Towards Migrating Security Policies along with Virtual Machines in Cloud

A Thesis in the Concordia Institute for Information Systems Engineering

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Applied Science in Information Systems Security
at the Concordia Institute for Information Systems Engineering
Concordia University, Montréal, Québec, Canada

Sahba Sadri

December 2013

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Sahba Sadri**
Entitled: **Towards Migrating Security Policies along with Virtual Machines in Cloud**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Andrea Schiffauerova, Ph.D. _____ Chair

Amr M. Youssef, Ph.D. _____ Examiner

Yan Liu, Ph.D. _____ Examiner

Mourad Debbabi, Ph.D. _____ Supervisor

Lingyu Wang, Ph.D., P.Eng. _____ Supervisor

Approved by _____
Chair of Department or Graduate Program Director

_____ 2013 _____
Christopher Trueman, Ph.D., Dean
Faculty of Engineering and Computer Science

Abstract

Towards Migrating Security Policies along with Virtual Machines in Cloud

Sahba Sadri

Multi-tenancy and elasticity are important characteristics of every cloud. Multi-tenancy can be economical; however, it raises some security concerns. For example, contender companies may have Virtual Machines (VM) on the same server and have access to the same resources. There is always the possibility that one of them tries to get access to the opponent's data. In order to address these concerns, each tenant in the cloud should be secured separately and firewalls are one of the means that can help in that regard. Firewalls also protect virtual machines from the outside threats using access control lists and policies. On the other hand, virtual machines migrate frequently in an elastic cloud and this raises another apprehension about what happens to the security policies that are associated with the migrated virtual machine.

In this thesis, we primarily contribute by proposing a novel framework that coordinates the mobility of the associated security policies along with the virtual machine in Software-Defined Networks (SDN). We then design and develop a prototype application called Migration Application (MigApp), based on our framework that moves security policies and coordinates virtual machine and security policy migration. MigApp runs on top of SDN controllers and uses a distributed messaging system in order

to interact with virtual machine monitor and other MigApp instances. We integrate MigApp with Floodlight controller and evaluate our work through simulations.

In addition, we prepare a test-bed for security testing in clouds that are based on traditional networks. We focus on virtual machine migration and use open-source utilities to equip this test-bed. We design an architecture based on GNS3 network emulator in order to provide a distributed testing environment. We then propose a virtual machine migration framework on Oracle VirtualBox; and finally, we enrich the security aspect of framework by adding firewall rule migration and security verification mechanisms into it.

Acknowledgments

I would like to sincerely thank my supervisors Dr. Mourad Debbabi and Dr. Lingyu Wang. They introduced me to interesting research projects on which I have enjoyed so much to work.

My appreciation also goes to Dr. Makan Pourzandi, for his guidance and advices which were helpful during some steps of this thesis.

I would also like to express my warmest gratitude to Dr. Yosr Jarraya and Arash Eghtesadi, my research mates in all of the projects.

Finally, I am thankful to all my colleagues in the department, particularly Gaby Dagher, Mert Kara and my mentor, Ashkan Rahimian who made a friendly environment for me and I had great time working and studying with them.

Thank you all

To my parents who encouraged me from the beginning and without their support I was not able to complete this step of my life ...

"If you're not prepared to be wrong, you'll never come up with anything original."

-Sir Ken Robinson

Contents

List of Figures

List of Tables

List of Acronyms

ACL	Access Control List
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
CIFS	Common Internet File System
CSP	Cloud Service Provider
GNS3	Graphical Network Simulator 3
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
NAT	Network Address Translation
NFS	Network File System
NIC	Network Interface Controller
NOS	Network Operating System
OS	Operating System
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer

SaaS	Software as a Service
SDN	Software-Defined Network
SFTP	Secure File Transfer Protocol
SMB	Server Message Block
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
UTM	Unified Threat Management
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network

Chapter 1

Introduction

1.1 Motivation

Cloud computing is currently known as one of the hottest topics not only in computer science, but also among recent technologies. Nowadays, many companies are adapted to cloud service model and others are aggressively starting to migrate their operations into this model. The market for public cloud computing services was $16 billion in 2008, went over $100 billion in 2012 and It is predicted that by 2016 it can exceed $206 billion [44].

Cloud computing paradigm is perhaps as old as the age of mainframes, when only big companies and academic centers had access to huge computing power. Over the time, processors got cheaper and faster which made it possible for many people around the world to have a personal computer. At the present time, various types of computers like laptops, tablets, smart phones and etc. with multiple processors are very common and popular. However, cloud computing also changed along with the computers during this evolution and turned into a service for companies and even home users. Fast improvements in computer networks infrastructure and emerging of virtualization, played the key role in this movement. Versatility of virtualization technologies and ubiquity of internet had the largest impact on maximizing the popularity of cloud computing and encouraged companies for adoption. Today, many

applications, storage services and development environments are available online and are based on cloud; others are also rapidly joining to this trend and trying to adapt their services to cloud infrastructures. This cloud-based approach reduces many costs and leads to a vast spectrum of opportunities and possibilities and expels a lot of difficulties and headaches from companies. In this approach, companies subscribe to a service that is offered by a cloud provider. To draw an analogy, a similar approach has been taken in most traditional utility services like water, electricity, telephony and etc. which end user (that can be an individual, a company or an organization) receives the service without concerning about how this service is hosted and provided to him. User consumes the service and pays a reasonable price for the amount of consumption. Normally, such a model is significantly cheaper and more convenient for the user than preparing or producing these by himself.

While, many companies are convinced that using this model has many advantages and benefits, some concerns and questions exist that need to be addressed by cloud providers. Like traditional utilities, there should be some standardized factors that ensure the quality of provided services. For example as users, we need to know that the water is safe to drink or the voltage of electricity power is always in the accepted range and doesn't hurt our electrical devices. For example, multi-tenancy which is a feature in cloud computing that enables different users to share same resources, can cause some security and privacy problems, as illustrated below. Suppose two contender companies are using same resources in a data center; Here, Cloud Service Provider (CSP) should guarantee that customers' data is confidential and has to show strong proof for each party that the opponent company cannot access their data. Generally, there is an agreement between the customer and the provider that indicates

the rights of each party and specifies fines for each of them, in case that they do not obey the contents of agreement. CSPs use Service Level Agreement (SLA) in order to define delivered services, performance measurements, customer responsibilities, compensations and reimbursement policies, security services and standards, disaster recovery and finally termination rules [102].

Recently, VM migration in cloud computing has been common within and between data centers. Basically, VM migration means moving a virtual machine from one host to another. It is an essential capability that supports service elasticity of a cloud. For example, it happens frequently that a VM has to move into another cluster within a data center due to load balancing issues or when a company decides to relocate its VMs into a larger data center. However, there is always a big concern on what happens to the security policy associated with the migrated machine. What should be done for a migrating VM in order to preserve its security and protect it from network attacks, after migration? This question highlights the importance of research on VM security during and after migration. In order to study VM migration and its related security concerns, an implementation of cloud migration is needed. Most of the current available simulators focus on performance and cost optimization and emulators that can imitate real services are normally owned by service providers. As a result, a customized testing environment that focuses on the security and provides possibility of migration can ease the mentioned research direction.

On the other hand, Software Defined network (SDN) has become one of the hottest topics in computer networking community lately and has drawn lots of attention from media and industry as well as academia because of the promising opportunities and undeniable benefits that it is offering. It is predicted that SDN market will grow

form $360 million in 2013 to $3.5 billion in 2015 and will exceed $25 billion by 2018 and as SDNcentral reported, the number of registered SDN companies grew from zero company in 2009 to 225 companies in April 2013 [80]. A significant number of vendors and famous companies like IBM, Cisco, Oracle and etc. are adapting to SDN and some others like NEC, Big Switch Networks and etc. started to produce products specifically for SDN. At the same time there is a big collaboration between academic institutions and industry leaders to redesign, adapt or invent network protocols in order to improve SDN and respond the huge demand from the market.

SDN introduces the physical separation of network data plane (aka forwarding plane) from control plane, and this separation gives possibility of managing multiple network devices by one controller [24]. SDN controller is directly programmable and the logic which is presented by the code inside the controller is responsible for managing the forwarding behavior of switches; therefore the network administrator can easily apply new changes on network based on business requirements without touching any switch which facilitates network management in a cost effective way. This approach enables the controller to manipulate switch flow tables by adding or removing flows and making a dynamic and flexible provision over data plane. Forwarding, dropping and modifying packets are examples of actions that flow entries can perform.

