p_{\text{out}}) = \textit{get\_in\_port}(\text{VR}_1, \text{outbound})$
9: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_vlan\_vid(vlan\_id = internal\_vid)”})$
10: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=}p_{\text{out}}\text{)”})$
11: $\textit{match} = \text{“ofp\_match(in\_port=}p_{\text{in}}\text{,}\textit{get\_match}(f_{\text{BU}})\text{)”}$
12: $\textit{flow\_mod}(sw, \text{ADD}, \textit{opts}, \textit{match}, \textit{fwdlist})$
13: $(sw, p_{\text{in}}) = \textit{get\_out\_port}(\text{VR}_1, \text{inbound})$
14: $(sw, p_{\text{out}}) = \textit{get\_in\_port}(\text{WANA}_1, \text{inbound})$
15: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_strip\_vlan\_vid()”})$
16: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=}p_{\text{out}}\text{)”})$
17: $\textit{match} = \text{“ofp\_match(in\_port=}p_{\text{in}}\text{,}\textit{get\_match}(f'_{\text{BU}})\text{)”}$
18: $\textit{flow\_mod}(sw, \text{ADD}, \textit{opts}, \textit{match}, \textit{fwdlist})$
19: $(sw, p_{\text{in}}) = \textit{get\_out\_port}(\text{WANA}_1, \text{inbound})$
20: $(sw, p_{\text{out}}) = \textit{get\_port}(\text{BU})$
21: $\textit{fwdlist} = \text{append}(\text{“ofp\_action\_output(port=}p_{\text{out}}\text{)”})$
22: $\textit{match} = \text{“ofp\_match(in\_port=}p_{\text{in}}\text{,}\textit{get\_match}(f'_{\text{BU}})\text{)”}$
23: $\textit{flow\_mod}(sw, \text{ADD}, \textit{opts}, \textit{match}, \textit{fwdlist})$
```

Figure 3.13: Actions taken on f_{BU} during state transition from C to E.

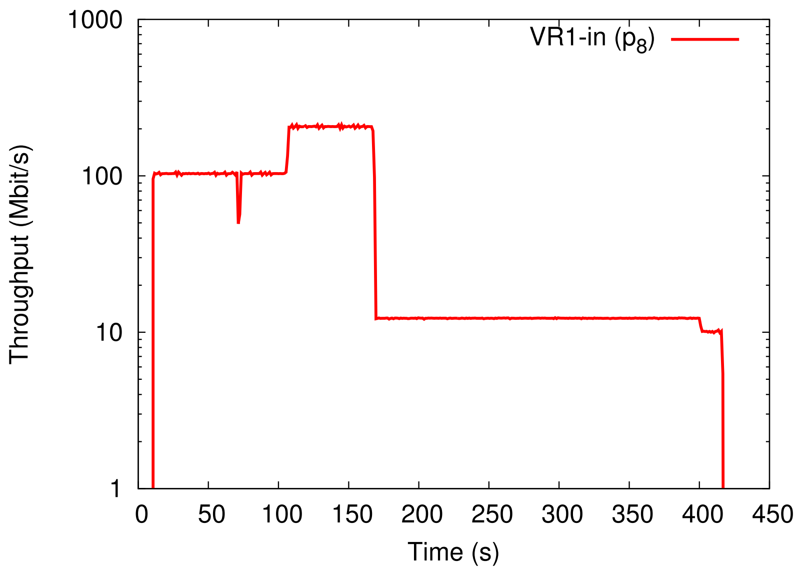
diately detected, causing also f_{BU} to move back to E. At this time, QoS is enforced on both flows: f_{BU} traffic is compressed by WANA$_1$, while f_{RU} traffic is shaped by TC, with throughput measured at VR$_1$ of around 2.2 Mbit/s and 10 Mbit/s, respectively.

We also led additional tests in order to evaluate the latency introduced by VNFs. We measured maximum, minimum, and average values of round trip time (RTT) and jitter experienced by UDP flows generated by RU. Mean values obtained from 20 experiments in states C and E are reported in Table 3.2. We did not find significant differences, mainly because in our setup VNFs are placed on the same server. We expect different results in case of more distributed VNF environments.

This work not only proves the feasibility of the proposed design methodol-



(a) Throughput measured at each VNF port.



(b) Throughput measured at VR_1 input port.

Figure 3.14: Throughput measured during the proof-of-concept experiment, showing correct traffic steering at controller state changes and final flow composition towards the destination.

Table 3.2: Latency and jitter under different chaining states.

	State	Max	Min	Average
RTT (ms)	*C*	14,840	1,525	3,972
	E	10,382	2,006	3,313
Jitter (ms)	*C*	0,322	0,080	0,125
	E	0,491	0,067	0,152

ogy, but also shows a possible approach to orchestrate the mutual dependence of different flows competing for the same network resources. This benefit is further motivated and proved in the following subsections, where we highlight the added value that dynamic software-defined networking control, coordinated with a flexible cloud management approach, brings to telco operators and service providers. In particular, we report some of the experimental achievements obtained in the framework of Activity "SDN at the Edges", funded in 2015 by the EIT-Digital[5] initiative under the "Future Networking Solutions" Action Line.

3.2.4 How can Telco industry and Service Providers benefit from stateful approach?

Telco industry and service providers are experiencing a rediscovered interest in SDN and NFV paradigms: on the one hand, SDN foster a more flexible communication resource programmability; on the other hand, NFV fosters the adoption of virtualization and a "cloud-like" utility-based approach applied to networking, resulting in more elastic and efficient resource management. This interest is most probably motivated by the novelty of the overall context, specifically concerning techno-economic sustainability. Therefore, the main goal of the joint collaboration with *Telecom Italia - Strategy and Innovation* and *Ericsson Telecomunicazioni S.p.A.*, was to investigate how to create the technical conditions for accelerating the practical deployment of SDN and NFV solutions in order to produce concrete socio-economic impacts. Specifically, one of the main arguments has been that a federation of interconnected test-beds/field trials on SDN and NFV would allow the development of a distributed experimental environment where it is possible to test (and, possibly in the future, to certify) open source software solutions in specific networking application use cases, at the same time attracting service providers, small to medium-sized enterprises (SMEs), and early

[5]`http://www.eitdigital.eu/`

adopters/users of newly enabled ICT services.

The use case described in subsection 3.2.5 leverages on Ericsson's cloud management and orchestration platform [49] and our stateful SDN framework described in the previous subsections (3.2.1, 3.2.2, [50]). The innovative aspect of the use case considered in the PoC is the added value that a dynamic SDN control, coordinated with a flexible cloud management approach, brings to telco operators and service providers deploying NFV solutions. In particular, the PoC demonstrates how telco services based on NFV can be made self-adaptive to the network conditions and/or to users' changing requirements.

3.2.5 A SDN use case for Service Provider Networks

The general use case scenario refers to an NFV deployment in a cloud-based telco edge environment similar to the one discussed in [12]. The full life cycle of a telco service to be deployed in a virtualized environment is considered, from the setup of a service level agreement (SLA) with the operator, to the deployment of the virtual network functions (VNFs) required to implement the service, up to the operations performed to ensure SLA compliance. While for the scope of this work a generic SLA could be considered, in the PoC presented later an SLA related to bandwidth availability is assumed.

The goal and the original contribution of this work is to demonstrate full automation in both the cloud deployment of a given set of VNFs and the capability to react to changes in the overall network conditions, safeguarding the SLA.

Typically, a given VNF forwarding graph [51] is implemented by properly chaining a number of VNFs required to deploy a given service, and steering the relevant traffic flows accordingly. Following the cloud paradigm, the VNFs are hosted in virtual machines (VMs) and shared among a set of users for scalability and efficiency reasons. A VM may host from one to several VNFs, depending on the associated workload. The two extreme cases are:

- When a VM hosts all the VNFs required for the service implementation
- When each VNF is hosted by a separate VM

The former case is the least challenging from the connectivity point of view, as shown in Fig. 3.15a: since the VNF forwarding graph is mostly implemented inside a single VM, traffic steering is limited to sending flows to the VM and does not need complex forwarding rules in the cloud virtual network infrastructure. However, this choice provides limited elasticity in resource management. The latter case is more suitable for a telco cloud-based

NFV deployment, where the placement of VMs (and consequently of VNFs) is adaptively distributed across a number of computing nodes with the purpose of optimizing performance and scalability. However, even in an intermediate case, such as the one shown in Fig. 3.15b, the distributed approach requires steering traffic along a data path that traverses multiple VMs in the proper order and/or combination, according to the forwarding graph. Therefore, the implementation of the forwarding graph must rely on more complex forwarding rules to be applied in the cloud virtual network infrastructure by means of a proper reconfiguration mechanism, possibly automated. This is where a dynamic SDN approach plays a crucial role.

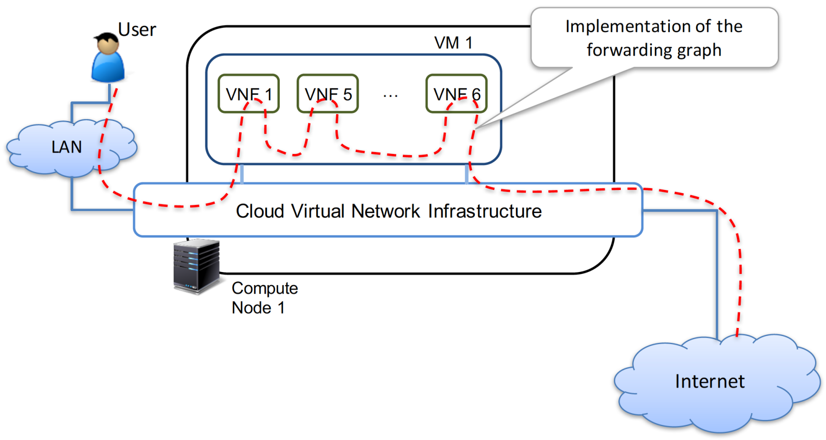
The use case presented here tackles the most challenging situation of distributed VNFs, with- out lack of generality. The VM images (VMIs) preconfigured with the installed VNFs are the basic bricks of the service implementation, while traffic steering is the glue to implement the forwarding graph. The most interesting feature is that the forwarding graph can be pruned, enhanced, and/or modified by starting/stopping VMs and/or modifying the traffic steering rules, with a degree of flexibility and within a time frame that are absolutely impossible to achieve using legacy deployments based on physical middle-boxes and vendor-locked equipment. The demonstration of these capabilities in a production-like environment is the core objective of this work, and this goal is achieved by implementing automated traffic steering and VNF forwarding graph management. The basic concepts behind the use case presented here are in line with some of the PoCs proposed within the European Telecommunications Standards Institute (ETSI) NFV framework.[6] In particular, PoC #28 and PoC #33 address similar issues. These PoCs were developed approximately at the same time as the activity reported. The findings of the PoCs are not yet publicly available. Therefore, this work can be considered complementary to the ETSI NFV use case analysis.

