BRUNO MEDEIROS DE BARROS

SECURITY ARCHITECTURE FOR NETWORK VIRTUALIZATION IN CLOUD COMPUTING

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Sao Paulo
2016

BRUNO MEDEIROS DE BARROS

SECURITY ARCHITECTURE FOR NETWORK VIRTUALIZATION IN CLOUD COMPUTING

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Área de Concentração:

Engenharia de Computação

Orientador:

Prof. Dr. Marcos Antonio Simplicio Junior

Sao Paulo
2016

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Medeiros de Barros, Bruno
Security Architecture for Network Virtualization in Cloud Computing / B. Medeiros de Barros -- versão corr. -- São Paulo, 2016.
154 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Computação em Nuvem 2.Virtualização de Redes 3.Segurança I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

”Problems cannot be solved by

the level of awareness that created them”

Albert Einstein

AGRADECIMENTOS

Em primeiro lugar eu quero expressar minha gratidão à minha família, cujo suporte foi fundamental para a conclusão deste trabalho. Em especial, eu agradeço minha mãe, Rosângela Medeiros, pelo exemplo de dedicação, coragem e perseverança; e meu avô, Astor Medeiros, por todas as vezes em que disse estar orgulhoso de mim. Também quero agradecer à Amanda Marques, mulher da minha vida, e sua família. Obrigado pelo apoio incondicional, e por encher meus dias de inspiração e felicidade.

Além disso, quero agradecer a meu orientador, Professor Marcos Antonio Simplicio Jr., por tudo que me ensinou na Academia e na vida. Obrigado por acreditar em mim e por me proporcionar a oportunidade de melhorar a cada dia. Também quero expressar minha gratidão à Fundação Para o Desenvolvimento Tecnológico da Engenharia (FDTE) e à Ericsson Research por financiarem meu trabalho.

Por último, mas não menos importante, manifesto minha gratidão a todos os membros do Laboratório de Arquitetura e Redes de Computadores (LARC), em especial à Professora Tereza Cristina, amiga e coorientadora, e a todos os membros da Ericsson Research com quem tive a oportunidade de trabalhar, em especial à Mats Näslund e à Makan Pourzandi. Obrigado pelas conversas agradáveis e inspiradoras.

Meus sinceros agradecimentos a todos vocês.

ACKNOWLEDGEMENTS

First, I want to express my gratitude to my family, whose support was essential for concluding this work. In special, I thank my mother, Rosângela Medeiros, for the example of dedication, courage and perseverance; and my grandfather, for every time he said he was proud of me. I also want to express my gratitude to Amanda Marques, the woman of my life, and her family. Thank you for your unconditional support, and for fill my days with inspiration and happiness.

Furthermore, I want to thank my adviser, Professor Marcos Antonio Simplicio Jr., for everything he has taught me in Academia and in life. Thank you for believing me, and for providing me with the opportunity to become better every day. I also want to express my gratitude to the FDTE (Fundação Para o Desenvolvimento Tecnológico da Engenharia – Foundation for the Technological Development of the Engineering) and Ericsson Research for sponsoring my work.

Last but not least, I express my gratitude to all members of the Laboratory of Computer Network and Architecture (LARC), in special to Professor Tereza Carvalho, my friend and co-adviser, and to all members of Ericsson Research with whom I have worked with, in special to Mats Näslund and to Makan Pourzandi. Thank you for the pleasant and inspiring conversations.

I am sincerely grateful to all of you.

RESUMO

Virtualização de redes é uma área de pesquisa que tem ganho bastante atenção nos últimos anos, motivada pela necessidade de se implementar sistemas de comunicação seguros e de alta performance em infraestruturas de computação em nuvem. Em particular, os esforços de pesquisa nesta área têm levado ao desenvolvimento de soluções de segurança que visam aprimorar o isolamento entre os múltiplos inquilinos de sistemas de computação em nuvem públicos, uma demanda reconhecidamente crítica tanto pela comunidade acadêmica quanto pela indústria de tecnologia. Mais recentemente, o advento das Redes Definidas por Software (do inglês *Software-Defined Networks – SDN*) e da Virtualização de Funções de Rede (do inglês *Network Function Virtualization – NFV*) introduziu novos conceitos e técnicas que podem ser utilizadas para abordar questões de isolamento de redes virtualizadas em sistemas de computação em nuvem com múltiplos inquilinos, enquanto aprimoram a capacidade de gerenciamento e a flexibilidade de suas redes. Similarmente, tecnologias de virtualização assistida por hardware como *Single Root I/O Virtualization – SR-IOV* permitem a implementação do isolamento de recursos de hardware, melhorando o desempenho de redes físicas e virtualizadas. Com o intuito de implementar uma solução de virtualização de redes que aborda de maneira eficiente o problema de isolamento entre múltiplos inquilinos, nós apresentamos três estratégias complementares para o isolamento de recursos de rede em sistemas computação em nuvem. As estratégias apresentadas são então aplicadas na avaliação de arquiteturas de virtualização de rede existentes, revelando lacunas de segurança associadas às tecnologias utilizadas atualmente, e abrindo caminho para o desenvolvimento de novas soluções. Nós então propomos uma arquitetura de segurança que utiliza as estratégias apresentadas, e tecnologias como SDN, NFV e SR-IOV, para implementar domínios de rede seguros. As análises teórica e experimental da arquitetura proposta mostram considerável redução das superfícies de ataque em redes virtualizadas, com um pequeno impacto sobre o desempenho da comunicação entre máquinas virtuais de inquilinos da nuvem.

Palavras-chave: Computação em Nuvem. Virtualização de Redes. Segurança.

ABSTRACT

Network virtualization has been a quite active research area in the last years, aiming to tackle the increasing demand for high performance and secure communication in cloud infrastructures. In special, such research efforts have led to security solutions focused on improving isolation among multiple tenant of public clouds, an issue recognized as critical both by the academic community and by the technology Industry. More recently, the advent of Software-Defined Networks (SDN) and of Network Function Virtualization (NFV) introduced new concepts and techniques for addressing issues related to the isolation of network resources in multi-tenant clouds while improving network manageability and flexibility. Similarly, hardware technologies such as Single Root I/O Virtualization (SR-IOV) enable network isolation in the hardware level while improving performance in physical and virtual networks. Aiming to provide a cloud network environment that efficiently tackles multi-tenant isolation, we present three complementary strategies for addressing the isolation of resources in cloud networks. These strategies are then applied in the evaluation of existing network virtualization architectures, exposing the security gaps associated to current technologies, and paving the path for novel solutions. We then propose a security architecture that builds upon the strategies presented, as well as upon SDN, NFV and SR-IOV technologies, to implement secure cloud network domains. The theoretical and experimental analyses of the resulting architecture show a considerable reduction of the attack surface in tenant networks, with a small impact over tenants' intra-domain and inter-domain communication performance.

Keywords: Cloud Computing. Network Virtualization. Security.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

API Application Programming Interface

ARP Address Resolution Protocol

ASON Automatically Switched Optical Network

CDN Content Delivery Network

CLI Command-line Interface

CSP Cloud Service Provider

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DDoS Distributed Denial of Service

DMZ Demilitarized Zone

DoS Denial of Service

DPDK Data Plane Development Kit

DPI Deep Packet Inspector

GMPLS Generalized Multiprotocol Label Switching

GRE Generic Routing Encapsulation

HAV Hardware-Assisted Virtualization

HIP Host Identity Protocol

IaaS Infrastructure-as-a-Service

IDS Intrusion Detection System

IPsec Internet Protocol Security

IPS Intrusion Prevention System

L1VPN Layer 1 Virtual Private Network

L2TP Layer Two Tunneling Protocol

L2VPN Layer 2 Virtual Private Network

L3VPN Layer 3 Virtual Private Network

MAC Media Access Control

MPLS Multiprotocol Label Switching

NFV Network Function Virtualization

NIC Network Interface Card

NIST National Institute of Standards and Technologies

NVGRE Network Virtualization using Generic Routing Encapsulation

ODL OpenDaylight

ONF Open Networking Foundation

OVSDB Open vSwitch Database Management Protocol

OVS Open vSwitch

PCI-SIG Peripheral Component Interconnect Special Interest Group

PCIe Peripheral Component Interconnect Express

PPTP Point-to-Point Tunneling Protocol

RIP Routing Information Protocol

SAL Service Abstraction Layer

SDN Software-Defined Network

SONET Synchronous Optical Network

SPoF Single Point of Failure

SR-IOV Single Root I/O Virtualization

STT Stateless Transport Tunneling

TLS Transport Layer Security

TNC Tenant Network Controller

TND Tenant Network Domain

TSO TCP Segmentation Offload

USP Universidade de São Paulo

VEPA Virtual Ethernet Port Aggregation

VF Virtual Function

VLAN Virtual Private Network

VMDq Virtual Machine Device Queues

VMM Virtual Machine Monitor

VM Virtual Machine

VNF Virtual Network Function

VNI VXLAN Network Identifier

VPC Virtual Private Cloud

VPN Virtual Private Network

vSwitch Virtual Switch

VTEP VXLAN Tunnel Endpoint

VTP VLAN Trunking Protocol

VXLAN Virtual Extensible Local Area Network

1 INTRODUCTION

Cloud computing shaped a new way of providing and consuming computational resources, aggregating unprecedented agility, flexibility and scalability to IT infrastructures. The numerous benefits provided by cloud computing (ARMBRUST et al., 2010) encouraged major players such as Google, Amazon, HP, and other technology companies to apply cloud computing for providing innovative services and products, as well as for running their own business (MARSTON et al., 2011).

The growing adoption of cloud computing has, however, contributed to the emergence of numerous security concerns (SUBASHINI; KAVITHA, 2011; KAUFMAN, 2009). In particular, the adoption of public cloud services has grown significantly in the last decade (NAG et al., 2016), along with unique security challenges resulted from its multi-tenancy characteristics (REN; WANG; WANG, 2012). Indeed, and despite the number of solutions proposed to address security issues in cloud computing (GONZALEZ et al., 2012), security still representing the major barrier for the adoption of cloud computing(AVRAM, 2014; Gartner, 2016).

Virtualization technologies are key elements for the analysis of such concerns, since they are responsible for implementing the isolation of computing resources in multi-tenant clouds. Furthermore, computer networks play an important role in cloud security, enabling the communication among multiple components of cloud platforms, the implementation of tenants' virtual networks, and the user access to cloud services. As such, network virtualization requires efficient security design and implementation,

since they may directly impact the security of the whole cloud system.

1.1 Motivation

The importance of cloud networking security has led to quite a few security solutions in the literature that apply network virtualization techniques to implement multi-tenant isolation. These solutions usually focus on the isolation of logical data paths by means of packet tagging techniques and tunneling protocols, providing the abstraction of tenant virtual networks that can be managed separately by the cloud orchestration system. However, the lack of isolation of virtual and physical network appliances applied to implement virtual networks uncover inherited attack surfaces, introducing numerous security vulnerabilities. Given to the limited scope of existing security solutions, as well as to the lack of interoperability among them, the conception of a comprehensive security architecture for cloud networking becomes very challenging.

However, novel opportunities emerged from the development and consolidation of technologies such as Software-Defined Network (SDN) (GREENE, 2009; JAMMAL et al., 2014), Network Function Virtualization (NFV) (ETSI, 2012), and Hardware-Assisted Virtualization (HAV) (SARATHY; NARAYAN; MIKKILINENI, 2010; KUTCH, 2011). These technologies provide, respectively, the abilities to: separate network control and data planes, enabling the centralization of network control; to virtualize network security functions inside physical and virtual standard appliances; and to enhance the performance of network virtualization. In other works, these technologies pave new paths to reach the isolation of network resources in the cloud systems, providing efficient control mechanisms to improve networking security against multi-tenant threats. This work is motivated by the opportunity to apply innovative technologies and virtualization strategies to enhance cloud network security by designing, implementing and evaluating a network virtualization architecture that can efficiently isolate virtual networks in multi-tenant clouds.

1.2 Goals

The main goal of this work is to design a secure network virtualization architecture that provides efficient isolation of network resources, mitigating security risks inherited by multi-tenant clouds. These security risks should be provided from a comprehensive analysis of the vulnerabilities associated with current network virtualization architectures, resulting on innovative proposals for addressing network isolation in cloud computing deployments. In addition, this work is intended to investigate the benefits and drawbacks regarding the application of innovative network virtualization technologies such as SDN, NFV and HAV to improve the security of multi-tenant cloud systems. Finally, the resulting proposal can be integrated to actual cloud deployments and evaluated according to security and additional requirements associated to the performance and manageability of the cloud.

1.3 Method

This work follows the hypothetical-deductive method, using relevant references in the specialized literature as the basis for defining the problem, specifying a solution hypotheses and then evaluating that hypothesis by means of a prototype implementation. The development of this work starts in a comprehensive research of existing network virtualization technologies, and how they have been applied to the implementation of virtual network deployments. In particular, our study explores innovative technologies and techniques provided by SDN, NFV, and HAV, which are applied on the design and implementation of the proposed architecture.

Next, from the evaluation of security vulnerabilities introduced by the lack of isolation in network virtualization deployments, we: (1) propose an innovative strategy for addressing multi-tenant isolation from a network resources' perspective; (2) evaluate the ability of current network virtualization architectures to provide efficient isola-

tion of virtual networks; and (3) identify the gaps and conceptual influences from the evaluated architectures that should be considered in our proposal.

The proposed architecture then applies the most promising technologies to address the isolation and additional requirements previously identified. Finally, the solution is specified and implemented in the form of a prototype, which can then be evaluated on a real cloud deployment considering relevant metrics, among them security, performance, scalability, and maintainability of cloud networks.

1.4 Contributions

The main contribution of this research is an architectural solution that enables to address relevant security threats on cloud networking, employing innovative concepts and technologies on network virtualization for this purpose. As additional contributions to the literature, it can be cited: (1) a comprehensive research of existing network virtualization technologies and projects that can be applied to enhance the security of cloud network deployments; (2) an evaluation of security vulnerabilities originated from shared network resources in multi-tenant cloud systems, and a resulting strategy for addressing the efficient isolation of cloud network resources; and (3) a research of current network virtualization architecture and their ability to implement efficient isolation of virtual networks.

1.5 Document Organization

This document is organized in chapters. Chapter 2 discusses network virtualization in cloud computing, providing an overview of the concepts and technologies hereby applied. Chapter 3 analyzes security issues in multi-tenant clouds, presenting three isolation strategies that can enhance the security of tenant networks, and analyzing how these isolation strategies are applied by current network virtualization architectures.

Chapter 4 presents the concept of isolated network domains, building upon the gaps and conceptual influences from existing network virtualization architectures in multi-tenant data centers. Chapter 5 describes the implementation of the proposed network virtualization architecture, whereas Chapter 6 present the evaluation our proposal. Finally, Chapter 7 presents our final considerations, contributions and future works.

2 BACKGROUND ON CLOUD NETWORK VIRTUALIZATION

This chapter presents a conceptual and technical background for understanding the importance of network virtualization in multi-tenant data centers and how it can be applied to cloud computing systems. We analyze different existing virtualization mechanisms, technologies and paradigms, and explain how they can be used to implement network virtualization in cloud systems.

2.1 Network Virtualization and Cloud Computing

Cloud computing has ushered the information technology (IT) field and service providers into a new era, redefining how computational resources and services are delivered and consumed. With cloud computing, distinct and distributed physical resources such as computing power and storage space can be acquired and used in an on-demand basis, empowering applications with scalability and elasticity at low cost. This allows the creation of different service models, generally classified as (MELL; GRANCE, 2011): Infrastructure-as-a-Service (IaaS), which consists in providing only fundamental computing resources such as processing, storage and networks; Platform-as-a-Service (PaaS), in which a development platform with the required tools (languages, libraries, etc.) is provided to tenants; and Software-as-a-Service (SaaS), in which the consumer uses the applications running on the cloud infrastructure.

To actually provide cost reductions, the cloud needs to take advantage of

economies of scale, and one key technology for doing so is resource virtualization. After all, virtualization allows the creation of a logical abstraction layer on top of the pool of physical resources, thereby enabling a programmatic approach to allocate resources wherever necessary while hiding the complexities involved in their management. The potential result is efficient resource utilization, easier manageability, on-demand and programmatic resource instantiation, and resource isolation for better control, accounting and availability.

Networking is a critical resource for cloud computing environments, since it connects the various distributed and virtualized components that compose the cloud, such as servers, storage elements, appliances and applications. For example, it is the network that allows aggregation of physical servers, efficient virtual machine (VM) migration, and remote connection to storage systems, effectively creating the perception of a large, monolithic resource pool. Furthermore, networks enable cloud-based services to be delivered to end users.

Whereas virtualization technologies have reached considerable maturity and are today applied to almost every component in cloud systems, the virtualization of network resources connecting those cloud components has not received as much attention. Actually, network virtualization techniques applied to cloud computing are somewhat restricted to data center virtualization, fulfilling a limited set of multi-tenancy and scalability requirements. Limited virtualization makes cloud networks increasingly complex, which in turn may hinder the scalability of the cloud itself as it evolves to provide new services with a wide variety of requirements. Fortunately, the emergence of network paradigms such as SDN and NFV has created a new set of virtualization techniques that may be employed for improving the development of cloud networks and, consequently, of the cloud itself.

2.1.1 Mechanisms for Network Virtualization

To understand the mechanisms that can implement network virtualization, first we need to understand which network layers and related resources can be virtualized in a network (CHOWDHURY; BOUTABA, 2009). In terms of resources, networks are basically composed by network interface cards (NICs), which are connected via the physical network layer (L1) and logically linked to a layer 2 (L2) network via a switch. These layer 2 networks can be connected through routers to form a layer 3 (L3) network, which in turn can be connected via routers to compose the Internet. Each of these network components — NIC, L1 network, L2 network, L2 switch, L3 networks, and L3 routers — can be virtualized. However, there are multiple (and often competing) mechanisms for virtualizing these resources:

- *Virtualization of NICs*: Every networked system is composed by at least one NIC. In virtualized environments with multiple connected VMs, it becomes, thus, necessary to provide every VM with its own virtual NIC (vNIC). The most common approach to fulfill this need is to use a hypervisor[1], such as Xen (The Xen Project, 2016) or KVM (KVM, 2016a), which is able to emulate vNICs to the VMs under its responsibility. The vNICs can be connected via a virtual switch (vSwitch), just like physical NICs can be connected via a physical switch to compose layer 2 networks. In addition, vNICs can be connected to server's physical NIC (pNIC) using vSwitches. This NIC virtualization strategy has benefits such as transparency and simplicity and, thus, is typically suggested by software vendors. There is also an alternative design proposed by pNIC (chip) vendors, which is to virtualize NIC ports using techniques such as SR-IOV (KUTCH, 2011) and Intel's Virtual Machine Device Queues (VMDq) (Intel, 2016b). These approaches enable the execution of VMs' packet forwarding functions directly

[1] A hypervisor, also known as a Virtual Machine Monitor (VMM), is a piece of software that sits between the hardware and one or more virtual machines that it supports (PORTNOY, 2012).

inside pNICs, potentially providing better performance (as it eliminates intermediary software) and resource isolation (as the traffic does not go through a shared vSwitch). Data plane packet processing technologies such as Intel Data Plane Development Kit (DPDK) (Intel, 2016a; DPDK, 2016) have also leveraged packet processing features of Network Interface Cards (NICs). Nevertheless, the number of ports in a typical switch is limited, and possibly lower than the number of physical machines that need to be connected to an L2 network. Therefore, several layers of L2 switches need to be connected to address network scalability requirements. To solve this issue, IEEE Bridge Port Extension standard 802.1BR (IEEE, 2012b) proposes physical bridges with an extensible number of ports, using physical or virtual port extenders.

- *Virtualization of L1 Networks*: Next-generation Synchronous Optical Networks (SONETs) (BERNSTEIN; RAJAGOPALAN; SAHA, 2003) technologies, along with network switching standards such as Generalized Multiprotocol Label Switching (GMPLS) (MANNIE; PAPADIMITRIOU, 2006) and Automatically Switched Optical Network (ASON) (JAJSZCZYK, 2005), enabled the emergence of Layer 1 Virtual Private Networks (L1VPNs) as a new service provisioning paradigm for enabling dynamic provisioning of logical topologies on top of physical transport networks (BENHADDOU; ALANQAR, 2007). L1VPN framework emerged from the need to extend L2/L3 packet-switching Virtual Private Network (VPN) concepts to advanced circuit-switching domains (TAKEDA, 2007). The main characteristic of L1VPNs is the ability to provide multi-service backbones on which customers can offer services with different payloads from any network layer (e.g., ATM, IP, TDM). This allows each service to have an independent address space, an independent Layer 1 resource view, independent policies, and, consequently, improved isolation. The L1VPN solution implements these features by assigning and managing clients' resources in a physical network (e.g.,

nodes, ports, and link capacities) according to their service needs. The similar purpose of virtualizing Layer 1 networks based on the SDN paradigm can be found in FlowVisor (SHERWOOD et al., 2009) and OpenVirtex (AL-SHABIBI et al., 2014) projects. FlowVisor and OpenVirtex are both network hypervisors that can create multiple virtual and programmable networks on top of a single physical infrastructure, enabling multiple clients to operate over different network slices using different SDN controllers and hence network applications and protocols.

- *Virtualization of L2 Switches*: Software switches are typically implemented by VMMs for providing layer 2 connectivity to controlled VMs. Examples of software implementations of L2 switches in VMMs can be found in VirtualBox (VirtualBox, 2016), VMware vSphere (VMware, 2016), Xen (Xen, 2016b) and KVM (KVM, 2016b). On Linux-based VMMs, the ability to bridge network traffic between controlled VMs and with the outside world is commonly implemented using built-in L2 switches, i.e., the Linux bridge (ROSEN, 2014). It is also possible to use OVS, another software switch built for virtualized environments, which is capable of exporting an external interface for fine-grained control of configuration state and forwarding behavior (PFAFF et al., 2009). OVS is an open source software that can operate both as a soft switch running within VMMs, or as the control stack for switching silicon (Open vSwitch, 2016c). In addition, physical switch vendors are able to provide layer 2 connectivity functions for virtual machines running inside virtualized servers using Virtual Ethernet Port Aggregation (VEPA) (IEEE, 2012a). VEPA enables the configuration of virtual channels between virtualized servers and physical switches, steering all VMs' traffic to the latter; the result is connectivity between the VMs running in the same physical server and the rest of the virtualized infrastructure. This approach not only frees up server resources, but also provides better visibility and control over the traffic between any pair of VMs.

- *Virtualization of L2 Networks*: In a multi-tenant data center, VMs in a single physical machine may belong to different clients and, thus, need to be in different Virtual Private Networks (VLANs) (IEEE, 2006). VLANs implement packet tagging, allowing L2 devices to isolate the clients' traffic in different logical networks, so different tenants can use the same addressing space. The most widely used VLAN example is probably the IEEE[2] 802.1Q standard (IEEE, 2014), which defines a VLAN tagging system for Ethernet frames and the corresponding procedures to be used by bridges and switches for handling such frames. Frames from the same VLAN carry a common VLAN ID in their MAC headers, and VLAN-enabled switches use both the VLAN ID and the destination MAC address to forward frames. Multiple VLAN on multiple switches can also be connected together using VLAN Trunking Protocol (VTP) (CISCO, 2014); this protocol, originally proposed by Cisco, propagates the definition of VLANs to the whole local area network and carries VLAN information to all the switches in a VTP domain. Furthermore, L2 network virtualization can be implemented through Layer 2 Virtual Private Network (L2VPN) (ROSEN; ANDERSSON, 2006; SERBEST; AUGUSTYN, 2006). L2VPNs provide end-to-end L2 connection between distributed sites by transporting L2 frames (typically Ethernet, but ATM and Frame Relay are also supported) inside pseudo wires (PATE; BRYANT, 2005). Pseudo wires are intended to provide the minimum functions to emulate a wired connection for network and telecommunication services. These functions include encapsulating service-specific bit streams arriving at switches ingress ports and carrying them across IP or MPLS tunnels, managing timing and ordering of frames. The Layer Two Tunneling Protocol (L2TP) (VALENCIA et al., 1999; PIGNATARO; MCGILL, 2009) is an example of encapsulating protocol applied to create L2VPN. There are two fundamentally different kinds of L2VPN

[2]The Institute of Electrical and Electronics Engineers (IEEE) is a professional association formed in 1963 to support educational and technical advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines.

services that a service provider could offer to a customer: Virtual Private Wire Service (VPWS), which supplies L2 point-to-point service; and Virtual Private LAN Service (VPLS) (MADSEN, 2005), which emulates LAN service across a Wide Area Network (WAN).

- *Virtualization of L3 Routers*: Network functions such as L2/L3 switches and routers can be implemented by combining different software-based modules able to run on physical and virtualized standard hardware. NFV (ETSI, 2012) provides the conceptual framework for developing and deploying virtual L3 switches and routers that can operate in both virtualized and underlying networks. It enables network functions to be dynamically instantiated or moved to different network locations, without adding new equipment to the physical infrastructure. Moreover, orchestration features provided by SDN and cloud computing technologies leverage NFV by providing a manageable and scalable architecture to deploy virtualized network functions (JAIN; PAUL, 2013). The development of virtual network functions with NFV has received considerable support from open source initiatives such as OPNFV (OPNFV, 2016), and the commercial interest in NFV routers is currently demonstrated by industry players such as Ericsson (Ericsson, 2015) and Brocade (Brocade, 2015). Layer 3 router virtualization can also be achieved by software-based router functions embedded in VMMs such as Xen (ANHALT; PRIMET, 2008) and KVM (ABENI et al., 2013), providing network layer services to controlled VMs. The same approach can be found in cloud orchestration solutions, such in OpenStack (OpenStack, 2016b) and Apache CloudStack (CloudStack, 2016b), which provide tenants' VMs with layer 3 connectivity. Finally, routing functions can also be implemented through virtual switches running custom L3 forwarding rules dynamically configured by SDN applications (ROTHENBERG et al., 2012).

- *Virtualization of L3 Networks*: When multi-tenant systems are extended to layer

3 networks, many virtualization approaches based on tunneling protocols can be applied. Tunneling protocols allow provisioning network services that are not natively supported by the underlying network. For instance, tunneling protocols allow a foreign protocol to run over a network that does not provide support for that particular protocol (e.g., IPv6-over-IPv4) (STEFFANN; BEIJNUM; REIN, 2013), and to provide corporate network addresses to remote users outside the corporate network, extending virtual networks across multiple sites. Tunneling protocols widely applied for network virtualization include: VXLAN (MAHALINGAM et al., 2014); Network Virtualization using Generic Routing Encapsulation (NVGRE) (WANG; GARG, 2015); and Stateless Transport Tunneling (STT) (DAVIE; GROSS, 2016). All these solutions ensure multi-tenancy support and efficient delivery within the underlying network by encapsulating Ethernet frames within layer 4 UDP packets (VXLAN) or within layer 3 IP packets (NVGRE and STT). Besides the encapsulation type, these solutions also present different design requirements. VXLAN's main purpose is to solve scalability issues in VLAN based solutions, such as limited address space. Although NVGRE protocol present similar scalability features when compared with VXLAN, it builds upon Generic Routing Encapsulation (GRE) (FARINACCI et al., 2000), a tunneling protocol widely deployed in existing hardware and software network stacks. One disadvantage, however, is that NVGRE does not use any standard transport protocol (TCP/UDP), so network functions that rely on these headers require custom implementations (YONG et al., 2016). In addition to addressing the aforementioned requirements for tunneling-based network virtualization (scalability, interoperability and multi-tenancy support), the STT protocol provides hardware acceleration by adding a stateless TCP header to take advantage of hardware-based TCP Segmentation Offload (TSO) (HUANG; BALDINE, 2012). Although STT packet resembles TCP/IP packets to the underlying network, it has a stateless operation (i.e., the TCP state is not maintained in the header) that

may cause an undesired behavior when STT packets pass through middle boxes that process the TCP header. Besides VXLAN, NVGRE and STT, other encapsulating protocols that are not multi-tenancy oriented are commonly applied to virtualize L3 networks in Layer 3 Virtual Private Networks (L3VPNs) (CARUGI; MCDYSAN, 2005; SUZUKI; CALLON, 2005). L3VPNs allows data to be carried between private network nodes distributed in different domains by using layer 3 protocols, such as IP and Multiprotocol Label Switching (MPLS) (GHEIN, 2006), in VPN backbones. Examples of encapsulating protocols commonly applied in L3VPNs are GRE (FARINACCI et al., 2000), Point-to-Point Tunneling Protocol (PPTP) (ZORN; PALL; HAMZEH, 1999), and security solutions such as IPsec (SEO; KENT, 2005).

These virtualization methods have been used by a wide range of network virtualization projects in academy and industry (for some illustrative examples, see (CHOWDHURY; BOUTABA, 2010)). The UCLP (BOUTABA; GOLAB; IRAQI, 2004; GRASA et al., 2008) project, for instance, provides a software abstraction of optical network resources, enabling the virtualization of the underlying L1 network and allowing users to divide light paths within single or across multiple domains. Layer 2 network virtualization projects such as VNET (SUNDARARAJ; DINDA, 2004) build upon L2TP to implement widespread VLANs that are agnostic to layer 3 protocols. As another example, the AGAVE project (BOUCADAIR et al., 2007) uses layer 3 network virtualization for providing support to multiple and coexistent Internet virtual infrastructures tailored to different end-to-end service requirements.

Figure 1 provides an overview of the analyzed network layers and resources suitable for network virtualization in cloud data centers. Likewise, Table 1 and Table 2 summarize the concepts, technologies and projects related to the virtualized layers and resources analyzed in this section.

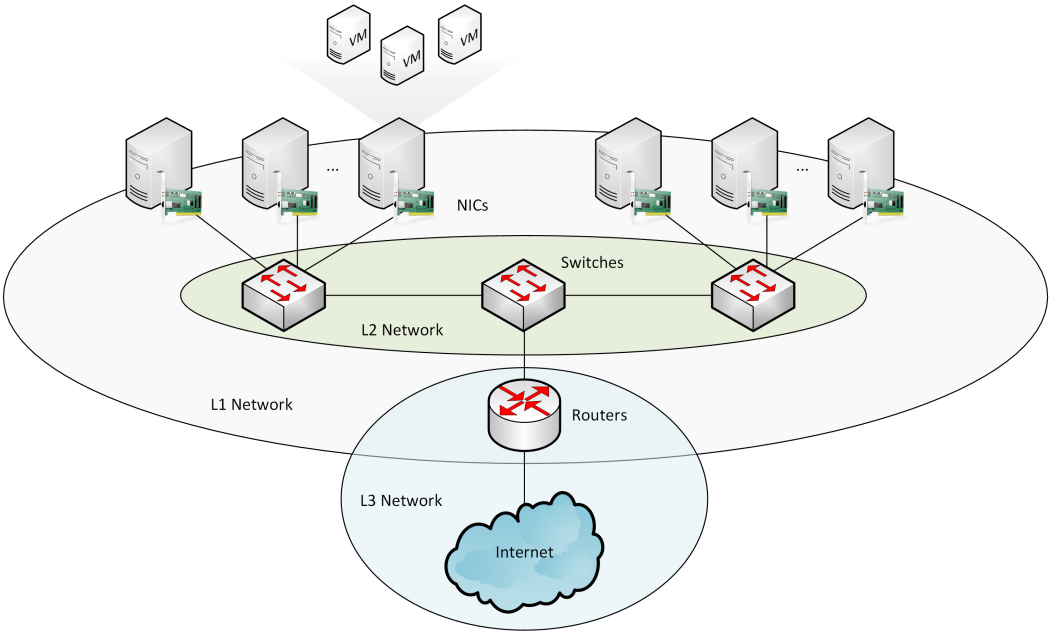

Figure 1: Mechanisms for network virtualization in cloud data centers.

Table 1: Concepts and technologies related to the virtualization of network layers.

Virtualized Layer	**Concepts**	**Projects and Technologies**
Physical Layer (L1)	L1VPN	UCLP (GRASA et al., 2008)
	SDN	Flowvisor (SHERWOOD et al., 2009), OpenVirtex (AL-SHABIBI et al., 2014)
Link Layer (L2)	Overlay	VLAN (IEEE, 2014), VTP (CISCO, 2014)
	Tunneling	L2TP (PIGNATARO; MCGILL, 2009)
	L2VPN	VNET (SUNDARARAJ; DINDA, 2004)
Network Layer (L3)	Overlay	VXLAN (MAHALINGAM et al., 2014), NVGRE (WANG; GARG, 2015), STT (DAVIE; GROSS, 2016)
	Tunneling	GRE (FARINACCI et al., 2000), IPsec (SEO; KENT, 2005)
	L3VPN	PPTP (ZORN; PALL; HAMZEH, 1999), AGAVE (BOUCADAIR et al., 2007)

Table 2: Concepts and technologies related to the virtualization of network resources.

Virtualized Resource	Concepts	Projects and Technologies
NIC	VMM Virtual NIC	VirtualBox (VirtualBox, 2016), vSphere (VMware, 2016), Xen (Xen, 2016b), KVM (KVM, 2016b)
	HAV	SR-IOV (KUTCH, 2011), VMDq (Intel, 2016b), DPDK (Intel, 2016a)
	Bridge Port Extension	802.1BR (IEEE, 2012b)
Switch	Virtual Switch Software	OVS (Open vSwitch, 2016c), Linux Bridge (ROSEN, 2014)
	VMM built-in Switch	VirtualBox (VirtualBox, 2016), vSphere (VMware, 2016), Xen (Xen, 2016b), KVM (KVM, 2016b)
	VEPA	802.1Qbg (IEEE, 2012a)
Router	NFV Router	OPNFV (OPNFV, 2016)
	SDN L3 Switch	OVS (Open vSwitch, 2016c)
	VMM built-in Router	Xen (ANHALT; PRIMET, 2008), KVM (ABENI et al., 2013)
	Cloud Router Service	OpenStack (OpenStack, 2016b), CloudStack (CloudStack, 2016b)

The technologies described in this section are building blocks for multiple network virtualization architecture, as will be explored in Chapter 3.2. Therefore, these technologies may considerably impact the security of such architectures, enhancing isolation or threatening virtual networks. In the next sections, we explore with more details what we consider key technologies for the implementation of the next generation network virtualization architectures.