Now that we expressed the importance and significance of SDN and cloud computing, we need to find out whether they can have a synergic collaboration or not. Fortunately, there is a common concept between these two topics. Thanks to the virtualization technology, data centers have to deal with numerous VMs that need to be handled remotely. Sometimes for a better performance, VMs have to migrate

to other servers. Switches and routers have to be monitored all around the clock. This is not the whole story, middleboxes are another vital part of each data center that have to be maintained constantly. Diversity and dispersion of different components, introduce an immediate need for a central management unit that can take care of all elements. Fortunately, SDN is about decoupling control plane and data plane that steers the concentration of management into a central place rather than having one control unit per each device. This separation, simplifies the supervision over the network and brings more flexibility to a data center. The convergence of cloud computing needs and opportunities which are introduced by SDN, motivate CSPs to support SDN who are always thirsty to adapt new technologies in the field. Security concerns are always one of the most important deterrent factors that make CSPs hesitant to move on. As it has been mentioned earlier, multi-tenancy can raise security concerns. Firewalling between tenants could be a solution to solve this issue. However, when a VM migrates to another location, all security policies that are associated with that VM, remain in the first place. Therefore, a security solution that address this problem in SDN not only solves an issue in its own filed, but also eliminates a barrier for widespread adoption of SDN by CSPs.

1.2 Objectives

The purpose of this master thesis is first to study cloud computing concepts, security concern in clouds and data centers, Software-Defined Networks and its importance for cloud computing, role of firewalls in either domains with particular focus on VM migration and its security concerns. Then, design and implement of frameworks and prepare test-beds which their purpose is testing and evaluating VM migration

procedures as well as firewall rule migration. Finally, produce a framework that leverages combination of SDN functionalities and cloud computing in order to produce an effective way of network management, especially from security perspective.

1.3 Contributions

Security policy migration of VMs has been discussed in the literature [89]. Although, at the time of writing this thesis, there is no implementation of that for SDN. Moreover, many physical devices in traditional networks are turned into software applications in SDN context and reside on top of controller; therefore, a new design that is applicable for new context is needed. Our work presents a framework that offers a solution for a collaboration between hypervisor and controller application firewall to migrate security context and perform a safe VM migration for both source and destination data centers as well as the VM. We also prepared a cloud security test-bed for traditional networks in order to perform VM migration experiments. Therefore, our main contributions are as follows:

- Design of a novel framework that enables the mobility of security policies during VMs migration in order to support cloud computing elasticity with an SDN-based architecture. The framework enables an east-west communication between management application within and across data centers.

- Implementation of the proposed framework in a prototype application integrated on the top of an existing OpenFlow controller, namely Floodlight.

- Test and validation of the security policy migration and evaluation of its performance using our framework deployed in a simulation and programming environment.

- Design and deployment of an architecture that introduces the way to use GNS3 for simulating multiple data center on multiple host machines.

- Design and implementation of a framework for migrating VMs on the test-bed by improving the features in VirtualBox.

- Design and implementation of a framework to migrate firewall rules in the case of migration in context of traditional networks.

1.4 Thesis Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we provide background information on cloud computing, Software Defined Networks and security issues in mentioned paradigms. In Chapter 3, we introduce our novel framework for migrating security context of virtual machines between SDN data centers and present our MigApp software. In chapter 4, we introduce and explain the customized test-bed and firewall rule migration framework in traditional networks, that we used for testing and verifying the reach-ability of VMs, before and after migration of VM and firewall rules. Finally in Chapter 5, we summarize the research and suggest future work directions.

Chapter 2

Background

This chapter covers some definition of main terms and reviews background knowledge for our research. We first discuss cloud computing and introduce its components and then describe security concerns in cloud. Then, we talk about SDN, its evolution and relation of SDN and cloud computing. We also give explanation about various controllers and describe the Floodlight [25] controller in detail. Finally, we explain different firewalls, particularly those who are applied in cloud computing and SDN.

2.1 Cloud Computing

Perhaps the evolving nature of cloud computing is the reason for not having a common and standard definition for that. However, when we search in the literature, the NIST definition of cloud computing is the first choice of most researchers. "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"[72].

In order to have a better understanding of this model, NIST described main characteristics of cloud computing and categorized it with deployment models and service models.

Essential Characteristics are elaborated as following [72]:

- **On-demand self-service.** Customer is able to manage resources remotely and automatically without the need to contact CSP.
- **Broad network access.** Customer access to capabilities through network and by standard mechanism. This feature has to be platform independent.
- **Resource pooling.** Sharing of the providers physical and virtual resources among multiple customers.
- **Rapid elasticity.** Described as an instant and on-demand possibility of scaling up and down the computing resources, by the customer.
- **Measured service.** This characteristic enables *pay as you go* feature with providing a complete report of resource usage for both provider and customer.

One important characteristic that is missing in above definition is *multi-tenancy* that enables coexistence of different customers' data on the same storage medium. This might be implicitly deductible from resource pooling, but we would like to express this characteristic separately, due to its importance from security perspective and our research path.

Cloud computing **Deployment Models** are described as following [72]:

- **Private cloud.** In this model, there is only one organization that uses and manages cloud infrastructure. However, it is possible to hand over the responsibility of management and maintenance to a third-party company. This model gives the highest level of control to organization, especially in terms of the security.

- **Community cloud.** In cases that organizations collaborate on projects or have shared data, the community model can be deployed to have a shared infrastructure between collaborative parties.

- **Public cloud.** An infrastructure that can be managed by any organization in order to provide services and host for general public consumers. Since this model is public, control over that is limited and sometimes is partially shifted to the users for their own security.

- **Hybrid cloud.** A combination of two or more aforementioned models can shape a hybrid infrastructure. This is the most complex method of infrastructure deployment and needs extra effort to make sure there is no security breach or lack of provision in different parts of infrastructure.

And finally, services are delivered to consumers based on the following **Service Models** [72]:

- **Software as a Service (SaaS).** Customer can access applications that are running on cloud infrastructure with various interfaces. Microsoft Office Live and Google Docs are good examples of such services.

- **Platform as a Service (PaaS).** Provider prepares a development environment including programming languages, libraries and different tools for customer in order to develop applications. Examples can be Microsoft Azure and Google App Engine.

- **Infrastructure as a Service (IaaS).** In this model, customer is served by customizable desired resources as processors, memory, network or even security

appliances. Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3) and RackSpace Cloud servers are examples of this type of service. Our research is mostly in this scope since network related issues, falls under this category.

2.1.1 Cloud Security and Security Appliances [1]

In fact, cloud computing made an enhancement to the overall security. Concentration of infrastructure in one location and ease of management and control, separation of virtual infrastructures and isolation of VMs were major improvements in security although they were just remedies for mitigation of some attacks. Security is still an issue, because not only most of the traditional attacks are still applicable, but also some new cloud-specific attacks manifest. In this part, we strive to categorize cloud-related security concerns and explain most important ways to protect with special focus on security appliance's role.

There is a rich literature on taxonomy of cloud computing security. Gartner [33], enumerates seven security risks as: *1. Privileged user access 2. Regulatory compliance 3. Data location 4. Data segregation 5. Recovery 6. Investigative support 7. Long-term viability.* CSA [43], defines cloud security in fourteen domains and grouped them in three sections as illustrated in Table 2.1:

While classification of cloud security concerns looks like an endless effort, it seems that many of them are inspired from CSA. Eventually, Gonzalez et al. [54], divided

[1]Note that, cloud computing security is different from cloud-based security services that are also know as *Security as a Service*.The latter, is out of the scope of this research.

i. Cloud Architecture	Cloud Computing Architectural Framework
ii. Governing in the Cloud	Governance and Enterprise Risk Management
	Legal Issues: Contracts and Electronic Discovery
	Compliance and Audit Management
	Information Management and Data Security
	Interoperability and Portability
iii. Operating in the Cloud	Traditional Security, Business Continuity- and Disaster Recovery
	Data Center Operations
	Incident Response
	Application Security
	Encryption and Key Management
	Identity, Entitlement, and Access Management
	Virtualization
	Security as a Service

Table 2.1: *Cloud Security Domains [43]*

cloud security issues into seven categories and then grouped them in three main domains, with respect to the most important key references (See Figure 2.1).