The nontrivial functional elements adopted in the considered use case include the following:

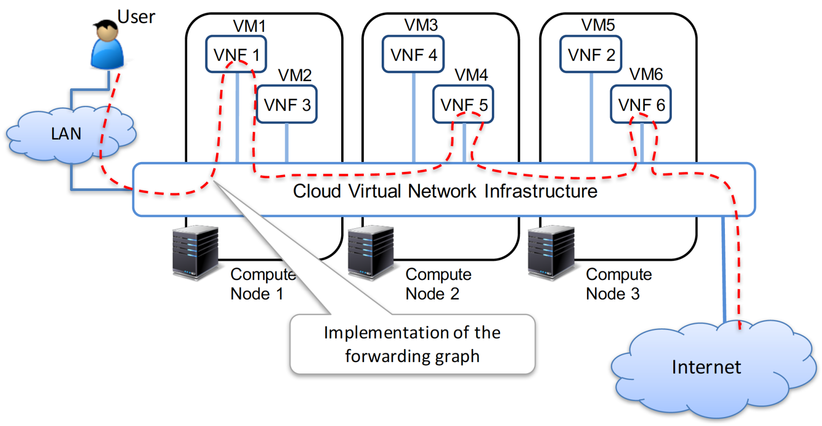
- A production-level cloud computing infrastructure (i.e., the Ericsson Cloud Lab described in the next subsection), equipped with a management and orchestration platform and an inventory of pre-defined VMIs to be picked up for deployment.

- An SDN controller, used to program the forwarding rules in the cloud network infrastructure in order to achieve proper traffic steering. The controller is designed according to our original stateful model that we

[6]`http://www.etsi.org/technolo-gies-clusters/technologies/nfv/nfv-poc/`


User
VM 1
Implementation of the forwarding graph
VNF 1
VNF 5
…
VNF 6
LAN
Cloud Virtual Network Infrastructure
Compute Node 1
Internet

(a) VNFs deployed as software entities running in a single VM hosted by a compute node in the cloud; the requested forwarding graph can mostly be implemented inside the VM.



(b) VNFs deployed as software entities running in different VMs hosted by different compute nodes. The requested forwarding graph must be implemented in the cloud virtual network infrastructure.

Figure 3.15: Schematic graph example of alternative implementations of the VNF forwarding graph.

developed at the University of Bologna, and that was previously described (subsection 3.2.1).

- A separate entity capable of monitoring the network conditions and interacting with the SDN controller through a northbound interface. This monitor notifies the controller with any change in network conditions that is meaningful for the traffic flow maintenance. We assume the presence of such monitoring tools since they are typical components of network management solutions.

3.2.6 Moving implementation to production-level environment: the Ericsson Cloud Lab

The Ericsson Cloud Lab [52] is an open and multi-vendor testing environment based on NFV and SDN technologies, where telco operators can join Ericsson to perform PoC experiments and field trials for new use cases and cloud services. The Ericsson Cloud Lab is a strategic infrastructure conceived and implemented with the scope to:

- Foster in-company competence build-up;
- Show specific and concrete “proof” points related to the cloud benefits;
- Implement customer demos for specific products;
- Demonstrate how issues and concerns can be managed to mitigate the risks.

To support the fulfillment of such a scope, the Cloud Lab implements activities ranging from:

- Validation and certification on customer-specific stack/solution;
- Fully customized PoC on customer premises;
- Deep dive on customer-specific requests;
- Standard customer demo.

Figure 3.16 shows the Ericsson Cloud Lab, which follows the ETSI NFV architecture. It exploits a heterogeneous environment mixing hardware from the major vendors of data center servers. The virtualization hypervisors are also mixed, with KVM running aside VMware, and different virtualized infrastructure managers (VIMs) are available. The whole infrastructure is managed by the proprietary Ericsson Cloud Manager (ECM) [49] on top of the VIM platforms. In the PoC presented here the VIM platform adopted is OpenStack [53].

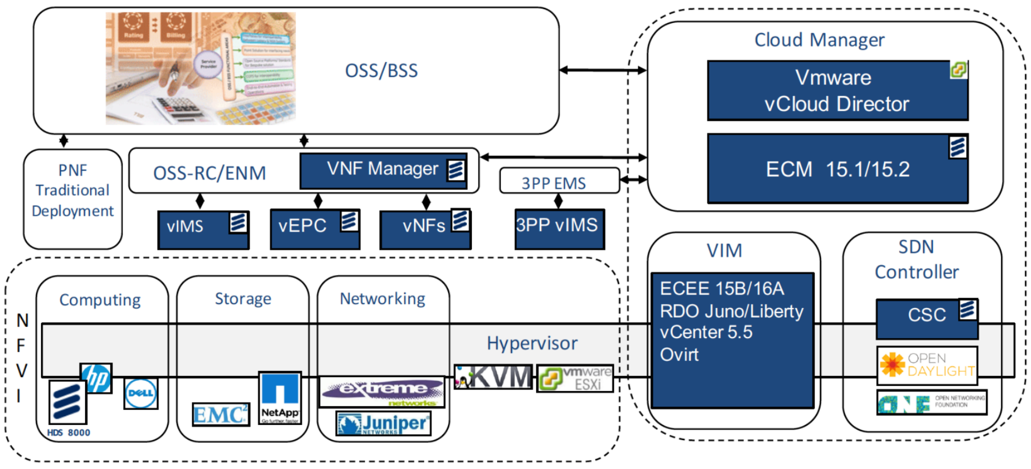


Figure 3.16: The Ericsson Cloud Lab NFV architecture.

The Cloud Lab allows the implementation of a workflow managing the full life cycle of a specific service and the set of VNFs used to implement it. In general terms, the operational workflow implemented in the use case here presented can be summarized as follows:

1. *Service creation*: Create a new VNF and make it available in the correct format for future utilization (pack a VMI with the NFV properly installed and configured).

2. *Site and VIM onboarding*: Let ECM be aware of the geographical site and VIMs to be considered as targets for the management activity.

3. *VNF/network service onboarding*: Deploy a new VNF (or a new release of a VNF) into ECM, including creation of an offer in the service catalog, import of VMIs, and so on.

4. *Instantiation and configuration*: Create or provision a new instance of the VNF including initial configuration (not customer-related).

5. *Runtime management*: Handle fault, configuration, accounting, performance, and security (FCAPS) operations and, in general, any change to the VNF.

6. *Decommissioning*: This is the disposal or retirement of a VNF instance, including all ancillary activities (e.g., secure destruction of VNF-related data).

The specific setup of the Ericsson Cloud Lab for the use case considered here is shown in Fig. 3.17. The virtualization infrastructure is running OpenStack Juno release and the KVM Hypervisor. The ECM hosts the inventory of VMIs with pre-installed VNFs and is responsible for the deployment of the logical building blocks necessary for the use case. The VNFs available in the inventory are:

- *Deep Packet Inspection* utility (based on the nDPI library[7]) that is used to classify incoming traffic flows
- *WAN Accelerator* (WANA), a compression utility (based on Traffic Squeezer[8]) that reduces the size of data packets to reduce transmission and waiting time along the WAN path, with an effect similar to an overall increase of bandwidth on the link
- *Traffic Conditioner* (TC), a traffic shaping function (implemented with tools available in the Linux kernel[9]) that limits the bit rate of a given set of flows by enforcing the incoming traffic SLA profile

An additional VMI (SDNc) is deployed to host the SDN controller (based on Ryu OpenFlow controller[10]) in charge of reconfiguring the underlying network infrastructure according to the different VNF forwarding graphs required by the users. This choice makes it possible to virtualize the network infrastructure and enables multiple potential service providers to coexist and separately control their slice of network resources. It is worth mentioning that each VNF listed above is not an Ericsson application, which emphasizes the multi-vendor capabilities of the ECM and the Ericsson Cloud Lab.

3.2.7 Proof of concept: VNF chaining for QoS enforcement & dynamic traffic steering

The PoC presented here focuses on VNF chaining for quality of service (QoS) enforcement, and assumes two classes of users sharing an access network segment with different SLAs concerning bandwidth availability:

[7]`http://www.ntop.org/products/deep-packet-inspection/ndpi/`
[8]`http://www.trafficsqueezer.org/`
[9]`http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html`
[10]`https://osrg.github.io/ryu/`

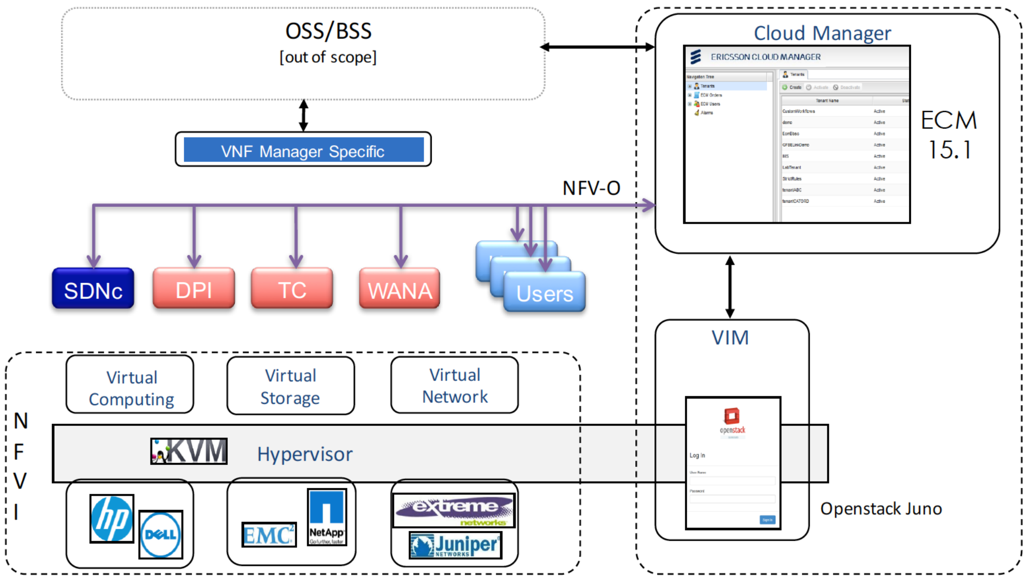


Figure 3.17: Use case Ericsson Cloud Lab setup.

- *BU* is a *business* (priority) *user*: In the case of adequate bandwidth availability, BU traffic follows traditional forwarding rules, but when the available bandwidth drops below a given threshold, the BU traffic is forwarded through a WAN acceleration service.
- *RU* is a *residential* (best effort) *user*: The RU traffic follows traditional forwarding rules until it starts competing with a BU for network resources; in this case, traffic from the RU is policed by a shaping function, thus limiting its bandwidth usage to a given value.

The access network is based on an SDN control plane designed to:

- Dynamically handle multiple simultaneous flows according to current network conditions
- Provide that some flows follow different forwarding graphs at different times
- Steer specific data flows toward the required VNF locations, thus achieving full dynamic service chaining

To this purpose, for each traffic flow, the SDN controller configures a set of forwarding rules that determine the specific behavior to be implemented in the forwarding graph.