2.1.2 Hardware Assisted Virtualization and SR-IOV

Virtualization technologies emerged mainly to optimize hardware utilization and to provide cost reductions, since they allow a more rational utilization of shared infrastructures. Today, virtualization has proven its benefits for solving bigger problems such as physical space in the data center, power/cooling costs, and resource management. Nonetheless, virtualization technologies introduce computational overheads and security vulnerabilities in shared environments. Especially regarding security issues, virtualization software introduces a new attack surface to the physical servers' network stack. There has been significant effort in the computer technology industry to tackle these issues, and hardware-assisted virtualization (HAV) technologies has been proved a prominent path to address the reduction of the software virtualization attack surfaces, as well as the security versus performance trade-offs. These attributes summarize the relevance of HAV technologies for this work.

HAV technologies consist in hardware-based implementation of virtualization mechanisms that provide high performance, secure and reliable shared computer resources. Hardware-embedded virtualization mechanisms started with processors, and was latter extended to other resources such as storage and networks. In its modern form, the application of HAV to networking is strongly related to the concept of I/O virtualization, whose main requirements are: to provide the same isolation that can be found when environments are running on separated physical machines; to provide scalability to support the number of VMs necessary to take advantage of idle resources; and to provide nearly native performance for I/O operations.

Despite its benefits, software-based mechanisms for I/O virtualization also have some drawbacks. Existing solutions are commonly based on a software-based virtualization layer, managed by the Virtual Machine Manager (VMM), which intercepts the traffic issued between custom guest drivers and I/O devices. These mechanisms provide only a subset of the total functionality provided by physical hardware and may

not have the ability to take advantage of advanced capabilities provided by the device. CPU overhead may also be required by the VMM to implement a virtual software-based switch for routing packets to/from the appropriate VM, reducing the maximum throughput on I/O devices.

To minimize such drawbacks, hardware vendors developed HAV technologies focused on I/O and network virtualization. Such technologies aim to directly expose the hardware to the guest OS and to allow native device drivers to be run, bypassing the VMM virtualization layer. The usual approach for directly assigning VMs to I/O hardware resources is based on memory addresses translation between host system and I/O devices, such as network ports. However, there is an important concern related to the scalability of direct assignment approach, since a physical device such as a network port can only be assigned to one VM. Recognizing the limitations of direct assignment, innovative HAV techniques have been proposed by the industry, leading to devices that are natively shareable. The devices replicate resources necessary for each VM can then be directly connected to the I/O device without VMM interference. In this context, Peripheral Component Interconnect Special Interest Group (PCI-SIG) (PCI-SIG, 2015) created a standard approach for implementing these devices.

The PCI-SIG Single Root I/O Virtualization and Sharing (SR-IOV) specification (KUTCH, 2011) defines a standardized mechanism to create natively shared devices. The goal of the PCI-SIG SR-IOV specification is to standardize a way of bypassing the VMM's involvement in data movement by providing independent memory space, interrupts, and direct memory access (DMA) streams for each virtual machine. Two network function types are introduced by SR-IOV:

- Physical Functions (PFs): Full-featured PCI Express (PCIe) functions that include SR-IOV extended capabilities used to configure and manage the SR-IOV functionality. As a full-featured PCIe function, the PF can be discovered and managed like any other PCIe device. PFs provide all the resources necessary to

configure and control the PCIe device, including the ability to move data from inside to outside the device and vice versa.

- Virtual Functions (VFs): Virtual PCIe functions provide the minimal set of configuration resources necessary for data movement. VFs cannot provide the same configuration set as the PFs because changing VF's configuration would affect the underline PF or PCIe device. Since the VFs cannot be configured like a full PCIe device, VF's configuration relies on SR-IOV support provided by the OS or the hypervisor software. SR-IOV technology also requires support from the BIOS to enable and manage SR-IOV features.

SR-IOV provides a mechanism by which a Single Root Function (for example a single Ethernet Port) can appear to be multiple separate physical devices. Figure 2 shows a configuration in which two VMs (VM_A and VM_B) are accessing dedicated resources within the Ethernet controller via Virtual Functions, while the Physical Function can be used by other two VMs (VM_C and VM_D) through a shared virtual switch controlled by the VMM.

An SR-IOV-enabled device can be configured (usually by the VMM) to appear in the PCI configuration space as multiple network functions. Each network function contains its own configuration space and Base Address Registers (BARs). Base address Registers (or BARs) hold the base memory address used by the device, as well as memory size and data type information. The VMM is able to assign one or more VFs to a VM by mapping the actual VF's device configuration space to the VM's device configuration space, as presented in Figure 3.

By improving I/O resource isolation, scalability and performance in virtualized multi-tenant environments, the use of SR-IOV has been explored in network virtualization solutions for cloud computing and data centers (DONG et al., 2012; MAUCH; KUNZE; HILLENBRAND, 2013; SHIEH et al., 2010; KELLER et al., 2010). Nonetheless,

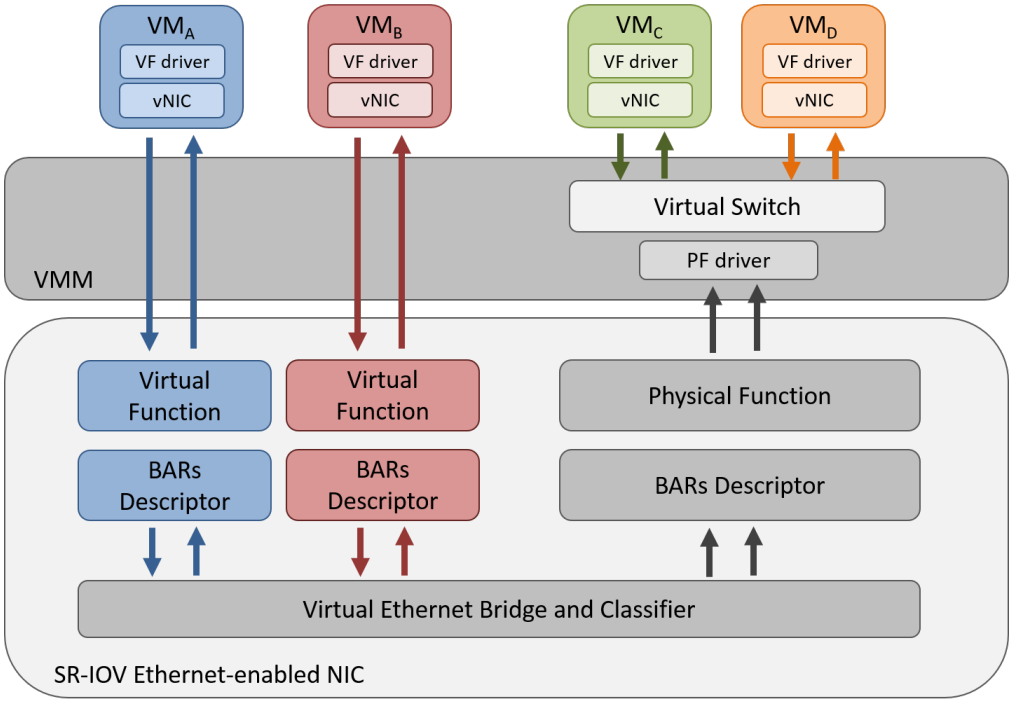

Figure 2: Physical and Virtual Functions running in the same SR-IOV-enabled device. Adapted from (KUTCH, 2011).

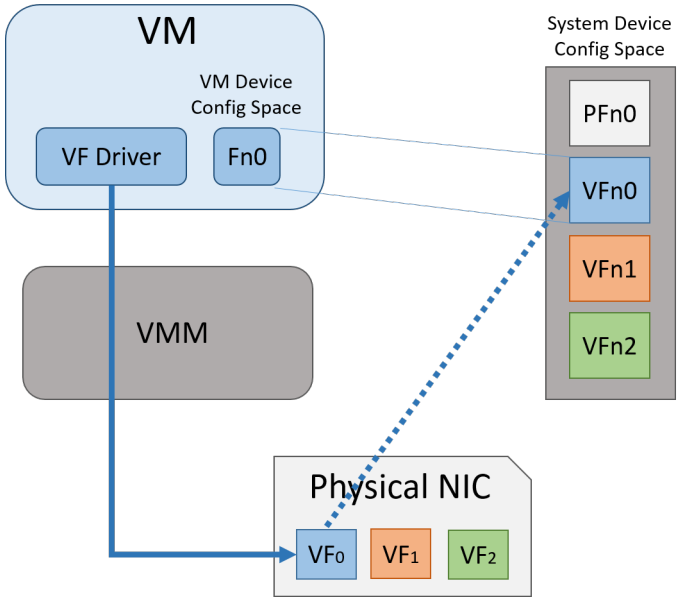

Figure 3: Mapping configuration space between VFs and VMs. Adapted from (KUTCH, 2011).

the contributions of SR-IOV technology for the security of such environments remains poorly investigated. This work intends to investigate how SR-IOV can be applied to implement secure network virtualization architectures for cloud computing systems

and analyze the gains on security, scalability and performance of these architectures.

2.1.3 Network Virtualization in Cloud Computing

The interest surrounding network virtualization has been fueled by cloud computing and its isolation and scalability requirements. All the network virtualization mechanisms presented in Section 2.1.1 can be applied to solve specific network issues in cloud computing, in especial for the implementation of multitenant data centers. Specifically, as depicted in Figure 4, a data center consists mainly of servers in racks interconnected via a top-of-rack Ethernet switch, which in turn connects to an aggregation switch, also known as an end-of-rack switch. The aggregation switches then connect to each other, as well as the other servers in the data center. A core switch connects the various aggregation switches and provides connectivity to the outside world, typically through layer 3 networks. In multitenant data centers, client VMs are commonly placed in a different server, connected through the L2 network composed by this switch-enabled infrastructure. The virtualization of L2 switches via mechanisms such as VLAN enables the abstraction of tenant L2 networks on the distributed cloud data center, allowing traffic isolation of tenant networks with a different logical addressing space. Similarly, the virtualization of L3 routers using technologies such as VXLAN and GRE tunneling enable the abstraction of layer 3 networks, connecting multiple data centers and allowing tenant networks to be distributed in different sites.

Another inherent characteristic of multitenant data centers is the virtualization of servers, enabling the instantiation of multiple VMs. VMs deployed in the same cloud server commonly belong to different tenants and share the same computing resources, including the network interface (NIC). Mechanisms to virtualize server NICs such as virtual switches (i.e., in software) and SR-IOV (i.e., in hardware) are necessary to address multi-tenancy. Besides virtual switches, other software-based virtualization mechanisms are enabled by the NFV approach. NFV consists on the virtualization of

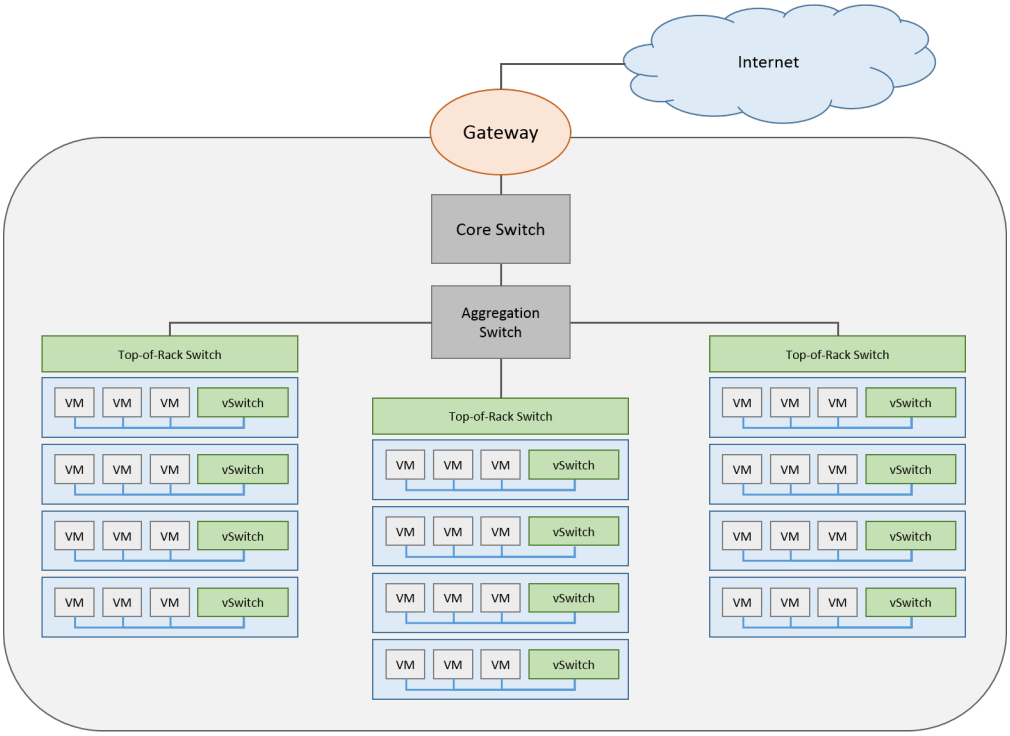

Figure 4: Network elements in the cloud data center.

network functional classes, such as routers, firewalls, load balancers and WAN accelerators. These appliances take the form software-based modules that can be deployed in one or more VMs running on top of servers.

2.2 Software-Defined Networks

The term SDN originally appeared in (GREENE, 2009), referring to the ability of OpenFlow (MCKEOWN et al., 2008) to support the configuration of table flows in routers and switches using software. However, the ideas behind SDNs come from the goal of having a programmable network, whose research started short after the emergence of the Internet, led mainly by the telecom industry. The networking industry has shown enormous interest in the SDN paradigm, given the expectations of reducing both capital and operational costs with service providers and enterprise data centers with programmable, virtualizable, and easily partitionable networks (SDXCENTRAL, 2015). Specifically, programmability is also becoming a strategic feature for network

hardware vendors, since it allows many devices to be programmed and orchestrated in large network deployments (e.g., data centers). In addition, discussions related to the Future Internet has led to the standardization of SDN-related application programming interfaces (API), with new communication protocols being successfully deployed on experimentation and real scenarios (KIM et al., 2013; PAN; PAUL; JAIN, 2011).

These features of SDNs make them highly valuable for cloud computing systems, where the network infrastructure is shared by a number of independent entities and, thus, network management becomes a challenge. Indeed, while the first wave of innovation in the cloud focused on server virtualization technologies and on how to abstract computational resources such as processor, memory, and storage, SDNs are today promoting a second wave with network virtualization (LIN et al., 2014). The emergence of large SDN controllers focused on ensuring availability and scalability of virtual networking for cloud computing systems (e.g., OpenDayLight (MEDVED et al., 2014), ONOS (ONOS, 2016) and Floodlight (Floodlight, 2016)) is a clear indication of this synergy between both technologies. Indeed, our previous work on the evaluation of current open source SDN controllers (BARROS et al., 2015b) demonstrates that OpenDaylight and Floodlight controllers are promising projects for network applications which require high performance, integration with cloud orchestration systems such as OpenStack, and support for recent versions of the OpenFlow protocol. In Chapter 5, we describe such characteristics as fundamental requirements for designing the security architecture proposed in this work. The support for Open vSwitch Database Management Protocol (OVSDB) protocol, the activity level of developer's community and the wide acceptance by industry are key differentiators that contributed to the adoption of the OpenDaylight.

2.2.1 Data Plane and Control Planes

Given that the separation between data and control planes is at the core of the SDN technology, it is important to discuss them in some detail. Figure 5 shows a simplified SDN architecture and its main components, showing that the data and control planes are connected via a well-defined programming interface between the switches and the SDN controller.

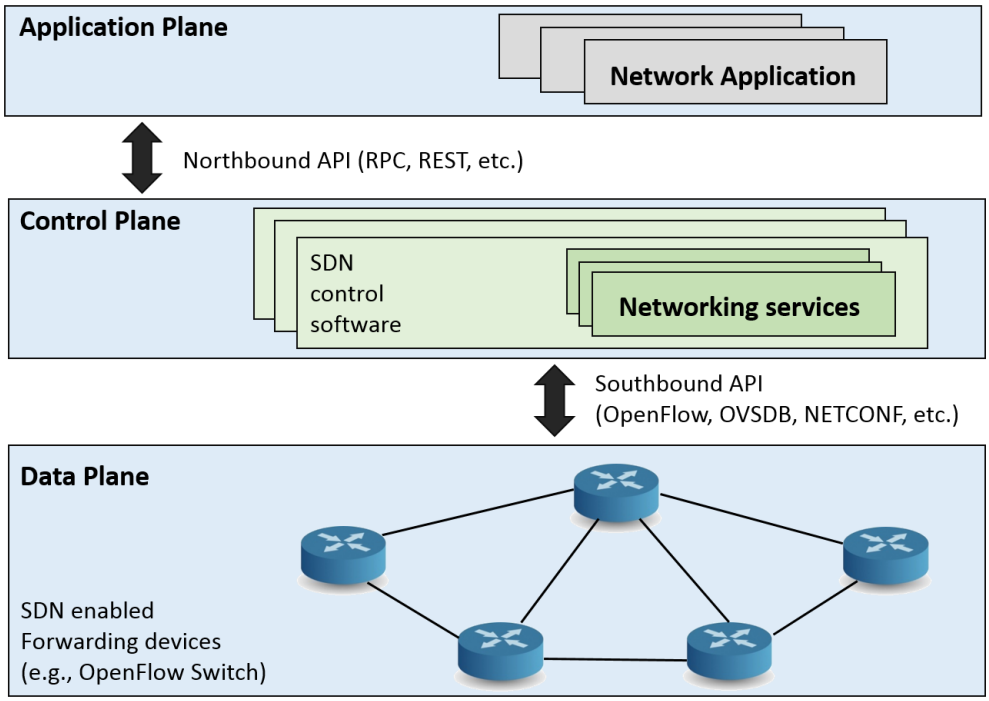

Figure 5: SDN architecture overview

The data plane corresponds to the switching circuitry that interconnects all devices composing the network infrastructure, together with a set of rules that define which actions should be taken as soon as a packet arrives at one of the device's ports. Examples of common actions are to forward the packet to another port, rewrite (part of) its header, or even to discard the packet.

The control plane, on its turn, is responsible for programming and managing the data plane, controlling how the routing logic should work. This is done by one or more software controllers, whose main task is is to set the routing rules to be followed by

each forwarding device through standardized interfaces, called the southbound interfaces. These interfaces can be implemented using protocols such as OpenFlow 1.0 and 1.3 (OPENFLOW, 2009; OPENFLOW, 2012), OVSDB (PFAFF; DAVIE, 2013) and NETCONF (ENNS et al., 2015) The control plane concentrates, thus, the intelligence of the network, using information provided by the forwarding elements (e.g., traffic statistics and packet headers) to decide which actions should be taken by them (KREUTZ et al., 2014).

Finally, developers can take advantage of the protocols provided by the control plane through the northbound interfaces, which abstracts the low-level operations for controlling the hardware devices similarly to what is done by operating systems in computing devices such as desktops. These interfaces can be provided by remote procedure calls (RPC), restful services and other cross-application interface models. This greatly facilitates the construction of different network applications that, by interacting with the control plane, can control and monitor the underlying network. This allows them to customize the behavior of the forwarding elements, defining policies for implementing functions such as firewalls, load balancers, intrusion detection, among others.

2.2.2 OpenFlow Protocol

The OpenFlows protocol is one of the most commonly used southbound interfaces, being widely supported both in software and hardware, and standardized by the Open Networking Foundation (ONF) (ONF, 2016). It works with the concept of flows, defined as groups of packets matching a specific (albeit non-standard) header (MCKEOWN et al., 2008), which receive may be treated differently depending how the network is programmed. OpenFlow's simplicity and flexibility, allied to the high performance at low cost, ability to isolate experimental traffic from production traffic, and to cope with vendors' need for closed platforms (MCKEOWN et al., 2008), are probably among the main reasons for this success.

Whereas other SDN approaches take into account other network elements, such as routers, OpenFlow focus mainly on switches (BRAUN; MENTH, 2014). Its architecture comprises, then, three main concepts (BRAUN; MENTH, 2014): (1) the network's data plane is composed by OpenFlow-compliant switches; (2) the control plane consists of one or more controllers using the OpenFlow protocol; (3) the connection between the switches and the control plane is made through a secure channel.

An OpenFlow switch is basically a forwarding device endowed with a Flow Table, whose entries define the packet forwarding rules to be enforced by the device. To accomplish this goal, each entry of the table comprises three elements (MCKEOWN et al., 2008): match fields, counters, and actions. The match fields refer to pieces of information that identify the input packets, such as fields of its header or its ingress port. The counters, on their turn, are reserved for collecting statistics about the corresponding flow. They can, for example, be used for keeping track of the number of packets/bytes matching that flow, or of the time since the last packet belonging to that flow was seen (so inactive flows can be easily identified) (BRAUN; MENTH, 2014). Finally, the actions specify how the packets from the flow must be processed, the most basic options being: (1) forward the packet to a given port, so it can be routed to through the network; (2) encapsulate the packet and deliver it to a controller so the latter can decide how it should be dealt with (in this case, the communication is done through the secure channel); or (3) drop the packet (e.g., for security reasons).

There are two models for the implementation of an OpenFlow switch (MCKEOWN et al., 2008). The first, consists in a dedicated OpenFlow switch, which is basically a "dumb" device that only forwards packets according to the rules defined by a remote controller.

In this case (see Figure 6), the flows can be broadly defined by the applications, so the network capabilities are only limited by how the Flow Table is implemented and which actions are available. The second, which may be preferable for legacy reasons,

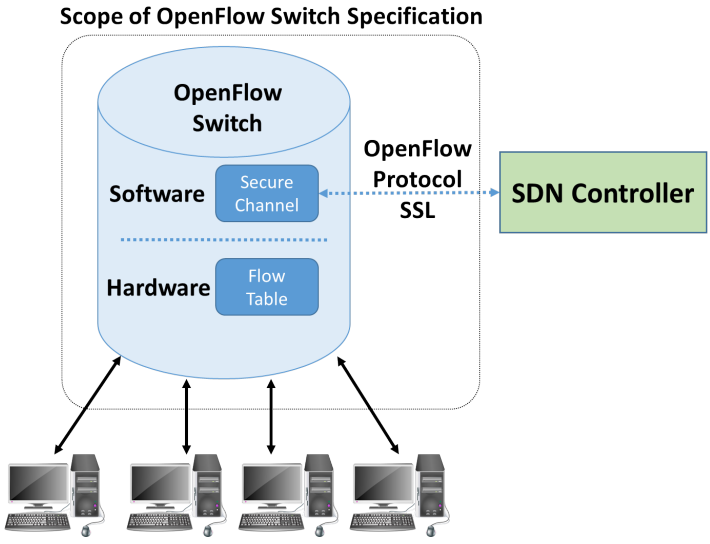

Figure 6: OpenFlow switch proposed by (MCKEOWN et al., 2008).

is a classic switch that supports OpenFlow but also keeps its ability to make its own forwarding decisions. In such hybrid scenario, it is more complicated to provide a clear isolation between OpenFlow and "classical" traffic. To be able to do so, there are basically two alternatives: (1) to implement one extra action to the OpenFlow Table, which forwards packets to the switches normal processing pipeline, or (2) to define different VLANs for each type of traffic.

Whichever the case, the behaviors of the switch's OpenFlow-enabled portion may be either reactive or proactive. In the reactive mode, whenever a packet arrives at the switch, it tries to find an entry in its Flow Table matching that packet. If such an entry is found, the corresponding action is executed; otherwise, the flow is redirected to the controller, which will insert a new entry into the switch's Flow Table for handling the flow and only then the packet is forwarded according to this new rule. In the proactive mode, on the other hand, the switch's Flow Table is pre-configured and, if an arriving flow does not math any of the existing rules, the corresponding packets are simply discarded (KREUTZ et al., 2014).

Operating in the proactive mode may lead to the need of installing a large number of rules beforehand on the switches, one advantage over the reactive mode is that in

this case the flow is not delayed by the controller's flow configuration process. Another relevant aspect is that, if the switch is unable to communication with the controller in the reactive mode, then the switch's operation will remain limited to the existing rules, which may not be enough for dealing with all flows. In comparison, if the network is designed to work in the proactive mode from the beginning, it is more likely that all flows will be handled by the rules already installed on the switches.

As a last remark, it is interesting to notice that implementing the controller as a centralized entity can provide a global and unique view of the network to all applications, potentially simplifying the management of rules and policies inside the network. However, as any physically centralized server, it also becomes a single point of failure, potentially impairing the network's availability and scalability. This issue can be solved by implementing a physically distributed controller, so if one controller is compromised, only the switches under its responsibility are affected. In this case, however, it would be necessary to implement synchronization protocols for allowing a unique view of the whole network and avoid inconsistencies. Therefore, to take full advantage of the benefits from a distributed architecture, such protocols must be efficient enough not to impact the overall network's performance.

2.2.3 SDN and Network Virtualization

Even though network virtualization and SDN are independent concepts, the relationship between these two technologies has become much closer in recent years. Network virtualization creates the abstraction of a network that is decoupled from the underlying physical equipment, allowing multiple virtual networks to run over a shared infrastructure with a topology that differs from the actual underlying physical network.

Even though network virtualization has gained prominence as a use case for SDN, the concept has in fact evolved in parallel with programmable networking. In especial, both technologies are tightly coupled by the programmable networks paradigm, which

presumes mechanisms for sharing the infrastructure (across multiple tenants in a data center, administrative groups in a campus, or experiments in an experimental facility) and supporting logical network topologies that differ from the physical network. In what follows, we provide an overview of the state of the art on network virtualization technologies before and after the advent of SDN.

The creation of virtual networks in the form of VLANs and virtual private networks has been supported by multiple network equipment vendors for many years. These virtual networks could only be created by network administrators and were limited to run the existing protocols, delaying the deployment of new network technologies. As an alternate, researchers started building overlay networks by means of tunneling, forming their own topology on top of a legacy network to be able to run their own control-plane protocols. In addition to the significant success of peer-to-peer applications, built upon overlay networks, the networking community reignited research on overlay networks as a way of improving the network infrastructure. Consequently, virtualized experimental infrastructures such as PlanetLab (CHUN et al., 2003) were built to allow multiple researchers to run their own overlay networks over a shared and distributed collection of hosts. The success of PlanetLab and other shared experimental network platforms motivated investigations on the creation of virtual topologies that could run custom protocols inside the underlying network (BAVIER et al., 2006), thus enabling realistic experiments to run side by side with production traffic. As an evolution of these experimental infrastructures, the GENI project (BERMAN et al., 2014) took the idea of a virtualized and programmable network infrastructure to a much larger scale, building a national experimental infrastructure for research in networking and distributed systems. These technologies ended up by leading some to argue that network virtualization should be the basis of a future Internet, allowing multiple network architectures to coexist and evolve over time to meet needs in continuous evolution (FEAMSTER; GAO; REXFORD, 2007; ANDERSON et al., 2005; TURNER; TAYLOR, 2005).

Researches on network virtualization evolved independently of the SDN concept. Indeed, the abstraction of the physical network in a logical network does not require any SDN technology, neither does the separation of a logically centralized control plane from the underlying data plane imply some kind of network virtualization. However, a symbiosis between both technologies has emerged, which has begun to catalyze several new research areas, since SDN can be seen as an enabling technology for network virtualization. Cloud computing, for example, introduced the need for allowing multiple customers (or tenants) to share a same network infrastructure, leading to the use of overlay networks implemented through software switches (e.g., OVS) that would encapsulate traffic destined for VMs running on other servers. It became natural, thus, to consider using logically centralized SDN controllers to configure these virtual switches with the rules required to control how packets are encapsulated, as well as to update these rules when VMs move to new physical locations.

Network virtualization, on its turn, can be used for evaluating and testing SDN control applications. Mininet (HANDIGOL et al., 2012; LANTZ; HELLER; MCKEOWN, 2010), for example, uses process-based network virtualization to emulate a network with hundreds of hosts, virtual switches and SDN controllers on a single machine. This environment enables researchers and network operator to develop control applications and easily evaluate, test and debug them on a full-scale emulation of the production data plane, accelerating the deployment on the real production networks. Another contribution from network virtualization to the development of SDN technologies is the ability to *slice* the underlying network, allowing it to run simultaneous and isolated SDN experiments. This concept of network slicing, originally introduced by the PlanetLab project (CHUN et al., 2003), consists in separate the traffic-flow space into different slices, so each slice has a share of network resources and can be managed by a different SDN controller. FlowVisor (SHERWOOD et al., 2010), for example, provides a network slicing system that enables building testbeds on top of the same physical

equipment that carries the production traffic.

2.2.4 SDN in Cloud Networking

As previously discussed, networking plays a key role in clouds, both as a shared resource and as part of the infrastructure needed for sharing other computational resources. As any infrastructure, the network in a cloud environment should attend some critical requirements of modern networks, in especial (CHENG et al., 2014): adaptability to new application needs, business policies, and traffic behavior, which new feature being integrated with minimal disruption of the network operation; automation of network changes propagation, reducing (error-prone) manual interventions; provision of high level abstractions for easier network management, so administrators do not need to configure each individual network element; capability of accommodating the nodes' mobility and security features as a core service, rather than as add-on solutions; and on-demanding scaling.

To fulfill these requirements, cutting-edge network equipment with advanced capabilities are likely to be needed. Nevertheless, the cloud cannot take full advantage of such resources without orchestration engines with deep knowledge of the available network capabilities. Unfortunately, such deep knowledge may require features that are very specific to a given proprietary network hardware and software, leading to vendor lock-in issues and, consequently, limiting the creation of new network features or services. In addition, it is often the case that the services must be orchestrated over multi-carrier and multi-technology communication infrastructure (e.g., composed by packet switching, circuit switching and optical transport networks) (AUTENRIETH et al., 2013), or even among different cloud providers (MECHTRI et al., 2013). SDNs tackle this issue by placing an abstraction layer with standard APIs over the (proprietary) hardware, facilitating the access to the corresponding features and, thus, to innovations. In an environment as dynamic as the cloud, the flexibility and programmability

provided by SDNs is essential to allow the network to evolve together with the services using it. Furthermore, SDN architecture can provide and support Application Programming Interfaces (APIs) that simplifies the implementation of common net-work services in cloud computing, such as slicing, virtualization, routing, multicast, security, access control, band-width management, traffic engineering, quality of service (QoS) and other forms of policy management (AZODOLMOLKY; WIEDER; YAHYAPOUR, 2013).

In (BARROS et al., 2015b), we describe different integration architectures for SDN controllers and cloud orchestration systems. This integration can be performed by means of third party applications, SDN functions incorporated to the cloud orchestrator, or well defined APIs for communication between cloud orchestrator and SDN controllers. The latter approach is usually adopted by cloud orchestrator systems such as OpenStack, which make use of SDN drivers to communicate with different SDN controllers. Although not mandatory, SDN controllers typically implement a network service application that optimize the communication API used by the cloud orchestrator.

2.3 Network Function Virtualization

The Network Function Virtualization (NFV) specification was developed by the European Telecommunications Standards Institute (ETSI) (ETSI, 2012) and proposes a novel approach for network virtualization. NFV aims to transform the way in which operators design their networks, applying standard IT virtualization technologies to consolidate network equipment onto industry-standard high volume servers, switches and storage devices, which could be located in data centers, network nodes or in the end-user premises. The idea behind NFV is that physical network functions (e.g., routers, firewalls, Deep Packet Inspectors (DPIs) and Content Delivery Networks (CDNs)) can be transformed: instead of software applications designed to run inside specialized network hardware, they can be decoupled applications deployed

on VMs, containers or standard servers. These applications are then called Virtual Network Functions (VNFs) and, as a decoupled application, the network function could then be instantiated in (or moved to) any network node in the data center, as illustrated in Figure 7. The NFV Infrastructure (NFVI) can, thus, provide computing capabilities comparable to those of an Infrastructure-as-a-Service (IaaS) cloud computing model, dynamically provisioning and orchestrating VNFs provided by independent software vendors.

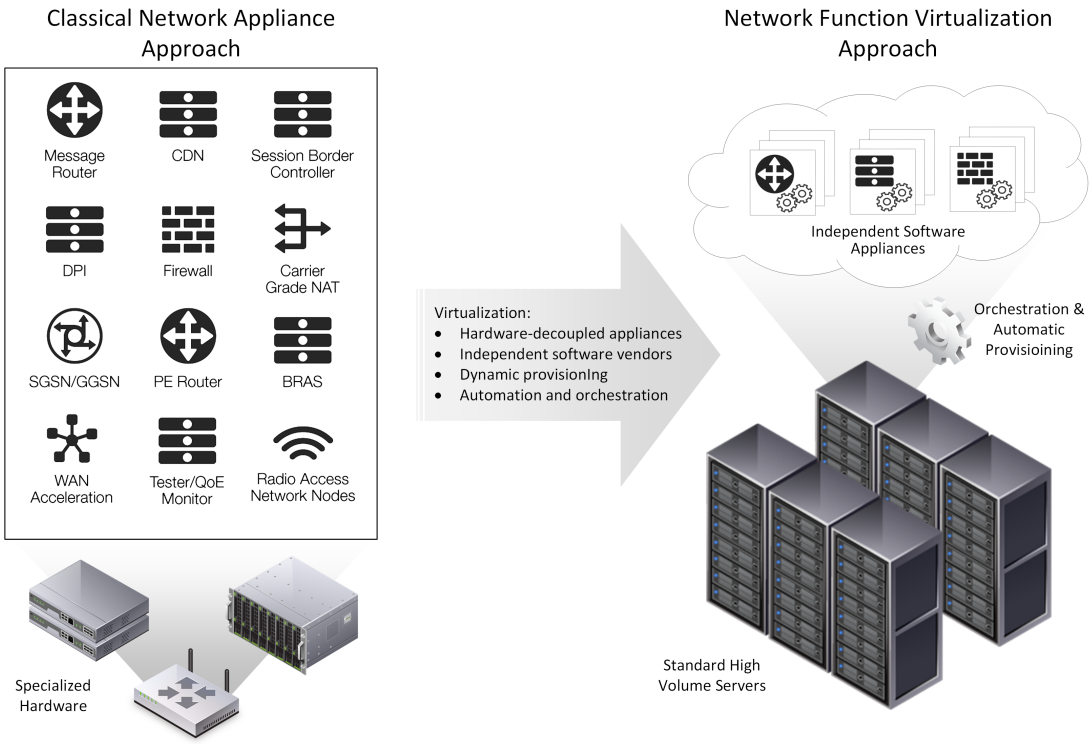

Figure 7: Concepts and vision of Network Function Virtualization (NFV). Adapted from (ETSI, 2012).