In order to safeguard a cloud, security should be applied in different layers. Thus, all dimensions that are shown in Figure 2.1, have to be studied carefully. For instance, data security can be improved by using stronger encryption techniques, interface breaches can be mitigated by hardening the code, and compliance issues can be solved by revising SLAs.

Security appliances that are also known as security middleboxes, are intermediary network devices that are meant to filter and inspect packets. Firewalls and Intrusion Prevention System/Intrusion Detection System (IPS/IDS) are common examples of security appliances that are extensively used in data centers for protecting it at the network level. In this regard, what type of security appliance will be used in future of cloud? Physical or Virtual? and the answer based on [69], is a hybrid approach.

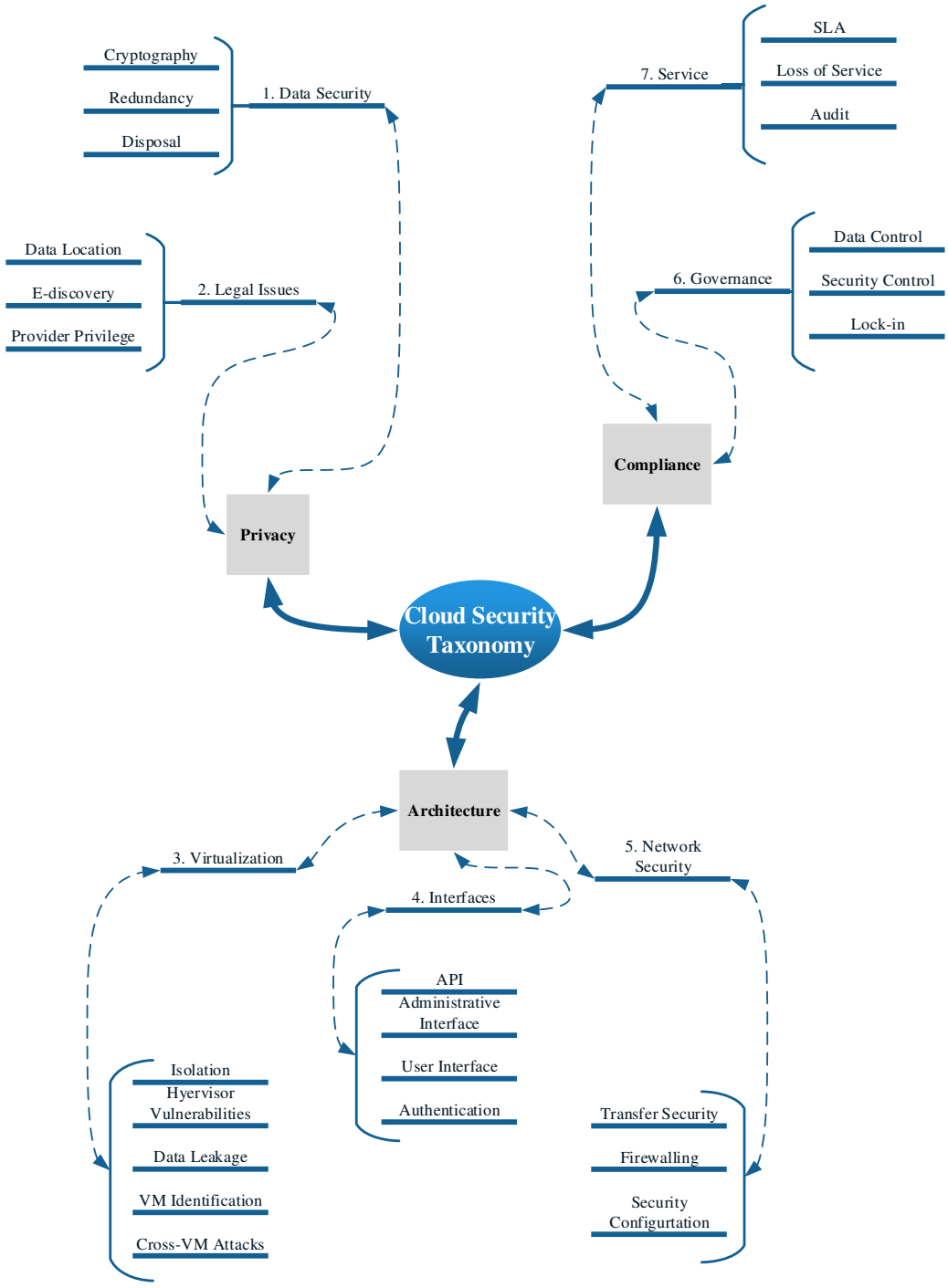

Figure 2.1: *Cloud Computing Security Domains [54]*

Neither of them are sufficient alone because some virtual traffic is always invisible from hardware layers and vice versa.

Our goal in this thesis is to study firewalls and security policy migration when a VM travels from one data center to another. Hence, Figure 2.1 clearly shows how our work is positioned.

2.1.2 Virtualization

Virtualization refers to the method for simulating software and/or hardware, rather than using the actual instance. This is performed by creating a layer of abstraction between layers of software and/or hardware, for the purpose of optimization, better organization and simplification of interaction between different elements. For instance, you can represent a single physical resource as multiple virtual (logical) resources and vice versa. This technique helps to eliminate many limitations in implantation and enables making use of resource in an efficient manner. For example, a company can put more load on its servers by using virtualization [84][31]. There are different forms of virtualization that VMware categorized them in five types as follows [17]: *Server Virtualization*, *Network Virtualization*, *Desktop Virtualization*, *Storage Virtualization* and *Application Virtualization* that all of them are applied in different layers of cloud computing. Since our research is about VM migration , we need to consider a type of virtualization that is called *full virtualization*. In this form, one or more OSs as well as the applications that are running on top it, shape Virtual Machine (VM). This OS, which is referred as a *guest OS*, runs on top virtual hardware that is provided by hypervisor and is controlled by Virtual Machine Monitor (VMM). In another word, full virtualization is a technique to simulate an entire hardware for

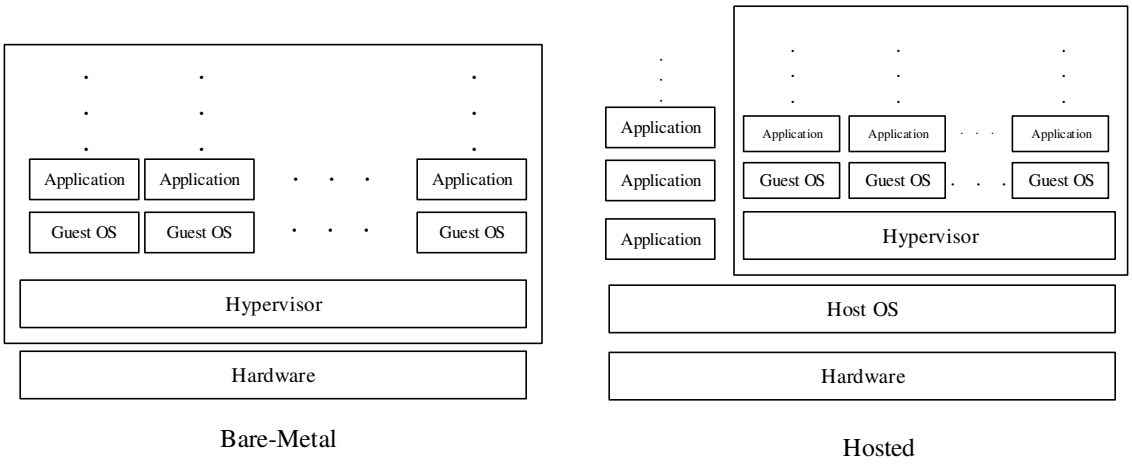

Figure 2.2: *Full-Virtualization Architectures*

a guest OS and applications. Flows of the instructions that traverse among VMs are provisioned by VMM.

As demonstrated in Figure 2.2, there are two forms of full virtualization. *Bare-metal*(aka *native*) and *Hosted*. The first one is where the hypervisor layer is exactly on top of underlying hardware and mostly is using for server virtualization. The latter is when an OS layer called *Host OS*, placed between physical hardware and hypervisor; this approach is regularly applied in desktop virtualization [84]. Full virtualization imitates the characteristics of the virtual hardware as the physical one, to the applications; therefore, there is no need to modify applications in order to run them on a VM.