As we said in section 3.2.1, as a general approach to the dynamic traffic steering configuration according to current network conditions, the behavior of the SDN control plane is modeled by means of a FSM where a pre-defined set of forwarding rules is associated with each state. The FSM state transitions may be triggered by either a change in network conditions, detected and notified by a network monitor, or an external command directly sent by the operator or user. This kind of communication reaches the controller through its northbound interface by means of an application programming interface (API) written for the purpose.

For the definition of the set of states, we refer to subsection 3.2.1, here we just remind that:

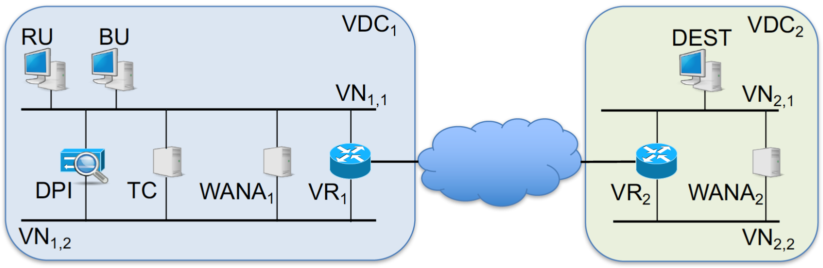
- *Init* is the *initialization* state.
- C is the *classification* state.
- E is the *enforcement* state.
- N is the *non-enforcement* state.
- D is the *drop* state.

When a new flow f arrives, a *PacketIn* message is received at the controller, which spawns a new thread and enters state C, steering f to the VNF in charge of flow classification. Depending on the classification result and current network conditions, the FSM state changes, and flow f can be:

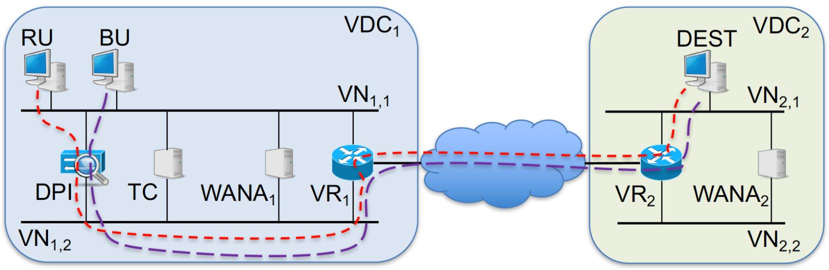
- Not accepted in the network (D state)
- Forwarded with no actions (N state)
- Treated according to what is required by the SLA to enforce the correct QoS (E state)

If the new flow is classified as SLA-compliant, a conservative approach would require it to be moved from state C to state E, thus enforcing the SLA for any new flow admitted in the network. Then, if network conditions are favorable, the new flow can move to state N, where packets can be forwarded directly to the network gateway without further processing by VNFs enforcing the negotiated SLA. Otherwise, in the case of critical conditions, all previously admitted flows move to the E state. As soon as the potential risk of congestion is solved (e.g., some flows terminate), active flows can progressively move back to state N. This is how the dynamic VNF chaining is accomplished, with proper traffic steering actions governed by cooperation among the SDN controller, the network monitoring system, and the VNF itself implementing the flow classification function.

The data plane virtual network topology implemented in the Ericsson Cloud Lab for the PoC described above is shown in Fig. 3.18a. Two virtual data centers (VDC_1 and VDC_2) are assumed to be in place. VDC_1 hosts BU and RU instances, and thus represents their access network, whereas VDC_2 hosts the destination to which all flows generated by the BU and RU are directed (DEST acts as a sink for PoC purposes). It is assumed that each VNF is implemented as a layer 3 virtual appliance connected to two separate broadcast domains (i.e., virtual networks $VN_{i,1}$ and $VN_{i,2}$ for VDC_i. Then a layer 3 virtual router (VR_i) is used to interconnect VDC_i to the external network. The provider maintains layer 3 connectivity to the user while deploying the required VNFs. This approach is typical of access/edge network operator deployments. Thus, the dynamic traffic steering and the SDN controller must be designed taking into account these transparency requirements.



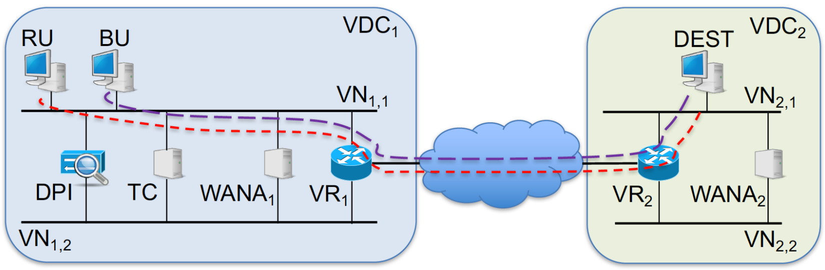
(a) Proof of concept data plane virtual network topology.



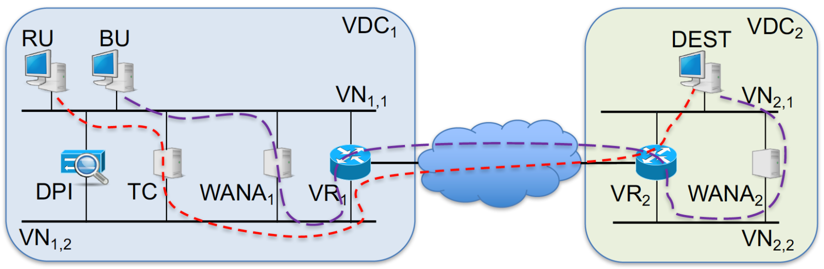
(b) C state, classification of traffic flows.

Figure 3.18: Proof of concept data plane virtual network topology and traffic flow steering in the C operating state.

In Fig. 3.18b traffic steering in the C state is shown. When a BU or an RU initiates a new flow directed to DEST, the flow is forwarded from VDC_1 to VDC_2 through gateways VR_1 and VR_2, but only after the SDN



(a) N state, no QoS enforcement.



(b) E state, QoS enforcement with VNF chaining.

Figure 3.19: Traffic flow steering in the N and E operating states.

controller configures the VDC$_1$ internal network to force the flow through the DPI. This VNF is in charge of classifying the flow, and detecting its QoS class and compliance with the respective SLA. Following the classification, the DPI communicates with the controller via the northbound interface (not shown in the figure) the set of information required to identify the flow (e.g., medium access control [MAC] and IP addresses, protocol, TCP/UDP ports) and its classification (whether BU or RU). No manual intervention is required as long as the DPI is made aware of any new user and related SLA at service setup. As a result of the classification, or if network conditions change, the SDN controller changes the flow state.

When network conditions are such that the SLAs are not at risk, the controller configures the active flows into state N, resulting in normal forwarding through the access LAN and the gateway as shown in Fig. 3.19a. Otherwise, if a potential congestion risk is detected by the network monitor (not shown), the controller moves the flows to state E and injects new forwarding rules to the network. With the aim to enforce SLA compliance for both BU and RU, their flows are forwarded to WANA$_1$ for improved QoS and to TC for bandwidth usage control, as depicted in Fig. 3.19b. The BU flow is also steered to WANA$_2$ in VDC$_2$ because compressed traffic should be restored to the original format before reaching DEST. This peculiar behavior has been included in the PoC to show how the stateful controller design is able to support interdependence between multiple VNF instances.

3.2.8 Experimental results on Ericsson Cloud Lab environment

The data plane virtual topology shown in Fig. 3.18a was deployed on the Ericsson Cloud Lab environment, following the procedure and instantiating the VMIs as described earlier. The logic to enforce the SLAs described earlier was programmed within the SDN controller. However, to simplify network reconfiguration in the PoC, but without any loss of generality, the service orchestration phase was implemented in a simpler and less conservative way:

- As long as a new flow is classified as RU, and if network conditions are not critical, the flow enters state N.
- As soon as a BU flow is detected, network conditions are considered critical, and all flows enter state E.

As a proof of the correct behavior of the dynamic traffic steering configured by the SDN controller according to the aforementioned orchestration logic, the throughput of four user flows (three generated by RU and one by BU) was measured on relevant ports of OpenStack's virtual switch that implements the virtual networks instantiated in VDC_1. Each flow consisted of a distinct TCP stream generated using the iperf tool[11] with the following time pattern:[12]

- Flow 1 from RU started at time T_1.
- Flow 2 from RU started at time $T_2 = T_1 + 30$ s.
- Flow 3 from RU started at time $T_3 = T_1 + 60$ s.
- Flow 4 from BU started at time $T_4 = T_1 + 90$ s.
- All flows were terminated at time $T_5 = T_1 + 200$ s.

Figure 3.21 shows the total throughput measured on the virtual switch ports connected to BU and RU when 3.21a a static network configuration is used, and 3.21b the SDN-based dynamic traffic steering is used to enforce

[11]`https://iperf.fr/`

[12]This time pattern was chosen to provide maximum readability to the results presented in the following (see comments to Figure 3.20 for instance). However the system discussed can cope with general arrival patterns.