It is interesting to notice that, even though the NFV and SDN concepts are considered highly complementary technologies, they do not dependent on each other (ETSI, 2012). Instead, both approaches can be combined to promote innovation in networking. The capability of SDN to abstract and programmatically control network resources are features meet NFV needs for efficiently create and manage dynamic and on-demand network environments. The synergy between these technologies has lead ONF and ETSI to work together with the common goal of evolving both approaches and pro-

vide a structured environment for their development. Table 3 compares SDN and NFV technologies, illustrating how they differently approach network virtualization.

Table 3: Comparison between SDN and NFV. Adapted from (JAMMAL et al., 2014).

	SDN	**NFV**
Motivation	Decoupling of control and data planes; Providing centralized controller and network programmability	Abstraction of network functions from dedicated hardware appliances to Commercial off-the-shelf (COTS) servers
Network Location	Data centers	Service provider networks
Network Devices	Servers and switches	Servers and switches
Protocols	OpenFlow	Not Applicable
Applications	Cloud orchestration and networking	Firewalls, gateways, content delivery networks
Standardization Committee	Open Networking Forum (ONF)	ETSI NFV group

In addition to the synergy between NFV and SDN, the transformation from services based on physical network functions to VNF services leads to the need of a management API for VNFs' orchestration. Cloud computing systems provide the ideal framework for orchestrating, provisioning and deploying VNFs in VM environments (HAN et al., 2015). Cloud solutions such as OpenStack provide a modular and extensible architecture that enables VNF deployments on virtualized platforms such as hypervisors, containers, or even on bare metal platforms (KAVANAGH, 2015). Potential benefits of deploying VNFs in the cloud are: faster development and deployment of network services; elastic scaling of VNF services, which results in optimization of hardware resources usage; support for various compute resources and flavors, offering deployment options such as bare metal, containers, and hardware virtualization; automated continuous deployment and rolling upgrades; and pluggable back-ends, allowing vendors and service providers to develop innovative solutions.

Furthermore, the suitability of cloud computing for NFV deployments enables the virtualization of different security functions in form of VNF. Those virtual functions can assume different roles such as firewall, DPI, and Intrusion Detection System (IDS), which can be deployed in one or more VNFs. Security functions may also be enabled by SDN (YOON et al., 2015), providing security management features through the net-

work control plane. We explore this approach by aggregating network and security functions in a special purpose VNF that enforces connectivity policies for tenants' VMs. This special purpose VNF is the key element for building our security architecture in the cloud, as described in details in Section 5.2.

2.4 Chapter Considerations

In this chapter we describe different technologies that can be applied to network virtualization in cloud computing, as well as the network abstractions they are able to provide. These network abstractions can be combined as building blocks for designing different network virtualization architectures, providing different security properties. Consequently, they also provide us with different techniques approach virtual network security in cloud data centers. Therefore, the virtualization technologies described in this chapter will serve as basis for our future evaluation of security aspects of existing network architecture, as well as for developing our security architecture.

The proposed network virtualization solution presented in the next chapters explores the synergy between multiple layer virtualization, building upon cloud orchestration, SDN control and hardware acceleration to implement multi-tenancy isolation with minimal impact over network performance. Therefore, we carefully analyze the concepts and technologies associated with HAV, SDN and NFV. In particular, SDN and NFV have changed network virtualization by introducing dynamic and flexible provisioning of network services and leveraging control over the network. We argue that both technologies are completely suitable for cloud computing deployments, leveraging the control capabilities on cloud networking. In Chapter 5 we build upon these technologies to enhance the control over security policies enforced by cloud network appliances. Furthermore, we describe how HAV technologies, in particular SR-IOV, can be applied to network virtualization architectures to enhance tenants' network isolation and to improve the performance of NFV appliances.

In Chapter 3 we explore the security concerns regarding multi-tenancy in cloud systems in order to identify how the network virtualization technologies described in this chapter can be improve isolation in cloud networks. We describe the security features provided by current applied technologies and analyze how the can leverage different multi-tenant isolation strategies in cloud networking.

3 SECURITY AND MULTI-TENANCY IN CLOUD COMPUTING

Historically, specialists from academia and industry have made a great effort to optimize networking inside data centers, in especial considering that the advent of server virtualization created a new network access layer connecting various VMs, containers and other virtual appliances. (BARI et al., 2013). In particular, server virtualization facilitates network mobility and scaling, but also leads to much stronger requirements in terms of isolation than most physical deployments (PFAFF et al., 2009). Such peculiarities of virtualized appliances are especially important today because, as observed in Gartner's latest magic quadrant report for server virtualization (BITTMAN; DAWSON; WARRILOW, 2015), about 75% of server workloads are currently virtualized. In addition, Gartner's latest market report (WARRILOW, 2016) states that, despite rumors about saturation, the server virtualization market is still growing: it is expected to reach $5.6 billion in 2016, an increase of 5.7 percent from 2015.

The success of virtualization is also observed in the context of cloud computing, where it is a key driver to create self-managed, highly scalable and highly available data centers (PORTNOY, 2012). In such environments, virtualization is employed to create an abstraction layer over the physical data center resources, exposing them as a pool of easily on-demand consumable and manageable virtual resources. These properties are key to enable public clouds, which allows companies and end-users to outsource their computing infrastructure to a cloud provider and then consume and pay for computing resources and services on-demand (MELL; GRANCE, 2011). Such cloud deployment

model is today very attractive: the latest Gartner forecast (NAG et al., 2016) predicted a growth of 16.5 percent in public cloud services in 2016, reaching a growth rate peak of 17.5 percent in 2017 and continuing to increase through 2020; the Infrastructure-as-a-Service (IaaS) model is the main responsible for this growth, being projected to advance 38.4 percent in 2016. Cisco Global Cloud Index (GCI) (Cisco, 2016b) also estimates that, by 2019, 86 percent of the worldwide workload will be processed by cloud data centers, of which 56 percent will be processed by public clouds.

Considering the numerous benefits and enormous success of public clouds, it is important to address the unique security challenges resulting from its multi-tenancy characteristics[1] (REN; WANG; WANG, 2012). Indeed, security is still one main factor that drives companies to adopt private and hybrid instead of public cloud strategies (Gartner, 2016; RightScale, 2016). Aiming to give a clear picture of such issues, this chapter explores the security implications of multi-tenancy characteristics in public clouds. We focus specifically on cloud networking, revisiting network virtualization approaches and technologies for improving isolation in this context. The reason is that networks are a critical resource both for CSP and for cloud consumers, being responsible for connecting physical and virtualized resources distributed across the underlying cloud infrastructure and providing ubiquitous access to cloud services. In Section 3.1 we discuss the need of isolation in network virtualization architectures applied to multi-tenant clouds. We thus give special focus on infrastructure services (IaaS), presenting different strategies to implement multi-tenant isolation in cloud network architectures in Section 3.2. Finally, in Section 3.3, we analyze the ability of existing cloud network architectures to implement multi-tenant isolation, proposing a novel approach to improve the security of cloud systems built upon the isolation of tenant network resources.

[1]Multi-tenancy allows multiplexing the execution of multiple virtual machines from disjoint customers upon the same physical hardware, enabling resource sharing by means of virtualization software.

3.1 The Need of Multi-tenant Isolation in Cloud Networks

Network access to public clouds commonly relies on standard protocols such as the Internet protocol (IP), which are susceptible to security threats, such as eavesdropping and man-in-the-middle attacks. Whereas traditional network-level security control mechanisms such as IP-based network zoning, IPsec and network-based vulnerability scanners can be employed for addressing such issues, the same approach is not necessarily enough inside the clouds. The reason is that network virtualization creates traffic that needs to be protected inside servers, which are commonly inaccessible by traditional network security appliances (GROBAUER; WALLOSCHEK; STOCKER, 2011; AZODOLMOLKY; WIEDER; YAHYAPOUR, 2013). In addition, they provide very limited control over the physical network infrastructure to tenants, so different consumers end up unknowingly sharing (potentially vulnerable) network infrastructure components such as DNS and DHCP services, as well as physical and virtual network elements. Hence, without proper isolation, the networks are also affected by security threats that are related to multi-tenancy in cloud environments.

More concretely, security vulnerabilities in multi-tenant clouds are mainly inherited from vulnerabilities in virtualization software as well as from co-residence based attacks. As an essential part of the cloud infrastructure, the network can also be affected by such threats and/or exploited by them. For example, a vulnerability on some service hosted in the cloud is usually accessed via its public interfaces (e.g., REST APIs, web applications or FTP services), so its exploitation involves packets sent via the cloud's network. Hence, the attack may impact not only the target service, but also have side effects on other services sharing the same network infrastructure (VARADARAJAN et al., 2015). In addition, there are many network based techniques that can be employed in co-residence based attacks. For instance, network techniques evaluated by (RISTENPART et al., 2009) to determine co-residence in Amazon EC2 (Amazon, 2016a) in-

clude: matching the IP address of the physical server hosting attacker's VM (Dom0 IP address), monitoring small packet round-trip times (RTT) for reaching target VM, and verifying numerically close internal IP addresses. These techniques are revisited in (INCI et al., 2015) and evaluated according to their applicability to current cloud provider infrastructures, suggesting additional covert-channels and side-channels techniques to enhance co-location capabilities in Amazon EC2. There are also alternative network-based co-residence check techniques that build upon active traffic analysis, enabling attackers to inject watermark signatures into the network flow of target resources (e.g., VMs), thereby being able to exfiltrate and broadcast co-residence information from the physical server hosting a victim's resources (BATES et al., 2012). This technique allows attackers to compromise multi-tenant isolation without relying on cover-channel or side-channels attacks, hindering Cloud Service Providers (CSPs)' detection capabilities by hiding malicious traffic into legitimate tenants' flows.

Given that cloud networks are a critical channel of attack in multi-tenant clouds, it is important to understand how security vulnerabilities on network virtualization technologies affect the cloud system. One example are virtual switches, which enable L2/L3 connectivity services inside virtualized host servers and are responsible for managing traffic to cloud tenants' resources, such as VMs. As such, they handle both internal traffic, exchanged between VMs inside the cloud infrastructure, hosted by the same or by different physical servers, as well as external traffic, exchanged between a tenant's VM entities external to the cloud (e.g., the Internet). Virtual switch instances are usually shared by multiple tenants' VMs hosted in the same physical server, being controlled by hypervisors and cloud network orchestration services to implement connectivity policies for different tenants (AZODOLMOLKY; WIEDER; YAHYAPOUR, 2013). This particular characteristic of multi-tenant cloud networking architectures creates many potential attack surfaces resulting from virtual switches' vulnerabilities, including Denial of Service (DoS), arbitrary code execution and overflow. For instance, Cisco

Nexus 1000V Virtual Switch (Cisco, 2016c) enables remote attackers to cause DoS via crafted TCP or CDP[2] packets (CVE, 2016a; CVE, 2016b), while malformed CDP packets may be employed to allow the execution of arbitrary code (CVE, 2013a). Vulnerabilities also exist in the widely deployed and supported Open vSwitch (Open vSwitch, 2016c), an open source virtual switch with support for multiple standard management interfaces and protocols (e.g., NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag) and automated control protocols (e.g., OpenFlow and OVSDB), and adopted by default by many open source hypervisor and cloud platforms (e.g., OpenStack (OpenStack, 2016f), OpenNebula (OpenNebula, 2016b), Xen Cloud Platform (Xen, 2016a) and CloudStack (CloudStack, 2016a)). One example of critical vulnerability reported on Open vSwitch allows remote attackers to execute arbitrary code via crafted MPLS packets, causing buffer overflow and potentially DoS (CVE, 2016c). Another common buffer overflow attack based on L2 Media Access Control (MAC) is known as a MAC flooding attack, which consists in generating packets with several random MAC addresses aiming to overflow the forwarding table within shared virtual switches, thus forcing them to broadcast packets to every interface; as a result, malicious tenants receive the packets from other, legitimate tenants, threatening the confidentiality of their communication (BULL; MATTHEWS, 2015). A less critical vulnerability reported on the Common Vulnerabilities and Exposures (CVE) database allows local attackers to delete and overwrite arbitrary files once they are able to bypass hypervisor isolation (CVE, 2012b). In addition, virtual switches that provide support to the OpenFlow protocol (MCKEOWN et al., 2008) are also exposed to openflow security vulnerabilities. (KLÃ¶TI; KOTRONIS; SMITH, 2013; BENTON; CAMP; SMALL, 2013)

Other virtualization technologies that can introduce security vulnerabilities to multi-tenant cloud networking include hypervisors (or VMMs) (CVE, 2014a), cloud network orchestration services (CVE, 2014b), and OS network virtualization features

[2]Cisco Discovery Protocol (CDP) is a proprietary Data Link Layer protocol developed by Cisco Systems to share information about other directly connected Cisco equipment (Cisco, 2016a).

(CVE, 2012a). In addition to vulnerabilities in virtualization technology that may impair the security of multi-tenant cloud network by causing DoS, bypass, overflow, and malicious code execution, cloud systems are exposed to network attacks based on standard protocol vulnerabilities, such as IP spoofing, Address Resolution Protocol (ARP) spoofing, Routing Information Protocol (RIP) based attacks, DNS poisoning, network flooding, DoS and Distributed Denial of Service (DDoS) attacks (MODI et al., 2013).

Given this wide threat scenario, we conclude that securing networks is critical to multi-tenant cloud systems, and that efficient isolation of network resources is important to enhance security principles such as confidentiality, integrity and availability. Unfortunately, however, current network virtualization strategies usually display a limited view on the isolation of network resources in multi-tenant clouds, focusing mainly on the isolation of data paths across shared network resources. As discussed further in what follows, this limited view can be expanded by taking advantage of network virtualization paradigms that enable different isolation strategies, such as NFV (ETSI, 2012), SDN (GREENE, 2009) and Hardware-Assisted Virtualization (HAV) (KUTCH, 2011; Intel, 2016b; Intel, 2016a).

3.2 Multi-tenant Isolation of what?

Multi-tenant isolation in cloud networking is usually implemented by means of network software appliances (e.g., virtual NIC, virtual router and virtual switches), VLANs (IEEE, 2006) and network overlay technologies (e.g., VXLAN (MAHALINGAM et al., 2014), NVGRE (WANG; GARG, 2015) and STT (DAVIE; GROSS, 2016)). Network software appliances, such as virtual switches, are deployed in cloud infrastructures to enable network services on the server virtualization layer, which are shared among multiple cloud tenants. Such appliances manage multi-tenant internal and external traffic (as defined in Section 3.1) by applying VLANs and tunneling protocols such as VXLAN to implement network traffic and IP address isolation, extending virtual

networks across the underlying cloud network infrastructure (WANG et al., 2013). Although virtual network appliances are intended to provide complete isolation among cloud tenants, security vulnerabilities in the corresponding software may expose the cloud's network infrastructure and its tenants to threats. Similarly, the underlying network components shared by multiple tenants are exposed to traditional network attacks that may lead to hardware resource overflow and DoS. It is, thus, important to consider how different network virtualization strategies for multi-tenant clouds can isolate vulnerable network resources and reduce attack surfaces.

In a nutshell, cloud networking consists of traffic from multiple VMs being generated inside virtualized servers and steered through software and hardware network appliances[3] across one or more data centers. Figure 8 illustrates a usual cloud networking infrastructure, including common software and hardware network appliances. In this figure, we can identify three key entities for cloud networking operation: tenant traffic, software appliances and hardware appliances. Hence, considering these elements, we argue that network isolation in multi-tenant clouds can be enhanced by applying three different strategies defined upon different groups of cloud network resources: (1) Data Path Isolation; (2) Isolation of Software Resources; and (3) Isolation of Hardware Resources. In what follows, we describe these strategies under the light of security issues and virtualization technologies discussed in Section 3.1, discussing the security requirements addressed by each of them and how they can be combined to improve isolation of network resources in multi-tenant clouds.

3.2.1 Data Path Isolation

Data path isolation refers to the manipulation of network traffic throughout the underlying cloud network infrastructure to create different logical paths that connect mul-

[3]Software appliance is a software application that might be combined with just enough operating system (JeOS) for it to run optimally on industry standard hardware or virtual machine, whereas a hardware or computer appliance is a separate and discrete hardware device and embedded software (firmware), designed to provide a specific computing resource (SMITH et al., 2007; DMTF, 2014).

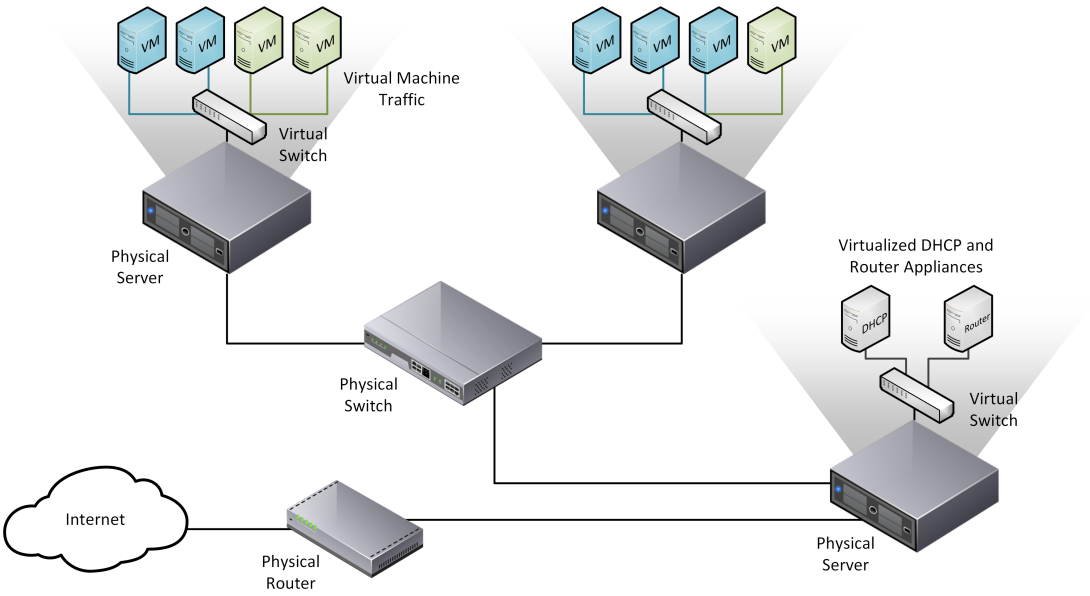

Figure 8: Cloud networking infrastructure including common software and hardware network appliances.

tiple tenants' resources. Logical data paths enable cloud tenants to run virtual networks and dependent services as though they were running in dedicated non-virtualized infrastructures, creating virtual isolated links between tenants' resources. In addition, data path isolation provides cloud tenants with independent IP addresses, allowing network resources and applications deployed by different tenants to use a same address space.

Logical data paths are usually implemented by manipulating traffic in the data link and the network layers, respectively, Layer 2 (L2) and Layer 3 (L3) of the OSI model (ISO, 1994). The most widely deployed technology for implementing data path isolation in L2 is probably the VLAN standard (IEEE, 2006; IEEE, 2014), which defines a tagging system for Ethernet frames, as well as procedures for handling tagged frames in network bridges and switches. VLAN tags, referred to as VLAN IDs, enable network bridges and switches to uniquely identify logical data paths inside a network, defining isolated broadcast domains within a shared media. The maximum number of VLANs on a given Ethernet network is limited to 4094, which corresponds to 4096 provided by 12-bit VID header field, minus the reserved values 0x000 and 0xFFF. This

reduced number of possible logical data paths can significantly impact the scalability of multi-tenant systems, in particular of public cloud systems, limiting the number of tenant networks. Tunneling protocols such as VXLAN (MAHALINGAM et al., 2014), NVGRE (WANG; GARG, 2015) and STT (DAVIE; GROSS, 2016) can help solving such scalability issues by means of larger identifiers (e.g., 24-bit network identifiers for VXLAN and NVGRE, and 64-bit context identifier for STT). It can also extend data path virtualization to the network layer, applying L3 encapsulation and tagging techniques to build isolated logical data paths upon IP networks, enabling data path virtualization across multiple data centers. L3 data path virtualization technologies also enable the creation of overlay networks, allowing originally not supported protocols to run over local isolated data paths built upon the IP protocol. Moreover, Virtual Private Cloud (VPC) services enhance isolation of logical data paths by providing private IP subnets and communication channels inside CSP networks by means of VLANs and VPN functions (Amazon, 2016b).

Current L2 and L3 data path virtualization technologies provide efficient and scalable mechanisms for isolating data traffic throughout the multiple elements of the cloud's network infrastructure. However, as discussed in Section 3.1, they are still threatened by vulnerabilities on hardware and software elements that compose the cloud network infrastructure, which can result in confidentiality, integrity and availability breaches. This issue can be addressed by means of security protocols such as IPsec (SEO; KENT, 2005) and TLS (DIERKS, 2008) for protecting the confidentiality and integrity of the tenants' network traffic in case of unauthorized access (STALLINGS, 2016; ZISSIS; LEKKAS, 2012). Nevertheless, besides incurring performance penalties, such protocols are unable to deal with attacks such as DoS, cover-channel, arbitrary code execution and bypass, which can seriously affect cloud network availability or result in CSP data breaches. It is, thus, necessary to explore virtualization strategies for isolating those potentially vulnerable software and hardware resources, reducing

the impact resulting from related attack surfaces.

3.2.2 Isolation of Software Resources

The isolation of software resources refers to provisioning network software appliances and services that will be consumed by no more than one tenant. Network software appliances include network virtualization software and virtualized network resources running inside physical host servers, such as virtual switches and virtual NICs. Virtualization software solutions are commonly provided as third party applications that can run upon host OS (sometimes with kernel support) (Open vSwitch, 2016c; Linux Kernel, 2016; KVM, 2016a), or directly upon standard hardware (The Xen Project, 2016; Open vSwitch, 2016a), managed by hypervisor or cloud orchestration software via a well-defined programmable interface. Network software appliances also include networking services provided directly by the OS, such as Linux bridges, namespaces, and virtual Ethernet interfaces (ROSEN, 2014). Such resources are commonly employed by hypervisor and cloud orchestration software to implement connectivity among virtualized resources such as virtual switches, virtual NICs, and virtual routers.

As previously discussed in Section 3.1, virtualization software is susceptible to security vulnerabilities, including widely deployed solutions such as Open vSwitch (CVE, 2016c). Hence, although a single Open vSwitch can provision multiple isolated bridges into one host system, thus mitigating bridge-oriented overflow attacks (BULL; MATTHEWS, 2015), the resulting system still exposed to Open vSwitch's vulnerabilities that allow bypassing and code execution attacks targeting its core services and managements interfaces (CVE, 2012b; CVE, 2016c). Moreover, traditional network virtualization architectures deployed by cloud systems usually share specialized bridges among multiple tenants hosted in the same physical server, increasing attack exposure in cloud networks (OpenStack, 2016d; OpenNebula, 2016a).

In this scenario, it is necessary (or at least advisable) to completely isolate software

network appliances by isolating virtualized network resources and core application services. An efficient approach for attaining such isolation is the adoption of NFV (ETSI, 2012). NFV enables the encapsulation of software network appliances within other virtualized appliances such as VMs and containers, extending the isolation originally provided by such virtualization technologies to cloud network functions. Moreover, cloud systems provide a compliant framework for orchestrating VNFs, enabling CSPs to implement multiple isolation strategies for software resources (KAVANAGH, 2015; HAN et al., 2015; MIJUMBI et al., 2016). It is worth mentioning that the isolation of network software appliances by means of NFV relies on the security provided by underlying virtualization technologies, such as a hypervisor or a containers management application. Nevertheless, the exploitation of such elements requires more powerful attackers and transcends the scope of cloud networking security; for this reason, dealing with such attacks is not in the scope of this work.

3.2.3 Isolation of Hardware Resources

The isolation of hardware resources requires the allocation of dedicated network hardware appliances (or portions of them), so the corresponding network functions and services are consumed by no more than one cloud tenant. Network hardware appliances include specialized network hardware such as NICs, switches, routers, and load balancers. In addition, using the NFV architecture, such appliances can also be virtual network functions deployed in commodity hardware, such as standard servers, storage and switches (ETSI, 2012; HAN et al., 2015). Therefore, we can extend the definition of network hardware appliances to standard servers and corresponding peripheral devices running network functions inside a CSP.

Network hardware appliances, like software network appliances, are susceptible to security vulnerabilities intrinsic to hardware and software (firmware) implementation of network functions and protocols, allowing attackers to perform co-location

checks (RISTENPART et al., 2009; INCI et al., 2015), side-channel and cover-channel attacks (BATES et al., 2012), or traditional network attacks based on standard network protocol vulnerabilities (MODI et al., 2013). By multiplexing the underlying network infrastructure among mutually distrusted consumers, multi-tenant clouds enable malicious tenants to explore these vulnerabilities attempting to cause, for example, the interruption of cloud network services (CVE, 2013b). In addition, the lack of isolation of hardware network resources on physical servers may allow malicious tenants to perform DoS attacks against collocated tenants by making excessive use of NIC resources (ALARIFI; WOLTHUSEN, 2013). Even though the saturation of NIC resources might be unadvised for the attacker because it can negatively impact its own VMs, such attack may actually benefit malicious tenants by means of resource-freeing attacks. Such attacks build upon the observation that the allocation of hardware resources for collocated cloud tenants can change unevenly based on their workloads. Hence, attackers may be able to *improve* their VMs performance by generating workloads for collocated VMs (VARADARAJAN et al., 2012).

Since shared hardware resources is an intrinsic characteristic of multi-tenant cloud systems, it is usually challenging to isolate those resources among tenants across without affecting the economy of scale of public clouds. A straightforward approach is to allocate dedicated hardware for tenant networks, which has been proposed in a few cloud architectures as discussed later in Section 3.3, but this leads to higher infrastructure costs. In contrast HAV allows better isolation of multiple resources inside cloud virtualized servers, such as CPU, memory, and I/O devices, while also improving the performance of virtualized appliances. Technologies such as SR-IOV (KUTCH, 2011) and VMDq (Intel, 2016b), for example, enable the isolation of a NIC's resources among multiple tenant VMs running under the same host server. In particular, SR-IOV allows tenant VMs to directly connect to the server's NIC, bypassing the hypervisor stack and providing near real-hardware communication performance (DONG et al., 2012).

SR-IOV achieves efficient isolation by allocating different portions of resources, such as network processor and I/O memory buffer, for each connected tenant VM in form of SR-IOV VFs. The DPDK technology can also be applied to perform multi-tenant isolation of NIC resources by programmatically allocating different memory vectors for software appliances consumed by different cloud tenants (KOURTIS et al., 2015). It is also worth mentioning that security of network hardware appliances can be enhanced by deploying data path virtualization technologies, such as tunneling protocols (e.g., VXLAN), which are able to encapsulate crafted malformed packets and mitigate attacks based on overflow and arbitrary code execution.

3.3 Isolation in Existing Cloud Network Architectures

The exact isolation strategy adopted directly influences the security of multi-tenant cloud infrastructures. In this section, we evaluate the capability of existing network virtualization architectures to implement different multi-tenant isolation strategies, considering the three classes discussed in Section 3.2, thereby exploring challenges and opportunities in this context. For the purpose of comparison, we assume that traditional architectures address multi-tenant isolation by means of server and data path virtualization, applying hypervisor technologies, VLAN tagging mechanism, and overlay protocols (CHAO, 2015; DENTON, 2016; SABHARWAL; SHANKAR, 2013). In addition, traditional network virtualization architectures deploy commodity hardware switches to provide connectivity between physical servers inside the data center, and commodity routers to provide connectivity with the Internet (AZODOLMOLKY; WIEDER; YAHYAPOUR, 2013; LEE, 2014). Hence, one main limitation of such traditional architectures is the lack of isolation of software and hardware resources, as well as security and scalability issues incurred by commonly applied data path virtualization mechanisms. In what follows, we analyze alternative architecture for data center network virtualization, discussing their suitability for deployment in public clouds.

3.3.1 NetLord

NetLord (MUDIGONDA et al., 2011) is a network virtualization architecture designed to enhance scalability in multi-tenant data centers in a cost-effective manner. The key idea behind the NetLord architecture is to encapsulate L2 packets from tenant virtual networks and transmit them over the underlying network infrastructure employing L2 and L3 encapsulation techniques. The additional L2 header specifies the source and destination MAC addresses for edge switches, whereas the L3 encapsulation header specifies the tenant ID and the egress port on destination switches through the IP source and destination header fields, respectively. The encapsulation process is performed by a software component called a NetLord Agent (NLA), deployed on every host inside the data center. Packets are encapsulated by NLAs and transferred over a data center network to the egress switch through the underlying L2 fabric, using specific data paths determined by a VLAN selection algorithm. Since the NLAs encapsulate VMs' MAC addresses within edge switch addresses, NetLord is able to reduce forwarding tables on the underlying switches, thus reducing hardware requirements.

Data path isolation inside virtualized servers is implemented by NLAs, which are able to handle tenant IDs using L3 encapsulation. However, such logical data path abstraction does not propagate to the underlying L2 network, once tenant IDs are not directly mapped to the VLANs that will be used to forward the traffic across the switching fabric. Instead, VLANs are essentially determined by the destination Ethernet address added by the NLAs, which corresponds to the address of the destination egress switch. This implies that multiple tenants attempting to reach the same egress switch may share the same VLAN, thereby not benefiting from VLAN isolation features that may be provided by software and hardware network appliances across the underlying network, such as traffic limiting and QoS. In addition, NetLord does not address isolation of a critical network software resource, the NLAs, which are shared among multiple tenants collocated on a physical server. Similarly, since NetLord relies on a

centralized configuration repository accessed by NLAs, which aggregates per-tenant configuration information, a tenant might exploit a vulnerable NLA to gain privileged access to configuration data from another tenant. Finally, the solution does not implement any isolation of network hardware resources.

3.3.2 SEC2

Secure Elastic Cloud Computing (SEC2) (HAO et al., 2010) is a cloud network virtualization architecture intended to enhance network isolation in multi-tenant data center. It does so by separating package forwarding and access control in two different architectural layers, named core and edge domains. SEC2 architecture is composed by one core network domain connecting multiple edge network domains. The core domain consists of L2 high-capacity Ethernet switches whose main function is to transport packets among edge data center domains, whereas edge domains consist of physical hosts connected through commodity Ethernet switches. Virtualized physical host inside edge domains are, in turn, composed by multiple tenant VMs provisioned by means of a hypervisor software and connected through a virtual switch, as in traditional network virtualization architectures. In addition, SEC2 uses different VLANs to isolate tenants' data path inside edge domains, and MAC-in-MAC encapsulation to send packets across the core domain.

The SEC2 virtualization architecture is supported by two main components, named Forwarding elements (FEs) and Central Controller (CC). FEs are essentially Ethernet switches responsible for forwarding traffic generated by edge domains across the core domain according to security policies defined by CC. Likewise, FEs enforce security policies for packets exchanged between two different VLANs inside the same edge domain, whereas packets sent to the same VLAN are directly delivered to the destination VM without policy checking. The connection between edge domains across the data center may involve one or more FEs. CC, in turn, is a centralized network controller

that maintains databases for address mapping and policies, and controls the operation of FEs through well-defined APIs such as OpenFlow. Furthermore, FEs located at the data center edge can serve as tunneling endpoints for VPN and VPC implementations, applying technologies such as GRE and IPsec.

Data path isolation in SEC2 architecture is implemented using different VLANs for traffic generated by different tenant VMs inside the edge domain. Therefore, SEC2 does not deal with security vulnerabilities resulting from the lack of isolation of software resources inside host servers to the edge domain's underlying network. SEC2 does not encapsulate tenant network traffic sent to different physical hosts, neither provides isolation of hardware resources. As a result, the underlying L2 infrastructure of edge domains becomes exposed to almost the same attacks that may be performed by malicious tenants against shared virtual switches and other software resources, including overflow, code execution and DoS attacks. In addition, since CC centralizes the control over all FEs, it becomes a potential target of attacks that can explore vulnerabilities on the edge domain's underlying networks by means of non-encapsulated packets sent by malicious tenants. Attackers may also attempt to perform DoS in CC by sending a large volume of malicious traffic between different edge domains, which may create an excessively high number of policy resolution requests.

3.3.3 vShield

The vShield (BASAK et al., 2010) network virtualization architecture builds upon the concept of security virtual appliances (SVA), implementing security services that can be deployed on host servers and offering distributed security functions for network flows across the data center. This approach aims to replace hardware security appliances that can become choke points, such as firewalls, flow monitors, and IDS. Those hardware appliances are substituted by distributed software appliances co-located with tenant VMs inside host servers, having the ability of handling VMs' incoming and out-

going traffic. As an example of SVA implementation, vShield architecture designs a Firewall (vShield Firewall) implemented by a vShield hypervisor module and a vShield SVA, a pre-installed and pre-configured VM running a hardened, specialized OS for handling firewall operations. The vShield hypervisor module effectively places a network packet filter between VM vNICs and the virtual switch deployed on each physical host, allowing incoming and outgoing traffic from vNICs to be inspected by the vShield Firewalls. VShield also provides a vShield Edge SVA, which provides network edge security and gateway services to the VMs attached to a specific port group of the host virtual switch. The vShield Edge SVA isolates connectivity with VMs attached to a switch port group, acting as a gateway for external communication, while providing services such as DHCP, VPN and NAT.