It is worth-mentioning that vulnerabilities in non-virtualized servers still remain after virtualization. Moreover, there is possibility of additional attacks that are specifically related to full virtualization characteristics. Hence, attack surface will increase; however, there are solutions to mitigate them; NIST grouped them as follows: *Guest OS Isolation*, *Guest OS Monitoring* and *Image and Snapshot Management* [84]. The

security, is also dependent on securing all the separate components such as physical hardware, hypervisor, host OS, guest OSs, storage units and applications which is out of the scope of this thesis. Our focus is on underlying network security, particularly firewall's role in protecting VMs. It can be either a physical or virtual firewall. In the following section we briefly discuss about VM migration and motivations to migrate a VM, and finally we talk about security concerns and countermeasures that should be taken into consideration in VM migration process.

2.1.3 VM Migration in clouds and Security Concerns

As it was mentioned in Chapter 1, virtualization is a key element in cloud industry trend. Data centers that are also known as server farms, require a huge amount of hardware and infrastructure facilities. On the other hand, due to the fast growing technologies that data centers are hinged on and customers asking for them, Cloud Service Providers have to go under major upgrades in every few years. Therefore, data centers need to make the best use of their facilities and this is exactly where virtualization can inevitably help. Server consolidation is a great example of virtualization impact on data center efficiency improvement [61].

Hypervisors have different features and capabilities depending on their vendor. As it has been found in the literature, there are three main features, common between most of them and according to VMware, which is the leader in virtualization technology, these features are *High Availability (HA)*, *Fault Tolerance (FT)* and *Live Migration*. The first and the second features are dependent on the last one.

A short description of named features, comes as following:

- **High Availability.** A technology that monitors all VMs constantly and in case of hardware failure, immediately restarts the VMs on another server. This feature does not transfer the current state of VM and only loads the VM from the stored image in storage unit [20].

- **Fault Tolerance.** This technology complements the previous feature. It runs identical copies of a VM in another place at the same time and in case of failure in origin location, duplicated VM will continue running without interruption. This feature is dependent on live migration of VM state [19].

- **Live Migration.** This feature provides the means, to transfer a VM and its running state from a server to another in real-time [21].

VM migration can be done in two ways; within a data center or across different data centers that sometimes are a located in different continents. In addition to the mentioned motivation for migration that complements other features of hypervisor, there are more incentives to migrate a VM. One of the main reasons and motivations for VM migration is load balancing between servers. Sometimes a task needs more processor power and there are not enough resources available in the server; in this situation, a migration to another server can solve the problem. Periodical maintenance is another typical reason for VM migration. Reducing power consumption is always a desire for data centers. It happens once in a while that many severs have minimum load and it is possible to shut down or hibernate them while transferring some VMs to another servers; these migrations can significantly reduce power consumption and consequently data center cost.

According to [74], virtualization security challenges can be categorized as follows:

- **Inactive VMs.** Normally VMs receive daily updates and security patches. VMs who are offline, are not able to receive those updates and become vulnerable when they go active. As a result, they turn into security threats for entire server.

- **VM Awareness.** All security solutions and appliances are not compatible with virtualized environment. The hypervisor security is also another concern that should be taken into consideration.

- **VM Sprawl.** VMs can be created with a click of a mouse. This is the key reason of rapid growth of virtualization; however, not only security cannot be achieved as easy as that, many security weaknesses can easily duplicate and spread all over the network. Each VM needs a special care and administrator cannot apply the same solution to all VMs.

- **Inter-VM Traffic.** Traffic between the VMs is not visible to traditional physical layer security appliances. Hence, monitoring and management of that traffic can be performed by an appliance that is integrated into hypervisor.

- **Migration.** When a VM migrates from one cloud to another, the security policies that are associated with that VM, remains in the place of origin.

Our contribution in this thesis, is exactly on the migration issue. We strive to address that by migrating also the security policies along with the migrating VM.

2.2 Software-Defined Networking

Software-defined network is a new paradigm in computer networks; however, the idea of programmable network has been around for a long time. Emergence of programmable networks was due to the need of researchers to deploy new protocols and ideas into the core network which was monopolized by big network companies such as Cisco and Juniper. Before manifestation of programmable networks, control and forwarding layers were tightly coupled to each other and implementations of network hardware was mostly on Application-Specific Integrated Circuit(ASIC) chips. The main property of ASIC is ultimate speed but their drawback is non-programmability. Improvements in x86 processors and popularity of using commodity hardware, encouraged network researchers to find a way to decouple control and forwarding layers, in order to deploy desired architectures and network control capabilities. To achieve this, various approaches have been taken. Active Network [90], OpenSig [37] and Juniper SDK [10] are examples of changing the routers in a way to make them programmable [67]. Software routers and switches that are installed on servers as well as virtualization technologies, added more flexibility to network architecture; in addition, test-beds enabled the researchers to perform experiments and verification on their own prototype environment [67]. Eventually, researchers came up with the idea of role abstraction in networking which led to emergence of SDN. SDN focuses on separation of control plane and data plane when the control unit is centralized and manageable by third party application. In order to program those applications, controller exposes an API; However, programmability in SDN is not limited only to ap-

plications and the main improvement is that the control units are also programmable by data plane exposed APIs.

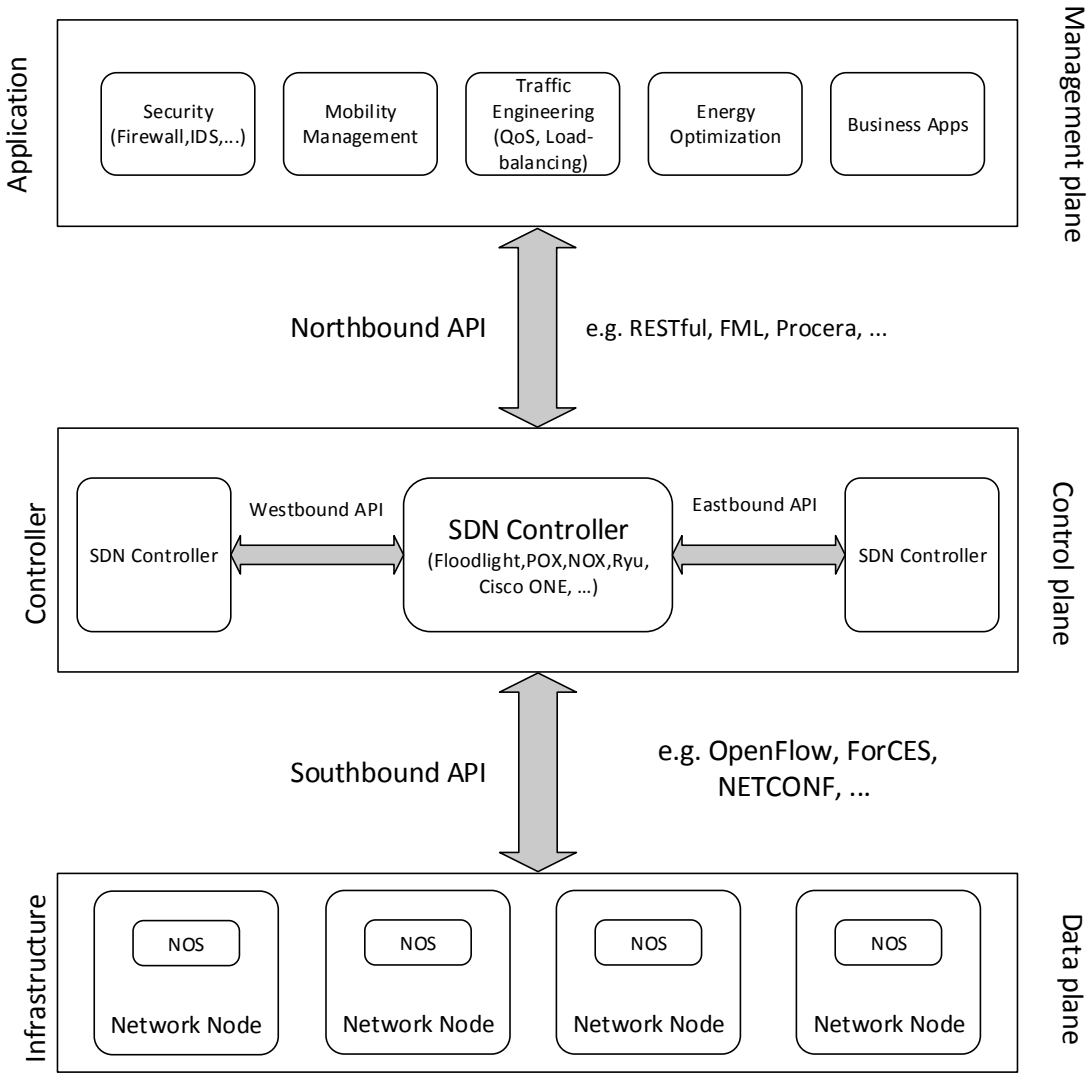

Figure 2.3: *Three Layers of SDN Architecture[85]*

As it is illustrated in Figure 2.3, there are three tiers in SDN architecture: 1) *Data plane* also known as forwarding plane, is the layers that infrastructure and particularly switches reside there. 2) *Control plane* is in the middle; controllers that