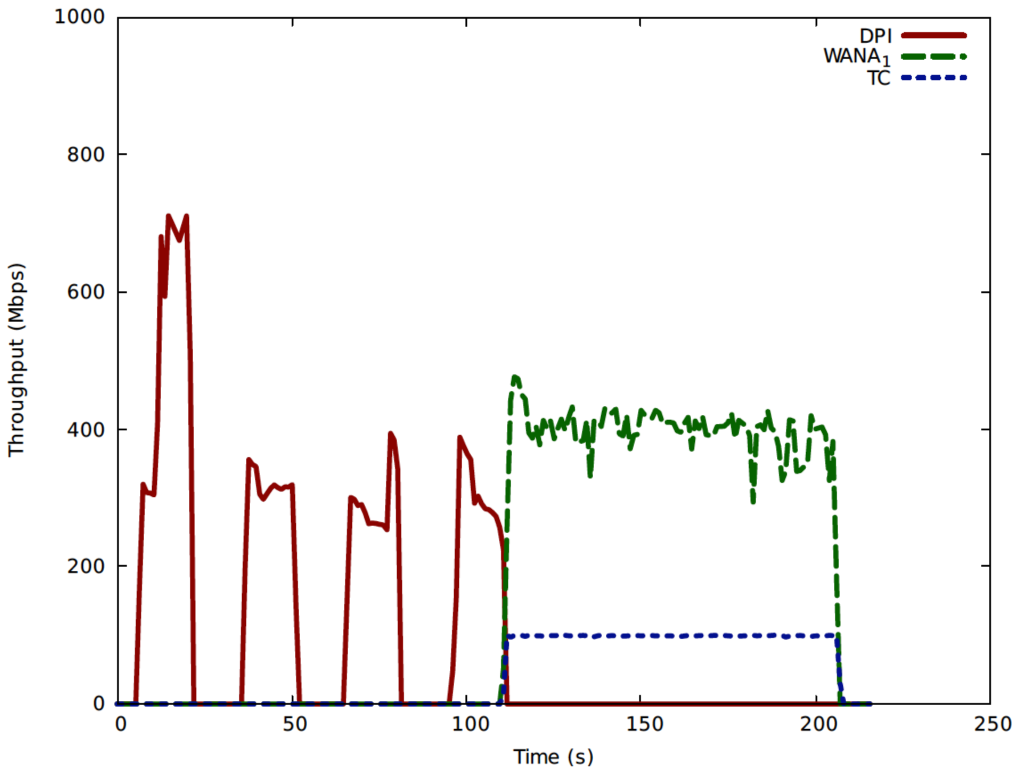
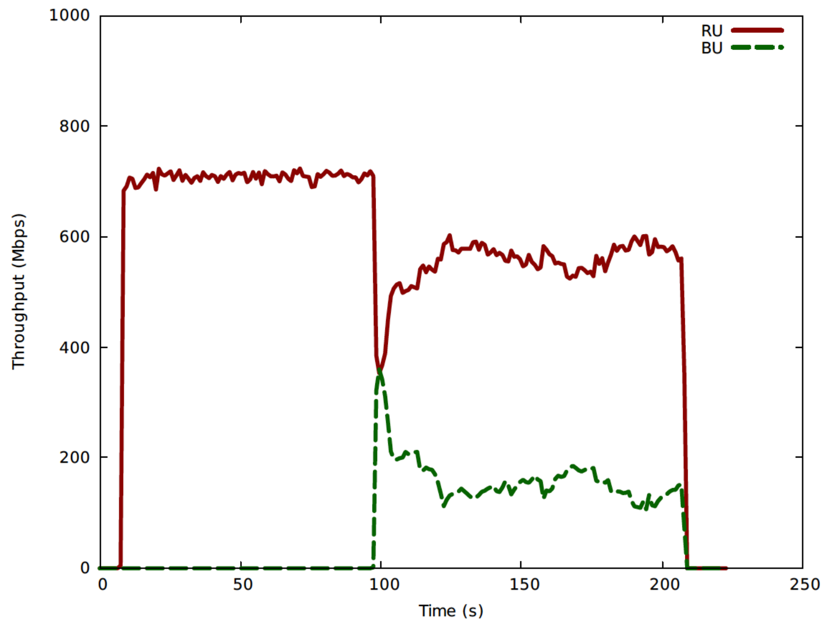


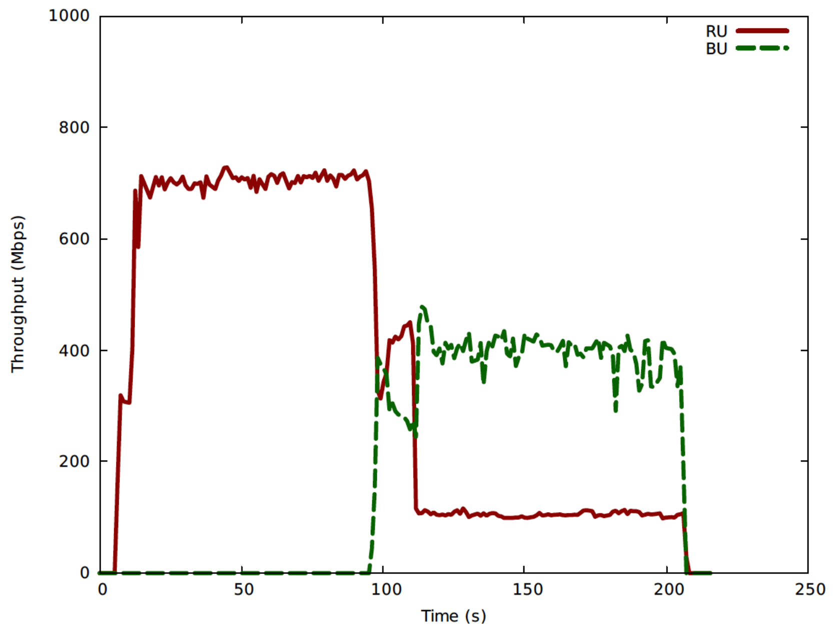
Figure 3.20: Measured throughput across the VNFs with dynamic traffic steering.

QoS. It is apparent how, without dynamic traffic steering, the avail- able bit rate is completely used by RU flows until flow 4 from BU starts at T_4 ∼100 s; then the four flows equally share the available bit rate, with BU using only about one fourth of the capacity. However, when dynamic traffic steering is enabled, after a small interval when flow 4 is in state C (between ∼100 s and ∼110 s), the controller moves all flows to state E and the QoS enforcement takes place: the total throughput of the three RU flows is limited to 100 Mb/s, whereas the BU flow throughput is not penalized.

Figure 3.20 highlights how the flows traverse the different VNFs when the dynamic traffic steering is applied. At the beginning, the three RU flows and the BU flow are processed by the DPI (state C). For each flow, the traffic steering to the DPI stops as soon as the classification is complete, which happens before the next flow starts. This results in four spikes interleaved by sudden drops of the throughput curve measured at the input of the DPI. After SLA compliance is determined for the three RU flows, since no BU



(a)



(b)

Figure 3.21: Measured total throughput for RU and BU: a) without dynamic traffic steering; and b) with dynamic traffic steering.

flow is active yet, the RU flows are sent directly to VR_1 and the QoS is not enforced[13] (state N). However, after the BU flow is classified and its compliance to the priority SLA is detected (approximately at time $T_1 + 110$ s), the QoS is enforced by properly steering all flows to the relevant VNFs (state E): the three RU flows are then sent to the shaping function TC, which limits their total bit rate to 100 Mb/s, whereas the BU flow is processed by the $WANA_1$ (and $WANA_2$) functions, as proved by the throughput measured at the output of the respective VNFs.

It is worth noting that, despite the bandwidth limitation applied to RU flows, the BU flow is not able to fully take advantage of the remaining capacity: this is due to the limited amount of computing resources in the OpenStack compute nodes used for the experiments, and the increased processing burden on the OpenStack virtual network infrastructure caused by the dynamic traffic steering [24]. The detailed evaluation of the trade-off between the performance improvement gained using dynamic traffic steering and the possible performance degradation due to cloud processing overload is beyond the scope of this work and is currently under investigation. Indeed, it is highly dependent on the hardware configuration and, at least to a certain degree, can be overcome by hardware improvement.

3.3 Conclusion

In this chapter several activities have been reported, each with its own primary goal but associated with the issue of implementing chain of virtual network functions; such issue has been investigated step-by-step.

Initially, two possible SDN scenario alternatives have been considered, namely Layer 2 and layer 3 approaches; since both are viable solutions, we wanted to study the complexity of the SDN control plane, focusing in particular on the actions to be programmed into OpenFlow switches in order to achieve dynamic service chaining. The Mininet emulation platform has been used to provide a former practical example of the feasibility and degree of complexity of the two approaches: in spite of the fact that the two approaches provide identical results, each scenario requires not only its own custom topology but it requires its own custom controller, as well. This is due to the fact that they face different issues and different actions when dealing with traffic steering.

Then, after a careful analysis of the practical issues posed by a given NFV deployment, and a smart definition of the set of actions to be programmed in the network, we extended our study by investigating the design of general

[13]For readability reasons, we do not show in Fig. 3.21 the throughput measured at VR_1.

traffic steering policies. To this purpose, we decided to leverage the Mealy Finite State Machine as a general approach to service chain reconfiguration and to model the behavior of the SDN control plane. The feasibility of such approach has been proved through the implementation of a proof-of-concept on the OpenStack platform, which highlighted also a possible approach to orchestrate the mutual dependence of different flows competing for the same network resources. Given the flexibility and cost-effectiveness of the solution of replacing hardware-based, vendor-dependent middle-boxes with software appliances, and in order to adapt to emerging network conditions, it is essential take into account that type, number and location of VNFs traversed by a given user data flow may change in time. One of the primary goal was to show that this approach allows space and time diversity in service chaining, with a higher degree of dynamism and flexibility with respect to conventional hardware based architectures. We believe that, considering both legacy forwarding and flexible traffic steering, it is possible to simplify the SDN controller design and successfully achieve dynamic service chaining.

Last, but not least, another important goal of the activity reported in this chapter was to investigate how to create the technical conditions for accelerating the adoption of SDN and NFV in order to produce concrete socio-economic impacts. In general, it is argued that the ongoing “softwarization” of telecommunications (which goes beyond the adoption of SDN and NFV in core networks, also reaching terminals and clouds) will allow virtualizing all network and service functions and executing them in horizontal software platforms fully decoupled from the physical infrastructure.

In this direction, section 3.2 reports the outcome of an experimental PoC aimed at demonstrating the capability of a software-controlled network to self-adapt by dynamically reconfiguring the VNF forwarding graph in order to guarantee the agreed SLAs to the user.

Chapter 4

Towards Network Management and Orchestration

Historically, network management is defined as being the plane devoted to monitoring, repairing and configuring network devices. This is usually done through Command Line Interface (CLI), that uses protocol such as SSH, Telnet and SNMP [54] to configure and maintain devices. With SDN, the management plane is defined as a set of applications that leverage the functions offered by the North Bound Interface to implement the network control and operation logic [3]. This vision is in line with the definition given by the IETF (Internet Engineering Task Force), which defines the management plane as the collection of functions (applications) responsible for monitoring, configuring, and maintaining one or more network devices or parts of network devices [55].

NFV requires to be managed as well; this is why the ETSI Industry Specification Group (ISG) defined the NFV Management and Orchestration (MANO) framework [6], which is composed of three functional blocks (figure 4.1):

1. NFV Orchestrator;
2. VNF Manager;
3. Virtualized Infrastructure Manager (VIM).

According to IETF, those entities could be management plane functionalities; the NFV architecture hints the need for cooperation and coordination among several functional blocks, thus, the orchestration can be seen as a subset of the management in the context of combining NFV and SDN (since the architecture might include both SDN resources and SDN controllers, which could be located in different part of the ETSI MANO framework, as it is briefly discussed in section 4.2). Therefore, there is the need of defining new standards and well defined interfaces among the different functional blocks that are in place in figure 4.1. One of the most critical interface is the one between the control platform and the NFV orchestrator. Such interface is often defined as the Northbound Interface (NBI), i.e., the interface exposed by the the SDN controller to the applications or to an higher level orchestrator.

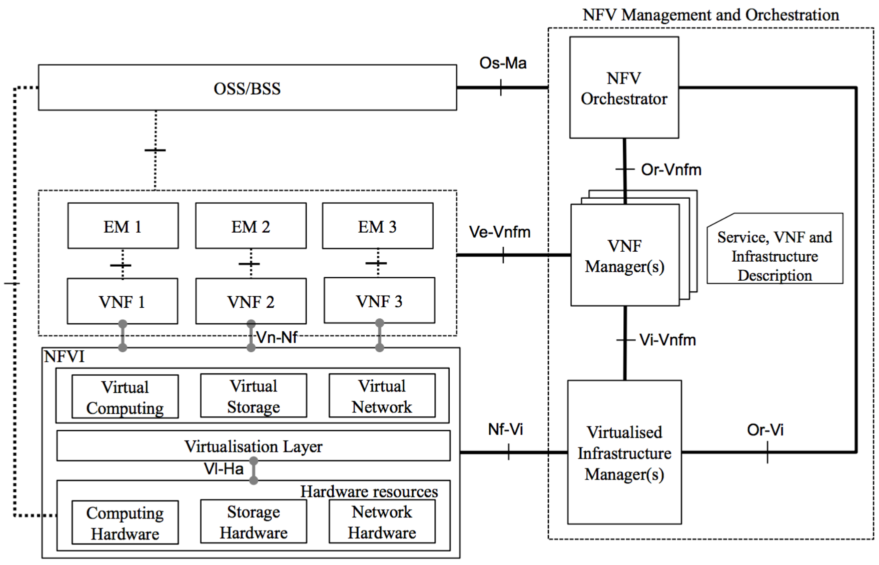


Figure 4.1: ETSI NFV architecture.