The vShield architecture is able to implement data path isolation similarly to traditional network virtualization architectures, deploying L2 tagging and L3 tunneling technologies to create and scale tenant overlay networks. However, SVAs handle packets from several VMs connected to a same vNIC, meaning that L2 packets from multiple tenant networks are not encapsulated by tunneling protocols. As a result, such software appliances become more exposed to overflow and code execution attacks than traditional hardware appliances that handle encapsulated traffic. Furthermore, the lack of isolation of software resources in SVAs that are not edge appliances, such as vShield Firewalls, allows malicious tenants to explore potential vulnerabilities in such resources to perform attacks such as DoS overflow. Although vShield Edge SVA provides better isolation of network software appliances by means of virtual switch port groups and gateway security appliances, it still relies on the virtual switch, a key network software appliance that is shared with other tenants' VMs collocated with those connected with vShield edge port groups. Finally, vShield was not designed to provide any isolation of network hardware resources among multiple tenants.

3.3.4 BlueShield

BlueShield (BARJATIYA; SARIPALLI, 2012) comprises a L2 network appliance designed to enhance traffic isolation and monitoring by blocking broadcast and multicast traffic among different tenants' VMs, using virtual switches. BlueShield's architecture builds upon vShield (BASAK et al., 2010) to provide a multi-tenant virtualization solution that does not depend on the hypervisor. The main component of BlueShield is a software agent, called simply a BlueShield Agent, that is installed in every data center VMs for enabling inter-VM communications. A BlueShield Agent captures outgoing ARP requests from its companion VM, converting the requests from broadcast to unicast queries forwarded to a directory-lookup server. Directory servers reply to BlueShield queries with a destination MAC address only if it is permitted by security policies, enabling the creation of isolated VMs. VMs belonging to a particular group communicate only with other VMs in the same group, whereas communication among VMs belonging to different groups is completely blocked by directory server policies, creating the illusion that a VM group is connected through a separate VLAN.

Even though BlueShield provides isolation features similar to VLANs, its implementation actually does not rely on VLAN technologies. In fact, data path isolation is achieved by means of the BlueShield agents, which blocking broadcast and multicast network traffic among untrusted VMs. Therefore, BlueShield is not able to create logical data paths that can be extended to the underlying L2 forwarding fabric. In addition, the solution does not provide any overlay technology to enable logical data paths to be extended and scaled across the data center's infrastructure. The lack of data path isolation by means of tagging and encapsulation technologies potentially increases the risk security threats involving the underlying L2 an L3 networks. For instance, since security policies are applied for broadcast and multicast traffic only, BlueShield becomes susceptible to different unicast-based attacks against the underlying network components. Furthermore, multi-tenant isolation vulnerabilities can be explored by malicious

tenants attempting to send crafted unicast packets to another legitimate tenant VM.

Even though each BlueShield agents is deployed inside a single tenant VM, thereby isolating software vulnerabilities that could be exploited against such software appliances, those agents create a new attack surface in the BlueShield architecture. The reason is that BlueShield agents require local configuration of the location (MAC address) of directory-lookup servers. Therefore, malicious tenants may try to subvert the software agent inside their VMs; they may then gain privileged access to such servers, potentially affecting configuration information from other tenants, or create many and malformed requests aiming to create a DoS situation. Even though BlueShield foresees the replication of the directory-lookup server, its architecture does not promptly provide any support for the isolation of tenant policies inside such software or hardware appliances.

3.3.5 DOVE

Distributed Overlay Virtual Ethernet (DOVE) (COHEN et al., 2013) leverages traditional network virtualization architectures, providing a policy-oriented mechanism to define and to enforce connectivity requirements for multiple tenants in a data center. DOVE builds upon the concept of policy domains, which represent aggregations of network endpoints ruled by a common set of network policies, to provide advanced abstraction for data center network services. Network services that compose policy domains are specified in the form of blueprints, which describe connectivity policies for domain endpoints. Connectivity policies are enforced then by two main components of DOVE's architecture: the DOVE virtual Switch (dSwitch) and the DOVE Policy Controller (DPC). DSwitch is a data plane component deployed on every host server, encapsulating and decapsulating all incoming and outgoing packets for tenant VMs provisioned on the corresponding host. In addition, dSwitch provides support for all packet handling functionalities necessary to implement network services specified by

DOVE blueprints, so it can enforce different connectivity policies. All the policies employed by dSwitches are obtained from the DPC, which manages DOVE's blueprints. The DPC is also responsible for maintaining the correlation between the blueprints and the physical infrastructure, including the physical location of virtual machines and network appliances, which allows it to resolve and validate policy requests.

DOVE implements data path isolation by using different VLANs for different tenants' network traffic forwarded by each dSwitch. In addition, dSwitch provides L3 encapsulation by means of tunneling protocols such as VXLAN, GRE and STT, including and removing outer headers that contain the physical addresses of source and destination dSwitches. Therefore, DOVE provides efficient data path isolation both inside virtualized servers and across the underlying data center network, implementing overlay networks that avoid attacks based on malformed packets, providing isolation of L2 and L3 address spaces, and building upon scalable technologies. However, DOVE still shares network software and hardware appliances, such as dSwitches and the DPC server. As a result, DOVE becomes exposed to the same threats as traditional network virtualization architectures, allowing malicious tenants to exploit potential vulnerabilities in dSwitches to affect the network of other tenants collocated on the same host. In addition, the DPC server is a critical resource that serves the entire DOVE architecture, potentially becoming a preferred target for DoS attacks.

3.3.6 Vagabond

Vagabond (DEY; MISHRA; KULKARNI, 2014) is a network virtualization architecture built upon SR-IOV, providing dynamic assignment of hardware network resources to tenant VMs. Vagabond's design relies on hardware support to SR-IOV technology to connect tenant VMs to the host server's NIC, bypassing the hypervisor network stack and improving performance and isolation of tenant virtual networks. SR-IOV allows multiple VMs to connects to VFs on the physical NIC, which abstract isolated portions

of NIC computing resources, such as memory buffer and time of network processor. In addition, Vagabond tries to preserve the advantages of software-based network virtualization, such as scalability and ease of management, by avoiding the direct assignment of SR-IOV VFs to VMs. Instead, it adopts a connectivity scheme based on a split-driver device model (FRASER et al., 2004; RUSSELL, 2008), a para-virtualization approach that requires a special network driver installed in all VMs (front-end driver). This driver coordinates with the host's network driver on the back-end for packet transmission and reception operations. Vagabond also enables the system to shift between the software-enabled split-driver and hardware-assisted SR-IOV network virtualization techniques by means of Vagabond Controller and Vagabond Switcher. Network traffic from VMs not connected to an SR-IOV VF is also routed through a software bridge that provides connectivity with the physical NIC and underlying network.

When connecting tenant VMs through SR-IOV VFs, Vagabond provides efficient isolation of hardware and software network resources inside host servers. However, the solution does not provide data path isolation, since there is no encapsulation of tenant VMs' traffic. Hence, malicious tenants can send non-encapsulated packets through the software bridge to forwarding elements of the underlying network. In addition, malicious tenants connected via split-driver may perform DoS attacks against the NIC's physical functions, e.g., by generating excessive network traffic. Finally, the lack of data path isolation in Vagabond, combined with the lack of isolation among network software resources, may allow malicious tenants to explore vulnerabilities in the software bridge's implementation when VMs are connected using the split-driver model.

3.3.7 HYVI

Hybrid Virtualization (HYVI) (DONG et al., 2014) adopts a hybrid approach that aims to combine the advantages of software and hardware virtualization technologies. Specifically, HYVI builds upon Xen hypervisor (The Xen Project, 2016) and HAV tech-

nologies, such as SR-IOV and Extended Page Tables (EPT), to create a manageable solution for high-performance and scalable network virtualization in multi-tenant environments. It also introduces optimizations on the Xen hypervisor, for better HAV support, including hybrid management of interrupt vectors consumed by SR-IOV VFs. The HYVI architecture aggregates VFs on SR-IOV-enabled NICs to compose a virtual network among attached VMs, including one additional VF that acts as proxy interface to communicate with the underlying physical network. The proxy interface is, in turn, connected to a service VM responsible for applying filtering policies for the virtual network's incoming and outgoing traffic. The service VM is connected to the underlying physical network by either a separate NIC or an additional VF.

The HYVI architecture can be extended to support multiple sub-networks connected to different service VMs, allowing distributed and isolated policy enforcement for multiple tenants. Such sub-networks should be connected to the service VM through different proxy interfaces, and be properly tagged. This approach considerably improves the isolation of software and hardware resources in multi-tenant environments, reducing the attack surface that can eb potentially exploited by malicious tenants. However, the lack of data path isolation provided by HYVI architecture enables non-encapsulated packets to reach both the VFs and the underlying physical network. Since the forwarding rules applied by the built-in hardware switch provided by SR-IOV are fundamentally based on L2 forwarding tables, malicious tenants may attempt to connect with legitimate tenant VMs by sending crafted non-encapsulated packets to them without passing through the proxy interface. Furthermore, since HYVI architecture does not implement any overlay technology, malicious tenants are able to send non-encapsulated packets across the underlying physical network aiming to exploit vulnerabilities in shared hardware and software network appliances.

3.3.8 Summary

The virtualization architectures described in this section provide a quite comprehensive, albeit not exhaustive, vision of the multi-tenant isolation strategies currently adopted on data center networks. Table 4 summarizes our analysis of the isolation strategies adopted by each network virtualization architecture. We can observe that most network virtualization architectures limit their approach for multi-tenant isolation to the adoption of data path isolation strategies. The techniques for implementing the isolation of addressing spaces and logical data paths vary considerably, ranging from VLANs in traditional cloud architectures to centralized lookup directories in SEC2 and BlueShield. Solutions that build upon a traditional cloud network virtualization architecture, such as DOVE and vShield, are able to enhance the security of the underlying physical networks by implementing tenant overlay networks across the data center. Moreover, the isolation of network software appliances provided by vShield and HYVI architectures demonstrate the efficiency of distributed per-tenant policy enforcement in reducing attack surfaces in a multi-tenant data center. Nevertheless, hardware-assisted virtualization architectures, such as Vagabond and HYVI, are mainly focused on performance benefits provided by SR-IOV technology, failing to provide efficient data path isolation.

3.4 Chapter considerations

As discussed along this chapter, isolation of hardware and software network appliances can significantly enhance the security cloud systems. However, designing a complete isolated network virtualization architecture can be very challenging given multi-tenancy characteristics of public clouds, such as high resource utilization and collocation of mutually untrusted tenants. We call attention to the need for understanding network isolation under the lights of network resources.

Table 4: Isolation strategies adopted in current network virtualization architectures.

	Data Path Isolation	Isolation of Software Resources	Isolation of Hardware Resources
Traditional Cloud	Yes	No	No
NetLord	Yes	No	No
SEC2	Yes	No	No
vShield	Yes	Yes	No
BlueShield	Yes	No	No
DOVE	Yes	No	No
Vagabond	No	No	Yes
HYVI	No	Yes	Yes

The analysis of current network virtualization architectures shows that most of them address multi-tenant isolation from a data path oriented strategy. Although we can identify the adoption of hardware and software isolation strategies, they do not co-exist with complementary strategies such as data path isolation, exposing multi-tenant cloud networks to serious security risks. This issue brings to light the need for novel network virtualization architectures that can provide a more comprehensive isolation of tenant network domains in multi-tenant clouds, considering the three complementary isolation strategies presented in this work: data path isolation, isolation of software resources, and isolation of hardware resources.

In the following Chapter 4, we develop the concept of secure tenant network domains, proposing a new network virtualization architecture intended to provide such comprehensive isolation of tenant network resources in cloud data centers.

4 VIRTUAL NETWORK DOMAINS

Although security vulnerabilities in cloud networking are mostly related to the lack of isolation among tenants' resources, security concerns also apply to the communication between physical hosts. After all, the traffic from the various tenants' VMs communicating are deemed to pass through shared network appliances deployed across the underlying networks infrastructure (e.g., physical, high-end switches connecting different hosts), just like traffic from different users sharing the same links, switches and routers on the public Internet. Therefore, a holistic security solution must enforce security policies throughout the whole virtualized data center network, providing abstractions that allow users to focus on the logical network as if it were actually decoupled from the underlying infrastructure.

In this chapter we explore the concept of virtual network domains, and how it has been defined and applied by network virtualization architectures to enforce security policies in cloud networks. Among the virtualization architectures evaluated in Section 3.3, DOVE stands out with a promising approach to manage network policies in the context of virtual network domains. Moreover, enlightening approaches for distributed policy enforcement are provided by other previously investigated architectures. Then we build upon previous approaches to introduce the concept of Tenant Network Domain (TND), an abstraction of the tenant network resources that operates under specific security and connectivity policies.

4.1 DOVE and Policy Domains

In addition to provide efficient mechanisms to implement data path isolation, and to create a multi-tenant overlay architecture distributed across the cloud data center, DOVE architecture provides an interesting approach for tackling the need for enforcing network policies in virtual networks. To the best of our knowledge, DOVE is one of the few (if not the only) solution in the literature that discusses the concept of policy domains in the context of network virtualization. In summary, DOVE provides a generic architecture for enforcing and verifying the configured policies prior to the network's deployment and instantiation of endpoints, bringing as main contributions:

- *Network management abstraction*: high level policies are translated into network configuration commands for the underlying infrastructure;
- *Policy isolation*: virtual networks belonging to different tenants are logically isolated, so that tenants can define and manage their networks independently of other tenants and of the physical infrastructure characteristics.

A policy domain, as thereby defined, consists of an abstraction that aggregates endpoints with common policy criteria, so that policies are specified and enforced among pairs of policy domains independently of the underlying infrastructure. Some policies criteria, in turn, specifies network services beyond the simple connectivity between network endpoints, and may include security, QoS and monitoring services. Policies are unidirectional and can define network services from a policy domain to another (default policy), as well as from a policy domain to itself. Both default policies and domain policies are enforced by virtual appliances deployed near the network endpoints. The concept of policy domain address important requirements for policy management in cloud systems, such as transparency (given its high-level policies), flexibility (there is no limitation on which kind of policies are applied) and portability of VM (only

the cloud infrastructure needs to be modified). Figure 9 illustrates DOVE's abstraction (called network blueprint) of a traditional network representation which includes three different groups network endpoints (ADC, Data Bases and Web Servers), plus a group of external endpoints, which are represented as four different policy domains. The endpoints are connected through three virtual networks, which provide connectivity between ADC and external endpoints (VN1), ADC and Web Servers (VN2), and Web Servers and Data Bases (VN3). Default policies and domain policies define the connectivity inside and between the different policy domains.

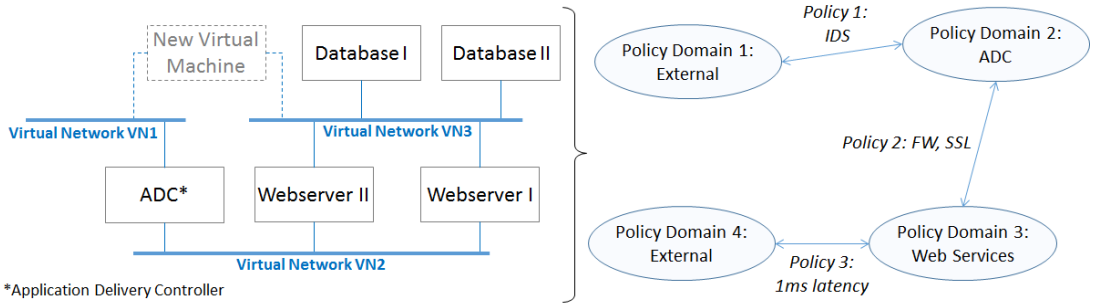

Figure 9: Policy domain representation of a traditional network model. Adapted from (COHEN et al., 2013).

In (COHEN et al., 2013), the authors also describe a concrete implementation of DOVE as a network virtualization architecture integrated with the OpenStack cloud orchestrator (OpenStack, 2016f). Indeed, DOVE builds upon OpenStack to provide an effective framework for managing connectivity and security policies applied to tenant virtual networks. The architecture implementation inherits the data path isolation properties from traditional cloud network virtualization architectures discussed in Section 3.3. Moreover, DOVE architecture centralizes policy resolution in a separate software appliance deployed in OpenStack controller nodes, which responds to distributed policy enforcement mechanisms deployed in multiple OpenStack compute nodes. Figure 10 illustrates the elements of DOVE's architecture, as well as their integration with elements of OpenStack cloud platform. The components of DOVE's architecture are described in what follows.

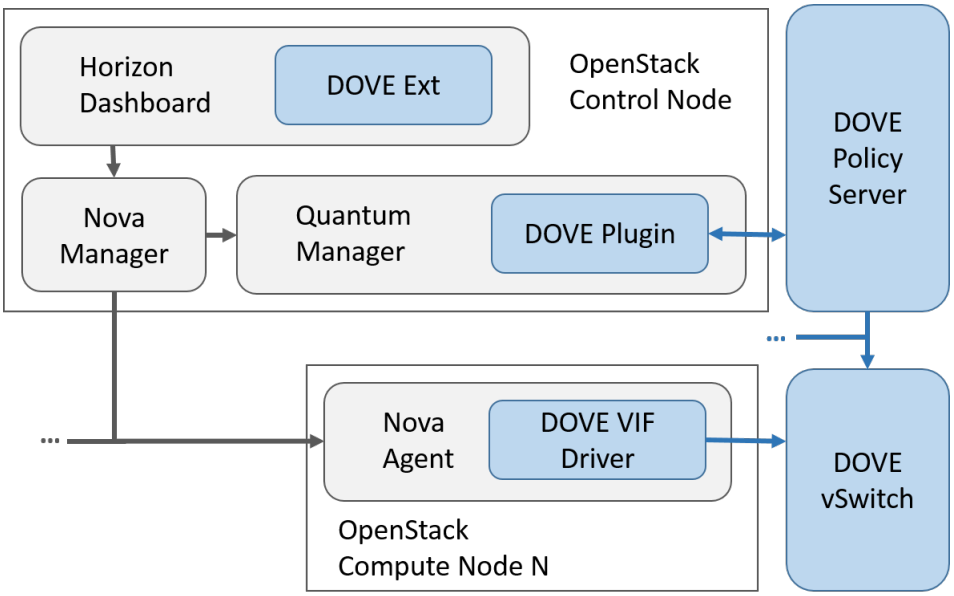

Figure 10: OpenStack DOVE's architecture. Adapted from (COHEN et al., 2013).

- DOVE Ext: An extension of the OpenStack Horizon service to provide user interface (UI) resources for definition and management of DOVE's network policy domains;

- DOVE plug-in: OpenStack network service (Quantum) plug-in that implements the communication interface between Quantum Manager and Dove Policy Server, enabling network policies defined in OpenStack to registered in DOVE's policy server;

- Dove Policy Server: Store tenants' policies and allows conflicts to be verified and resolved. The Policy server is also responsible for deploying network policies on DOVE's virtual switches;

- DOVE vSwitches: Virtual switches which are responsible for handling the packets from all virtual machines running inside a specific compute node, enforcing the policies defined by the Dove Policy Server whenever applicable.

- DOVE Virtual Interface (VIF) Driver: Communicate network management information to DOVE's virtual switches;

DOVE describes a framework to implement policy-driven network domains in cloud computing. Although it can be applied to enforce security policies for isolating the communication between multiple tenant networks, DOVE does not provide any native mechanism intended to isolate software and hardware network resources used by different domains. However, DOVE provides cloud systems with promising control mechanisms to create and manage virtual network domains. As demonstrated further in Chapter 5, DOVE design describes components that can be reused for designing and implementing a better isolated network virtualization architecture. Indeed, components such as the DOVE plugin are required to integrate OpenStack to any mechanism able to enforce network management policies in cloud virtual networks (BARROS et al., 2015b). The concept of policy-driven virtual network domains can also be extended, as demonstrated in Section 4.3, to encompass all three multi-tenant isolation strategies presented in 3.2, as well as to provide the fundamental design goals for a novel security architecture for network virtualization in cloud computing presented in Chapter 5.

4.2 Other Conceptual Influences

Although DOVE architecture provides the main conceptual influence regarding the creation of virtual network domains in cloud systems, the design and implementation of efficiently isolated domains requires the exploration of the benefits provided by different architectural approaches. Different elements (and respective implementations) proposed by the network virtualization architectures analyzed in Section 3.3 can provide us with valuable insights into the design and implementation of isolated network domains. This section presents conceptual influences from previously analyzed network virtualization architectures, other than the policy domain concept explored in Section 4.1. The concepts, as following described, explore the benefits of isolation strategies adopted by the analyzed architectures, aiming to understand their implications on the definition of isolated virtual network domains in multi-tenant cloud

infrastructures.

- **Isolation of L2 addressing space**: The reuse of MAC addresses by different tenants allows better scalability in multi-tenant data centers and can leverage the reuse of VLANs to implement and manage virtual networks from multiple tenants. However, this concept relies on the implementation of tunneling techniques that are able to encapsulated reused MAC addresses (and potentially VLANs) in novel L2 and/or L3 packets to be forwarded across the underlying data center network. Such approach can be seen in NetLord, which proposes a custom semantic for additional L2 and L3 encapsulation, aiming to reduce forwarding tables across the switch fabric.

- **Overlay networks**: The utilization of overlay networks can also improve the scalability of cloud network infrastructure by applying L3 virtualization technologies, such as VXLAN, to encapsulated tenant network traffic forwarded through the underlying data center network. In addition, as discussed in sections 3.1 and 3.3, overlay network can enhance data path isolation, protecting the underlying network infrastructure from malformed packets. Solutions that build upon traditional cloud network virtualization architecture, such as DOVE, are usually technology agnostic, being suitable with multiple tunneling protocols (e.g., VXLAN, NVGRE and STT).

- **Centralized policy control**: Most of evaluated network virtualization architectures that provide policy management capabilities in multi-tenant systems, such as SEC2, BlueShield, relies on centralized control. Although centralized controllers may become a potential target for malicious tenants' attacks, the risks can be mitigated by encapsulating tenant network traffic, for instance, deploying overlay technologies. Furthermore, centralized policy control can be readily integrated with traditional cloud architectures, which usually rely on centralized controllers that orchestrate different cloud services. As discussed in the previous

sections SDN controllers are commonly applied to the orchestration of cloud networking services. The centralized approach provided by SDN has also security implications in cloud network architectures. Therefore, to reduce the impact of SDN-based control in cloud network security, it is necessary to investigate distributed SDN approaches, with isolated control units and policy enforcement locations, which can ensure the security of virtual network domains.

- **Virtualized security appliances**: The virtualization of network security appliances leverages the isolation of software resources in multi-tenant data centers. By encapsulating security appliances inside VMs, as in vShield, network architectures can benefit from the isolation provided by underlying virtualization technologies, such as hypervisor and containers. Virtualized security appliances can be deployed in a distributed fashion across multiple cloud servers, leveraging per-tenant network resource isolation. Such approach can be seen in vShield Edge component. Furthermore, the virtualization of security appliances may simplify the lifecycle management of security services, including installs and upgrades, by leveraging underlying virtualization primitives such as dynamic deployment and VM mobility.

- **Per-tenant network appliances** Network virtualization elements such as vShield Edge and HYVI Service VM enable the deployment of network software appliances that can enforce connectivity and security policies in a per-tenant fashion. This approach provides valuable benefits for the isolation of tenant network resources inside multi-tenant data centers. As previously discussed in sections 3.2 and 3.3, the efficiency of this approach may rely on appropriate encapsulation of tenant's network traffic. Furthermore, HYVI demonstrates encouraging performance benefits on connecting per-tenant network appliances using HAV technologies.

The design of isolated tenant network domains requires efficient implementation of isolated policy enforcement mechanisms. This section describes different concepts that can be applied by novel network virtualization architecture intended to improve the isolation of such mechanisms. The efficient isolation of policy enforcement mechanisms in a per-tenant per-host fashion, enables the configuration and enforcement of network policies in virtual network domains as they would be configured in real private networks. Hereafter, we build upon the conceptual influences previously described to develop the concept of isolated tenant network domains, as well as to define the main design goals of a network virtualization architecture that can provide effective isolation of tenant network resources.

4.3 Tenant Network Domains

Network domains can be seen as 'trust zones', i.e., groups of tenant machines that follow the same set of security policies and that trust each other. In practice, a domain in the cloud would normally correspond to the set containing all VMs from a same tenant; alternatively, it could be a subset of these VMs if that tenant prefers to clearly separate them into different security groups, e.g., separating intranet from extranet VMs, as is the common use case for Demilitarized Zone (DMZ) network arrangements. Even though DOVE's generic concept of policy domains translates to security domains quite easily, its OpenStack implementation as depicted in Figure 10 does not directly translate to a scenario in which resource isolation measures can be easily applied, once it relies on shared virtual switches that can be target of multiple attacks, as discussed in sections 3.1 and 3.3. In addition, DOVE's implementation does not provide specific mechanisms for isolating physical resources used by different policy domains.

To create efficiently isolated tenant network domains, it is necessary to build upon all three network isolation strategies described in Section 3.2, extending the concept of policy-based domains and merging promising techniques proposed by previous net-

work architectures to build a novel cloud network virtualization approach. To achieve this goal, we introduce the concept of Tenant Network Domain (TND), which encompasses (1) a set of isolated physical and logical network resources owned by a tenant, as well as (2) a set of connectivity and security policies applied to these resources. In other words, the conceptual model of isolated TNDs deployed in multi-tenant cloud data centers, in which tenants' VMs are distributed among multiple physical hosts, should meet following two design goals:

1. The communication of VMs running in the same physical host (intra-host communication), but pertaining to different TNDs, should be physically and logically isolated by going through different virtual network appliances and consuming different resources from the physical network infrastructure.

2. The communication of VMs running in different physical hosts (inter-host communication), but pertaining to the same TND, should be logically isolated from the communication involving other TNDs as the packets travel from one host to another.

We notice that, even though physical isolation in inter-host communications would be even better in terms of security, this requirement is clearly impractical in high density network environments such as multi-tenant cloud data centers or the Internet. Hence, instead of having separated physical networks to isolate tenant's inter-host communications, the best that can be achieved is to make use of overlay technologies such as VXLAN to isolate tenant's logical data paths, and logical data path from underlying network data paths. In addition, data path isolation may include security mechanisms traditionally applied to VPNs, such as IPsec (SEO; KENT, 2005), to provide confidentiality, integrity and authentication services for tenants' virtual networks. This approach is explored in details in Chapter 5, where we describe a practical implementation for efficient data path isolation in a secure network virtualization architec-

ture. Nevertheless, achieving physical isolation of network resources inside virtualized server can be far more challenging. Indeed, multiplexing and demultiplexing shared physical resources among different tenants and applications are intrinsic to server virtualization software, such as hypervisors. However, approaches based on HAV technologies, such as in Vagabond and HYVI, have proven to be very efficient in isolating portions of NIC resources consumed by different VMs or group of VMs. Therefore, HAV technologies such as SR-IOV can be applied together with efficient data path isolation strategies to provide great end-to-end isolated logical links for tenant VMs.

Although SR-IOV provide numerous benefits on isolation and performance by allowing tenant VMs to bypass shared hypervisor network stack, it introduces new challenges in policy management and policy enforcement for multi-tenant clouds. VMs directly connected to the underlying data center network through SR-IOV VFs rely on external network appliances to implement, for example, connectivity and security services. Modern cloud platforms, such as OpenStack and CloudStack, usually perform policy enforcement near virtualized endpoints, in this case, tenants' VMs (SABHARWAL; SHANKAR, 2013; DENTON, 2016). This approach enables cloud orchestrator to provide connectivity and security services to tenant VMs without relying on external traffic, implementing tenant virtual networks by means of software network appliances, usually virtual bridges and switches. Such approach can also be seen in network virtualization architectures such as DOVE and vShield. To provide isolated per-tenant enforcement of connectivity and security policies, as well as to preserve the isolation of hardware resources provided by HAV technologies, specifically built-in NIC virtualization, we introduce a new control element named Tenant Network Controller (TNC). Figure 11 illustrates the design of TNC and how it provides per-tenant policy enforcement isolation on virtualized server hosts.

TNC provides per-tenant policy enforcement by isolating the software appliances that implement connectivity and security services for tenant VMs belonging to the

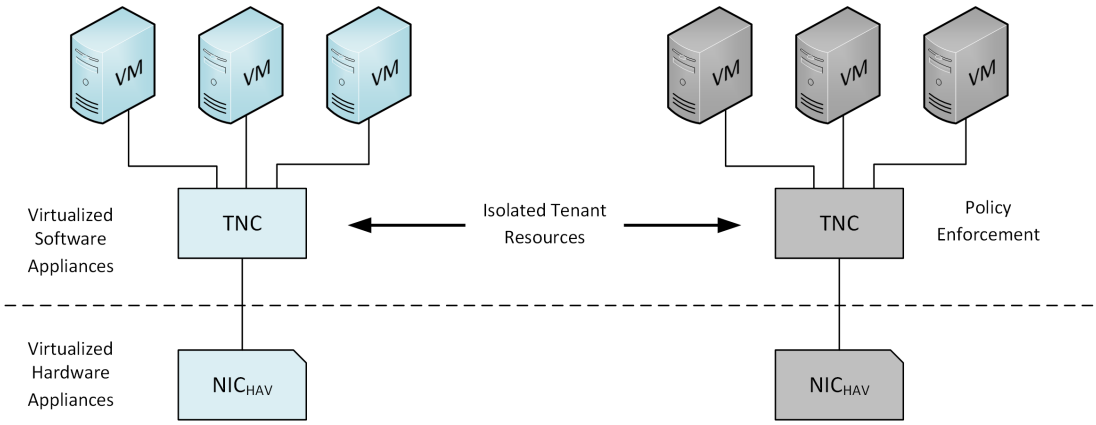

Figure 11: Policy enforcement isolation provided by TNC.

same TND. As discussed in Section 3.1, this approach can avoid security attacks against network appliances shared between malicious and legitimate tenants of the cloud. The TNC aggregates connectivity and security services in a novel virtualized network appliance that implements communication for TND VMs provisioned at one or more physical servers inside the cloud infrastructure. While connectivity services implement the virtual links necessary for the communication among the different VMs of a TND, security services may vary from stateless firewalls to IDS and Intrusion Prevention System (IPS) systems.

As a policy enforcement mechanism responsible for controlling tenant network traffic across the data center network, the TNC appliance should be decoupled from tenant VMs and controlled exclusively by CSPs. Communication between TND VMs that involves different physical hosts are performed through the hardware-assisted virtualized NIC, represented in 11 as NIC_{HAV}. The group of TND resources, including VMs, TNC and virtual NICs, deployed in the same physical host, is called TND slice. In addition, for scalability purposes, a TND slice may contain more than one TNCs, being mandatory the minimum of one TNC to configure a TND slice.

The multiple TNCs distributed across the cloud data center require a policy management framework that can be readily integrated with cloud orchestration systems. As previously discussed in Section 4.2, centralized policy management mechanisms

have proven to be compliant with cloud orchestration systems, leveraging its control capabilities over the complete virtualized infrastructure. In Section 2.2.4, we present SDN as a prominent control framework to implement and manage virtualized networks in cloud data centers. In addition to provide centralized control over distributed network appliances (virtual and physical), modern SDN controllers provide readily integration with cloud platforms. Furthermore, SDN can leverage the implementation of security functions across virtual and physical network infrastructure (YOON et al., 2015). Following, we summarize the design goals of a network virtualization architecture intended to provide a practical implementation of the TND conceptual model here described. These design goal will guide the implementation of the TND Architecture along the Chapter 5.

1. The TND architecture should provide efficient isolation of tenants' network resources, including logical data paths, software network appliances and hardware network appliances, among different TNDs;
2. A TND is composed necessarily by one or more TND slices;
3. The TND architecture should deploy one or more TNC appliances per TND slice;
4. The TNC appliance provides connectivity and security services for TND VMs deployed in a single TND slice;
5. The TNC appliance should be decoupled from tenants' VMs and controlled exclusively by the CSP;
6. Policies to be enforced by TNCs should be managed and deployed through a centralized policy controller, named TNC controller.

The network abstraction provided by the TND model is illustrated in Figure 12, which shows a simple Demilitarized Zone (DMZ) network with three security do-

mains: internal servers, external servers, and workstations. In this example, the external servers and workstations follow different security policies, but are not physically or logically isolated from each other, as they are all under the TND B. Hence, the corresponding VMs can share the same virtual switch and NIC ports if collocated in a given physical host, and the corresponding policies are then enforced as needed in the connections points between those machines (e.g., the switch's ports). The external servers, on the other hand, are placed on TND A. Therefore, even if VMs from TND A are collocated in the same physical host as those from TND B, they should be isolated in such a manner that they do not share the same logical and physical network resources.

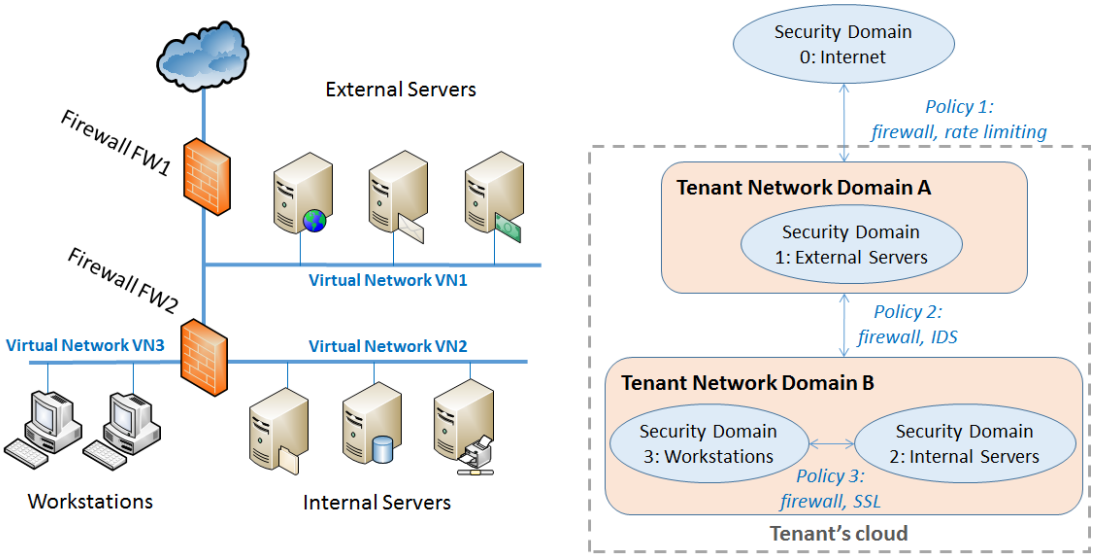

Figure 12: TND abstraction of real network deployments.