have a comprehensive view of the network and are able to control switches are placed there. 3)*Management plane* is on the top and applications which are replacements for many devices in traditional networks such as middleboxes, load-balancers and etc., as well as some new business and management applications are located on this layer. In order to make a connection between these layers, different APIs are used to provide programmable interface. As it is shown in the Figure 2.3, there are three types of API: 1)*Southbound API* is provided to make a link between control and forwarding layers. OpenFlow [71], NETCONF [46] and ForCES [45] are examples of southbound API. 2)*Northbound API* exposes an interface to application layer in order to interact with control plane. REST API is a common example for this type of programming interface; however, it completely depends on the choice of the vendor who develops the controller. 3)*Eastbound-Westbound API* is for making connections between elements in the same layer. ALTO [55] and Hyperflow [91] are examples of that.

SDN tries to simplify the network architecture and make a flat network with eliminating many middleboxes and shift their functionality to application layer. In SDN, switches have forwarding unit and a simple Network Operating System (NOS) that communicates with controller through API. Although the development of SDN is mainly due to the emergence of OpenFlow, it must be noted that the original idea is coming from Ethane [39] project and its predecessor SANE project [40]. These projects introduced a centralized controller and switches that contain flow tables and communicate with controller via a secure channel [73]. OpenFlow standardize the communication between data plane and control plane[2]. Since OpenFlow is well supported by ONF, it gained momentum among all APIs and turned into a standard

[2]OpenFlow uses Transport Layer Security (TLS) with mutual authentication to secure the connection between controller and switches.

for southbound communications; whereas there is not any commonly accepted API for northbound yet.

Like any new and evolving paradigm in computer world, there are some security breaches that are specific to that technology and have to be addressed even though it does not happen very quickly. Many shortages in design and vulnerabilities unveil after widespread deployment. Regarding to the shift of the middleboxes from physical layer to the application layer in SDN architecture, we believe that there is some space to have a contribution in security with focus on VM migration. In Chapter 3, we strive to investigate and address one of the security issues in SDN environment when a VM migration happens.

2.3 Firewalls in Cloud and SDN

As we discussed before and it is shown in Figure 2.1, *network security* that falls under architecture domain, can affect cloud security. Firewalling is one of the solutions that have significant impact on improving network security. Firewall is a security appliance that filter packets and control incoming and outgoing traffic. Still there is not enough information that gives us a clear view about firewall. Therefore, it is needed to go another level deeper and define what are main purposes and benefits of using these devices on which data centers invest huge amount money on them. Middleboxes are essential intermediary devices in the network that mainly optimize performance and security of the network. Some of them such as WAN optimizers and IDSs specifically focus on one issue; respectively performance and security, and others can be helpful in both directions (See Figure 2.4).

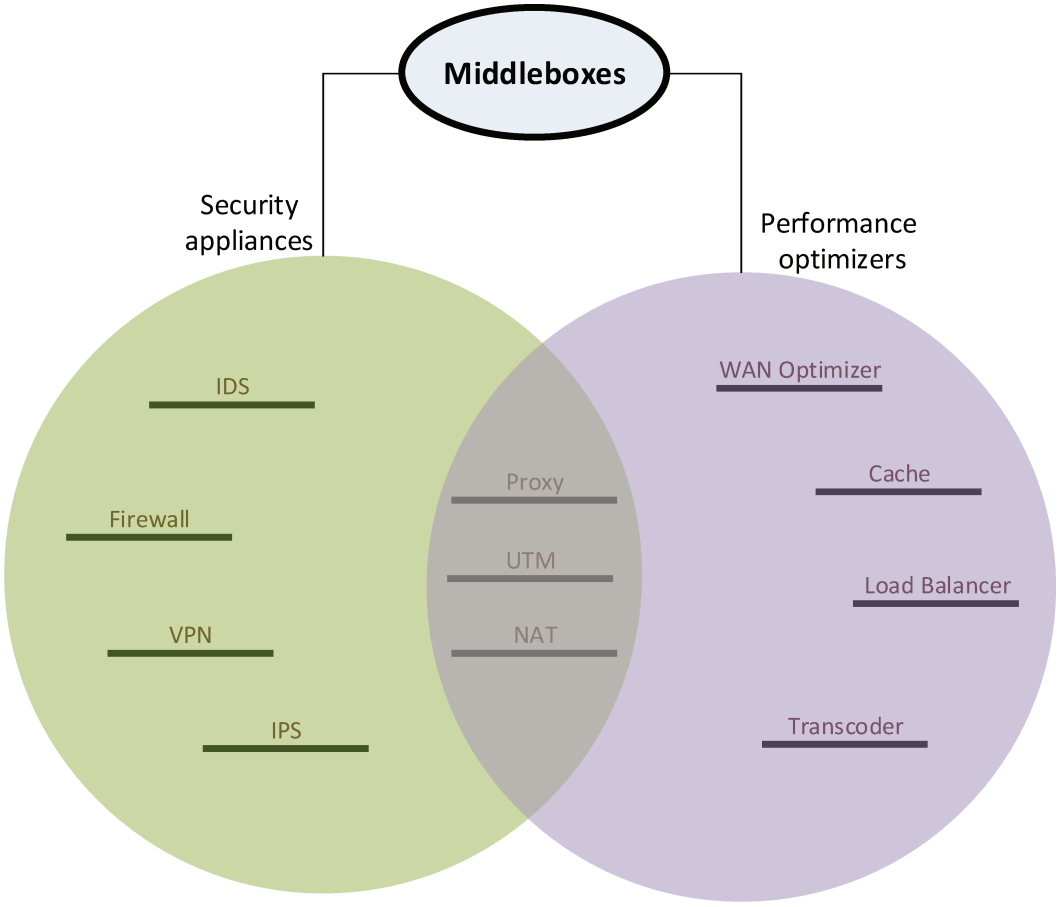

Figure 2.4: *Middleboxes*

The term "middlebox" may mistakenly imply a physical existence of a separate device whereas, it can be a virtual appliance that is running on a commodity hardware along with another functionalities or even as a software solution [38]. The authors in [15] state that security appliances can be distinguished by their types (See Figure 2.5). *Active* devices can filter and modify the traffic. Common examples of active devices are anti-virus, content-filtering device and firewalls. *Passive* devices basically monitor the network, make reports and alerts whenever they detect a suspicious activity; like IDS. Another type of devices are *Preventive* ones such as IPS or vulnerability assessment tools that unveil threats before incident occurs. Finally, there are *Unified Threat Management(UTM)* devices that integrate multiple security features into one box.

Knowing the place of firewall among all middleboxes allows higher accuracy in giving a definition of the firewall. As a result, firewalls are software or hardware components, that separate a network into different security levels by enforcing rules. These rules specify access level as well as limitations of each network entity or program. According to NIST [83], there are ten different firewall technologies for production networks and two types for individual hosts and home networks.

- **Packet Filtering.** Most basic feature on any firewall that specifies ACCEPT or DROP of a packet, based on information in the header.
- **Stateful Inspection.** Keep track of connection states in a table called *state table* and inspect packets by matching their state with connection state.
- **Application Firewalls.** Stateful protocol analysis, also known as *deep packet inspection*. Allow or deny based on application behavior on the network.