A standard for the NBI is not available yet, but its definition and principles have recently been clarified by the Open Networking Foundation (ONF) [7]. The proposed approach is to adopt a so-called *intent-based* interface that allows to declare service policies rather than specify networking mechanisms, i.e., the focus is on *what* should be achieved rather than on *how* it should be achieved. This is only one aspect of the more general problem of finding suitable service model descriptions for NFV environments [56]. In section 4.1, we present an intent-based approach to the definition of the NBI between the VIM and the higher management and orchestration layers defined in the ETSI MANO framework. In particular we focus on the network infrastructure programmability functions of the VIM, assuming a general network scenario where multiple SDN domains are interconnected by non-SDN domains.

4.1 An Intent-based Approach to Virtualized Infrastructure Management

Deploying a given SFC in traditional infrastructures requires time-consuming configuration and management tasks on vendor-specific appliances, often resulting in static packet processing and forwarding rules, unaware of changing user needs and network conditions. On the contrary, by adopting a “softwarized” infrastructure the SFC deployment paradigm can be completely changed. We have already demonstrated in practice (section 3.2) that the SFC dedicated to a given customer (or set of customers) can be *dynamically controlled and modified over a relatively small time scale*, according to external commands or by reacting to changing service conditions.

Key to an effective implementation of these concepts is a proper northbound interface (NBI) through which high-level orchestration and management entities are allowed to control the underlying NFV and SDN platforms and implement the dynamic SFC features. In this section we present a proof of concept (PoC) implementation of a vendor-independent, technology-agnostic, intent-based NBI for controlling dynamic service function chaining on top of a SDN infrastructure. The NBI and the overall PoC architecture are compliant with both the ETSI MANO framework specifications and the aforementioned NBI description by ONF. The PoC was designed to provide adaptive quality of service (QoS) enforcement to a multi-tenant NFV scenario. In particular, here we focus on performance and scalability aspects of the proposed NBI, demonstrating the feasibility of our approach.

4.2 Reference NFV architecture

The NFV architecture considered in this work, sketched in Fig. 4.2, is in line with the ETSI MANO framework [6]. The resources needed to deploy the VNFs are located in several SDN domains, where the specific control and management platform includes:

- a Virtualized Infrastructure Manager (VIM) in charge of controlling and managing the compute, storage, and network resources in a given domain;

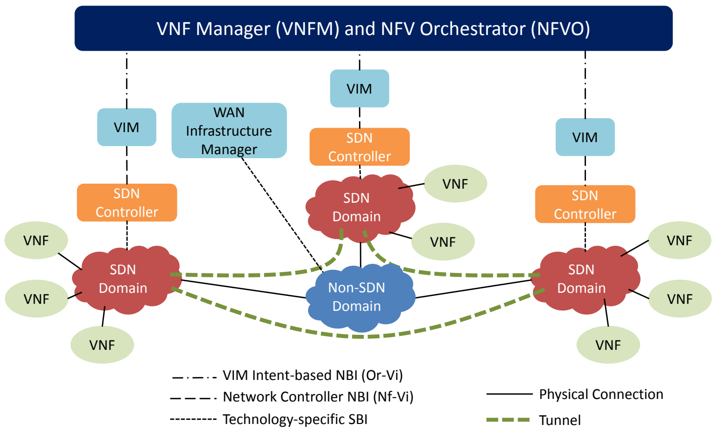


Figure 4.2: Reference NFV architecture.

- an SDN controller programmed to properly steer traffic flows across the VNFs, according to a specified SFC.

The VIM/controller couple can represent:

- an instance of the same VIM/controller distributed across different domains;
- a stand-alone VIM/controller federated with the other ones;
- a stand-alone VIM/controller completely unaware of the other ones.

It is important to outline that this reference architecture considers also the possibility that SDN domains are interconnected through non-SDN domains. This assumption stems from the fact that it appears reasonable that a network operator will deploy SDN technologies mainly within data center infrastructures where the VNF resources will be located—e.g., in the operators' points of presence or central offices— rather than in backbone networks. In this case, traffic flows that must traverse a number of SDN domains, according to a given SFC, can be properly routed by adopting some form of tunneling or overlay network technology across the non-SDN domains, such as Generic Routing Encapsulation (GRE) [57], Virtual eXtensible Local Area Network (VXLAN) [58], or the emerging Network Service Header (NSH) [59], to name a few.

The overarching VNF Manager (VNFM) and NFV Orchestrator (NFVO) are responsible for programming the underlying VIMs/controllers in order to implement and maintain the required SFC in a consistent and effective way, both intra- and inter-domain. The interfaces located between the VNFM/NFVO and the VIM are called `Vi-Vnfm` and `Or-Vi` in the ETSI MANO specifications. The former is related to VNF management tasks, whereas the latter is dedicated to orchestration tasks. The MANO specifications make clear that the `Or-Vi` is "[...] used for exchanges between the NFV Orchestrator and the VIM".

Nonetheless, the ETSI specifications leave the implementation details completely open. This is where we contribute with our approach, by showing with a suitable PoC the effectiveness of an intent-based implementation of the NBI provided by the VIM to the orchestration module. The intent-based approach focuses on *what* should be done and not on *how* it should be done, and aims at decoupling the abstracted service descriptions issued by applications and orchestrators from the technology-specific control and management directives that each VIM/controller must send to its respective devices through the southbound interface (SBI) [56]. This would give different domains the freedom to choose different SDN technologies to control traffic flows and different cloud computing platforms to manage VNF instances, as long as the common NBI is exposed.

Our approach is also in line with the ONF description of the Intent-based NBI [7], even though it does not make use of the Boulder module [60], as it is still immature and unstable. Given that a general purpose intent syntax and framework is not available yet, we decided to take advantage of SDN controller-dependent intent interfaces. More specifically, we choose the Open Network Operating System (ONOS) platform for our PoC [61]. However, the use of ONOS standalone interface does not provide the required abstraction levels for a general and technology-independent NBI. While the ONOS Intent Framework [62] does allow users to directly specify a high-level policy, it also requires some knowledge of low-level details, such as IP addresses and switch port numbers of hosts and devices to be interconnected. This is needed in order to perform operations such as flow rules being installed on OpenFlow switches or tunnel links being provisioned.

On the other hand, the NBI implementation used in our PoC allows the user to specify high-level service policies without the need to know any network detail: these details are grabbed and handled by our domain-specific solution, and the high-level policies are resolved and mapped into suitable ONOS intents, which will then be compiled and translated into proper flow rules. Although in principle our approach can be applied to any SDN control plane technology, in our PoC we opted for the ONOS platform instead of OpenDaylight (ODL) [63] because we estimated an ODL-based solution to be more complex in terms of implementation: indeed, it would have required to set up a working environment composed by the Network Intent Composition module [64] and the Service Function Chaining module [65], as well as the development of an additional application to deal with the NBI for management purposes.

4.3 Intent-based NBI for dynamic Service Function Chaining

As already mentioned, the definition of an open, vendor-agnostic, and interoperable NBI will foster improved interactions between high-level applications and orchestration services and the underlying NFV and SDN platforms. The powerful abstraction level offered by the intent-based approach allows the specification of policies rather than mechanisms through the NBI. Therefore, an intent-based NBI represents a key feature in a multi-domain NFV scenario such as the one assumed here.

When a given service request is received, the NFV Management and Orchestration framework must convert that request into a suitable service graph and pass it to the relevant VIMs in charge of any domain involved in the SFC composition. Then each VIM must:

- verify the availability and location of the VNFs required in its own domain, instantiating new ones if needed;
- interact with the relevant SDN controller to program traffic steering rules and deploy a suitable network forwarding path in the SDN domain.

In order to provide an abstracted yet flexible definition of the requested service graph, without knowledge of the technology-specific details (such as devices, ports, addresses, etc.), the NBI exposed by the VIMs should allow to specify not only the sequence but also the nature of the different VNFs to be traversed, which is strictly related to the service component they implement. For instance, consider the case of a customer requesting connectivity to a given remote content or service. Assume that, according to the service level agreement (SLA), the network operator must provide bidirectional WAN acceleration as well as intrusion detection (IDS) features. In addition, the operator should apply deep packet inspection (DPI) and traffic shaping to monitor and police customer-generated traffic. Both customer requirements and operator needs must be expressed as intents and passed to the NFV orchestrator, resulting in the SFC illustrated in Fig. 4.3. Therefore, the NBI should allow an abstracted representation of the topological characteristics of each VNF, specifying the different ways VNFs can be inserted in a SFC.

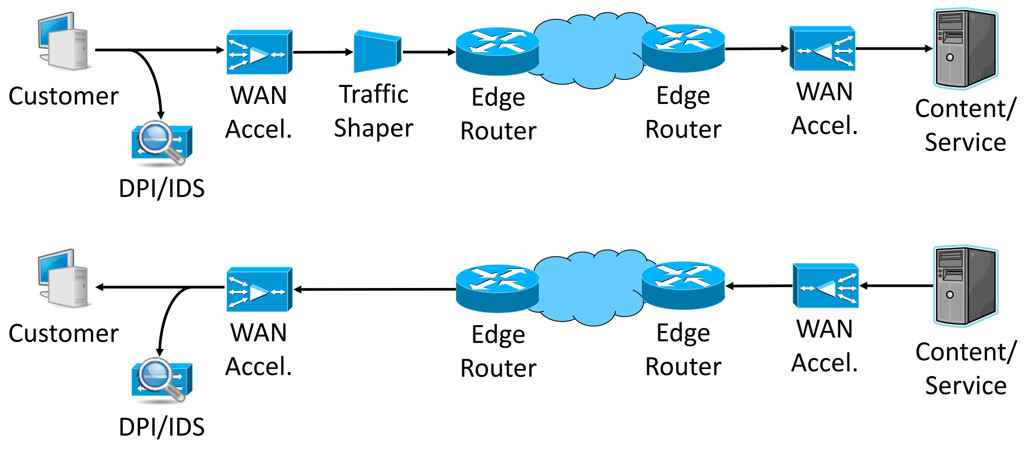


Figure 4.3: Example of SFC representing both customer requirements and operator needs. Top: upstream traffic. Bottom: downstream traffic.

In this work the following topological abstractions are considered:

- A VNF can be *terminating* or *forwarding* a given traffic flow. In the example, DPI/IDS is terminating the flow, whereas traffic shaper and WAN accelerator are forwarding it.