The design concepts introduced by TNDs, as well as their implementation by the TND architecture, may result in significant impact over different characteristics of multi-tenant clouds. Even though the ultimate goal of the TND architecture is to provide complete isolation of tenant network resources in cloud deployments, thereby enhancing the security of current network virtualization architectures, we call attention to additional requirements that should be considered when implementing the TND architecture. These secondary requirements are described as follows.

- *Transparency*: The burden of configuring the underlying cloud's network in a secure manner should not be placed on the tenants. Therefore, tenant should be provided with high-level policy configuration, which should be then enforced by the CSP through the TND architecture.

- *Flexibility*: Tenants should be able to deploy the same security services on its TNDs as they would be able to deploy in private data centers.

- *Portability*: TND architecture should support the deployment of tenant VMs regardless of the OS and applications installed, providing full abstraction of infrastructure resources.

- *Mobility*: The migration of one or more VMs of a TND from one physical host to another should not only be possible, but the overhead incurred in such operations should be minimized by the TND architecture.

- *Performance*: TND architecture should not significantly impact the communication throughput between VMs belonging to the same TND beyond the impact inherent to the implement of required security policies.

- *Scalability*: TND architecture should take into account the costs involved when scaling the number of VM in a host, whether those machines are from the same or different TNDs.

- *Availability*: TND architecture should avoid creating elements that can become single points of failures for one or multiple TNDs, mitigating DoS attacks other availability issues.

- *Accountability*: TND architecture should allow CSPs to easily account and charge for tenants' consumption of network resources and security services (e.g., physical isolation, firewall, IDS/IPS).

- *Maintainability*: Maintenance operations over one TND network resources, should have little or no impact on the operation of other TNDs.

These additional requirements are non-functional characteristics that may affect both the quality of the services provided by CSPs and the manageability of cloud infrastructures. Therefore, aiming at leveraging the adoption of the TND architecture, it is necessary to understand the impact of the resource isolation techniques proposed by TNDs on the characteristics of current cloud network architectures.

4.4 Chapter Considerations

This chapter introduces the concept of Tenant Network Domain (TND), a network abstraction built upon the assumption of efficient isolation among tenants' network resources. TND builds upon the concept of policy domains introduced by DOVE, extending it to address connectivity and security policies inside and among isolated domains. Our proposal enhances the security of policy domains by isolating the policy enforcement appliances employed by different tenants, enabling the creation of completely isolated tenant networks. Despite the security benefits discussed in Chapter 3, the isolation of tenant networks may enhance other characteristics of cloud networks, described in this chapter as secondary requirements for the TND architecture.

In addition, we describe other conceptual influences provided by the analysis of existing network virtualization architectures described in Chapter 3. This analysis provides us with different virtualization concepts and technologies that can be applied to implement the isolation of tenant network resources and the policy control framework required by the TND architecture. The application of such concepts and technologies is detailed in Chapter 5, where we describe the design and implementation of the TND architectural components and correspondent operation.

Moreover, we describe the main design goals of the TND architecture, a network

virtualization architecture that can enforce isolation of network resources in cloud computing, providing efficient and secure policy enforcement mechanisms for cloud tenants. These design goals are addressed as requirements for the practical implementation of the TND architecture presented along the Chapter 5. In addition, we take into account the secondary requirements described in this section, aiming to improve the suitability of our proposal to production deployments.

5 TND ARCHITECTURE

In this chapter we propose a novel network virtualization architecture for cloud data centers, aiming to provide efficient isolation of tenants' network resources, thereby enhancing the security of cloud deployments. Our architecture, hereinafter referred as TND architecture, addresses the isolation requirements established for TNDs by applying different network virtualization technologies presented in previous chapters. TND architecture implementation builds upon the concepts and technologies provided by cloud computing, SDN, NFV and HAV. In particular, we build upon SDN and NFV to implement an efficient framework for managing and enforcing network policies on TNDs. In addition, we apply HAV technologies to provide isolation of hardware network resources while leveraging the performance of key NFV appliances, such as TNCs. In what follows we present the technologies applied to the implementation of the TND architecture, as well as the integration model which leverages the synergy among them. A layered view of the TND architecture and correspondent technologies applied in each layer is provided by Figure 13.

We can observe from Figure 13 that TND architecture builds upon traditional SDN architectural layers, with the addiction of the HAV layer. As previously discussed in Section 2.1.2, the application of SR-IOV technology in the HAV layer intends to enhance the isolation of hardware network resources among cloud tenants and to reduce the impact of security mechanisms over the performance of VM communication. In the sections 5.1 and 5.2, we describe our approach to implement effective isolation TNDs. Then we describe in Section 5.4 a framework for deployment and enforcement

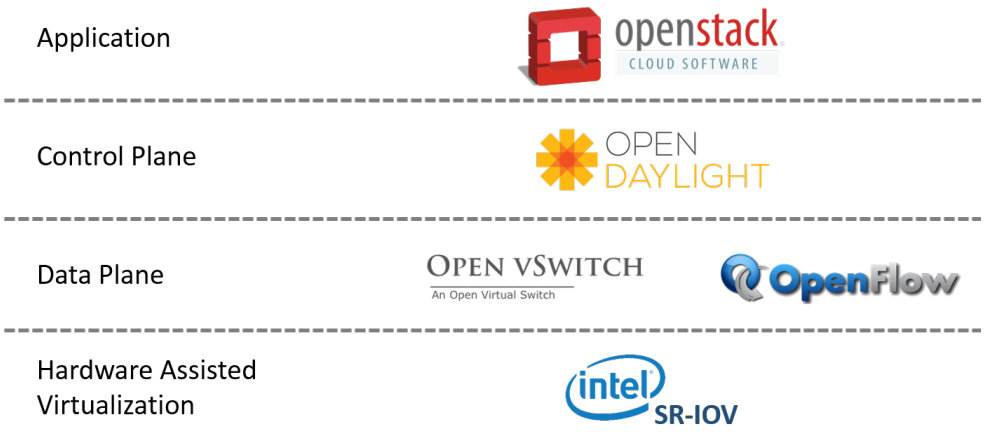

Figure 13: Layered view of the TND architecture and correspondent technologies.

of network policy in isolated TNDs. Finally, in Section 5.5, we describe the software components necessary to integrate the TND architecture to an SDN-enabled cloud networking infrastructure provided by OpenStack and OpenDaylight.

5.1 Data Paths Isolation

This section describes the implementation of isolated logical data paths inside the TND architecture. Current OpenStack network implementation provides data path isolation by means of VLAN tagging mechanisms and tunneling protocols such as GRE and VXLAN (DENTON, 2016). However, these mechanisms are not deployed with the ultimate purpose of providing isolation among tenant networks. Although current network deployments involve the utilization of different VLANs to isolate tenant virtual networks managed by distributed tenant switches, GRE or VXLAN tunneling protocols are usually applied to encapsulate the traffic generated by VMs inside a single physical host, regardless of the tenant who owns the traffic. Such approach, illustrated in Figure 14 has also implications on the cloud network scalability, since the number of virtual networks is limited by the number of VLANs supported.

VXLAN is a network virtualization technology developed to address the scalability issues of VLANs in cloud deployments (MAHALINGAM et al., 2014). VXLAN is able

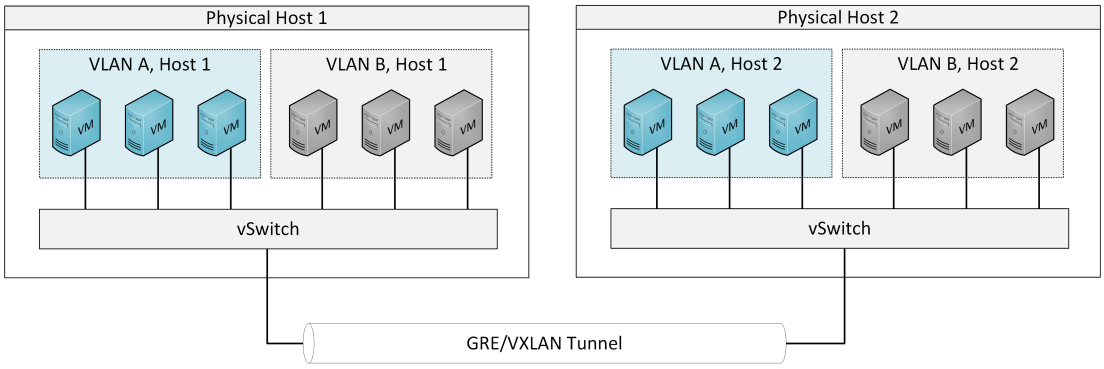

Figure 14: Shared OVS bridges for tenant virtual networks.

to encapsulate layer 2 Ethernet frames within layer 4 UDP packets, creating logically isolated tunnels between endpoints called VXLAN Tunnel Endpoint (VTEP). In addition, VXLAN extends the capabilities of VLAN tagging mechanism from 4094 to up to 16 million virtual networks supported, implemented through a 24-bit identifier embedded into VXLAN encapsulation. Despite virtual switch appliances provide full support for VXLAN encapsulation, the utilization of VXLAN tunnels to isolate tenants' data paths inside the shared OVS bridges deployed in OpenStack hosts may be challenging. VXLAN ports in shared OVS bridges are usually configured as host VTEPs, which aggregate the communication of hosted VMs with other physical hosts by means of VXLAN tunnels (Open vSwitch, 2016b). One possible approach for implementing data path isolation among tenants is to replace host VTEPs for VTEPs configured on each tenant VM, using different VXLANs for different tenants. This approach generates configuration overheads and lack of transparency for cloud services. Alternatively, an efficient solution to provide data path isolation and network scalability by means of VXLAN is to isolate the OVS bridges consumed by the different tenants collocated in the same physical host, configuring VXLANs tunnels that connect the tenant OVS bridges distributed across multiple TND slices. This approach, illustrated in Figure 15, can efficiently address the data path isolation requirements imposed by the TND architecture.

In our TND architecture, isolated OVS bridges are implemented by means of

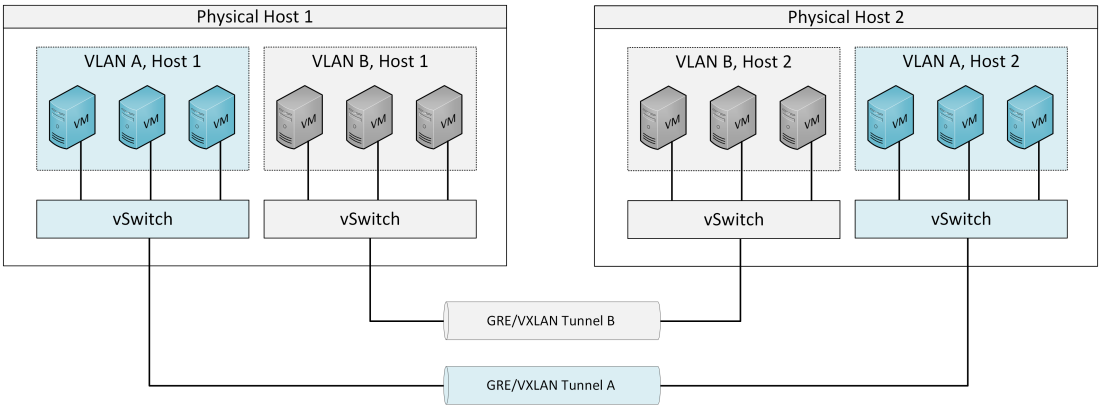

Figure 15: Isolated OVS bridges for tenant virtual networks.

TNCs, as detailed further in Section 5.2. Moreover, applying VXLAN to connect TNC OVS bridges distributed along TND slices allow us to create TND overlays across the underlying cloud infrastructure. TND overlays, as illustrated in Figure 16, can enhance cloud network manageability by providing high level abstraction of TND networks, which can be readily deployed in the underlying cloud network infrastructure using different VXLAN Network Identifiers (VNIs).

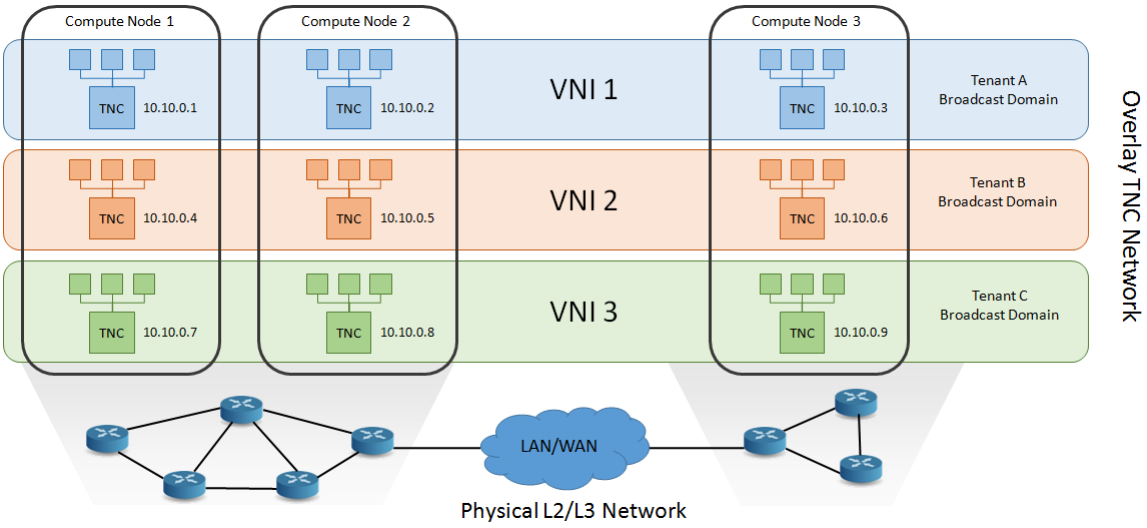

Figure 16: VXLAN overlay network in TND architecture.

Aiming at enhancing data path security on TND architecture and, at the same time, to take advantage of the scalability features provided by VXLAN, we apply IPsec on the TND architecture using the transport mode of operation. IPsec transport mode allows to provide encryption and authentication for data exchanged between

TND VTEPs without modify VXLAN IP headers, taking advantage of routing services such as multicasting (IETF, 1989). Inside the TND architecture, multicast allows TND VTEPs deployed in distributed TND slices to send IP packets to all other VTEPs in the same TND with a single transmission using a multicast address, thus defining an isolated broadcast domain. Figure 17 illustrates how IPsec security mechanisms are applied to the VXLAN encapsulation to provide secure communication between isolated OVS bridges implemented through TNC appliances deployed in two TND slices.

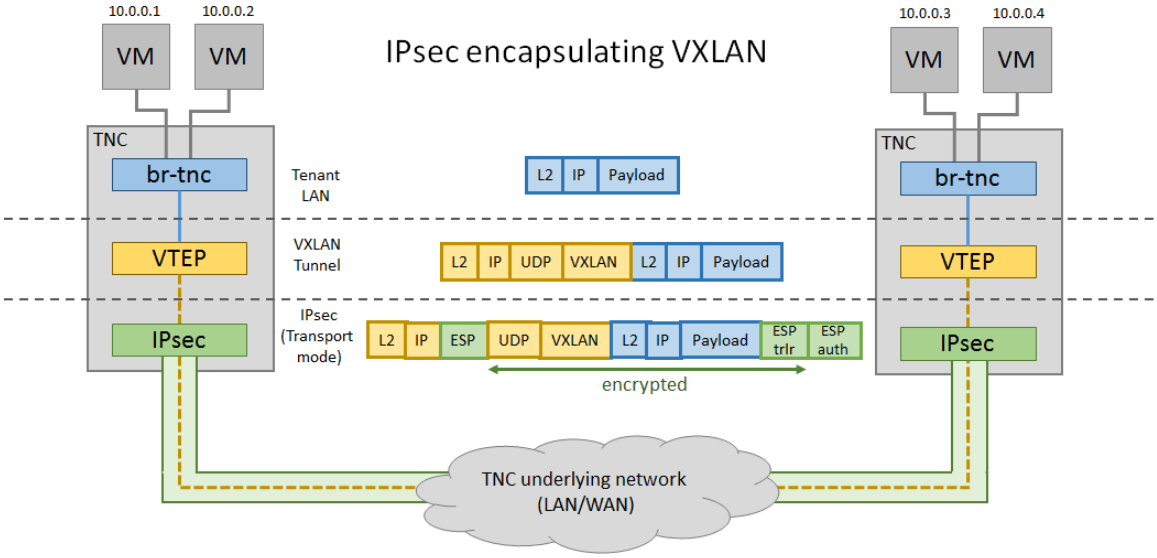

Figure 17: IPsec encapsulation of VXLAN packets in the communication between TND slices.

The design builds upon OVS bridges commonly deployed on OpenStack cloud setups (BARROS et al., 2015b). As introduced in Section 4.3, the TNC appliance aggregates the connectivity services provided by the TND architecture, thus implementing the communication among VMs deployed in the same or in different TND slices. In Section 5.2 we describe the implementation of the TNC appliance, and how it can be leveraged by SR-IOV to address the TND requirements for isolation of software and hardware network appliances.

5.2 Isolation of Hardware and Software Resources

Although efficient data path isolation can improve the security of virtual networks, we still need to protect software and hardware network appliances from attacks that result from co-residency in multi-tenant clouds. To accomplish this goal, the entry point of each TND is controlled by a component named Tenant Network Controller (TNC), which provides TND VMs with the following properties:

- A virtual switch appliance that is shared among the TND's VMs in a given TND slice, enforcing connectivity and security policies for packets originated from the corresponding TND or from external sources (e.g., other TNDs or the Internet);
- A direct connection to a SR-IOV VF in the physical host's NIC, ensuring the packets sent/received by VMs from a given TND are physically isolated from packets coming from/to another TND, even if the corresponding VMs are hosted at the same physical server;
- A OVS bridge VTEP that provides VXLAN encapsulation for the communication with other TNCs, creating a TND overlay network with isolated broadcast domain and identified by a unique VNI;
- IPsec encapsulation for the communication with other TNCs or to external networks, ensuring confidentiality, integrity and authentication to logical data paths across the cloud underlying network.

The IPsec tunnels established between TNCs that belong to the same TND must be automatically established, creating the impression that all the corresponding VMs are in the same subnet despite eventually being in different physical hosts. In addition, if a TND should be connected to external networks (e.g., a service inside the cloud provider's infrastructure or the Internet), another IPsec tunnel is also configured to one or more of the cloud's border routers. This is slightly different from what happens

when a tenant explicitly creates separate TNDs for different groups of machines under their responsibility, but want that the VMs from those TNDs to be able to communicate with each other. In this latter case, the TNC from the different TNDs may connect via IPsec tunnels dedicated to this purpose, instead of seeing each other as external networks and connecting via the IPsec tunnels established with the corresponding border router.

Even though TNC is a general concept, a concrete implementation could take the form of a software or hardware network appliance that aggregates and enforces network policies for VMs in a corresponding TND slice. Aiming to obtain a feature-rich implementation, as well as to take advantage of the isolation properties inherent to full virtualization platforms, we implement TNC component in form of a virtual machine endowed with OVS and, eventually, additional security software not necessarily available as internal components of commodity virtual switches and bridges (ETSI, 2012; MIJUMBI et al., 2016). This approach can actually provide better isolation of the TNDs' software resources, as then each TNC would have its own dedicated VM, and improve manageability by allowing configuration changes to be applied locally and independently of other tenants' resources. On the downside, the full virtualized VM-based approach is likely to provide lower communication performance than simple bridges, since the latter does not involve an extra virtual machine network stack. Hence, an actual implementation of the TNC concept requires careful consideration of those trade-offs, as the best approach depends on the target application's requirements.

Furthermore, our TNC implementation makes use of SR-IOV technology to provide the TNC VM with direct access to the underlying cloud network as illustrated in Figure 18. These approach allow us to extend the isolation provided by TNC VM to the hardware layer, in this case, to the physical NIC. Indeed, SR-IOV allows a TNC VM to communicate with other TNCs on different TND slices bypassing the hypervisor and other network stacks, benefiting from the performance and isolation features provided

by this technology (DONG et al., 2012; KOURTIS et al., 2015). The TNC VM connects to the physical NIC through a VF, which provides an abstraction of the isolated NIC resources allocated to a single VM.

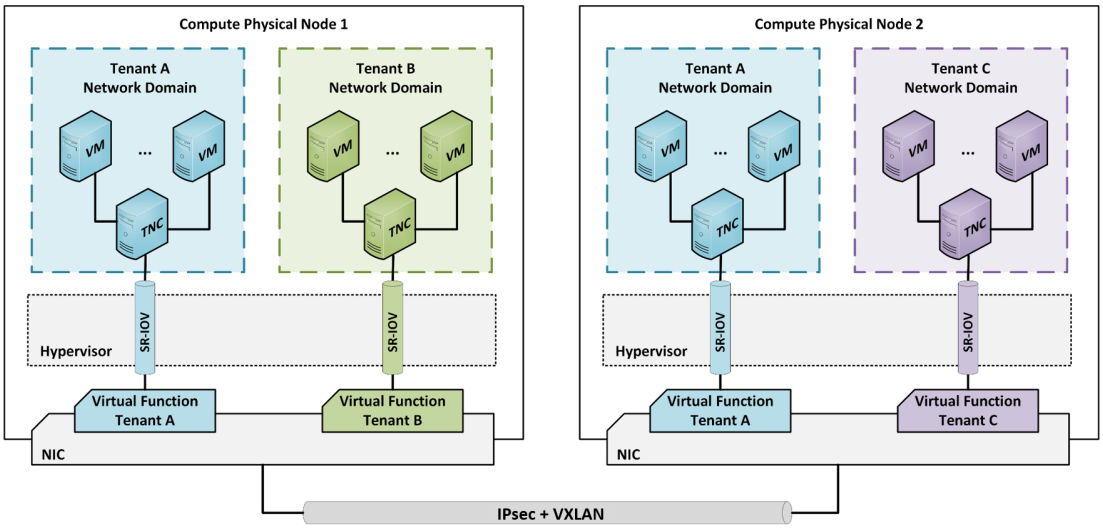

Figure 18: TNCs connected to SR-IOV VFs.

The communication between two TNCs deployed in the same TND slice is implemented through the SR-IOV built-in virtual switch, which implements communication between NIC VFs with minimal performance impacts. The performance impact is caused by the need for transferring the packet to the physical layer before being able to forward it to the destination TNC, regardless if it is located in the same physical host. As additional drawback, mapping SR-IOV VFs to TNC network interfaces require additional configuration, which may impact provisioning and migration processes for TNC VMs. It is worth noticing that this drawback can be minimized by applying automated and/or pro-active provisioning of TNC VMs.

The TND concept is illustrated by Figure 19, which shows 4 different TNDs (A, B, C and D) distributed among four physical hosts (1, 2, 3 and 4). Domain A has VMs placed on Hosts 1 and 4; for this reason, these VMs are connected via two TNCs (A1 and A2), which are themselves connected using a common, intra-domain IPsec configuration 'IPsec A:A'. The VMs from Domains A and B are owned by the same tenant

but follow different security policies; hence, to facilitate management and improve isolation of their logical and physical resources, that tenant preferred to place them into different TNDs. For example, going back to the DMZ scenario depicted in Figure 12, domain B contains a web server that communicates with external clients, while domain A contains workstations and also internal servers that support the said web server (e.g., providing database and business intelligence services). For better performance and isolation, the TNCs from both domains have their corresponding virtual functions connected using dedicated IPsec tunnels (named 'A:B' in the figure), which may not even be necessary if their virtual functions are on the same port of the NIC (which is indicated with the dotted line connecting VF_{A1} and VF_{B1}). Since the VMs from those domains need to communicate with each other, their TNCs must implement connectivity and security policies that allow the passage of packets between them. Domains C and D, on the other hand, only communicate internally (again, using intra-domain IPsec configurations) and with external networks via a border router (a connection that, for conciseness, is not shown in the figure), but have their network resources isolated from each other and from TNDs A and B.

This section describes the TNC appliance and how it can be leveraged by SR-IOV technology to provide isolation of software and hardware network appliances in cloud virtualized servers of the TND architecture. Along with the data path isolation strategy described in Section 5.1, TNC implementation can achieve all the design goals established for the TND architecture in Section 4.3.

5.3 Communication Use Cases

This section is intended to illustrate the packet flow across the TND architecture. For this purpose, we describe common use cases for communication among VMs deployed in one or more TND slices across the cloud infrastructure. These use cases aim at providing details on the operation of fundamental network services, such as ARP

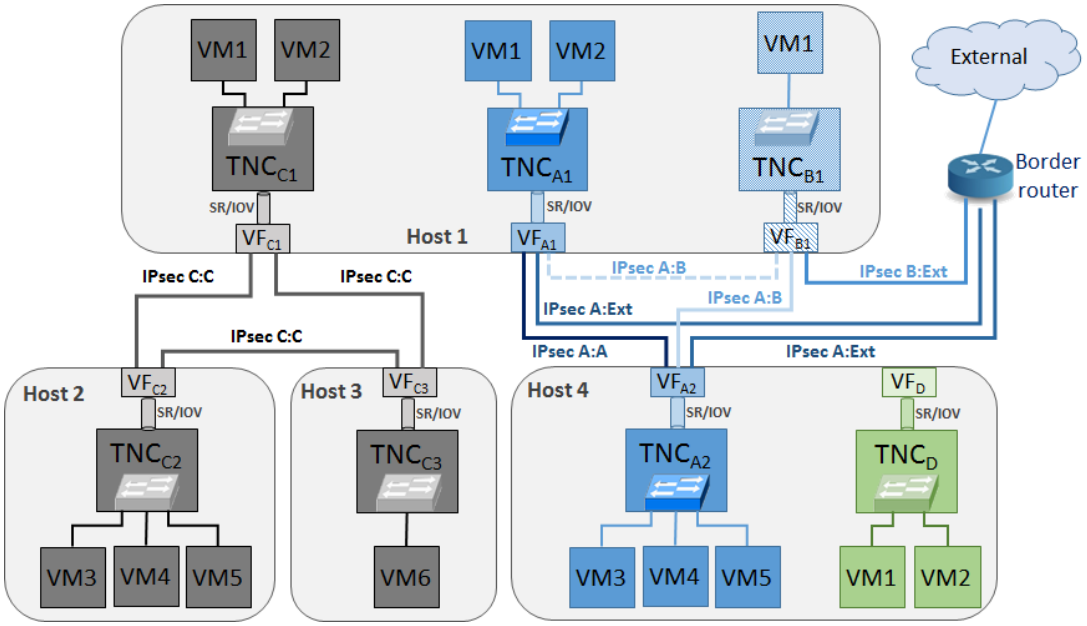

Figure 19: Overview of the TND architecture: TNDs A and B (blue) are managed by the same tenant, but follow different sets of security policies when communicating with each other or with external networks; VMs from TNDs C (gray) and D (green) are controlled by different tenants (their connection to external networks are omitted for conciseness).

abstraction and Internet access.

5.3.1 ARP Request in Different TND Slices

To illustrate how VMs can communicate in an inter-host environment, we can evaluate how ARP messages are distributed in the architecture. For this purpose, we consider a scenario with the following characteristics:

- Two VMs are instantiated: VM1, with IP address 10.0.0.1 and MAC address AA::AA , and VM2, with IP 10.0.0.2 and physical address BB::BB;
- Two TNCs are instantiated: TNC1, to which VM1 is attached, and TNC2, to which VM2 is attached. Each TNCn is then connected to a VF on IP address 192.168.10.n;
- The TNCs are configured to use VXLAN with a network identifier (VNI) of 100, and the remote multicast IP address ”remoteIP = 239.1.100”;

- Finally, the TNCs (or, more precisely, their IPsec modules associated to their VXLAN tunnel endpoints – VTEPs) are each configured in Encapsulating Security Payload (ESP) with authentication, using the following IPsec security associations (SAs): a pair of SAs for their unicast communications, having as source and destination IPs the addresses ”localIP = 192.168.10.1” and ”remoteIP = 192.168.10.2” (and vice-versa), hereby denoted SAnu for TNCn; and one additional SA pair for their multicast communications, both having as destination IP their common multicast address ”remoteIP = 239.1.100” but one having as source address the VTEP’s own address (i.e., ”localIP = 192.168.10.n” for TNCn), employed for sending multicast packets, and the other having as source address their counterpart’s VTEP address (i.e., ”localIP = 192.168.10.(n+1 mod 2)” for TNCn), which is used for the reception of multicast packets, which are hereby denoted SAnm.

Then, the sequence of steps performed by the architecture when processing an ARP request between VM1 and VM2 in Figure 20 are:

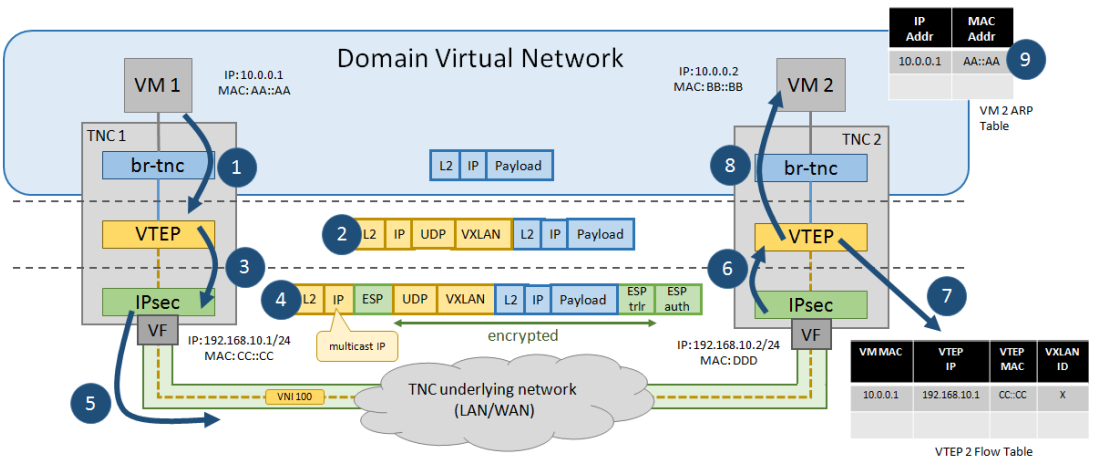

Figure 20: VXLAN-based inter-host ARP: request.

1. VM1 sends an ARP broadcast to tenant virtual network to find out the VM2 MAC address;

2. VTEP on TNC1 encapsulates the Ethernet broadcast packet into a VXLAN UDP header with multicast address 239.1.1.100 (multicast group address associated with the network identifier VNI 100) as the destination IP and TNC1 address 192.168.10.1 as the source IP;

3. VTEP on TNC1 delivers the VXLAN packet to the IPsec module on TNC1;

4. The IPsec module on TNC1 inserts into the packet the ESP header and the ESP trailer corresponding to SA1m, encrypts the transport segment and adds the ESP authentication field after ESP trailer;

5. The physical network delivers the multicast packet to the TNCs that joined the multicast group address 239.1.1.10;

6. The IPsec module on TNC2 validates the ESP authentication, decrypts the transport segment and removes ESP trailer and ESP header. If the authentication succeeds, the IPsec module on TNC2 delivers the VXLAN packet to the VTEP on TNC2;

7. The VTEP on TNC2 receives the encapsulated packet. Based on the outer and inner header, it makes an entry in the forwarding table that shows the mapping of the VM1 MAC address and the VTEP IP on TNC1. VTEP on TNC2 also checks the VNI in the external header to decide if the packet has to be delivered on the TNC2 or not;

8. The VXLAN packet is de-encapsulated and delivered to the VM2 through the br-tnc on TNC2;

9. VM2 inserts an entry in the ARP table mapping the VM1 MAC address and the VM1 IP address.

When the ARP request finally reaches VM2, its response is treated as follows (see Figure 21):

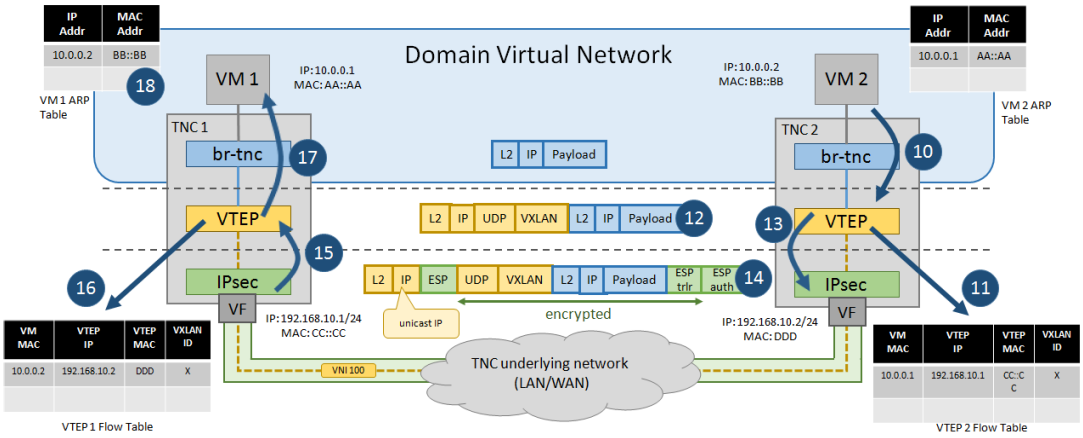

Figure 21: VXLAN-based inter-host ARP: response.