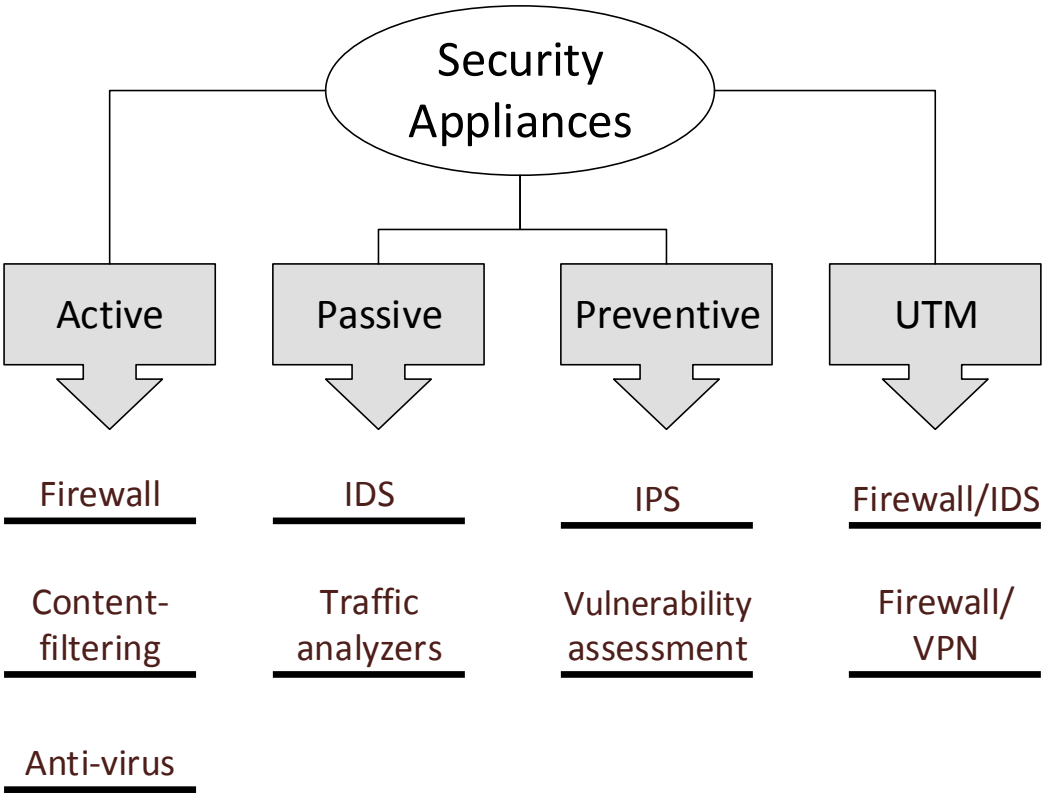

Figure 2.5: *Different Security Appliances [15]*

- **Application-Proxy Gateways.** Another deep inspection method that act as a middleman between two hosts and prevents a direct connection.
- **Dedicated Proxy Servers.** Placed behind firewall because they have very limited firewalling features and act as an intermediary between firewall and internal network.
- **Virtual Private Networking.** Using additional protocols in order to encrypt and decrypt traffic between two gateways or a host and a gateway.
- **Network Access Control.** Granting access to a client based on its credential and health check result.
- **Unified Threat Management.** Combination of multiple features like firewalling and intrusion detection into one system.
- **Web Application Firewalls.** A special type of application firewall for protecting web servers.
- **Firewalls for Virtual Infrastructure.** Software firewalls that can monitor virtualized network traffic.

While firewalls usually protect network to some extent, still there are some attacks that can pass through firewall and reach the internal network. In order to protect the hosts in the network, there are firewalls designed to deploy particularly on host machines instead of network. Firewalls for individual hosts and home networks are as following [83]:

- **Host-Based Firewalls and Personal Firewalls.** Software firewalls that are installed on OS for servers and personal computers like Windows firewall. They

can have other capabilities such as logging, intrusion prevention and application based firewalling.

- **Personal Firewall Appliances.** Small hardware firewalls that are used in home or small offices. They have more advanced features than host-based firewalls and add another layer of protection.

In a cloud environment there are high capacity storage devices that store customers' information, plenty of high performance servers that handle different tasks and virtual machines that are running on top of servers. Virtual network inside servers and sophisticated physical network that connects data center to the world are assets that should be protected very carefully. Firewalls are always deployed on the front line of the network in order to safeguard the internal network. In a cloud, network can be extremely complicated due to the huge number of servers and also virtual machine which create a combination of virtual and physical traffic. Therefore, firewalling only on one layer, cannot properly protect all assets. Moreover, virtual traffic is not visible to physical firewall. Co-tenancy of different customers on the same sever is another matter of contention because they may be owned by contender companies. Generally, different VMs in cloud are not considered as trusted to each other and there is a need for firewalling via VMs. In addition, security demand vary between different customers and in some cases, customized security services is demanded depend on the importance of their VMs. Thus, firewalling has to be applied in different layers from outside of data center to inside by physical firewalls with sophisticated features and within the virtual instances inside servers by virtual firewalls (or firewall that comes with hypervisor as a component). In our work, we considered both physical and virtual firewalls since they share same concepts in terms of access control, appli-

cation filtering, stateful inspection and other technologies (their difference is mainly in implementation).

Firewalling in a pure SDN network is shifted to the application layer (See Figure 2.3). SDN application[3] firewalls are similar to virtual firewalls, from implementation point of view, and are closer to physical firewalls, from the view scope and functionality perspective. In general, SDN firewalls can rule and protect the portion of the network that is controlled by SDN controller and they follow the main concepts of firewalling as traditional networks. However, it is very unlikely that we see a pure SDN data center in future; instead, it is more probable to have a hybrid architecture which is a combination of traditional network and SDN. Hence, virtual, physical and SDN firewalls will collaborate and protect the entire architecture. Our proposed framework in this thesis, focuses on SDN firewall policy migration when a VM migration happens.

[3]Here "application" refers to the SDN firewall implementation as an application and should not be confused with firewalls that control applications

Chapter 3

Towards Migrating Security Policies of Virtual Machines in Software-Defined Networks

Virtual machine migration is an essential capability that supports cloud service elasticity. However, there is always a big concern on what happens to the security policy associated with the migrated machine. Recently, Software-Defined Networking (SDN) has gained momentum in both research and industry. It has shown great potential to be used in cloud data centers, particularly for inter-domains migration of virtual machines. In a distributed settings, where more than one physical SDN controller is used, particularly in different network domains, coordinating security policies during migration is an important issue. In this chapter, we propose a novel framework, to be deployed in an SDN environment, that coordinates the mobility of the associated security policy along with the migrated virtual machine. We implemented our framework into a prototype application, called MigApp, that runs on top of SDN controllers. Our application interacts with the virtual machine monitor and other instance of MigApp through messaging system to achieve security migration. In order to evaluate our framework, we integrate our application with the Floodlight controller and use it with a simulation environment.

3.1 Overview

Virtualization technology has become a commonplace in cloud computing data centers. Specifically, Virtual Machine (VM) technology has recently emerged as an essential building block for such an environment as it allows multiple tenants to share the same infrastructure. The capability of VM migration brings multiple benefits such as high performance, improved manageability, and fault tolerance. While this mobility represents a valuable asset to the cloud computing model, it may introduce critical security flaws or violate the tenants requirements, if the security policy does not follow the VM to the destination after migration. Manual reconfiguration of security policies is not an acceptable solution as it is error prone and is inadequate for live VM migration. To the best of our knowledge, existing works on VM migration have not addressed security policy migration in SDN.