- A forwarding VNF can be *port symmetric* or *port asymmetric*, depending on whether or not it can be traversed by a given traffic flow regardless of which port is used as input or output. In the example, the WAN accelerator is port asymmetric, because it compresses or decompresses traffic based on the input port used. The traffic shaper can be considered port symmetric, if we assume that the shaping function is applied to any output of the VNF.
- A VNF can be *path symmetric* or path asymmetric, depending on whether or not it must be traversed by a given flow in both upstream and downstream directions. In the example, according to the service requirements, WAN accelerator and DPI/IDS are path symmetric, whereas the traffic shaper is path asymmetric.

In order to implement the topological abstractions described above, we define a sort of ETSI MANO deployment template [6], adopting the well-known JSON format. An in-depth discussion of the formal specification of our proposed intent-based NBI is out of the scope of this work and is left for future works.

With reference to the example in Fig. 4.3, let us assume the following VNF identifiers:

- `DPI_IDS` for the DPI/IDS;
- `WA_1` for the WAN accelerator located at the customer edge network;
- `WA_2` for the WAN accelerator located at the content/service edge network;
- `TS` for the traffic shaper.

Then the resulting SFC can be expressed as follows:

```
{
  "src": "Customer",
  "dst": "Content/Service",
  "vnfList": [DPI_IDS, WA_1, TS, WA_2]
  "dupList": [DPI_IDS]
}
```

where `src` and `dst` represent the endpoint nodes of the SFC, `vnfList` is an ordered list of VNFs to be traversed, `dupList` is a (possibly empty) list of VNFs towards which the traffic flow must be duplicated, and each VNF is described in terms of its topological abstractions as follows:

```
DPI_IDS ::= {
  "name": "DPI_IDS",
  "terminating": "true",
  "port_sym": "null",
  "path_sym": "true"
}

WA_1 ::= {
  "name": "WA_1",
  "terminating": "false",
  "port_sym": "false",
  "path_sym": "true"
}

TS ::= {
  "name": "TS",
  "terminating": "false",
  "port_sym": "true",
  "path_sym": "false"
}

WA_2 ::= {
  "name": "WA_2",
  "terminal": "false",
  "port_sym": "false",
  "path_sym": "true"
}
```

The NBI, which can be implemented through the mechanism of a Representational State Transfer (REST) Application Programming Interface (API), must provide the following set of operations:

- add a new SFC
- update an existing SFC
- delete an existing SFC.

These actions are basically in line with the operations foreseen by the ETSI MANO specifications with reference to the `Or-Vi` interface.

It is worth highlighting that the NBI description given above can be considered based on intents. VNFs and SFCs are indeed specified in a high-level, policy-oriented format without any knowledge of the technology-specific details. A non-intent-based description of the SFC shown in Fig. 4.3, e.g. using the OpenFlow expressiveness to steer traffic flows and compose the network forwarding path, would require the application or the orchestrator to specify multiple flow rules in each forwarding device for each traffic direction, and include explicit actions to mirror traffic to `DPI_IDS`, involving technology-dependent details such as IP and MAC addresses, device identifiers and port numbers.

4.4 Implementation of the VIM on the ONOS platform

To demonstrate the capabilities of our Intent-based NBI, we built a PoC implementation of the VIM according to our reference architecture. We adopted the ONOS platform (version 1.7) as the SDN domain controller for our first PoC [61], although the approach is more general and will be extended to other controllers in the future. ONOS provides a set of built-in intents that can be used to program the SDN domain and deploy the required network forwarding paths. However, the ONOS intent-based interface requires some knowledge of the data-plane technical details, while in our approach we prefer to expose only high-level abstractions to the orchestrator. Therefore, one of the main functions of our VIM is to implement new, more general and abstracted intents that can be expressed according to the NBI specifications given in section 4.3. Then the VIM takes advantage of the network topology features offered by ONOS to discover VNF location and connectivity details, and eventually it is able to compose native ONOS intents and build more complex network forwarding paths as per the SFC specification.

The VIM was developed as an application running on top of the ONOS platform. It is offered as an ONOS service called *ChainService*, which provides the capability of dynamically handling the SFC through the abstracted NBI. To achieve extensibility and modularity, the ChainService implementation is delegated to a module called *ChainManager*, which is in charge of executing all the required steps to translate the high-level SFC specifications into ONOS native intents. An interaction diagram of the ChainService components is given in Fig. 4.4. The input to the ChainManager can be given through either the ONOS Command Line Interface (CLI) or a REST API. The latter is preferable because it allows remote applications to use standard protocols (e.g., HTTP) to access resources and configure services. In our implementation, the REST API provides the following service endpoints:

```
POST /chaining/{action}/{direction}
DELETE /chaining/flush
```

In the former endpoint, the `action` variable indicates the operation that the user wants to perform on a specified SFC (add, update, or delete), whereas in case of an update the `direction` variable (forth, back, or both) defines whether the modified SFC specification refers to the existing forwardng path from source to destination, the opposite way, or both directions. So the basic operations of this endpoint are as follows:

- If the *add* action is given, this will result in defining a new SFC, based on the JSON specification included in the message body. This means that a forwarding path will be created for traffic flowing from source to destination (say `SrcToDst`), and another one in the opposite direction (`DstToSrc`). Note that `DstToSrc` does not necessarily replicate `SrcToDst` in reverse order, since it must follow the topological abstractions defined by the NBI.

- If the *update* action is given, then the direction is taken into account and the forward path, backward path, or both paths of the specified existing

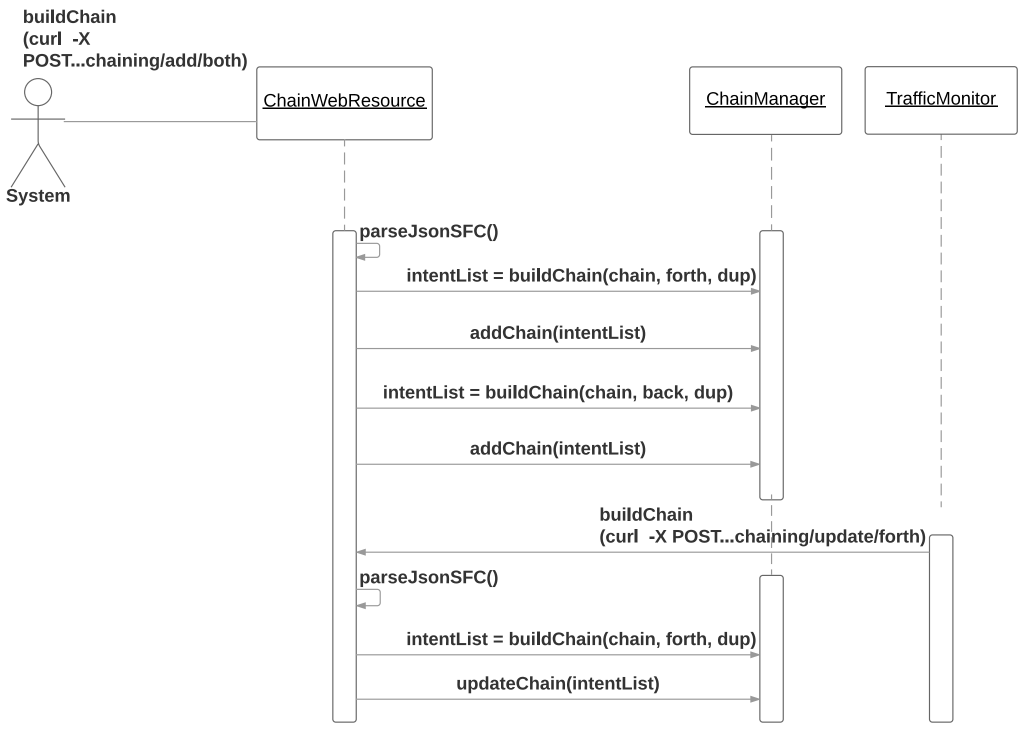

buildChain
(curl -X
POST...chaining/add/both)
System
ChainWebResource
ChainManager
TrafficMonitor
parseJsonSFC()
intentList = buildChain(chain, forth, dup)
addChain(intentList)
intentList = buildChain(chain, back, dup)
addChain(intentList)
buildChain
(curl -X POST...chaining/update/forth)
parseJsonSFC()
intentList = buildChain(chain, forth, dup)
updateChain(intentList)

Figure 4.4: Interaction diagram for SFC deployment by VIM. A system administrator creates a new SFC, resulting in bidirectional forwarding paths being installed. A traffic monitoring agent dynamically updates the SFC by changing the forwarding path in one direction only. Both use a REST interface.

SFC are changed. In fact, a user may be interested in changing only a segment of the forwarding path and only in one direction, to reduce the latency and limiting the impact that a path change can have on the traffic flows.

- If the *delete* action is given, then both forwarding paths of the specified existing SFC are removed.

ChainService provides also the *flush* operation through another endpoint, thus offering the possibility of deleting in a single step the forwarding paths of all the SFCs previously created.

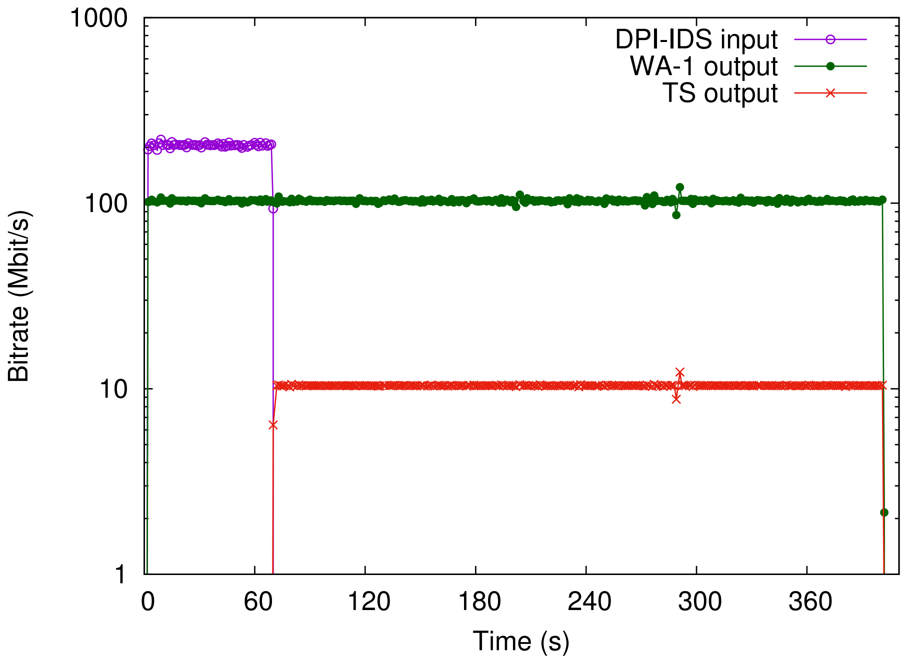


Figure 4.5: Bitrate measured at relevant VNF ports.