10. VM2 responds to the ARP request by sending a unicast packet to VM1 with Destination Ethernet MAC address as AA::AA;

11. After receiving the unicast packet, the VTEP on TNC2 performs a lookup in the forwarding table and gets a match for the VM1 MAC address AA::AA. The VTEP on TNC2 now knows that to deliver the packet to VM1 it has to send it to VTEP on TNC1 with the destination IP address 192.168.10.1;

12. The VTEP on TNC2 creates a VXLAN unicast packet with destination IP address as 192.168.10.1 and delivers it to the IPsec module on TNC2;

13. The IPsec module on TNC2 inserts into the packet sent by VM2 the ESP header and the ESP trailer corresponding to SA2u, encrypts the transport segment and adds the ESP authentication field after the ESP trailer;

14. The IPsec module on TNC2 delivers the encrypted and authenticated packet to the TNC1 through the underlying network;

15. The IPsec module on TNC1 validates the ESP authentication, decrypts the transport segment and removes ESP trailer and ESP header. If the authentication succeeds, the IPsec module on TNC1 delivers the VXLAN packet to the VTEP on TNC1;

16. The VTEP on TNC1 receives the encapsulated packet. Based on the outer and inner header, it makes an entry in the forwarding table that shows the mapping of the VM2 MAC address and the VTEP IP on TNC2. VTEP on TNC1 also checks the VNI in the external header to decide if the packet has to be delivered on the TNC1 or not;

17. The VXLAN packet is de-encapsulated and delivered to the VM1 through the br-tnc on TNC1;

18. VM1 inserts an entry in the ARP table mapping the VM2 MAC address and the VM2 IP address;

19. The subsequent communications between VM1 and VM2 should involve only unicast packets being transferred from/to TNC1 and TNC2.

5.3.2 ARP Request in the Same TND Slice

To illustrate how VMs can communicate in an intra-host environment, we can evaluate how ARP messages are distributed in the architecture. For this, we consider a scenario with the following characteristics:

- Two VMs are instantiated: VM1, with IP address 10.0.0.1 and MAC address AA::AA, and VM2, with IP address 10.0.0.2 and physical address BB::BB;
- One TNC is instantiated and attached to VM1 and VM2. The TNC is then connected to a virtual function;
- The VXLAN tunnel and IPsec protocol is configured as described in Section 5.1.

The sequence of steps performed by the architecture when processing an ARP request between VM1 and VM2 are described below:

1. VM1 sends an ARP broadcast to the TND to find out the VM2 MAC address.

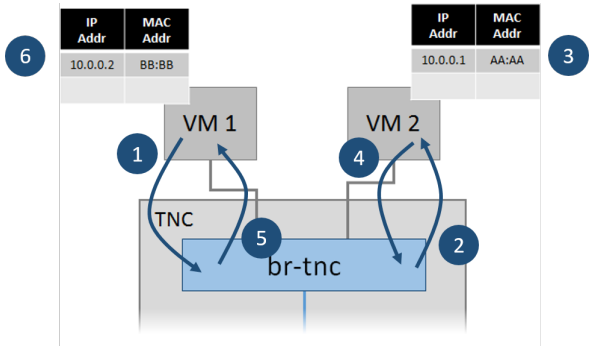

Figure 22: Intra-host ARP in a Domain.

2. The TNC's bridge (br-tnc) forwards the broadcast packet to all of its active ports.

3. VM2, after receiving the request, inserts an entry in the ARP table mapping VM1's destination MAC and IP addresses.

4. VM2 responds to the ARP request by sending a unicast packet to VM1 with Destination Ethernet MAC address as AA::AA.

5. The TNC's bridge (br-tnc) forwards the unicast packet to the correct port connected to VM1.

6. VM1 inserts an entry in its ARP table mapping VM2's destination MAC and IP addresses.

7. The subsequent communications between VM1 and VM2 should involve br-tnc internal traffic only.

5.3.3 Internet Access

To illustrate how VMs can communicate with external networks, we can evaluate the message flow throughout the distributed elements of the architecture. For this purpose, we consider a scenario with the following characteristics:

- VM1 is instantiated with IP address 10.0.0.1 and MAC address AA::AA;

- Two TNCs are instantiated: TNC1, to which VM1 is attached, and TNC1 Net, which contains an external bridge (br-ext) and a Neutron qRouter element to provide external connectivity for the TND;
- The TNCs are configured to use VXLAN and IPsec tunnels as described in Section 5.1.

The message flow performed by the architecture when exchanging packets between a TND VM and the Internet is illustrated in Figure 23 and described as it follows.

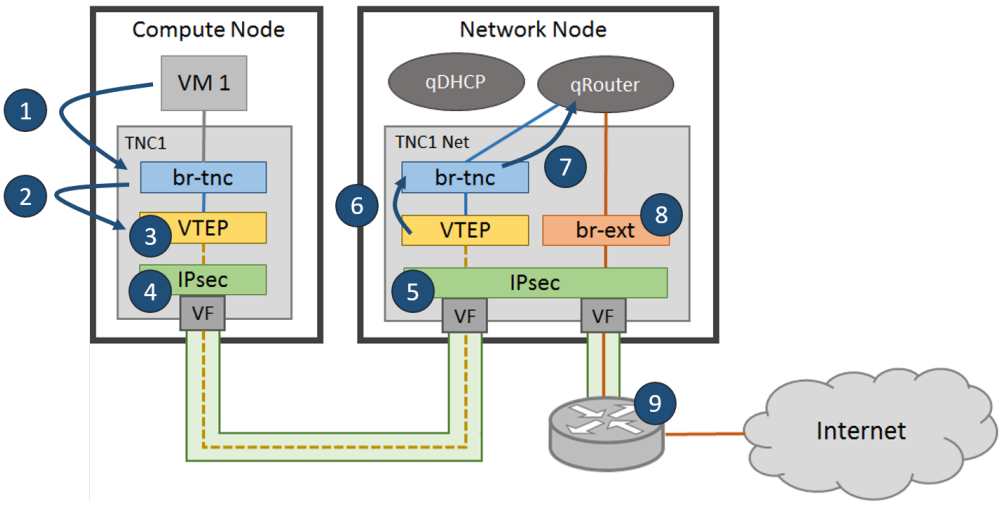

Figure 23: VM Internet access.

1. VM1 sends the packet to the TNC1's br-tnc bridge;
2. TNC1's br-tnc bridge forwards the broadcast packet to all of its active ports;
3. The VTEP on the compute node creates a VXLAN packet with destination IP equals to the network node VTEP;
4. The packet is encrypted and authenticated using IPsec in transport mode inside TNC1 and sent to TNC1 Net;
5. The IPsec module on TNC1 Net validates the IPsec packet and delivers the VXLAN packet to its VTEP;

6. The VTEP on the Network Node removes the VXLAN header and delivers the packet to TNC1 Net's br-tnc bridge;

7. br-tnc forwards the packet to br-ext through the qRouter namespace, used to configure the tenant's external network;

8. br-ext forwards the packet to a border router through an encrypted channel using the IPsec module in TNC1 Net;

9. The border router provides Internet connectivity to the Network Node.

5.4 Policy Management

The TNC is a key component for implementing traffic forwarding inside TNDs, and therefore to enforce connectivity and security policies defined by cloud orchestration system. In this section we describe a framework to deploy connectivity and security policies on TNC appliances. Tenants can define high-level security policies to the domains they control, as well as interconnect such domains if so desired. The configuration of these policies can be performed as usual, via the OpenStack dashboard service (OpenStack, 2016a), or by OpenStack Command-Line Interface (CLI) (OpenStack, 2016c). The security policies are then translated to low-level policies dependent on the underlying network technologies and topology, with the TNCs being responsible for enforcing those policies before any packet is allowed to enter or leave the corresponding slice of the domain (i.e., a group of machines from that TND that are collocated in a same physical host). This process is illustrated in Figure 24 for the same TNDs A and B depicted in Figure 19, showing that each of them receives a different set of policies.

Modifying policies for any given TND in this scenario consist of sending the appropriate configuration requests to all TNCs pertaining to that TND, so changes remain transparent to the VMs themselves. Adding a VM to any given host, as well as migrat-

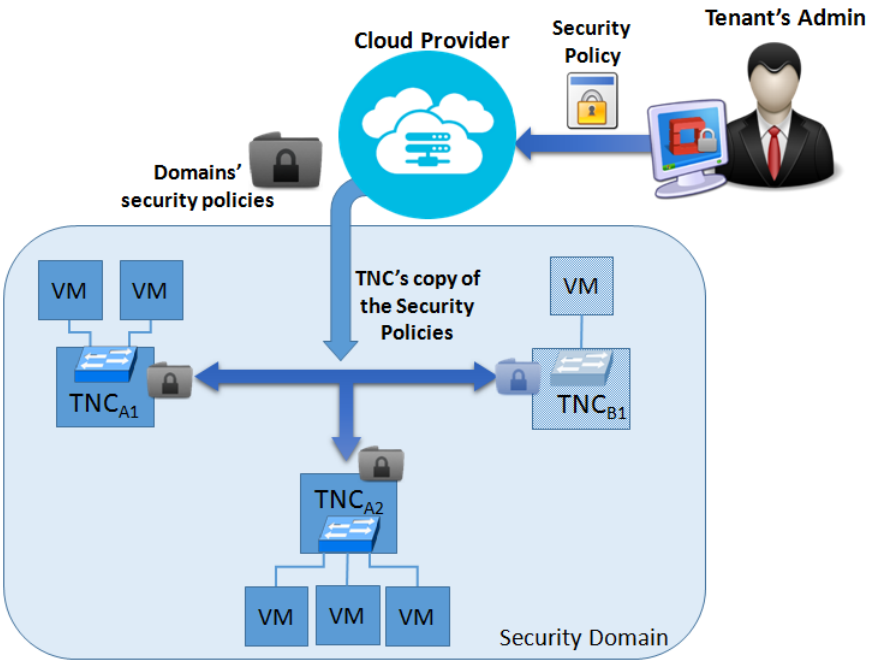

Figure 24: TND security policy configuration.

ing VMs from one host to another, are also straightforward tasks. Once the target host already a TND slice, it suffices to connect the new VM to the corresponding TNC and, otherwise, a copy of the TNC should be created in the target host and the VM should be attached to it. Whichever the case, during VM migrations, if all VMs from a TND originally in host H are moved to other hosts, then the TNC should also be removed from host H to save computational resources. To illustrate these situations, we consider some examples based on the scenario depicted in Figure 19.

- Adding a VM to TND C in Host 1: the new VM is directly connected to TNC$_{C1}$. No further modification is required.
- Adding a VM to TND A in Host 3: a new TNC has to be spawned at Host 4, with a virtual function VF$_{A3}$ attached to it, and then the new VM is attached to this TNC. The virtual functions VF$_{A1}$ and VF$_{A2}$, as well as VF$_{B1}$ (which is not part of TND A, but communicates directly with it via the IPsec tunnel 'A:B') are then reconfigured to take into account the new slice of TND A.
- Moving TND C's VM3 from Host 2 to Host 1: VM3 is directly connected to

TNC$_{C1}$. No further modification is required.

- Moving TND C's VM6 from Host 3 to Host 1: VM6 is directly connected to TNC$_{C1}$. TNC$_{C3}$ can then be shut down at Host 3, and the corresponding IPsec tunnels at the virtual functions VF$_{C1}$, VF$_{C2}$, and VF$_{C3}$ are reconfigured so only the tunnel between Host 1 and Host 2 remains active.

- Moving TND C's VM6 from Host 3 to Host 4: TNC$_{C3}$ can be simply moved together with VM6 to the new host, since no VM from TND C remain at Host 3. By cloning VF$_{C3}$'s configuration on Host 4 and connecting TNC$_{C3}$ to it, there is no need to reconfigure any VF to which VF$_{C3}$ is connected (i.e., VF$_{C1}$ and VF$_{C2}$ in our example), so no further modification is required.

5.5 Software Components

TND architecture builds upon the integration of two technological platforms: the OpenStack cloud operating system (OpenStack, 2016f); and the OpenDaylight SDN controller (Linux Foundation, 2016). Both platforms are consolidated and largely adopted open source software for cloud systems and SDN control plane implementations, respectively. This integration directly involves OpenStack's network module, named Neutron, responsible for implementing tenant networks and associated network services across the underlying cloud infrastructure. The core Neutron functionality explored by TND architecture is the VM layer 2 connectivity, implemented by the Modular Layer 2 (ML2) Neutron plug-in. ML2 makes use of mechanism drivers to implement layer 2 services using different virtualization mechanisms, among them the OpenDaylight controller. OpenDaylight controller, in turn, provides a collection of open source and independent network services that can be used to improve network features in OpenStack deployments. The integration between the OpenStack Neutron ML2 plugin and the OpenDaylight SDN controller is a fundamental piece of the TND

architecture implementation and evaluation. In (BARROS et al., 2015b), we describe Neutron ML2 plugin and how it integrates with OpenDaylight SDN controller to provide connectivity services to tenant VMs.

The TND architecture builds upon the integration of Neutron ML2 and OpenDaylight to implement the software components necessary to manage and deploy network policies to be enforced by TNC L2 appliances. All the implementation details described in this work assumes the utilization of the Icehouse stable release of OpenStack software (OpenStack, 2016e), and the Hydrogen release of the OpenDaylight controller (OpenDaylight, 2016). TND architecture employs a custom OpenDaylight SDN application to deploy connectivity and security policies inside the TNC OVS bridges. To this end, it is necessary to implement the following three software components described as following (see Figure 25).

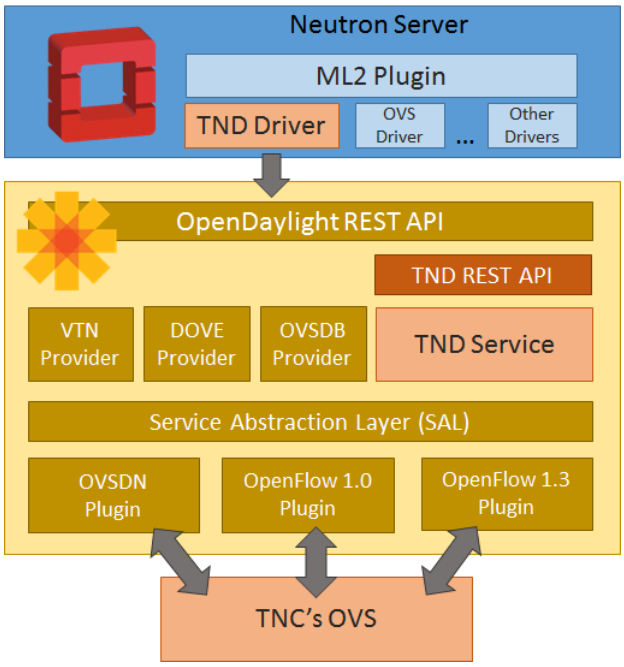

Figure 25: Software components to integrate TND architecture to OpenStack and OpenDaylight.

- **TND Modular Layer 2 (ML2) Driver**: based on ML2 ODL driver, it is responsible for modifying the Neutron Server's REST API interactions, allowing it to communicate with a new ODL service (in our case, the "TND Service");

- **TND OpenDaylight (ODL) Service**: based on Neutron ODL service, it is a service responsible for identifying and programing a domain's virtual switches in the TNC virtual machine;

- **TNC's Open Virtual Switch (OVS)**: the OVS instance running inside the TNCs, for creating a local virtual network in the domain slice.

The deployment of connectivity and security policies using the described software components can be organized in two message flows, the OpenStack and the OpenDaylight message flows. To illustrate the operation of these message flows, we describe the deployment of a connectivity policy responsible for adding a VM to a TND. The first phase corresponds to the messages exchanged inside the OpenStack network components, including Neutron API, ML2 plugin and the mechanism driver, whose messages flow is described as follows (see Figure 26):

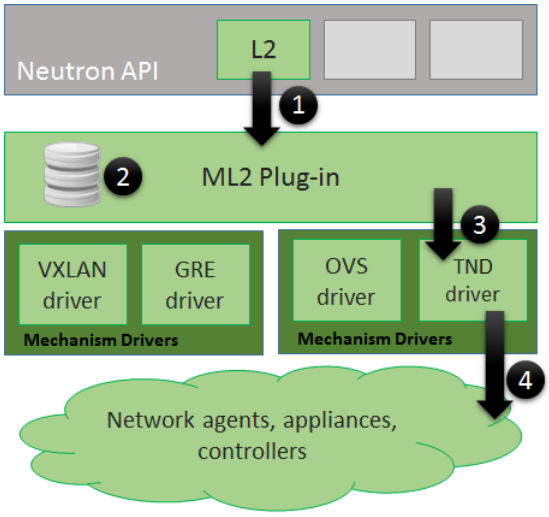

Figure 26: Connecting a VM to a TND: OpenStack flow.

1. Neutron ML2 responds to Neutron API call for connecting the VM interface in the identified virtual network;

2. Neutron ML2 registers the call in the Plugin DB;

3. Neutron ML2 sends a request to the mechanism driver that executes the network service (TNC driver);

4. TNC driver calls the OpenDaylight REST API, sending all the necessary parameter to insert the new VM on the tenant network.

After the VM is initialized, its networking capabilities must be configured by the OpenDaylight API, through the following steps (see Figure 27):

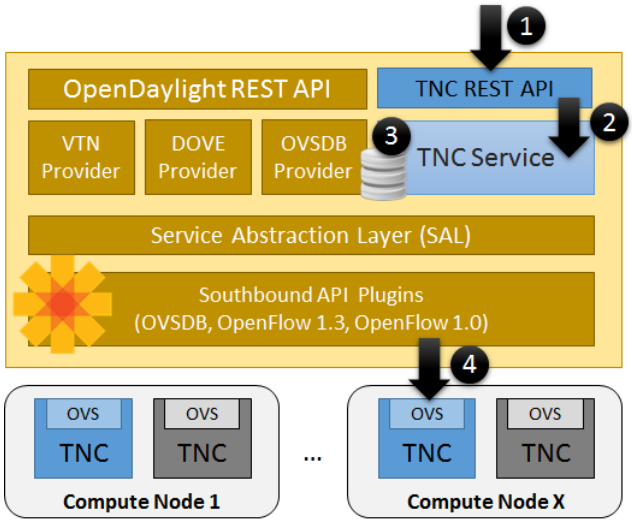

Figure 27: Connecting a VM to a TND: OpenDaylight flow.

1. Neutron TNC driver calls TNC REST API on OpenDaylight;

2. TNC Service receives TNC REST API request;

3. TNC Service consults the location of the TNC virtual switch that should be configured;

4. TNC Service configures the selected TNC OVS.

The components described here enable the deployment and complete integration of the TND architecture features to Vanilla OpenStack. These software components,

in addition to the virtualization technologies applied to provide isolation of tenant network resources, allows TND architecture to reuse OpenStack policy definition mechanisms for creating connectivity and security policies. In other words, TND components are used to translate network configurations provided by Neutron ML2 to the TNC appliances executed inside TND slices. Figure 28 illustrate the control framework provided by the TND Architecture to deploy connectivity policies TNCs deployed into different TND slices at the same host.

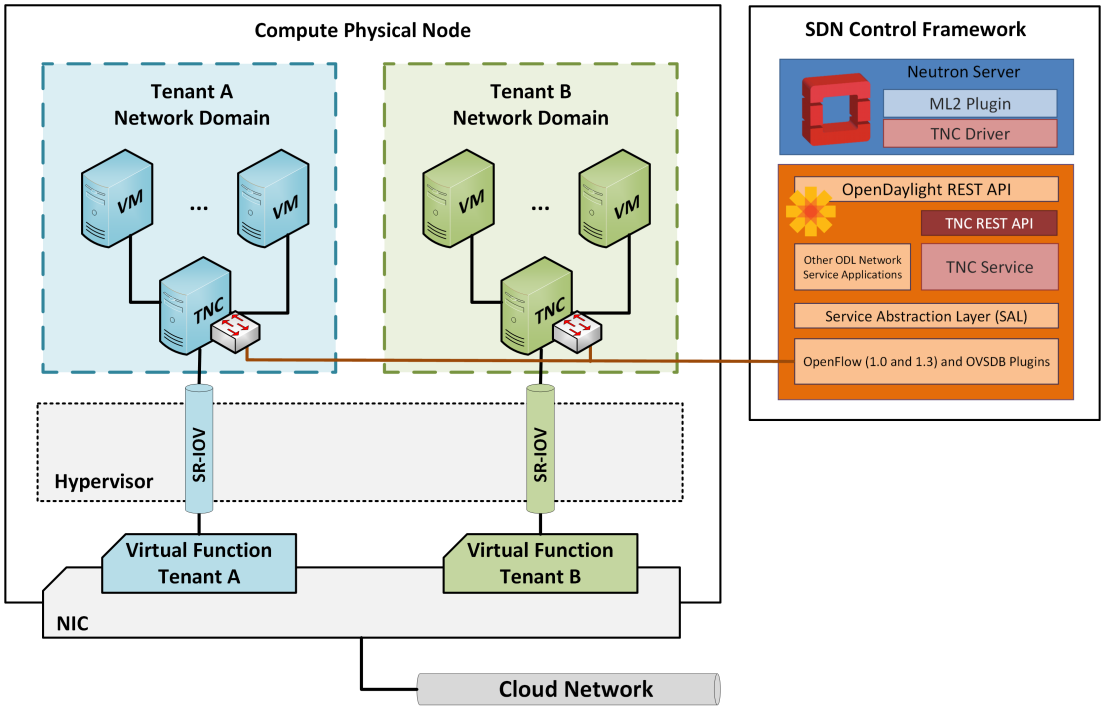

Figure 28: TND control framework.

Despite the software architecture described is intended to implement connectivity and basic security policies (e.g., security groups), currently defined in OpenStack through the Neutron ML2 plug-in, it can be extended to integrate with more sophisticated security services provided by OpenStack, such as Firewall-as-a-Service (FWaaS) and LoadBalancer-as-a-Service (LBaaS). The flexibility provided by our TNC appliance implementation (TNC VM), also leverage the implementation of custom security services for the TND architecture, that may be enforced by the TNC OVS bridge, or other security applications deployed on TNC.

5.6 Chapter Considerations

In this chapter we describe the implementation of the TND architecture, presenting the virtualization technologies and strategies adopted to address the isolation required by TNDs. We discuss how TND architecture can enhance the security traditional network virtualization architecture by improving the isolation of software and hardware network resources. The TNC network appliance is described as a key element of the TND architecture that aggregates the isolation and policy enforcement functions in TND slices. We detail the TNC operation by describing the VXLAN and IPsec encapsulation processes performed by OVS bridges deployed inside the TNC VM. Furthermore, we describe a promising approach to achieve isolation of hardware resources on TNC architecture by connecting TNC to SR-IOV VFs.

In addition, we describe a practical implementation of the TND architecture over a SDN-enabled cloud environment built upon the integration of OpenStack Neutron and OpenDaylight. We present the software components developed to integrate the TND architecture within this cloud network framework, in order to manage and deploy connectivity and security policies to be enforced by the TNC appliances. The practical implementation of the TND architecture described in this chapter will serve as basis for the comparative analysis with Vanilla OpenStack in Chapter 6.

6 TND ARCHITECTURE EVALUATION

This chapter presents a comparative analysis between the proposed TND architecture and a regular implementation of Openstack as described in Section 3.3, which we call Vanilla Openstack. The goal is to evaluate the level of isolation provided by the TND architecture, considering the three isolation strategies described in Section 3.2: (1) data path isolation, (2) isolation of software network appliances, and (3) isolation of hardware network appliances. In addition, we evaluate the TND architecture considering the non-functional requirements described in Section 4.3, including an experimental analysis of the communication performance obtained.

6.1 Evaluation of Network Resource Isolation

Vanilla OpenStack provides data path isolation by means of VLAN tagging mechanisms and tunneling protocols, a strategy commonly adopted by traditional network virtualization architectures. As discussed in Section 3.2, this isolation strategy relies on shared software and hardware network appliances, such as virtual switches and physical NICs, and therefore does not provide secure isolation for tenant networks if a malicious tenant is able to exploit vulnerabilities in such appliances. For example, as the tenants' network resources are not isolated, Vanilla OpenDaylight is susceptible to attacks such as MAC flooding and DoS, as previously described in Chapter 3.

The TND architecture addresses this isolation issue by means of TNCs, network appliances that encapsulate virtual switches for each tenant network, enforcing con-

nectivity and security policies for that tenant. The fact that each portion of a tenant's security domain in any given host has its own virtual switch, which is not shared with VMs from any other domain, creates logically isolated networks for each domain. As a result, even if a malicious VM exploits vulnerabilities on the switch to which it is connected, the potential damage is restricted to VMs from the same security domain, while other domains remain unaffected. In other words, a tenant can only damage its own VMs when attacking the underlying virtual switch. For instance, if a malicious tenant performs a MAC flooding attack inside a TND and does create a hub-like behavior on the underlying virtual switch (BULL; MATTHEWS, 2015), the packets that will be affected are only those from the same domain, as illustrated in Figure 29.

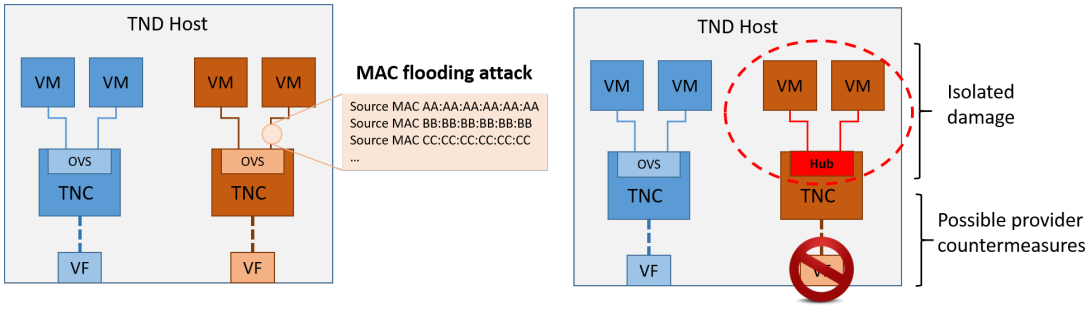

Figure 29: Example of MAC flooding attack against the TND architecture: only VMs from the same domain are affected, frustrating the attack.

Furthermore, the TND architecture provides isolation of hardware resources by connecting TNC appliances to SR-IOV VFs, which provide built-in isolation of physical NIC resources consumed by the attached VMs. Physical isolation in the TND architecture is obtained with the provisioning of one VF for each TNC and, thus, to each portion of a domain inside a host. As a result, attacks that exploit resources of the shared NIC are restricted to VFs of the corresponding domain, limiting the damage on other domains. This separation between packet flows from different domains also facilitates corrective actions (e.g., enforcement of traffic rate limiting policies) from the cloud provider in case of misbehavior, as a single VF needs to be configured instead of the whole NIC. We note that a similar effect could be obtained simply by connecting each VM to a separate VF, but having only one VF per security domain

tends to facilitate managing and configuration. To illustrate how this protective measure works, we can consider a DoS attack attempt against the host's physical NIC: if a malicious/misconfigured tenant VM generates an excessive amount of traffic, only its own domain's bandwidth share will be affected (see Figure 30).

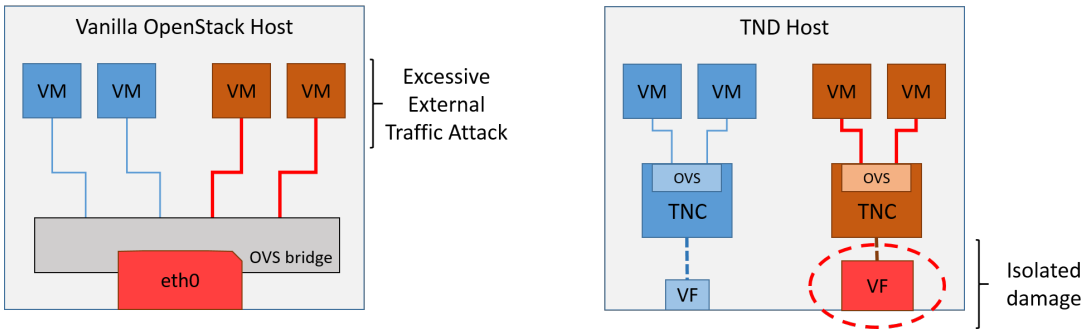

Figure 30: Example of excessive external traffic in TND architecture: only VMs from the same TND are affected.

In addition, the application of cryptographic techniques provided by the TND architecture provides additional security services for TNDs, improving data path protection. Namely, as discussed in Section 5.1, IPsec in transport mode provides confidentiality, integrity and authentication for tenants' network traffic, without compromising routing features provided by the VXLAN protocol.

6.2 Evaluation of Additional Requirements

This section presents a qualitative evaluation of the TND Architecture considering the additional non-functional requirements described in Section 4.3. Although not part of TND's core goal of isolating tenants' networks, those requirements are relevant for its potential adoption in actual cloud deployments.

6.2.1 Transparency

In TND, tenants only have to define high-level security policies just like in non-cloud environments (e.g., firewall or IDS rules to be applied to each domain), consid-

ering the network's logical topology. The security provisions for enforcing physical and logical isolation between domains in the actual multi-host physical topology, as well as any tenant-defined security rule, are then carried out transparently by the cloud provider. The latter is, thus, responsible for configuring the corresponding TNCs and VFs in each host accordingly.

In addition, modifications on the underlying technology can be handled without the tenants' intervention. For instance, different physical isolation technologies besides SR/IOV, as well as different generations of SR/IOV-enabled hardware, can be employed in the hosts without the need of installing any specific driver on the tenant's VMs. We note that this would not necessarily be the case if each tenant VM were directly connected to its own VF, as in that case the VMs themselves would need to have the required drivers for handling the corresponding hardware-enabled network virtualization technology.

6.2.2 Flexibility

The TND architecture does not impose any restriction on the security functionalities that can be deployed on their security domains. Indeed, as long as the corresponding functionality can be supported by the TNC itself and/or implemented in the underlying network hardware, it can be deployed in the TND architecture as an appliance (i.e., without the need of modifying the tenants' VMs).

6.2.3 Portability

TNCs work as an abstraction layer between the VMs and the network software/hardware employed for providing physical and logical isolation between domains. Therefore, a domain's network security configurations can be deployed simply by installing the required software on the corresponding TNCs, independently of the specific details regarding the VMs behind those TNCs. This approach is especially

interesting for facilitating the creation of domains in cloud environments in which the physical hosts have heterogeneous hardware: as long as the TNC spawned in a given physical host has support to all required drivers and security software, any VM can be initialized in (or moved to) that host without modifications.

Once again, it is interesting to note that portability issues would likely arise if physical isolation with SR/IOV was provided by connecting VMs directly to virtual functions: in that case, the VM itself would need to support SR/IOV drivers, which are not yet available for all operating systems. Even if support to SR/IOV becomes commonplace, the continuous development of this and similar hardware-assisted virtualization technologies could end up making the already installed software obsolete, hindering the deployment of outdated VMs in cloud environments with heterogeneous hardware.

6.2.4 Mobility

Since the TNC concentrates all of the domain's configurations and is itself a VM, the task of moving VMs and their corresponding security network configurations from one physical host to another is reduced to the task of migrating VMs. This is the case only for the migration of stateless domain configurations, since the migration of stateful elements would require the migration of the security context across different TND slices (TAVAKOLI; MEIER; VENSMER, 2012). In fact, if the host to which the tenant's VM is being moved already has VMs from the same domain, it would suffice to attach the moved VM to the existing TNC, so the overhead introduced by the TND architecture to the migration process would be minimal. This characteristic of the TND architecture can also take advantage of auxiliary mechanisms for VM allocation that place VMs from the same tenant in the same host for better security (e.g., avoiding cache timing attacks) or efficiency.

6.2.5 Performance

On the one hand, the TND architecture provides better isolation by adding abstraction layers to the communication between different VMs (the TNC, for all communications, and IPsec tunnels, for inter-host packet exchanges); hence, it is expected that the security gains provided by the solution also leads to some impact on network performance. On the other hand, this performance hit is somewhat counterbalanced by the fact that communications among different domains have lower influence on each other (except, obviously, for the fact that the host's processor is still shared by networking elements). Hence, (1) the dedicated TNC switch should provide reasonably efficient intra-host communications, especially if each TNC is endowed with its own processing core for processing packets; and (2) inter-host communications benefit from SR-IOV performance gains, so the throughput obtained should not be much lower than what is obtained with an implementation where there is an OVS shared by all collocated VMs.

To evaluate how those two complementary factors actually affect the network performance when the TND architecture is deployed, we performed a series of experiments for measuring the latency and bandwidth of the intra- and inter-host communication between VMs. Namely, we (1) executed 10 ping requests between two VMs and measured the average latency of those executions, and (2) used the iperf tool to measure the average bandwidth for 10 transmissions of a few hundreds of megabytes. The intra-host tests were repeated in some scenarios (see Figure 31): vanilla Openstack with a shared OVS, with IPsec enabled or disabled on the VMs; the TND architecture with and without IPsec enabled on the VMs and for TNCs configured with different amounts of resources.

The results obtained for the tests are shown in Figure 32, for bandwidth, and in Figure 33, for the observed latency. The figures consider two flavors of TNCs: one smaller, configured with 1 vCPU, 2 GB of RAM, and 20 GB of hard disk, named simply "TNC"; and a larger one, called "L TNC", which has 4 vCPU, 8 GB of RAM,

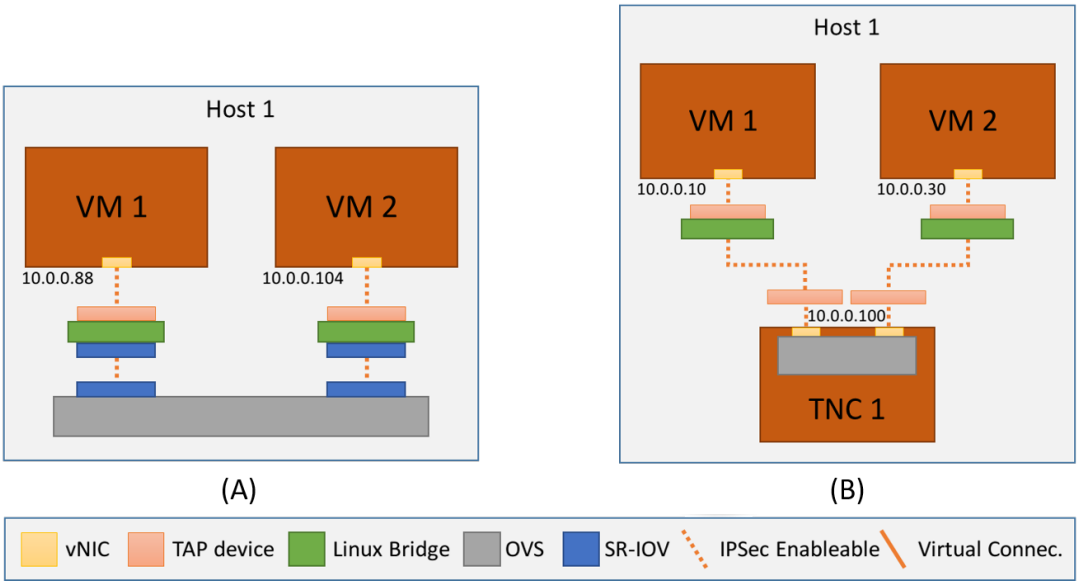

Figure 31: Intra-host communication scenarios: (A) Vanilla OpenStack and (B) TND architecture.

and 80 GB of hard disk. In all cases, the host in which the tests are performed are Intel Servers S1400SP2, with the following hardware configuration: Intel Xeon E5-2430 processor (15M Cache, 2.20 GHz, 7.20 GT/s Intel®QPI): 48GB (6x8GB) of RAM memory (ECC 1333, DDR3); Intel® Gigabit ET2 82576GB Quad Port Server Adapter.