In this thesis, we address security policies mobility with VMs in IaaS cloud computing, and demonstrate how to solve security policy migration in SDN-based cloud. Software-Defined Networking (SDN) is an emerging networking paradigm that aims at separating the control and data planes while offering logically centralized network's intelligence and state [76] that are connected to the forwarding elements at the data plane. This architecture allows for a dynamic and flexible provisioning over data plane and facilitates network management in a cost effective way. Particularly, the network administrator can program and manage multiple network devices based on business requirements without the need to deal with each network device separately. At the management plane, the administrator can specify various policies (e.g. quality of service, security, etc.) that are then used by a set of applications to program the

controller through a northbound API (e.g. REST API [81]). The controller, programs the forwarding devices at the data plane through a southbound API. The most popular protocol that offers an implementation of such an API is OpenFlow [78], maintained by the Open Networking Foundation (ONF). SDN provides new ways to solve age-old problems in networking while simultaneously enabling the introduction of sophisticated capabilities. For instance, GoGrid [23] is an example of IaaS providers that have adopted an SDN approach to architecture the cloud. The configuration and control is put into customers' hands so that they can design their own cloud platform with virtualized services such as firewalls and load balancers, managed using the management console or a public REST API. In order to deal with security groups, GoGrid implements security groups as global objects that are automatically synched across data centers [23]. A recent IDC study [59] projected that the SDN market will increase from $360 million in 2013 to $3.7 billion in 2016. There are several organizations worldwide including Google [93], NDDI [96], and GENI [22] running and testing OpenFlow networks. A significant number of vendors such as HP [57], Cisco [41], and IBM [58] are contributing by manufacturing SDN-enabling products.

Research initiatives [89, 101, 56, 18] supported by industry acknowledge the challenge and the importance of security context migration as a part of cloud elasticity mechanism. Many research initiatives have proposed to leverage the SDN paradigm to benefit cloud computing environments. Particularly, [70, 32] propose SDN strategy to enable live and offline migration of VM within and across multiple data centers. However, the reviewed solutions either circumvent the problem (e.g. using traffic tunneling techniques) or do not fully address it, if not at all. In this thesis, we design and implement a framework for migrating security policies along with the VMs

within the same or between data centers based on SDN. Our solution, as opposed to vendor-specific ones, is meant to be open source, secure and interoperable with any SDN controller that provides a REST API and a virtual security appliance such as a firewall. To coordinate the migration, we propose to use a distributed messaging system, namely RabbitMQ. The latter is based on Advanced Message Queuing Protocol (AMQP), a highly scalable publish and subscribe message protocol that is increasingly used in cloud architectures (e.g. VMware vCenter Orchestrator [1]). Thus, our main contributions are as follows:

- Design of a novel framework, namely MigApp, that enables the mobility of security policies during VMs migration in order to support cloud computing elasticity with an SDN-based architecture. The framework enables an east-west communication between management application within and across data centers.

- Implementation of the proposed framework in a prototype application integrated on the top of an existing OpenFlow controller, namely Floodlight.

- Test and validation of the security policy migration and evaluation of its performance using our framework deployed in a simulation and programming environment.

This chapter is organized as follows: Section 3.2 briefly describes the most relevant research initiatives under the discussed topic. Section 3.3 elaborates on the preliminaries of our work notably by describing main SDN concepts and Floodlight controller as well as the foundation of messaging systems. Section 3.4 is dedicated to detail our

[1] http://www.vmware.com/support/orchestrator/doc/amqp-plugin-102-release-notes.html

proposed approach. Therein, we present the design, implementation, and deployment of our prototype framework. In order to demonstrate the viability of our framework, Section 3.5 describes the experimental results that we obtained by deploying our framework using a simulation environment.

3.2 Related Work

In the context of VM migration in traditional data centers, Tavakoli et al. [89] proposed a mechanism for VM security context migration based on an extension made to the virtual machine monitor. However, the approach only deals with hypervisor-based firewalls and considers a non-SDN context. Recently, a number of research initiatives [32, 62, 70] have investigated VM migration based on SDN and OpenFlow. Boughzala et al. [32] proposed an OpenFlow-based infrastructure-as-a-services (IaaS) middleware solution to interconnect different data centers. This enables inter-domains VM migration, howbeit, only entries in the flow table of switches are transferred to re-establish the existing connectivity of the migrating VMs. Similar to us, they used mininet but for a different purpose which is to evaluate network setup and migration delay using their solution. Keller et al. [62] propose LIME, an SDN-based approach for live migration of a network consisting of a group of related VMs along with the data-plane state. The prototype was implemented on top of a Floodlight controller. LIME clones one or more switches, and then separately migrates each VM to the new location. Migrating openflow entries from the source switches and statically inserting them into the destination switches does not completely solve the problem for two reasons. First, static insertion of all flow entries contradicts an important SDN concept, which is to dynamically generate and insert the flow entries by the controller

at the reception of the first packet of a flow. Second, for the newly incoming flows, which are different from the static ones, the absence of the adequate policies (specified at the management plane) may lead the controller's decisions to violate the security of these new flows. In contrast to LIME, which focus was solely on network forwarding state, we propose to migrate the security policy as specified by the VM administrator (or tenant). CrossRoads [70] is a framework for seamless live and offline VM mobility across multiple data centers based on SDN. It has been implemented as a module on top of NOX controller. In order to avoid firewall reconfiguration, CrossRoads uses a mechanism for tunneling the packets at layer 2. Ghorbani et al. [52] tackles a different problem than us as it considers the migration of a network of VMs while preserving some correctness conditions such as preserving bandwidth guarantees and loop-freedom. Their approach only entails the modification of the forwarding state using a set of generated OpenFlow instructions sent to the switches at the destination location.

Gember et al. [51] propose a Software-Defined MiddleBox Networking (SDMBN) framework, which applies SDN strategy to networks of middleboxes and thus allowing a fine-grained programmatic control over their states. This solution requires the modification of middleboxes to expose programmatic interface (southbound API similar to OpenFlow), a middlebox controller with a northbound API to allow programming middleboxes, and scenario-specific control applications that orchestrate middlebox and network changes in tandem. Control applications, such as live migration and elastic scaling, are enabled by leveraging the proposed northbound API for MBs and OpenFlow. Koorevaar [64] proposes an approach for leveraging the SDN architecture for automatic security policy enforcement using (Elastic Enforcement Layer)EEL-

tags. These tags are added by the hypervisor into the flow of packets emitted by the VM. They allow identifying the security policies, which actually refer to a chain of middleboxes to be traversed by the secured traffic. However, the work assumes a trustful hypervisor and tenants' traffic isolation, though isolation mechanism was left out of scope. Similarly, Fayazbakhsh et al. [48] highlight the importance of having flow tracking capability in order to ensure consistent policy enforcement for security middleboxes. They propose a SDN-based architecture called FlowTags where tags are embedded inside outgoing packets headers for policy enforcement. Works proposing insertion of tags within the packet for policy enforcement suffer from security problems as they are prone to malicious packets injection and does not consider attacks against packets integrity or packet authenticity. Although tenants' traffic isolation is proposed, the isolation mechanism was left out of scope.

Williams et al. [100] propose VirtualWire, a system that exposes to the cloud tenant an interface to create logical network topologies in which VMs can be live migrated within or between clouds. [87] propose an implementation of an elastic IP and security group service, similar to the Amazon EC2 services, using the OpenFlow protocol and the OpenNebula system. The implementation relies on the integration of an OpenFlow controller (NOX) with the EC2 server.

With respect to the related works, only few works consider security policy migration in the cloud context that makes room for us to discuss more on this matter.

3.3 Preliminaries

In this section, we briefly describe the background by presenting the main SDN concept, the Floodlight controller and its architecture, and distributed messaging systems, particularly RabbitMQ.

3.3.1 SDN and Floodlight Controllers

In an SDN architecture, control plane of network devices are brought "outside the boxes" and are logically centralized to facilitate configuration and management. Programming network devices is achieved by exchanging control messages between the controller and the switches. OpenFlow is an open standard network protocol that implements such control messages and provides a packet-forwarding abstraction to the controller for managing the behavior of network devices [71]. The controller instructs the data plane on how to act on the incoming traffic flows. On the other side, switches inquire the controller's decision about a new incoming flow, inform about the occurrence of network events, or send statistics about the received traffic. When switches on a given network path receive the controller's decision as a set of flow rules, they update their flow tables with these entries in order to act on subsequent packets of the flow. In such a setting, security policies specified (programmed) at the management plane are applied by the controller at the switch level.