4.5 Validation of the PoC

The first validation of our PoC implementation was performed on top of an OpenFlow network infrastructure emulated with Mininet [66]. The considered network topology was spread across three different SDN domains, where each domain was emulated by a separate Mininet instance running on a dedicated virtual machine. The three SDN domains were interconnected through an IP-based network that emulated the non-SDN domain shown in Fig. 4.2. A single ONOS platform was configured as a distributed SDN controller across the three domains. Endpoint nodes and VNFs composing a given SFC can be part of the same domain or can be located across different domains. This means that the traffic steering rules needed to deploy a given SFC forwarding path must be capable of reaching both co-located VNFs and distributed ones. To this purpose, static GRE tunnels were established between the edge nodes of each pair of SDN domains.

To demonstrate that our approach works, the bitrate of two flows generated by different customers was measured at the ports of some relevant VNFs. Figure 4.5 shows the bitrate measured at the input of `DPI_IDS` and at the outputs of `WA_1` and `TS`. Two UDP flows were generated with *iperf* at

100 Mbit/s each. At the beginning of the test, the orchestrator specified the same SFC for both customers through the NBI:

```
"vnfList": [DPI_IDS, WA_1, WA_2]
"dupList": [DPI_IDS]
```

The rate measured at the input of `DPI_IDS` was equal to the sum of the two flows (200 Mbit/s), whereas the rate at the output of `WA_1` was cut to 100 Mbit/s, as this was the maximum speed of the VNF interface. Then, at around $t = 70s$, the DPI detected that one of the two customers was sending high priority traffic, while the other one was sending best effort traffic. Therefore, the orchestrator dynamically updated the SFCs as follows:

```
"vnfList": [WA_1, WA_2],  "dupList": []
```

for high priority traffic and

```
"vnfList": [TS],  "dupList": []
```

for the best effort flow. This resulted in the former flow still crossing `WA_1` at 100 Mbit/s, while the latter flow was correctly shaped by `TS` at 10 Mbit/s.

4.6 Experimental results

To assess the NBI response time and scalability, we measured the latency experienced by an *add*, *update*, *delete* or *flush* action for an increasing number of SFCs. To get statistically valid measurements, we performed 20 different runs for each point and computed the average response time value with 95% confidence intervals, as shown in Figs. 4.6 and 4.7. Clearly, the trend shows that the higher the number of SFCs on which each action must be performed, the higher the load on the VIM and the number of operations to be executed by the SDN platform, resulting in an increased response time. Nonetheless, the latency of each single action does not exceed 2s, showing the correct functionality of the NBI and the scalability potentials of the proposed approach.

The *update* and *delete* actions show higher response times than *add* and *flush*, due to the additional time required by the VIM and the ONOS intent framework to search for the intents related to the specified SFC to be updated or deleted. In particular, a *delete* takes typically longer than a *flush*, because in the former case ChainManager is instructed to remove the SFCs one by one, whereas with the *flush* action ChainManager proceeds directly with removing all the forwarding paths. However, the wider confidence intervals obtained with more than 100 SFCs denote the extremely variable response time measured when the VIM was overloaded.

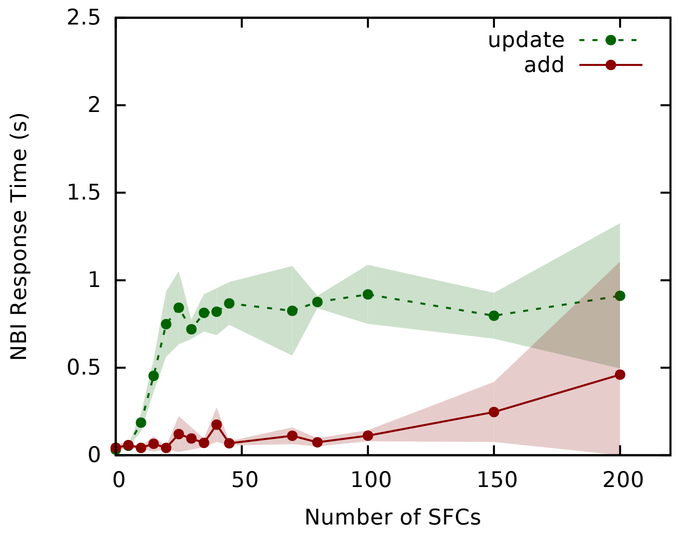


Figure 4.6: Average NBI response time and 95% confidence interval when SFC *add* and *update* actions are performed, as a function of the number of SFCs.

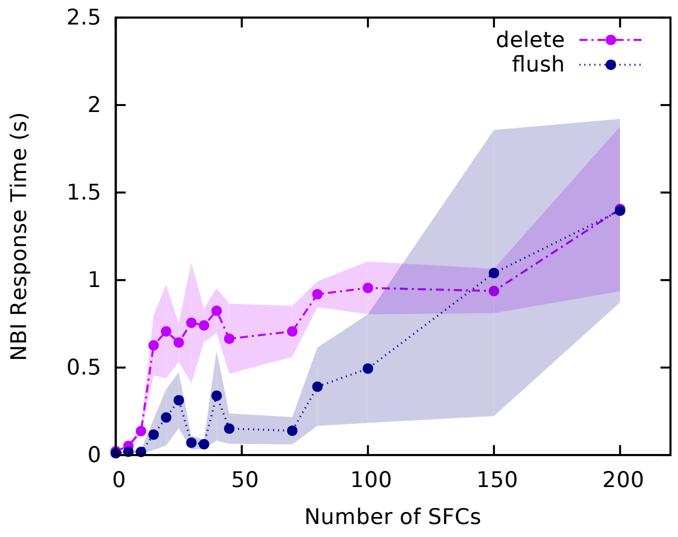


Figure 4.7: Average NBI response time and 95% confidence interval when SFC *delete* and *flush* actions are performed, as a function of the number of SFCs.

4.7 Conclusion

In this chapter we presented an intent-based approach to the definition of the NBI between the VIM and the higher management and orchestration layers defined in the ETSI MANO framework. In particular we focused on the network infrastructure programmability functions of the VIM, assuming a general network scenario where multiple SDN domains are interconnected by non-SDN domains. In line with recent ONF specifications, we defined a high-level, vendor-independent, intent-based NBI which provides semantic expressing "what" should be done in term of service function chaining, and not "how" it should be done, thus hiding the implementation details to the high-level orchestrator and making the service requests agnostic to the specific technology adopted by each SDN domain. We reported on a proof-of-concept implementation of the VIM and its NBI on top of the ONOS SDN platform. We tested it on an emulated multi-domain SDN infrastructure, where we were able to successfully create, update, delete and flush the forwarding paths according to the intent specified via the NBI. We also measured the responsiveness of our VIM implementation under increasing load, proving the correct functionality of the NBI and the scalability potentials of the proposed approach. This solution represents an interesting starting point to create more complex and general interactions between the NFV management and orchestration layers and the VIM.

Chapter 5

Conclusions

Software Defined Networking (SDN) and Network Function Virtualization (NFV) seem to be promising solutions that, properly combined together, can bring several benefits and open opportunities to several players, such as network operators, service and infrastructure providers and enterprise networks. To overcome todays network high costs and manageability concerns, players could take advantages of flexibility introduced by both SDN and NFV, and leverage well matured virtualization technologies already in use in data centers networks.

In this thesis, we mainly focused on scenarios that are of interest for telecommunications operators networks, which are trying to evaluate the impact that such technologies can have on their business, and how they can profitably migrate to more flexible and agile networks. In this evolution process, software will play an unprecedented dominant role, but network "softwarization" process also brings several challenges.

5.1 Summary of Contributions

In this dissertation, we focused on performance and management concerns, with particular attention to the service function chaining network capability. Such focus has been explored at all different network operational planes:

- **data plane:** since Telco Central Offices will likely be replaced by cloud data centers located at the edge, we decided to provide some insights on how an open-source cloud computing platform such as OpenStack implements multi-tenant network virtualization and how it can be used to deploy NFV, focusing in particular on packet forwarding performance issues.

The main goal of this work was to identify performance bottlenecks in the cloud implementation of the NFV paradigms, and try to answer few questions that naturally rise when dealing with system integration, data center management, and packet processing performance: 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?
As we said, the answer to those questions is a function of the cloud management platform considered; therefore, an ad hoc set of experiments were designed to evaluate the OpenStack performance under critical load conditions, in both single tenant and multi-tenant scenarios. It is worth investigating if virtualized networks will provide performance comparable to those achieved with current physical networks. Therefore, performance of underlying platform should be known in order to achieve a better planning about how properly dimension networks and virtual appliances placement.

Performance analysis of the two basic software switching elements natively adopted by OpenStack, namely Linux Bridge and Open vSwitch, prove that the Linux Bridge is the critical bottleneck of the architecture, while Open vSwitch showed an almost optimal behavior. The main outcome of this work is that an open-source cloud computing platform such as OpenStack can be effectively adopted to deploy NFV. 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. These limitations can be mitigated with a careful redesign of the virtual network infrastructure and an optimal planning of the virtual network functions.

We successfully accomplished the main goal of this work, even if we believe that, in order to fully answer all previous questions, further and deeper investigation would be required. To the best of our knowledge, at the time of such investigation, not much work was reported about the actual performance limits of network virtualization in OpenStack cloud infrastructures under the NFV scenario.

- **control plane:** future edge network data center infrastructure must be flexible enough to allow dynamic and conditional Virtual Network

Functions (VNFs) chaining; thus, properly steer the traffic flows to make them traverse the chain of VNFs that implement the correct Service Level Agreements (SLAs) for each user will be key.

The main goal of this work was, first, investigate the issues of implementing chains of network functions in a "softwarized" environment and, in particular, the complexity of the SDN control plane within a cloud-based edge network implementing NFV. At the beginning, the focus was mainly on describing the actions to be programmed into OpenFlow switches in order to achieve dynamic service chaining for two representative case studies, namely Layer 2 and Layer 3 edge network function implementations. Then, we investigated a design methodology for implementing a SDN control plane capable of steering specific data flows toward the required VNF locations and achieving fully dynamic service chaining.
Initially, we adopted the Mininet emulation platform to provide a former practical example of the feasibility and degree of complexity of the two case studies. Whereas later on, we decided to propose a stateful approach by adopting a Mealy Finite State Machine to model control plane behavior to implement fully dynamic and adaptive service chaining. The feasibility of such approach was proved through the implementation of a proof-of-concept (PoC) on the Ericsson Cloud Lab environment, which was based on an OpenStack deployment.

The outcome of this work is twofold: on the one hand, we showed that our approach allows space and time diversity in service chaining, with a higher degree of dynamism and flexibility with respect to conventional hardware based architectures. We believe that, considering both legacy forwarding and flexible traffic steering, it is possible to simplify the SDN controller design and successfully achieve dynamic service chaining. On the other one, the stateful approach proved to be feasible in a real life production environment: a software-controlled network is indeed capable to self-adapt by dynamically reconfiguring the VNF forwarding graph in order to guarantee the agreed SLAs to the user. Last, but not least, investigate how to create the technical conditions for accelerating the adoption of SDN and NFV in order to produce concrete socio-economic impacts is key.