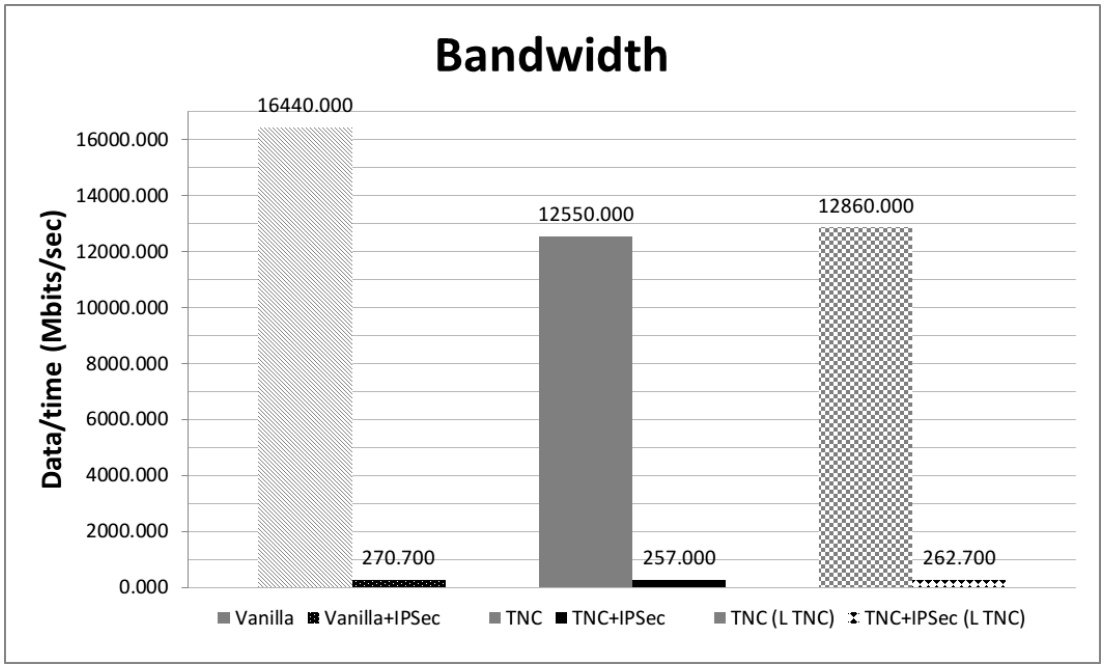

Figure 32: Bandwidth for intra-host communication in different settings.

For inter-host communications, the same test settings were considered, but this

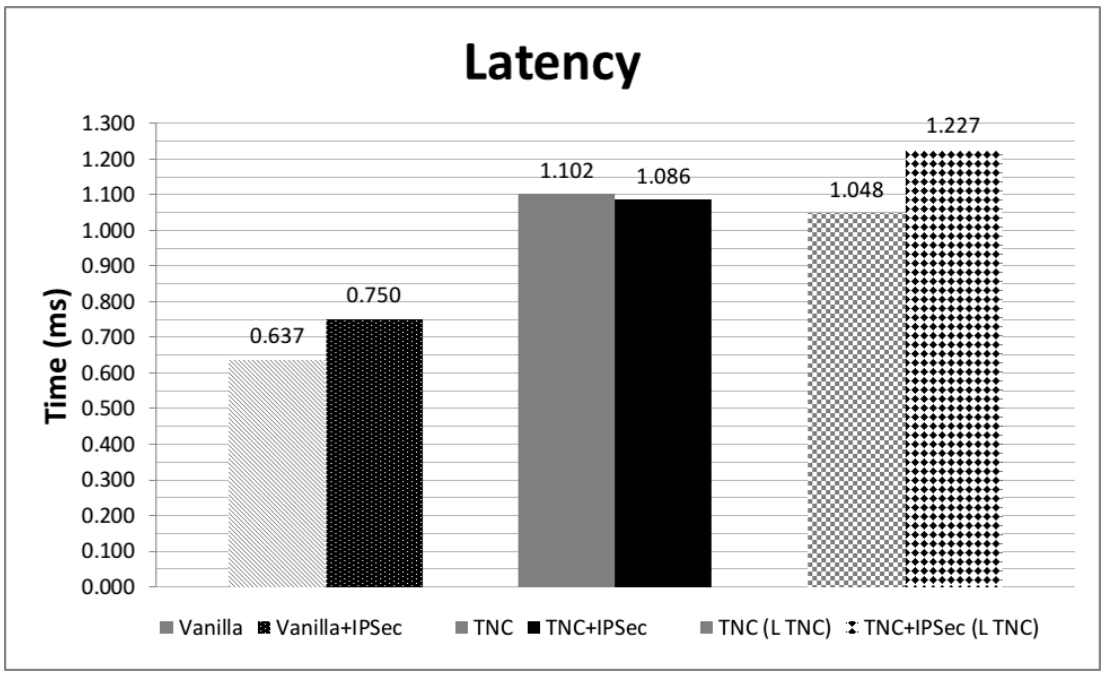

Figure 33: Latency for intra-host communication in different settings.

time we also evaluated the impact of using VXLAN and SR-IOV. The result is a larger number of scenarios and technologies, all of which are depicted in Figure 34.

The measurements obtained for the total bandwidth and perceived latency in these scenarios are depicted, respectively, in Figure 35 and Figure 36. The bandwidth measurements show that the TND architecture's overhead when compared to vanilla OpenStack is basically compensated by the performance gains provided by SR-IOV, whether or not VXLAN is used. The same applies to the scenario in which IPsec is employed: once again, SR-IOV compensates the performance impact introduced by TND's additional abstraction layers. These measurements also show the interest of employing technologies such as IPsec offloading for potentially improving the network's overall bandwidth; this specific feature was not tested, however, due to the technical difficulties encountered when trying to activate it in the available hardware. Finally, in terms of latency, once again the impact of the TND architecture is quite high, in especial when the solution is compared with vanilla OpenStack combined with SR-IOV. Nevertheless, in absolute numbers, the latency remains quite low: under 1.63 ms.

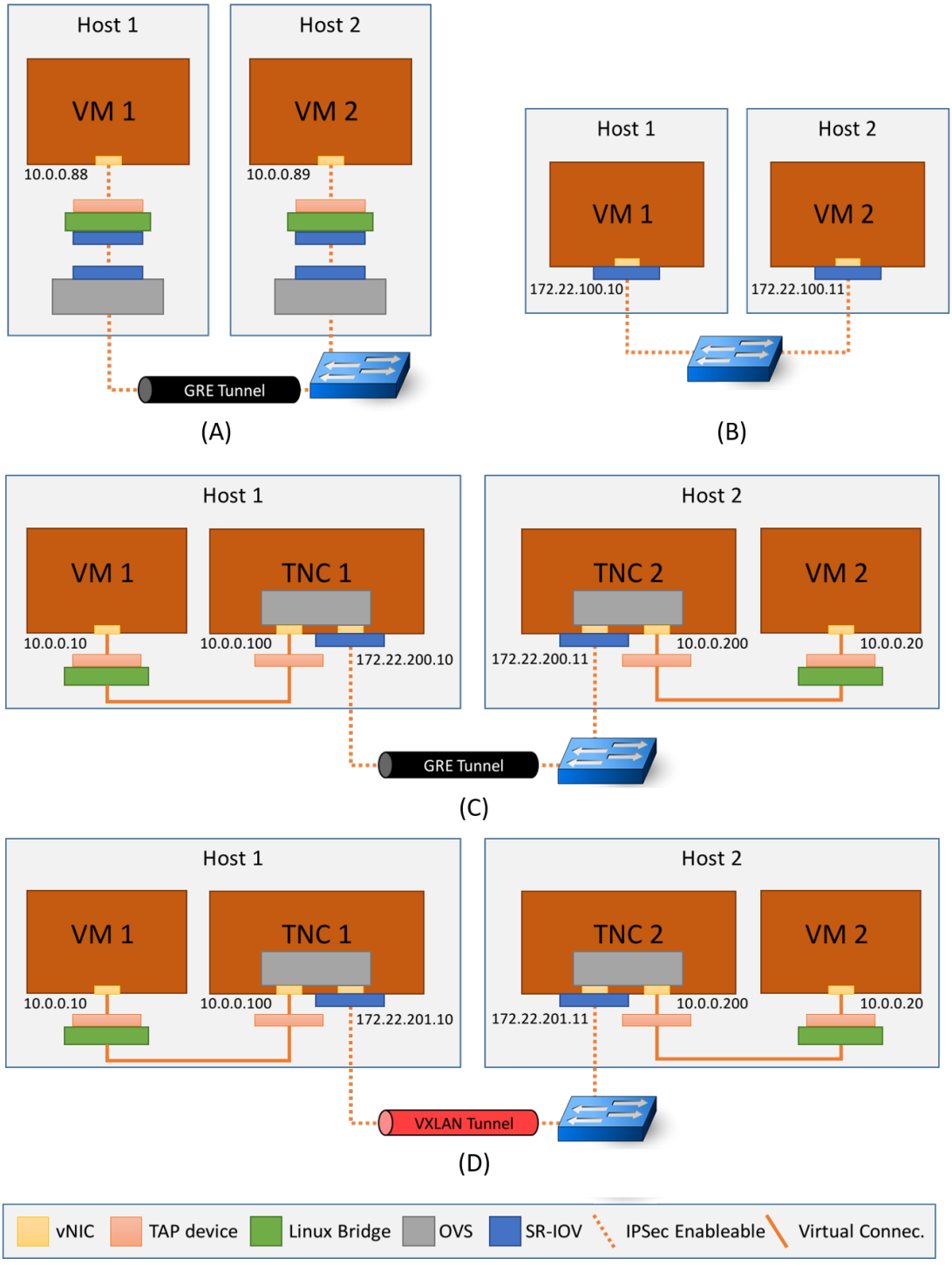

Figure 34: Inter-host communication scenarios: (A) Vanilla OpenStack, (B) Vanilla OpenStack with SR-IOV and, (C) TND using GRE, and (D) TND using VXLAN.

6.2.6 Scalability

The TND architecture addresses several scalability aspects, namely:

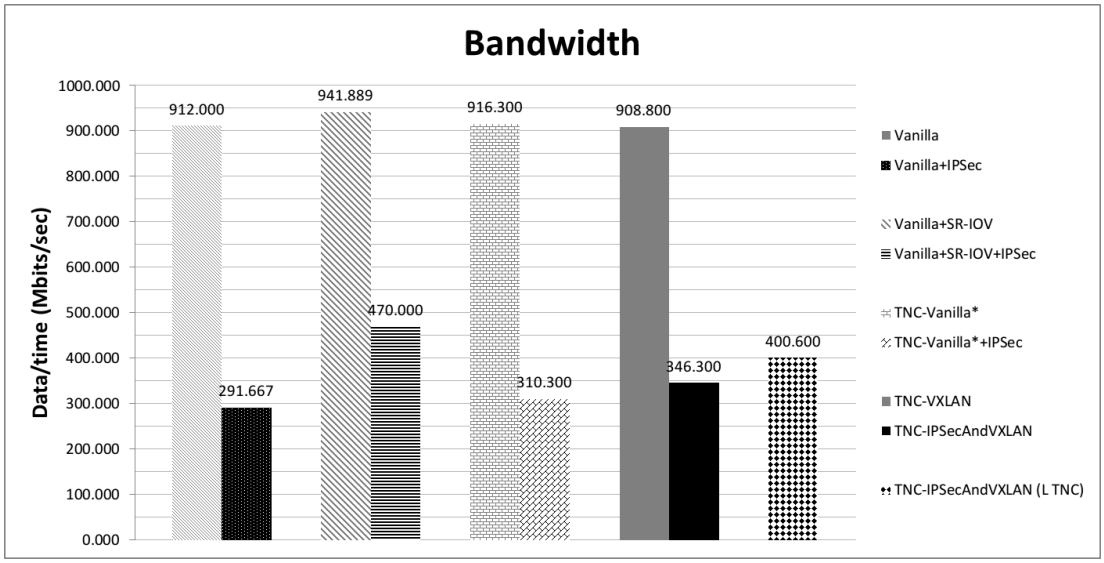

Figure 35: Bandwidth for inter-host communication in different settings.

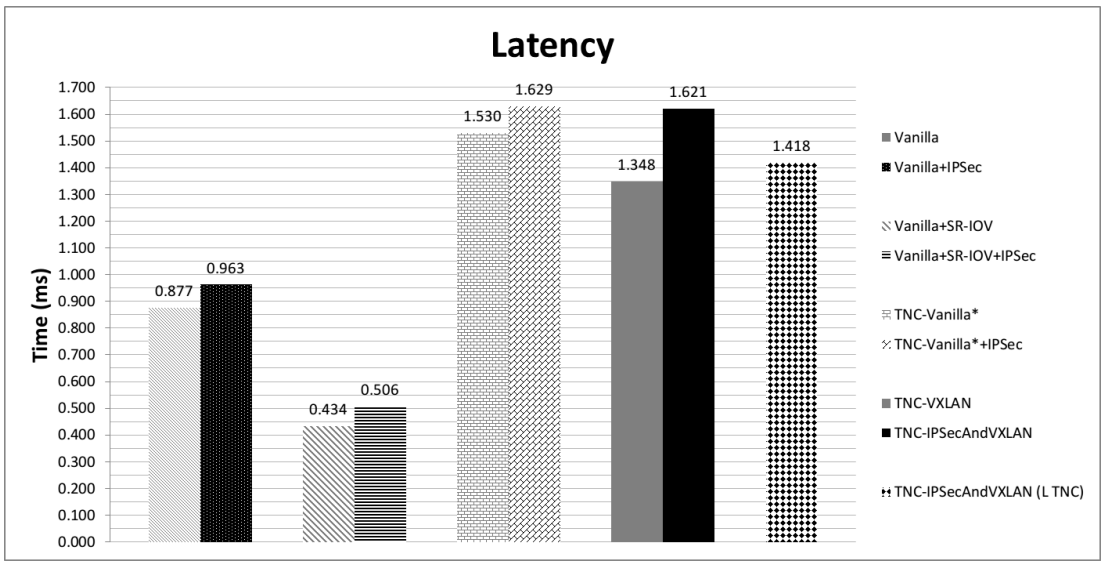

Figure 36: Latency for inter-host communication in different settings.

1. *Scaling the number of VFs per NIC*: modern network cards that support network virtualization have a fixed number of VFs, depending on the exact hardware details. Therefore, if every VM hosted has its own dedicated VF for improved physical isolation, the maximum number of VMs in a host ends up being limited by the number of NICs and their corresponding availability of VFs. Since NICs are physical devices, (de)allocating them dynamically is not an easy task, meaning that cloud providers are likely to compensate this limitation by over-

provisioning NICs in the different physical hosts. The TND architecture, on the other hand, helps alleviating this issue by concentrating multiple VMs from a same domain into a single VF, potentially allowing for a higher number of VMs per NIC in a given physical host. For example, if all VMs in a same physical host belong to a same security domain, only one VF is used, while the others remain free to be dynamically allocated for VMs from other domains. The worst case scenario in terms of VF allocation is, thus, when every tenant VM in a host belongs to a different domain, in which case there will still be a 1:1 relationship between VFs and VMs. With a carefully designed VM allocation mechanism, however, this issue can be easily solved without the need of extra hardware.

2. *Scaling the number of virtual networks*: the adoption of VXLAN for handling the communications between VMs from the same domain hosted in different machines allows the coexistence of a larger number of virtual networks in comparison with what could be obtained with regular 12-bit VLAN tags.

3. *Scaling the number of VMs per physical host*: one drawback of the TND architecture is that each domain slice inside a given physical host has an extra VM (the TNC) in addition the tenant VMs themselves. Nevertheless, the TNCs are expected to be more lightweight than regular tenant VMs, since they basically act as dedicated appliances for providing connectivity, besides eventual security and accountability services. There can even be a base "template VM" that could be used for instantiating the TNCs from several tenants, thus saving disk memory, with the required tenant-oriented customizations being applied when the TNC is initialized. In addition, a single TNC can handle $n \geqslant 1$ VMs in any given domain slice, so the TNC's resource usage should not be higher than $1/n$ of the domain slice's total cost. Therefore, the overhead introduced by TNCs should not significantly impact the system's scalability, especially for a large n.

6.2.7 Availability

As a direct consequence of providing better isolation between VMs from different domains (and, thus, of potential damages caused by/to them), the TND architecture enhances the availability of the cloud infrastructure. For example, if a misconfigured or compromised VM starts sending a large amount of traffic, the virtual switch and VF shared by VMs from that same domain are the elements more directly affected. Hence, they are the only ones that will need to be policed to avoid damages to other tenants and/or to the underlying network infrastructure, having their traffic rate reduced or even being disconnected from the network. The other domains running in the same host may be affected due to the higher consumption of resources in the shared hardware, but, in principle, they do not have to suffer any drastic measures.

Similarly, the TND architecture avoids the creation of cross-domain bottlenecks or Single Point of Failures (SPoFs), which would be the case of the virtual switch shared by all VMs in the same physical host. Even though the TNC can be seen as a SPoF for a given domain slice, the fact that it can be instantiated with similar configurations in any physical host facilitates its recovery analogously to what is done with tenant VMs in case of failure; therefore, similar disaster recovery mechanisms can be employed.

6.2.8 Accountability

Another consequence of higher traffic isolation between domains is that it should be easy to account (and, thus, charge for) the resource consumption of each domain. This control can be implemented in a distributed manner, by placing accounting mechanisms on the TNCs themselves, since they act as gateways for all traffic coming to/from each domain. For example, the tenant can be charged by the total amount of traffic entering and/or leaving its own domain, which can be learned simply by analyzing the traffic on all TNCs of that domain. This is actually quite similar to some scalable billing-oriented approaches discussed in the literature, such as sFlow (PHAAL;

LAVINE, 2004), which can be integrated into the TND architecture in a straightforward manner.

In addition, each TNC can be empowered with different security services, such as firewalls, traffic analyzers, intrusion detection/prevention systems (IDS/IPS), among others. Therefore, the cloud provider may apply different charges to each (set of) security services provided to a tenant's domain, and then install the corresponding mechanisms in all TNCs belonging to that domain. Some providers may even offer a "basic security package" free of charge, which would be part of all TNC images, while other security services from a "premium security package" could be installed on demand.

Finally, the information collected by the TNCs about how many and which attacks were detected/prevented on the domains covered by them can be shared with the corresponding tenants. The result is that tenants would be able to visualize the threats to which they are exposed, a vital piece of information for risk analysis and something expected in any Security-as-a-Service (SecaaS) utility. At the same time, the tenants do not learn any information about attacks on their neighbours, a property that would likely require filtering if the domains were not clearly separated by TNCs.

6.2.9 Maintainability

The isolation between domains in the TND architecture is such that actions taken toward a domain have no direct impact on the operation of other domains. For example, if a tenant decides to acquire a higher-speed connection for the VMs in a domain, applying those changes would be basically a matter of configuring the VFs attached to that domain's TNCs. Similarly, adding/removing/configuring security services that should apply to a domain would involve changing those services in those TNCs. This facilitates policy management, as the only policies that need to be verified for conflicts when making network configuration changes are those already installed in the target

domain's TNCs.

Furthermore, when discussing availability, corrective measures taken in response to attacks can be more precisely applied with the TND approach. For example, if the cloud provider detects that one domain is behaving inappropriately (e.g., generating malicious traffic, such as spoofed packets), that domain or parts of it can be isolated from the rest of the cloud by deactivating the corresponding VFs. Then, detection mechanisms such as an IDS can be installed in the corresponding TNCs to verify which are the domain portions affected, as well as the corresponding VMs responsible for the problem; whenever a part of the security domain is identified as not being the source of the malicious traffic, it can be reconnected to other parts of the domain that have also been considered "clean", gradually restoring the whole domain. Finally, the malicious VMs can be isolated and/or have their traffic limited by configuring the corresponding ports of the virtual switches connecting those VMs to the domain. This whole process, from detection to mitigation, is transparent to any VM except for those belonging to the compromised domain.

6.3 Chapter Considerations

In this chapter we compare the TND architecture with the vanilla OpenStack network virtualization architecture, aiming to understand the impact of our proposal over current cloud network deployments. The comparison considers the main design goals described for the TND architecture regarding the isolation of network resources, as well as the secondary characteristics affected by the TND proposal. Table 5 summarizes the results from the comparison between vanilla OpenStack and the TND architecture, resulting from the reasoning described in this section.

We observe that our implementation of the TND architecture achieves its main design goals, which consist of improving multi-tenant network isolation in cloud sys-

Table 5: Comparative analysis of vanilla Openstack and TND architecture.

Requirements	Vanilla OpenStack	TNC Architecture	Comparative Analysis Type
Data Path Isolation	✓	★	**T,E**
Software Resource Isolation	✗	★	**T,E**
Hardware Resource Isolation	✗	★	**T,E**
Transparency	✓	✓	**T**
Flexibility	✓	✓	**T**
Portability	✓	✓	**T**
Mobility	✓	★	**T**
Performance	★	✓	**T,E**
Scalability	✓	✓	**T**
Availability	✓	★	**T**
Accountability	✓	★	**T**
Maintainability	✓	★	**T**
★: Best Option for Addressing the Requirement	✓: Requirement Addressed	✗: Requirement Not Addressed	**T**: Theoretical, **E**: Experimental

tems, by enhancing the isolation of logical data paths, software and hardware network resources. However, the analysis of secondary requirements demonstrates that there is space for improvements on different aspects of cloud networking. In particular, our performance evaluation demonstrated a significant impact of TND isolation mechanisms over the tenant VMs communication, specially in intra-domain communication. However, we assume that the performance impact provided by the TND mechanisms, such as IPsec should be known and deliberately applied aiming at achieving the necessary security requirements. Although we observe that such impact can be mitigated by resource-rich TNC appliances and HAV technologies, these solutions may impact other mentioned characteristics of the cloud, such as scalability and portability. Therefore, the evaluation of the TND architecture demonstrates the viability of implementing ef-

ficiently isolated virtual network domains and its impact over relevant cloud characteristics, paving the path for future research and improvements on the proposed solution.

7 CONCLUSIONS AND FINAL CONSIDERATIONS

The increasing demand for public cloud services has raised the security concerns regarding multi-tenancy. In particular, the shared nature of cloud network resources represents a massive source of threats in multi-tenant cloud infrastructures. Multiple network virtualization technologies have been applied to different network architectures intended to ensure the isolation of virtual networks, most of them focused on the isolation of logical data paths through the implementation of tunneling and tagging techniques. However, to reduce attack surfaces in multi-tenant clouds and to ensure the security of tenant networks, it is also necessary to isolate software and hardware appliances applied to the implementation of such networks.

The security network virtualization architecture proposed in this work, named TND architecture, significantly increase multi-tenant isolation in cloud networking. In comparison with existing architectures, including vanilla OpenStack, TND architecture provides better isolation of tenant networks by addressing the isolation of software and hardware network appliances deployed in cloud network infrastructures. In addition, the TND architecture enhances the security of usual data path isolation schemes through the IPsec encapsulation of VXLAN packets, benefiting from the security services provided by IPsec, as well as from the multicasting and scalability features provided by VXLAN.

The TND architecture design builds upon innovative network technologies, such NFV, SR-IOV and SDN. These technologies are employed in the development of the

TNC network appliance, a special purpose network function controlled by the CSP to provide policy enforcement for different virtual network domains distributed across the cloud infrastructure. The TNC implementation in form of VM take advantage of full virtualization platforms to implement the isolation of connectivity and security services consumed by different TNDs, enabling the deployment and measurement of custom network services. The performance drawbacks of inserting a new network element on current cloud network architectures are mitigated by the utilization of SR-IOV VFs, which allow TNC appliances to bypass the hypervisor network stack and connects to the physical NIC of the server host. Moreover, the utilization of an SDN control framework for the TND architecture enables the integration of the TND architecture with cloud systems such as OpenStack, as well as the readily deployment of security functions in form of SDN applications.

The evaluation of the TND architecture demonstrates that it achieves the isolation requirements defined by its design goals, and analyze the impact of the proposed design in other requirements of cloud networking. This analysis paves the path for future research and improvements of the TND architecture. In the next sections, we describe the main contributions and future works resulting from this work.

7.1 Contribution

The main contribution of this work is the proposal of a security architecture for network virtualization in cloud systems, named TND architecture. The proposed architecture provides efficient isolation of network resources in multi-tenant cloud systems, reducing network attack surfaces and mitigating associated security threats. In comparison with other existing network virtualization architectures, the TND architecture addresses virtual network isolation from three different perspectives: the isolation of logical data paths, the isolation of software network appliances, and the isolation of hardware network appliances.

The proposal of an efficient approach for isolating virtual networks in multi-tenant clouds based on three different strategies is an additional contribution of this work. From the evaluation of different security attacks against shared network resources in multi-tenant data centers, we call attention to the need for isolating logical data paths, software and hardware network appliances. Furthermore, we analyze the ability of current network virtualization architecture to implement the proposed strategies. In addition, we developed a comprehensive research on network virtualization techniques and technologies that can be applied to implement multi-tenant isolation in future network architectures.

7.2 Future Work

Although TND architecture provides valuable enhancements on network isolation in multi-tenant cloud environments, it still requires deeper evaluation of its impact on cloud non-functional requirements. Indeed, our theoretical and practical evaluation demonstrates prominent paths for improvements of the TND architecture implementation. In particular, the scalability and performance issues inherited from our TNC implementation may should be revisited under the lights of new network virtualization technologies, such as DPDK (Intel, 2016a) and other packet process offloading techniques (LIN et al., 2016; TSENG et al., 2016). Therefore, future works include to perform a deeper evaluation of TND impact over cloud requirements, and improvements on the implementation of the TND architecture.

In addition, the TND architecture demonstrates the viability of the implementation of efficiently isolated network domains in cloud deployments through the adoption of the proposed resource isolation approach. Although the security benefits provided by such isolation can be theoretically stated based on the reduction of attack surfaces, the TND implementation should be tested under different attack scenarios to strengthen such statements. As future work, the TND architecture should be deployed on different

cloud scenarios, possibly on different cloud platforms, to have its security assumptions validated against practical attacks.

Finally, TND architecture provides a framework for the implementation and deployment of multiple security functions for cloud networks. These security functions can be implemented in form of SDN network applications (YOON et al., 2015), or deployed as current network applications running inside TNC VMs (e.g., Firewall, IDS, IPS, DPI). Thus, the extension of the security features supported by the TND architecture, including support for custom tenant network applications, is also a prominent research path to motivate the adoption of isolated network domains in cloud deployments. The definition of different security policies using SDN also requires the design of an appropriate policy definition language. Therefore, future works include the creation of a comprehensive security policy language to be used by the TND architecture, allowing different tenants to specify security requirements to be enforced through isolated network services provided by TNCs. The TND security policy language should be further applied to develop the semantic layer of the TND architecture, which should allow the development of novel security management solutions that can extend the features of the TND architecture.

7.3 Publications

The following publications, submissions and envisioned papers are a direct or indirect result of the research effort carried out during this dissertation:

- *Conference Papers*: in (BARROS et al., 2015a), we develop a comprehensive research on security threats in cloud networking and proposed a threat classification that provides support for the development and evaluation of security solutions; in (BARROS, In Productionc), we apply the proposed threat classification on the analysis of the state-of-the-art on current cloud network virtualization so-

lutions; in (ALMEIDA et al., 2016), we provide a practical evaluation of the use of IPsec encapsulation along with SR-IOV technology in cloud systems.

- *Journal Papers*: in (BARROS, In Productionb), we discuss the security threats originated from shared resources in cloud networking, proposing a novel strategy to achieve efficient multi-tenant isolation and evaluating the ability of current network architectures to implement the proposed strategy; in (BARROS, In Productiona), we describe the TND architecture, discussing how it can achieve efficient isolation of tenant network resources, as well as its impact over additional requirements commonly applied to cloud networking.

- *Book Chapters*: in (BARROS et al., 2015b), we present a theoretical study of the network virtualization technologies that can be applied to the implementation of cloud networking systems. In addition, we describe a practical implementation of the integration between the OpenDaylight SDN controller and the OpenStack cloud orchestration system, which provided us with the basis for designing the policy control framework of the TND architecture; in (MIERS et al., 2014), we contribute to the security analysis of cloud computing platforms, describing the security threats associated to usual network virtualization architectures and discussing possible countermeasures.

REFERENCES

ABENI, L.; KIRALY, C.; LI, N.; BIANCO, A. Tuning kvm to enhance virtual routing performance. In: *2013 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2013. p. 3803–3808. ISSN 1550-3607.

AL-SHABIBI, A.; DE LEENHEER, M.; GEROLA, M.; KOSHIBE, A.; PARULKAR, G.; SALVADORI, E.; SNOW, B. Openvirtex: Make your virtual sdns programmable. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 25–30. ISBN 978-1-4503-2989-7.

ALARIFI, S.; WOLTHUSEN, S. Robust coordination of cloud-internal denial of service attacks. In: *Cloud and Green Computing (CGC), 2013 Third International Conference on*. Karlsruhe: IEEE, 2013. p. 135–142.

ALMEIDA, T. R. M.; ANDRADE, E. R.; BARROS, B. M.; JR, M. A. S. Avaliação de desempenho em nuvens computacionais utilizando ipsec em conjunto com sr-iov. In: BRAZILIAN TELECOMMUNICATIONS SOCIETY. *Porceeedings of 34th Brazilian Telecommunications and Signal Processing Symposium*. Santarem, PA, Brazil, 2016. p. 959–963.

Amazon. *Amazon Elastic Compute Cloud (Amazon EC2) - Virtual Server Hosting*. 2016. Available at: https://aws.amazon.com/ec2/. Accessed: 2016-12-7.

Amazon. *Amazon Virtual Private Clouds (VPC)*. 2016. Available at: https://aws.amazon.com/vpc/. Accessed: 2016-12-7.

ANDERSON, T.; PETERSON, L.; SHENKER, S.; TURNER, J. Overcoming the Internet impasse through virtualization. *Computer*, v. 38, n. 4, p. 34–41, April 2005. ISSN 0018-9162.

ANHALT, F.; PRIMET, P. *Analysis and evaluation of a XEN based virtual router*. [S.l.], 2008. 60 p. Available at: https://hal.inria.fr/inria-00320620/file/RR-6658.pdf. Accessed: 2016-12-7.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782.

AUTENRIETH, A.; ELBERS, J.-P.; KACZMAREK, P.; KOSTECKI, P. Cloud orchestration with SDN/OpenFlow in carrier transport networks. In: *15th Int. Conf. on Transparent Optical Networks (ICTON)*. [S.l.: s.n.], 2013. p. 1–4. ISSN 2161-2056.

AVRAM, M.-G. Advantages and challenges of adopting cloud computing from an enterprise perspective. *Procedia Technology*, Elsevier, v. 12, p. 529–534, 2014.

AZODOLMOLKY, S.; WIEDER, P.; YAHYAPOUR, R. Cloud computing networking: challenges and opportunities for innovations. *IEEE Communications Magazine*, v. 51, n. 7, p. 54–62, July 2013. ISSN 0163-6804.

BARI, M. F.; BOUTABA, R.; ESTEVES, R.; GRANVILLE, L. Z.; PODLESNY, M.; RABBANI, M. G.; ZHANG, Q.; ZHANI, M. F. Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, v. 15, n. 2, p. 909–928, Second 2013. ISSN 1553-877X.

BARJATIYA, S.; SARIPALLI, P. Blueshield: A layer 2 appliance for enhanced isolation and security hardening among multi-tenant cloud workloads. In: *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*. Chicago, IL: IEEE, 2012. p. 195–198.

BARROS, B. M.; IWAYA, L. H.; JR., M. A. S.; CARVALHO, T. C. M. B.; MÉHES, A.; NÄSLUND, M. Classifying security threats in cloud networking. In: *Proceedings of the 5th International Conference on Cloud Computing and Services Science*. [S.l.: s.n.], 2015. p. 214–220. ISBN 978-989-758-104-5.

BARROS, B. M.; JR, M. A. S.; CARVALHO, T. C. M. B.; ROJAS, M. A. T.; REDIGOLO, F. F.; ANDRADE, E. R.; MAGRI, D. Book Chapter, *Applying Software-defined Networks to Cloud Computing*. Porto Alegre, RS, Brazil: Sociedade Brasileira de Computação (SBC), May 2015. 1–54 p. 33rd Brazilian Symposium on Computer Networks and Distributed Systems. (Short Courses). Available at: http://sbrc2015.ufes.br/wp-content/uploads/livro-texto-Minicursos.pdf. Accessed: 2016-12-7.

BARROS, B. M. et al. An efficient network virtualization architecture for multi-tenant isolation in cloud computing. In: *To be defined*. [S.l.: s.n.], In Production.

BARROS, B. M. et al. Multi-tenant isolation of what? revisiting network virtualization to provide efficient isolation in cloud computing. In: *To be defined*. [S.l.: s.n.], In Production.

BARROS, B. M. et al. A survey on security solutions for threats in cloud networking. In: *To be defined*. [S.l.: s.n.], In Production.

BASAK, D.; TOSHNIWAL, R.; MASKALIK, S.; SEQUEIRA, A. Virtualizing networking and security in the cloud. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 44, n. 4, p. 86–94, dez. 2010. ISSN 0163-5980.

BATES, A.; MOOD, B.; PLETCHER, J.; PRUSE, H.; VALAFAR, M.; BUTLER, K. Detecting co-residency with active traffic analysis techniques. In: *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*. New York, NY, USA: ACM, 2012. (CCSW '12), p. 1–12. ISBN 978-1-4503-1665-1.

BAVIER, A.; FEAMSTER, N.; HUANG, M.; PETERSON, L.; REXFORD, J. In VINI veritas: Realistic and controlled network experimentation. In: *Proc. of the 2006 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*. New York, NY, USA: ACM, 2006. p. 3–14. ISBN 1-59593-308-5.

BENHADDOU, D.; ALANQAR, W. Layer 1 virtual private networks in multidomain next-generation networks. *IEEE Communications Magazine*, v. 45, n. 4, p. 52–58, April 2007. ISSN 0163-6804.

BENTON, K.; CAMP, L. J.; SMALL, C. Openflow vulnerability assessment. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 151–152. ISBN 978-1-4503-2178-5.

BERMAN, M.; CHASE, J.; LANDWEBER, L.; NAKAO, A.; OTT, M.; RAYCHAUDHURI, D.; RICCI, R.; SESKAR, I. GENI: A federated testbed for innovative network experiments. *Computer Networks*, Elsevier, v. 61, p. 5–23, 2014.

BERNSTEIN, G.; RAJAGOPALAN, B.; SAHA, D. *Optical Network Control: Architecture, Protocols, and Standards*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0201753014.

BITTMAN, T. J.; DAWSON, P.; WARRILOW, M. *Magic Quadrant for x86 Server Virtualization Infrastructure*. [S.l.], July 2015. Available at: https://www.gartner.com/doc/3093222. Accessed: 2016-12-7.

BOUCADAIR, M.; LEVIS, P.; GRIFFIN, D.; WANG, N.; HOWARTH, M.; PAVLOU, G.; MYKONIATI, E.; GEORGATSOS, P.; QUOITIN, B.; SANCHEZ, J. R.; GARCIA-OSMA, M. L. A framework for end-to-end service differentiation: Network planes and parallel internets. *IEEE Communications Magazine*, v. 45, n. 9, p. 134–143, September 2007. ISSN 0163-6804.

BOUTABA, R.; GOLAB, W.; IRAQI, Y. Lightpaths on demand: a web-services-based management system. *IEEE Communications Magazine*, v. 42, n. 7, p. 101–107, July 2004. ISSN 0163-6804.

BRAUN, W.; MENTH, M. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, Multidisciplinary Digital Publishing Institute, v. 6, n. 2, p. 302–336, 2014.

Brocade. *Brocade 5600 vRouter Datasheet*. San Jose, CA USA, December 2015. Available at: http://www.brocade.com/content/dam/common/documents/content-types/datasheet/brocade-5600-vrouter-ds.pdf. Accessed: 2016-12-7.

BULL, R. L.; MATTHEWS, J. N. *Exploring Layer 2 Network Security Issues in Virtualized Environments Categories and Subject Descriptors*. August 2015. White Paper presented at DEFCON 23, Las Vegas, Nevada, USA.

CARUGI, M.; MCDYSAN, D. *Service Requirements for Layer 3 Provider Provisioned Virtual Private Networks (PPVPNs)*. [S.l.]: RFC Editor, April 2005. RFC 4031. (Request for Comments, 4031).

CHAO, L. *Cloud Computing Networking: Theory, Practice, and Development*. Boca Raton, FL, USA: CRC Press, Taylor & Francis Group, 2015. ISBN 978-1-4822-5482-2.

CHENG, Y.; GANTI, V.; LUBSEY, V.; SHEKHAR, M.; SWAN, C. *Software-Defined Networking Rev. 2.0*. Beaverton, OR, USA, 2014. Available at: http://www.opendatacenteralliance.org/docs/software_defined_networking_master_usage_model_rev2.pdf. Accessed: 2015-08-01.

CHOWDHURY, N.; BOUTABA, R. A survey of network virtualization. *Comput. Netw.*, New York, NY, USA, v. 54, n. 5, p. 862–876, 2010. ISSN 1389-1286.

CHOWDHURY, N. M. M. K.; BOUTABA, R. Network virtualization: state of the art and research challenges. *IEEE Communications Magazine*, v. 47, n. 7, p. 20–26, July 2009. ISSN 0163-6804.

CHUN, B.; CULLER, D.; ROSCOE, T.; BAVIER, A.; PETERSON, L.; WAWRZONIAK, M.; BOWMAN, M. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, ACM, v. 33, n. 3, p. 3–12, 2003.

CISCO. *Understanding VLAN Trunk Protocol (VTP)*. [S.l.], September 2014. Available at: http://www.cisco.com/c/en/us/support/docs/lan-switching/vtp/10558-21.html. Accessed: 2016-12-7.

Cisco. *Cisco Discovery Protocol Version 2*. 2016. Available at: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cdp/configuration/15-mt/cdp-15-mt-book/nm-cdp-discover.html. Accessed: 2016-12-7.

Cisco. *Cisco Global Cloud Index: Forecast and Methodology, 2014-2019 White Paper*. [S.l.], April 2016. Available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html. Accessed: 2016-12-7.

Cisco. *Cisco Nexus 1000V Switch for VMware vSphere*. 2016. Available at: http://www.cisco.com/c/en/us/products/switches/nexus-1000v-switch-vmware-vsphere/. Accessed: 2016-12-7.

CloudStack. *Apache CloudStack: Open Source Cloud Computing*. 2016. Available at: https://cloudstack.apache.org/. Accessed: 2016-12-7.

CloudStack. *Managing Networks and Traffic*. 2016. Available at: http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.8/networking_and_traffic.html. Accessed: 2016-12-7.

COHEN, R.; BARABASH, K.; ROCHWERGER, B.; SCHOUR, L.; CRISAN, D.; BIRKE, R.; MINKENBERG, C.; GUSAT, M.; RECIO, R.; JAIN, V. An intent-based approach for network virtualization. In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. Ghent: IEEE, 2013. p. 42–50.

CVE. *Vulnerability Details : CVE-2011-4326*. 2012. Available at: https://www.cvedetails.com/cve/CVE-2011-4326/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2012-3449*. 2012. Available at: https://www.cvedetails.com/cve/CVE-2012-3449/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2013-1178*. 2013. Available at: https://www.cvedetails.com/cve/CVE-2013-1178/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2013-1178*. 2013. Available at: http://www.cvedetails.com/cve/CVE-2013-1178/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2011-1763*. 2014. Available at: https://www.cvedetails.com/cve/CVE-2011-1763/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2014-0187*. 2014. Available at: https://www.cvedetails.com/cve/CVE-2014-0187/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2015-0718*. 2016. Available at: https://www.cvedetails.com/cve/CVE-2015-0718/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2016-1465*. 2016. Available at: https://www.cvedetails.com/cve/CVE-2016-1465/. Accessed: 2016-12-7.

CVE. *Vulnerability Details : CVE-2016-2074*. 2016. Available at: https://www.cvedetails.com/cve/CVE-2016-2074/. Accessed: 2016-12-7.

DAVIE, B.; GROSS, J. *A Stateless Transport Tunneling Protocol for Network Virtualization (STT)*. [S.l.], April 2016. Work in Progress. Available at: https://tools.ietf.org/html/draft-davie-stt-08. Accessed: 2016-12-7.

DENTON, J. *OpenStack Networking Essentials*. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1785283277.

DEY, K.; MISHRA, D.; KULKARNI, P. Vagabond: Dynamic network endpoint reconfiguration in virtualized environments. In: *Proceedings of the ACM Symposium on Cloud Computing*. New York, NY, USA: ACM, 2014. (SOCC '14), p. 21:1–21:13. ISBN 978-1-4503-3252-1.

DIERKS, T. *The Transport Layer Security (TLS) Protocol Version 1.2*. [S.l.]: RFC Editor, August 2008. RFC 5246. (Request for Comments, 5246).

DMTF. *Open Virtualization Format White Paper*. Portland, OR, USA, April 2014. Available at: https://www.dmtf.org/sites/default/files/standards/documents/DSP2017_2.0.0.pdf. Accessed: 2016-12-7.

DONG, Y.; YANG, X.; LI, J.; LIAO, G.; TIAN, K.; GUAN, H. High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing*, Elsevier, v. 72, n. 11, p. 1471–1480, 2012.

DONG, Y.; ZHANG, X.; DAI, J.; GUAN, H. Hyvi: A hybrid virtualization solution balancing performance and manageability. *IEEE Transactions on Parallel and Distributed Systems*, v. 25, n. 9, p. 2332–2341, Sept 2014. ISSN 1045-9219.

DPDK. *Data Plane Development Kit*. 2016. Available at: http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html. Accessed: 2016-12-7.

ENNS, R.; BJORKLUND, M.; BIERMAN, A.; SCHÖNWÄLDER, J. *Network Configuration Protocol (NETCONF)*. [S.l.]: RFC Editor, oct 2015. RFC 6241. (Request for Comments, 6241).

Ericsson. *Ericsson Virtual Router Datasheet*. 200 Holger Way, San Jose, CA 95134, August 2015. Available at: http://archive.ericsson.net/service/internet/picov/get?DocNo=3/28701-FGB1010557&Lang=EN. Accessed: 2016-12-7.

ETSI. *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action*. Darmstadt, Germany, October 2012. Available at: https://portal.etsi.org/NFV/NFV_White_Paper.pdf. Accessed: 2016-12-7.

FARINACCI, D.; HANKS, S. P.; MEYER, D.; TRAINA, P. S. *Generic Routing Encapsulation (GRE)*. [S.l.]: RFC Editor, March 2000. RFC 2784. (Request for Comments, 2784).

FEAMSTER, N.; GAO, L.; REXFORD, J. How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review*, ACM, v. 37, n. 1, p. 61–64, 2007.

Floodlight. *Floodlight OpenFlow Controller*. 2016. Available at: http://www.projectfloodlight.org/floodlight/. Accessed: 2016-12-7.

FRASER, K.; H, S.; NEUGEBAUER, R.; PRATT, I.; WILLIAMSON, M. Safe hardware access with the xen virtual machine monitor. In: *In Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure*. [S.l.: s.n.], 2004.

Gartner. *Focus on Three Areas of Cloud Security*. 2016. Available at: http://www.gartner.com/smarterwithgartner/three-areas-cloud-security/. Accessed: 2016-12-7.

GHEIN, L. D. *MPLS Fundamentals*. 1st ed. 800 East 96th Street, Indianapolis, IN 46240 USA: Cisco Press, 2006. ISBN 1-58705-197-4.

GONZALEZ, N.; MIERS, C.; REDÍGOLO, F.; SIMPLÍCIO, M.; CARVALHO, T.; NÄSLUND, M.; POURZANDI, M. A quantitative analysis of current security concerns and solutions for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, v. 1, n. 1, p. 1–18, 2012.

GRASA, E.; JUNYENT, G.; FIGUEROLA, S.; LOPEZ, A.; SAVOIE, M. Uclpv2: a network virtualization framework built on web services [web services in telecommunications, part ii]. *IEEE Communications Magazine*, v. 46, n. 3, p. 126–134, March 2008. ISSN 0163-6804.

GREENE, K. TR10: Software-defined networking. *Technology Review (MIT)*, 2009.

GROBAUER, B.; WALLOSCHEK, T.; STOCKER, E. Understanding cloud computing vulnerabilities. *IEEE Security Privacy*, v. 9, n. 2, p. 50–57, March 2011. ISSN 1540-7993.

HAN, B.; GOPALAKRISHNAN, V.; JI, L.; LEE, S. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 0163-6804.

HANDIGOL, N.; HELLER, B.; JEYAKUMAR, V.; LANTZ, B.; MCKEOWN, N. Reproducible network experiments using container-based emulation. In: ACM. *Proc. of the 8th Int. Conf. on Emerging networking experiments and technologies*. [S.l.], 2012. p. 253–264.

HAO, F.; LAKSHMAN, T. V.; MUKHERJEE, S.; SONG, H. Secure cloud computing with a virtualized network infrastructure. In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. Berkeley, CA, USA: USENIX Association, 2010. (HotCloud'10), p. 16–16.

HUANG, S.; BALDINE, I. Performance evaluation of 10ge nics with sr-iov support: I/o virtualization and network stack optimizations. In: SCHMITT, J. B. (Ed.). *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance: 16th International GI/ITG Conference, MMB & DFT 2012, Kaiserslautern, Germany, March 19-21, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 197–205. ISBN 978-3-642-28540-0.

IEEE. IEEE standard for local and metropolitan area networks virtual bridged local area networks. *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)*, p. 1–285, 2006.

IEEE. Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks–amendment 21: Edge virtual bridging. *IEEE Std 802.1Qbg-2012 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, IEEE Std 802.1Qbf-2011, and IEEE Std 802.aq-2012)*, p. 1–191, July 2012.

IEEE. IEEE standard for local and metropolitan area networks–virtual bridged local area networks–bridge port extension. *IEEE Std 802.1BR-2012*, p. 1–135, July 2012.

IEEE. IEEE standard for local and metropolitan area networks–bridges and bridged networks. *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, p. 1–1832, Dec 2014.

IETF. *Host extensions for IP multicasting*. RFC Editor, August 1989. RFC 1112. (Request for Comments, 1112). Disponível em: <https://rfc-editor.org/rfc/rfc1112.txt>. Acesso em: 07/12/2016.

INCI, M. S.; GULMEZOGLU, B.; IRAZOQUI, G.; EISENBARTH, T.; SUNAR, B. *Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud*. 2015. Cryptology ePrint Archive, Report 2015/898. Available at: http://eprint.iacr.org/2015/898. Accessed: 2016-12-7.

Intel. *Data Plane Development Kit (DPDK)*. 2016. Available at: http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html. Accessed: 2016-12-7.

Intel. *Virtual Machine Device Queues (VMDq)*. 2016. Available at: http://www.intel.com.br/content/www/br/pt/ethernet-products/converged-network-adapters/io-acceleration-technology-vmdq.html. Accessed: 2016-12-7.

ISO. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Vernier, Geneva, Switzerland, 1994. Available at: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. Accessed: 2016-12-7.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, v. 51, n. 11, p. 24–31, November 2013. ISSN 0163-6804.

JAJSZCZYK, A. Automatically switched optical networks: benefits and requirements. *IEEE Communications Magazine*, v. 43, n. 2, p. S10–S15, Feb 2005. ISSN 0163-6804.

JAMMAL, M.; SINGH, T.; SHAMI, A.; ASAL, R.; LI, Y. Software-defined networking: State of the art and research challenges. *CoRR*, abs/1406.0124, 2014.

KAUFMAN, L. M. Data security in the world of cloud computing. *Security & Privacy, IEEE*, IEEE, v. 7, n. 4, p. 61–64, 2009.

KAVANAGH, A. Openstack as the api framework for nfv: the benefits, and the extensions needed. *Ericsson Review*, n. 3, April 2015.

KELLER, E.; SZEFER, J.; REXFORD, J.; LEE, R. B. Nohype: virtualized cloud infrastructure without the virtualization. In: ACM. *ACM SIGARCH Computer Architecture News*. [S.l.], 2010. v. 38, p. 350–361.

KIM, D.; GIL, J.-M.; WANG, G.; KIM, S.-H. Integrated sdn and non-sdn network management approaches for future internet environment. In: *Multimedia and Ubiquitous Engineering*. Daegu, Korea: Springer, 2013. p. 529–536.

KLÃ¶TI, R.; KOTRONIS, V.; SMITH, P. Openflow: A security analysis. In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2013. p. 1–6. ISSN 1092-1648.

KOURTIS, M. A.; XILOURIS, G.; RICCOBENE, V.; MCGRATH, M. J.; PETRALIA, G.; KOUMARAS, H.; GARDIKIS, G.; LIBERAL, F. Enhancing vnf performance by exploiting sr-iov and dpdk packet processing acceleration. In: *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. [S.l.: s.n.], 2015. p. 74–78.

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440, 2014. Disponível em: <http://arxiv.org/abs/1406.0440>. Acesso em: 07/12/2016.

KUTCH, P. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. *Intel application note*, p. 321211–002, 2011.

KVM. *Kernel Virtual Machine*. 2016. Available at: http://www.linux-kvm.org. Accessed: 2016-12-7.

KVM. *KVM Networking*. 2016. Available at: http://www.linux-kvm.org/page/Networking. Accessed: 2016-12-7.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. *Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. [S.l.], 2010. p. 19.

LEE, G. *Cloud Networking: Understanding Cloud-based Data Center Networks*. Walthamm MA, USA: Morgan Kaufmann, Elsevier, 2014. ISBN 978-0128007280.

LIN, A. d.; FRANKE, H.; LI, C. s.; LIAO, W. Toward performance optimization with cpu offloading for virtualized multi-tenant data center networks. *IEEE Network*, v. 30, n. 3, p. 59–63, May 2016. ISSN 0890-8044.

LIN, Y.; PITT, D.; HAUSHEER, D.; JOHNSON, E.; LIN, Y. Software-defined networking: Standardization for cloud computing's second wave. *Computer*, v. 47, n. 11, p. 19–21, 2014. ISSN 0018-9162.

Linux Foundation. *OpenDaylight, a Linux Foundation Collaborative Project*. 2016. Available at: http://www.opendaylight.org/. Accessed: 2016-12-7.

Linux Kernel. *Open vSwitch datapath developer documentation*. 2016. Available at: https://www.kernel.org/doc/Documentation/networking/openvswitch.txt. Accessed: 2016-12-7.

MADSEN, T. *Provider Provisioned Virtual Private Network (VPN) Terminology*. [S.l.]: RFC Editor, March 2005. RFC 4026. (Request for Comments, 4026).

MAHALINGAM, M.; SRIDHAR, T.; BURSELL, M.; KREEGER, L.; WRIGHT, C.; DUDA, K.; AGARWAL, P.; DUTT, D. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. [S.l.]: RFC Editor, August 2014. RFC 7348. (Request for Comments, 7348).

MANNIE, E.; PAPADIMITRIOU, D. *Generalized Multi-Protocol Label Switching (GMPLS) Extensions for Synchronous Optical Network (SONET) and Synchronous Digital Hierarchy (SDH) Control*. [S.l.]: RFC Editor, August 2006. RFC 4606. (Request for Comments, 4606).

MARSTON, S.; LI, Z.; BANDYOPADHYAY, S.; ZHANG, J.; GHALSASI, A. Cloud computing - the business perspective. *Decision Support Systems*, Elsevier, v. 51, n. 1, p. 176–189, 2011.

MAUCH, V.; KUNZE, M.; HILLENBRAND, M. High performance cloud computing. *Future Generation Computer Systems*, Elsevier, v. 29, n. 6, p. 1408–1416, 2013.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008.

MECHTRI, M.; HOUIDI, I.; LOUATI, W.; ZEGHLACHE, D. SDN for inter cloud networking. In: *IEEE SDN for Future Networks and Services (SDN4FNS)*. [S.l.: s.n.], 2013. p. 1–7.

MEDVED, J.; VARGA, R.; TKACIK, A.; GRAY, K. Opendaylight: Towards a model-driven sdn controller architecture. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. [S.l.: s.n.], 2014. p. 1–6.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. [S.l.], September 2011. Available at: http://dx.doi.org/10.6028/NIST.SP.800-145. Accessed: 2016-12-7.

MIERS, C. C.; KOSLOVSKI, G. P.; JR, M. A. S.; CARVALHO, T. C. M. B.; REDIGOLO, F. F.; RODRIGUES, B. B.; BARROS, B. M.; GONZALEZ, N. M.; ROJAS, M. A. T.; IWAYA, L. H. Book Chapter, *Análise de Segurança para Soluções de Computação em Nuvem*. Porto Alegre, RS, Brazil: Sociedade Brasileira de Computação (SBC), May 2014. 194–243 p. 32rd Brazilian Symposium on Computer Networks and Distributed Systems. (Short Courses). Available at: http://www.sbrc2014.ufsc.br/?pg=minicursos&id=5. Accessed: 2016-12-7.

MIJUMBI, R.; SERRAT, J.; GORRICHO, J. L.; BOUTEN, N.; TURCK, F. D.; BOUTABA, R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X.

MODI, C.; PATEL, D.; BORISANIYA, B.; PATEL, H.; PATEL, A.; RAJARAJAN, M. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, v. 36, n. 1, p. 42 – 57, 2013. ISSN 1084-8045.

MUDIGONDA, J.; YALAGANDULA, P.; MOGUL, J.; STIEKES, B.; POUFFARY, Y. Netlord: A scalable multi-tenant network architecture for virtualized datacenters. In: *Proceedings of the ACM SIGCOMM 2011 Conference*. New York, NY, USA: ACM, 2011. (SIGCOMM '11), p. 62–73. ISBN 978-1-4503-0797-0.

NAG, S.; LAM, L. ling; DHARMASTHIRA, Y.; ESCHINGER, C.; ANDERSON, R. P.; TORNBOHM, C.; GRANETTO, B. F.; TRAMACERE, G.; BLACKMORE, D.; CORREIA, J. M.; WURSTER, L. F.; EID, T.; CONTU, R.; BISCOTTI, F.; PANG, C.; SINGH, T.; SWINEHART, H. H.; SOOD, B.; MONTGOMERY, N.; DOMINY, M.; PETRI, G.; YEATES, M.; HARE, J.; WOODWARD, A.; CORRIVEAU, J. *Forecast: Public Cloud Services, Worldwide, 2014-2020, 1Q16 Update*. [S.l.], April 2016. Available at: https://www.gartner.com/doc/3273418. Accessed: 2016-12-7.

ONF. *Open Networking Foundation*. 2016. Available at: https://www.opennetworking.org/. Accessed: 2016-12-7.

ONOS. *ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out*. 2016. Available at: http://onosproject.org/. Accessed: 2016-12-7.

Open vSwitch. *How to Port Open vSwitch to New Software or Hardware*. 2016. Available at: https://github.com/openvswitch/ovs/blob/master/PORTING.md. Accessed: 2016-12-7.

Open vSwitch. *Openvswitch Native tunneling configuration guide*. 2016. Available at: http://openvswitch.org/support/config-cookbooks/userspace-tunneling/. Accessed: 2016-12-7.

Open vSwitch. *Production Quality, Multilayer Open Virtual Switch*. 2016. Available at: http://openvswitch.org/. Accessed: 2016-12-7.

OpenDaylight. *OpenDaylight Helium SR3*. 2016. Available at: https://www.opendaylight.org/software/downloads/helium-sr3. Accessed: 2016-12-7.

OPENFLOW. *Specification, OpenFlow Switch – v1.0.0*. 2009.

OPENFLOW. *Specification, OpenFlow Switch – v1.3.0*. 2012.

OpenNebula. *OpenNebula 5.0.2 Documentation - Open vSwitch Networks*. 2016. Available at: http://docs.opennebula.org/5.0/deployment/open_cloud_networking_setup/openvswitch.html. Accessed: 2016-12-7.

OpenNebula. *OpenNebula: Flexible Enterprise Cloud Made Simple*. 2016. Available at: http://opennebula.org/. Accessed: 2016-12-7.

OpenStack. *Horizon: The OpenStack Dashboard Project*. 2016. Available at: http://docs.openstack.org/developer/horizon/. Accessed: 2016-12-7.

OpenStack. *Neutron OVS DVR - Distributed Virtual Router*. 2016. Available at: https://specs.openstack.org/openstack/neutron-specs/specs/juno/neutron-ovs-dvr.html. Accessed: 2016-12-7.

OpenStack. *OpenStack Command-Line Interface Reference*. 2016. Available at: http://docs.openstack.org/cli-reference/. Accessed: 2016-12-7.

OpenStack. *OpenStack Documentation - Scenario: Classic with Open vSwitch*. 2016. Available at: http://docs.openstack.org/mitaka/networking-guide/scenario-classic-ovs.html. Accessed: 2016-12-7.

OpenStack. *OpenStack Icehouse*. 2016. Available at: https://releases.openstack.org/icehouse/index.html. Accessed: 2016-12-7.

OpenStack. *OpenStack: Open Source Cloud Computing Software*. 2016. Available at: https://www.openstack.org/. Accessed: 2016-12-7.

OPNFV. *Open Platform for NFV (OPNFV)*. 2016. Available at: https://www.opnfv.org/. Accessed: 2016-12-7.

PAN, J.; PAUL, S.; JAIN, R. A survey of the research on future internet architectures. *Communications Magazine, IEEE*, IEEE, v. 49, n. 7, p. 26–36, 2011.

PATE, P.; BRYANT, S. *Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture*. [S.l.]: RFC Editor, March 2005. RFC 3985. (Request for Comments, 3985).

PCI-SIG. *PCI-SIG: the community responsible for developing and maintaining the standardized approach to peripheral component I/O data transfers*. 2015. Available at: http://pcisig.com/. Accessed: 2015-08-01.

PFAFF, B.; DAVIE, B. *The Open vSwitch Database Management Protocol*. [S.l.]: RFC Editor, dec 2013. RFC 7047. (Request for Comments, 7047).

PFAFF, B.; PETTIT, J.; AMIDON, K.; CASADO, M.; KOPONEN, T.; SHENKER, S. Extending networking into the virtualization layer. In: *Hotnets*. New York City, NY: ACM, 2009.

PHAAL, P.; LAVINE, M. *sflow version 5*. 2004. Available at: http://www.sflow.org/sflow_version_5.txt. Accessed: 2016-12-7.

PIGNATARO, C.; MCGILL, N. *Layer 2 Tunneling Protocol Version 3 (L2TPv3) Extended Circuit Status Values*. [S.l.]: RFC Editor, August 2009. RFC 5641. (Request for Comments, 5641).

PORTNOY, M. *Virtualization Essentials*. 1st ed. [S.l.]: Sybex, 2012.

REN, K.; WANG, C.; WANG, Q. Security challenges for the public cloud. *IEEE Internet Computing*, v. 16, n. 1, p. 69–73, Jan 2012. ISSN 1089-7801.

RightScale. *RightScale 2016 State of the Cloud Report*. [S.l.], 2016. Available at: http://assets.rightscale.com/uploads/pdfs/RightScale-2016-State-of-the-Cloud-Report.pdf. Accessed: 2016-12-7.

RISTENPART, T.; TROMER, E.; SHACHAM, H.; SAVAGE, S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2009. (CCS '09), p. 199–212. ISBN 978-1-60558-894-0.

ROSEN, E.; ANDERSSON, L. *Framework for Layer 2 Virtual Private Networks (L2VPNs)*. [S.l.]: RFC Editor, September 2006. RFC 4664. (Request for Comments, 4664).

ROSEN, R. *Linux kernel networking: Implementation and theory*. [S.l.]: Apress, 2014.

ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; CORRêA, C. N. A.; LUCENA, S. Cunha de; RASZUK, R. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: ACM, 2012. (HotSDN '12), p. 13–18. ISBN 978-1-4503-1477-0.

RUSSELL, R. Virtio: Towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 42, n. 5, p. 95–103, jul. 2008. ISSN 0163-5980.

SABHARWAL, N.; SHANKAR, R. *Apache CloudStack Cloud Computing*. Birmingham, UK: Packt Publishing, 2013. ISBN 978-1782160106.

SARATHY, V.; NARAYAN, P.; MIKKILINENI, R. Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure. In: IEEE. *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. Larissa: IEEE, 2010. p. 48–53.

SDXCENTRAL. *SDN and NFV Market Size Report*. [S.l.], 2015.

SEO, K.; KENT, D. S. T. *Security Architecture for the Internet Protocol*. [S.l.]: RFC Editor, December 2005. RFC 4301. (Request for Comments, 4301).

SERBEST, Y.; AUGUSTYN, W. *Service Requirements for Layer 2 Provider-Provisioned Virtual Private Networks*. [S.l.]: RFC Editor, September 2006. RFC 4665. (Request for Comments, 4665).

SHERWOOD, R.; GIBB, G.; YAP, K.-K.; APPENZELLER, G.; CASADO, M.; MCKEOWN, N.; PARULKAR, G. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, p. 1–13, 2009.

SHERWOOD, R.; GIBB, G.; YAP, K.-K.; APPENZELLER, G.; CASADO, M.; MCKEOWN, N.; PARULKAR, G. M. Can the production network be the testbed? In: *OSDI*. [S.l.: s.n.], 2010. v. 10, p. 1–6.

SHIEH, A.; KANDULA, S.; GREENBERG, A.; KIM, C. Seawall: performance isolation for cloud datacenter networks. In: USENIX ASSOCIATION. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. [S.l.], 2010. p. 1–1.

SMITH, B.; HARDIN, J.; PHILLIPS, G.; PIERCE, B. *Linux Appliance Design: A Hands-on Guide to Building Linux Appliances*. [S.l.]: No Starch Press, 2007.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 7th ed. Upper Saddle River, New Jersey, USA: Pearson, 2016. ISBN 13:978-0-13-335469-0.

STEFFANN, S.; BEIJNUM, I. van; REIN, R. van. *A Comparison of IPv6-over-IPv4 Tunnel Mechanisms*. [S.l.]: RFC Editor, November 2013. RFC 7059. (Request for Comments, 7059).

SUBASHINI, S.; KAVITHA, V. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, Elsevier, v. 34, n. 1, p. 1–11, 2011.

SUNDARARAJ, A. I.; DINDA, P. A. Towards virtual networks for virtual machine grid computing. In: *Virtual machine research and technology symposium*. [S.l.: s.n.], 2004. p. 177–190.

SUZUKI, M.; CALLON, R. *A Framework for Layer 3 Provider-Provisioned Virtual Private Networks (PPVPNs)*. [S.l.]: RFC Editor, July 2005. RFC 4110. (Request for Comments, 4110).

TAKEDA, T. *Framework and Requirements for Layer 1 Virtual Private Networks*. [S.l.]: RFC Editor, April 2007. RFC 4847. (Request for Comments, 4847).

TAVAKOLI, Z.; MEIER, S.; VENSMER, A. A framework for security context migration in a firewall secured virtual machine environment. In: ____. *Information and Communication Technologies: 18th EUNICE/ IFIP WG 6.2, 6.6 International Conference, EUNICE 2012, Budapest, Hungary, August 29-31, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 41–51. ISBN 978-3-642-32808-4.

The Xen Project. *The Xen Project, the powerful open source industry standard for virtualization*. 2016. Available at: http://www.xenproject.org/help/documentation.html. Accessed: 2016-12-7.

TSENG, J.; WANG, R.; TSAI, J.; EDUPUGANTI, S.; MIN, A. W.; WOO, S.; JUNKINS, S.; TAI, T. Y. C. Exploiting integrated gpus for network packet processing workloads. In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. [S.l.: s.n.], 2016. p. 161–165.

TURNER, J.; TAYLOR, D. Diversifying the Internet. In: *IEEE Global Telecommunications Conference (GLOBECOM'05)*. [S.l.: s.n.], 2005. v. 2, p. 6–pp.

VALENCIA, A. J.; ZORN, G.; PALTER, W.; PALL, G.-S.; TOWNSLEY, W. M.; RUBENS, A. *Layer Two Tunneling Protocol "L2TP"*. [S.l.]: RFC Editor, August 1999. RFC 2661. (Request for Comments, 2661).

VARADARAJAN, V.; KOOBURAT, T.; FARLEY, B.; RISTENPART, T.; SWIFT, M. M. Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2012. (CCS '12), p. 281–292. ISBN 978-1-4503-1651-4.

VARADARAJAN, V.; ZHANG, Y.; RISTENPART, T.; SWIFT, M. A placement vulnerability study in multi-tenant public clouds. In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015. p. 913–928. ISBN 978-1-931971-232.

VirtualBox. *VirtualBox Virtual Networking*. 2016. Available at: https://www.virtualbox.org/manual/ch06.html. Accessed: 2016-12-7.

VMware. *vSphere Networking*. Update 2. Palo Alto, CA, 2016. Available at: http://www.linux-kvm.org/page/Networking. Accessed: 2016-12-7.

WANG, A.; IYER, M.; DUTTA, R.; ROUSKAS, G. N.; BALDINE, I. Network virtualization: Technologies, perspectives, and frontiers. *Journal of Lightwave Technology*, v. 31, n. 4, p. 523–537, Feb 2013. ISSN 0733-8724.

WANG, Y.-S.; GARG, P. *NVGRE: Network Virtualization Using Generic Routing Encapsulation*. [S.l.]: RFC Editor, September 2015. RFC 7637. (Request for Comments, 7637).

WARRILOW, M. *Market Trends: x86 Server Virtualization, Worldwide, 2016*. [S.l.], April 2016. Available at: https://www.gartner.com/doc/3285825. Accessed: 2016-12-7.

Xen. *XAPI: Open source software to build private and public clouds*. 2016. Available at: https://www.xenproject.org/developers/teams/xapi.html. Accessed: 2016-12-7.

Xen. *Xen Networking*. 2016. Available at: http://wiki.xenproject.org/wiki/Xen_Networking. Accessed: 2016-12-7.

YONG, L.; XU, X.; CRABBE, E.; HERBERT, T. *GRE-in-UDP Encapsulation*. [S.l.], July 2016. Work in Progress. Available at: https://tools.ietf.org/html/draft-ietf-tsvwg-gre-in-udp-encap-16. Accessed: 2016-12-7.

YOON, C.; PARK, T.; LEE, S.; KANG, H.; SHIN, S.; ZHANG, Z. Enabling security functions with sdn: A feasibility study. *Computer Networks*, v. 85, p. 19 – 35, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128615001619>. Acesso em: 07/12/2016.

ZISSIS, D.; LEKKAS, D. Addressing cloud computing security issues. *Future Generation Computer Systems*, v. 28, n. 3, p. 583 – 592, 2012. ISSN 0167-739X.

ZORN, G.; PALL, G.-S.; HAMZEH, K. *Point-to-Point Tunneling Protocol (PPTP)*. [S.l.]: RFC Editor, July 1999. RFC 2637. (Request for Comments, 2637).