We successfully accomplished the main goal of this work, and we believe that our stateful approach could be adopted to develop and analyze the performance and scalability of more complex cloud network scenario.To the best of our knowledge, at the time of such investigation, this was the first attempt that leverages a finite state machine as approach to design and implement the control plane behavior.

- **management plane:** northbound interface in SDN-NFV architecture is a topic yet to be explored, since a well defined, common interface does not exist.

  The main goal of this work was to propose an intent-based approach to the definition of the North Bound Interface (NBI) between the Virtualized Infrastructure Manager (VIM) and the higher management and orchestration layers defined in the ETSI Management and Orchestration (MANO) framework. The focus was, in particular, on the network infrastructure programmability functions of the VIM, assuming a general network scenario where multiple SDN domains are interconnected by non-SDN domains.
  To this purpose, we decided to leverage the well known JSON format as a possible implementation of the interface between the VIM and high level orchestrator. To show the feasibility of such approach, we designed a PoC aimed at providing adaptive quality of service (QoS) enforcement to a multi-tenant NFV scenario. The focus was mainly on performance and scalability aspects of the proposed NBI.

  The main outcome of our first validation is that such approach proved to be successful, since we are capable of creating, updating, deleting and flushing the forwarding paths according to the intent specified via the NBI. Moreover, the measured responsiveness of our implementation shows promising scalability potentials.

  This is an early stage proposal of a possibile way of how to implement a vendor-agnostic, technology-independent intent NBI between the SDN control platform and higher level orchestrator. Such interface was designed specifically to orchestrate dynamic service chaining of VNFs, but the approach is general enough to be adopted also in non-OpenFlow infrastructure (e.g., Internet of Things domain).

5.2 Future Work

In spite of successful accomplishments, we believe many interesting aspects of the overall work are left open as a follow up. Few examples of such interesting aspects in the field of data, control and management plane are given in the following.
Data plane: virtual network infrastructure implemented by OpenStack, together with the hosting hardware maximum capacity, can pose some limitations to the network performance. 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, in order to provide useful insights on how to proper dimension networks and virtual appliances placement.

Control plane: suitable methodologies that simplify SDN controller design are yet to be explored. We believe that our stateful SDN approach could be extended to develop and analyze more complex cloud network scenario, where VNF instances can be dynamically created, moved, and terminated to cope with the current workload.

Management plane: how to implement well defined interfaces between SDN control platform and higher orchestration layer is still unclear, as it is still unclear which could be a standard testing methodology or any reference performance parameters to be used. We believe that our intent North Bound Interface approach can be improved and extended to support other application requirements, besides other complex communication scenario, such as, for example, inter-domain communication. Moreover, we intend to: i) extend the implementation of our proposed Virtualized Infrastructure Manager to further investigate the performance of our proposed approach; ii) investigate the comparison of our approach with possible existing alternative approches.

List of Tables

List of Figures

Bibliography

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] "Etsi nfv white paper #1." https://github.com/mininet/mininet/wiki/Mininet-VM-Images.

[5] "Verizon sdn-nfv reference architecture." http://innovation.verizon.com/content/dam/vic/PDF/Verizon_SDN-NFV_Reference_Architecture.pdf.

[6] T. E. T. S. Institute, "Network Functions Virtualization (NFV); Management and Orchestration," http://www.etsi.org/technologies-clusters/technologies/nfv, ETSI GS NFV-MAN 001 V1.1.1, December 2014.

[7] The Open Networking Foundation, "Intent NBI - Definition and Principles," https://www.opennetworking.org/sdn-resources/technical-library, ONF TR-523, October 2016.

[8] 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.

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

[10] O. Committee *et al.*, "Software-defined networking: The new norm for networks," *Open Networking Foundation*, 2012.

[11] G. ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.

[12] 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.

[13] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, 2015.

[14] A. Al-Shabibi and L. Peterson, "Cord: Central office re-architected as a datacenter," *OpenStack Summit*, 2015.

[15] 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.

[16] "Openstack project website," http://www.openstack.org.

[17] F. Sans and E. Gamess, "Analytical performance evaluation of different switch solutions," *Journal of Computer Networks and Communications*, vol. 2013, 2013.

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

[19] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications.* IEEE, 2014, pp. 1285–1293.

[20] P. Rad, R. V. Boppana, P. Lama, G. Berman, and M. Jamshidi, "Low-latency software defined network for high performance clouds," in *System of Systems Engineering Conference (SoSE), 2015 10th.* IEEE, 2015, pp. 486–491.

[21] S. Oechsner and A. Ripke, "Flexible support of vnf placement functions in openstack," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–6.

[22] M. Banikazemi *et al.*, "Openstack networking: It's time to talk performance *OpenStack Summit 2015*, vancouver, canada,," May 2015.

[23] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of network virtualization in cloud computing infrastructures: The openstack case," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 132–137.

[24] ——, "Performance of multi-tenant virtual networks in openstack-based cloud infrastructures," in *2014 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2014, pp. 81–85.

[25] 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*. ACM, 2012, pp. 253–264.

[26] P. Bellavista, F. Callegati, W. Cerroni, C. Contoli, A. Corradi, L. Foschini, A. Pernafini, and G. Santandrea, "Virtual network function embedding in real cloud environments," *Computer Networks*, vol. 93, pp. 506–517, 2015.

[27] "The linux foundation, linux bridge," http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge, November 2009.

[28] J. Yu, "Performance evaluation on linux bridge," in *Telecommunications System Management Conference*, 2004.

[29] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

[30] 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.

[31] G. Santandrea, "Show my network state project website, 2014," https://sites.google.com/site/showmynetworkstate.

[32] "Rude & crude: Real-time udp data emitter & collector for rude." http://sourceforge.net/projects/rude/.

[33] "iperf3: A tcp, udp, and sctp network bandwidth measurement tool." https://github.com/esnet/iperf.

[34] "ndpi: Open and extensible lgplv3 deep packet inspection library." http://www.ntop.org/products/ndpi/.

[35] N. ISG, "Network functions virtualisation (nfv)-network operator perspectives on industry progress,?" ETSI, Tech. Rep, Tech. Rep., 2013.

[36] G. Gibb, H. Zeng, and N. McKeown, "Initial thoughts on custom network processing via waypoint services," in *WISH-3rd Workshop on Infrastructures for Software/Hardware co-design, CGO*, 2011.

[37] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using sdn," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 27–38, 2013.

[38] F. Risso, A. Manzalini, and M. Nemirovsky, "Some controversial opinions on software-defined data plane services," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for.* IEEE, 2013, pp. 1–7.

[39] J. Hwang, K. Ramakrishnan, and T. Wood, "Netvm: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.

[40] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2015.

[41] "Mininet vm images." https://github.com/mininet/mininet/wiki/Mininet-VM-Images.

[42] F. López-Rodríguez and D. R. Campelo, "A robust sdn network architecture for service providers," in *2014 IEEE Global Communications Conference.* IEEE, 2014, pp. 1903–1908.

[43] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 19–24.

[44] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: simplifying sdn programming using algorithmic policies," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 87–98, 2013.

[45] M. Monaco, O. Michel, and E. Keller, "Applying operating system principles to sdn controller design," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, 2013, p. 2.

[46] J. Yao, Z. Wang, X. Yin, X. Shiyz, and J. Wu, "Formal modeling and systematic black-box testing of sdn data plane," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 179–190.

[47] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[48] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[49] "Ericsson cloud manager," http://www.ericsson.com/ourportfolio/products/cloud-manager?nav=productcategory008, June 2016.

[50] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Sdn controller design for dynamic chaining of virtual network functions," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, 2015, pp. 25–30.

[51] ETSI Industry Specification Group (ISG) NFV, "Etsi gs nfv 001 v1.1.1: Network function virtualization. use cases," October 2013.

[52] "Ericsson opens a cloud lab in italy for faster co-creation and innovation;," http://www.ericsson.com/news/1923781, June 2015.

[53] "Openstack: Open source software for creating private and public clouds;," http://www.openstack.org, June 2016.

[54] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple network management protocol (snmp)," Tech. Rep., 1990.

[55] Internet Engineering Task Force, https://tools.ietf.org/html/rfc7426.

[56] J. Garay, J. Matias, J. Unzilla, and E. Jacob, "Service description in the nfv revolution: Trends, challenges and a way forward," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 68–74, 2016.

[57] D. Farinacci *et al.*, "Genericrouting encapsulation (gre)," http://www.rfc-editor.org/info/rfc2784, IETF RFC 2784, March 2000.

[58] M. Mahalingam *et al.*, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," http://www.rfc-editor.org/info/rfc7348, IETF RFC 7348, August 2014.

[59] P. Quinn and U. Elzur, "Network service header," https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh, IETF Internet-Draft draft-ietf-sfc-nsh-10, September 2016.

[60] "Project boulder: Intent nbi," http://opensourcesdn.org/projects/project-boulder-intent-northbound-interface-nbi/.

[61] ONOS: Open Network Operating System, http://onosproject.org.

[62] ONOS Intent Framework, https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[63] The OpenDaylight Platform, https://www.opendaylight.org/.

[64] "OpenDaylight Network Intent Composition Project," https://wiki.opendaylight.org/view/Network_Intent_Composition:Main.

[65] "OpenDaylight Service Function Chaining Project," https://wiki.opendaylight.org/view/ServiceFunctionChaining:Main.

[66] "Mininet," http://mininet.org.

Acknowledgments

First of all, I would like to express my enormous gratitude to my PhD advisor, Franco Callegati, and co-advisor, Walter Cerroni: thank you so much for this wonderful opportunity,and the trust, teaching, advices, support and funding you gave me during all PhD. Without you, such wonderful 3 years could not have been possible. I would also like to thank you not only from a professional point of view but also from a personal one: thank you both for being the such wonderful people you are, I feel really lucky.

I would like to thank my thesis committee: Flavio Esposito, PhD, and Stuart Clayman, PhD, for their precious comments that encouraged me to enhance my thesis.

A special thanks goes to those who shared with me these 3 years (or at least an important part of it) on the field, sharing happiness, hard work, anxiety, and unforgettable business trip that distinguish a PhD student life: again, Franco Callegati and Walter Cerroni, Giuliano Santandrea, Francesco Foresta, Giuseppe Portaluri e Francesco Lucrezia.

I also would like to thank the Esposito-Schwetye family: thanks for the great experience, I felt like if I were a member of your wonderful family. Thanks to my lab mates of the NetLab at Bologna: Gianluca Davoli, Federico Tonini, Bahare Khorsandi and Andrea Melis.

Last, but for sure not least, I would like to thank all my family, mamma Carla, babbone Secondo and my sister Daniela: thanks for your support and the enthusiasm you always showed me. Thanks to my boyfriend Andrea Bessi and to his family: Roberta, Adriano and Lucianina; thanks to my best friend Meris Michelacci and to my friends Andrea Arnoffi and Federico Foschini.