Floodlight [25] is a Java-based open source SDN controller initially developed by Big Switch Networks and currently supported by a community of developers. A number of module applications can be developed and compiled within Floodlight in order to perform additional functionalities. These modules make use of the Flood-

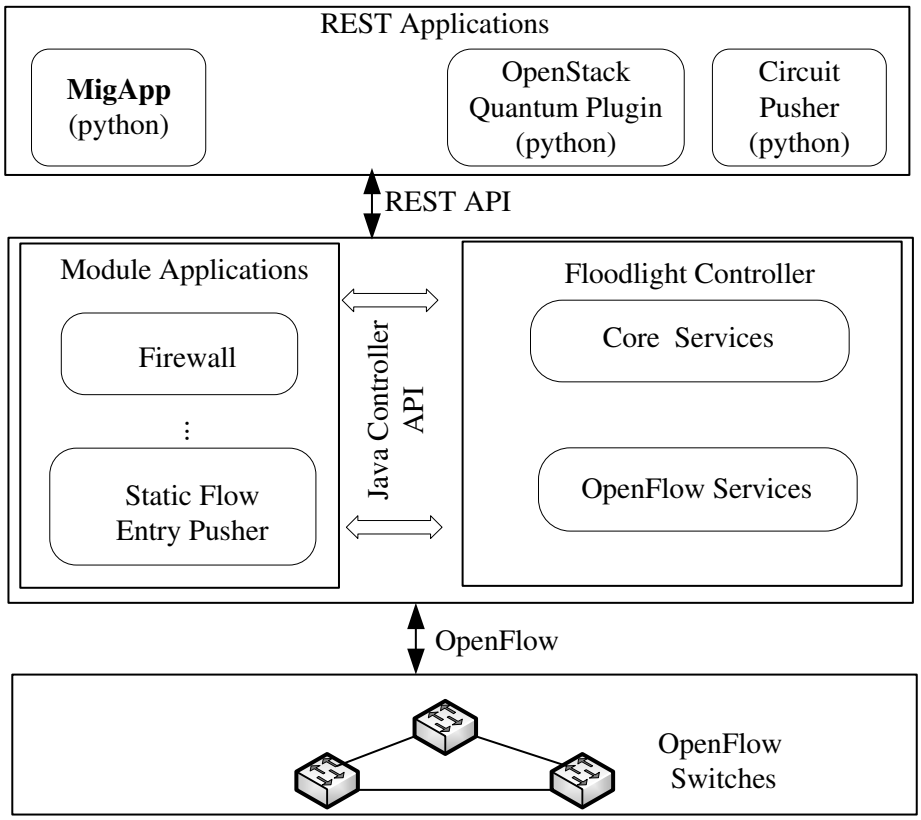

Figure 3.1: *Architecture of SDN and Floodlight Controller with our MigApp*

light API to communicate with the controller. An example of a module applications is the "Floodlight Firewall" [97]. Floodlight exposes its own REST API as well as the one of its modules for northbound communication with REST applications. REST stands for REpresentational State Transfer and it an architectural style for designing networked applications for distributed systems that runs over web. REST offers a general interface, scalable component interactions, and an independent components deployment while reducing latency, enforcing security and encapsulating legacy systems. It allows developing applications in any programming language (Java, python, etc.) that communicate with the controller and its modules in a standard way. Figure 3.1 illustrates the SDN architecture composed of separated data and control planes as well as the management plane. It also shows the relationship among the Floodlight controller, the applications modules compiled with Floodlight, and the REST applications. Once the controller is running, the modules running within it expose their REST API via a specific REST port to the management plane. Thus, any management application can send HTTP REST commands through the listening REST port in order to retrieve information and invoke services from the controller or the supervised network. In this context, we built our MigApp as a REST application that uses REST API to retrieve and update information stored by the firewall module and to retrieve the network state. The firewall keeps the specified Access Control List (ACL) in a dedicated Floodlight storage that is consulted by the controller to take decisions on newly incoming flows. The rules conditions in the Floodlight firewall module matches a set of fields to their values, including switch id, the receiving switch port, the source and destination MAC and IP, the protocol with the source

and destination ports and a priority number. The matched ACL rule with the highest priority determines the action (allow/deny) to be taken on the flow.

3.3.2 Distributed Messaging System

In our solution, there is a need for a mechanism to coordinate the communication between the involved parties so that control messages and security policies can be exchanged. Constraints such as security, reliability, and performance have to be considered while selecting the right mechanism.

To this end, we opt for a distributed messaging system based on the Advanced Message Queuing Protocol (AMQP). Interesting features of the AMQP protocol are that all resources used for storing messages are dynamically created and destroyed by clients as they need them, there is no need for static pre-configuration, and there is a number of free client libraries available in various programming languages. We chose RabbitMQ [79], an open source broker implementation of the AMQP protocol. RabbitMQ is installed and used by clients on existing cloud services such as Amazon EC2 [28]. RabbitMQ is a message broker that basically accepts messages from a client program, called a producer, and store the messages in a queue that works as a mailbox inside the server. Other client programs, called consumers, wait to receive messages from the queue. The queue can store any number of messages and it is substantially an infinite buffer, which can run up to almost 30000 messages per second depending on the properties of the message. Many producers can send messages to one queue. On the other hand, many consumers can receive messages from one queue. A nice characteristic of such system is that the producer, consumer, and the broker do not

have to necessarily reside on the same machine. In fact, as distributed applications, they are scattered around different machines.

In RabbitMQ, the producer do not send messages directly to queues but to a specific entity called exchange, which receives and pushes them to queues. Thus, one needs to first create an exchange, then creates the queues, and finally binds the exchange to the queue. A binding is a relationship between an exchange and a queue which means that the queue is interested in messages from that particular exchange. The exchange must know what to do with a received message: either to append it to a specific or multiple queues, or even to discard it. The fate of a message is determined according to the exchange type, which can be direct, topic, headers or fanout. Moreover, among various messaging patterns, the Publish-and-Subscribe is one of the most used pattern in which messages can be received by multiple consumers with a fanout exchange that broadcasts produced messages to all subscribers. On the other hand, in some cases, the consumers are supposed to selectively receive some messages. It means that one message should be delivered to a specific consumer, whereas another message is destined for another consumer, and so on. In this scenario, the direct exchange will be used in addition to an extra routing key parameter that determines which consumer will receive which message. In our case, we use this latter exchange type so that any party only receives the messages intended to it.

With respect to server's deployment, a RabbitMQ broker can be deployed in a centralized settings where multiple clients connect to the same server. However, it is also possible to use a distributed architecture, where multiple brokers are clustered and federated in order to ensure scalability and reliability of the messaging service

and to interconnect multiple administrative domains. Particularly, a shovel may be used to link multiple RabbitMQ brokers across the Internet and provide more control than federation. The latter deployment enables dealing with inter data center security rules migration. With respect to security, RabbitMQ supports encrypted SSL connection between the server and the client and has pluggable support for various authentication mechanisms. Thus, these specific features enable to strengthen the security of our framework by only allowing authorized MigApps and hypervisors to produce and consume messages and protecting the underlying communications. In the next section we discuss our approach in more details.

3.4 Approach

In this section, we describe our framework that enables security policies to accompany the corresponding virtual machines during their migration. We assume cloud data centers deployed according to the SDN strategy. In such a setting, an SDN controller (or multiple controllers such as in Onix [65] and HyperFlow [92]) is in charge of controlling and configuring a set of OpenFlow-enabled switches. In the following, we detail the design, implementation, and deployment of our framework.

3.4.1 MigApp Design

In order to address the need for security rules migration, we designed our solution as a distributed REST application that communicates with other MigApp instances and the hypervisor through a distributed messaging system, namely RabbitMQ. MigApp uses the REST API of Floodlight and its modules, particularly the firewall module,

in order to retrieve the matching ACL rules at the source of the migration or to update them at the destination. In our migration scenario, MigApp at the source of migration receives a request from hypervisor and starts a communication with the peer application that resides at the destination side. The source MigApp, identifies the rules whom correspond to the migrating VM, from the source firewall. Then, the two MigApps exchange the rules while coordinate the migration of VM through communication with hypervisors. This supervision ensures that the VM does not start running at the destination before the rules migration. At the end, the source MigApp deletes unnecessary rules and sends a message to the hypervisor in order to let the VM running at the destination place. As far as the communication through messaging is concerned, we designed various types of structured messages that contain the information needed for successfully coordinating the firewall rules migration. Each type of message is associated with a numerical identifier that identifies the message purpose. A message is structured as follows:

```
{ MsgType: <number>,
   Sender:<Sender_IP>,
    MigVM:<VM_IP>,
      Src:<MigApp_src_IP>,
      Dst:<MigApp_dst_IP>,
     Data:<Policies>
}
```

such that "MsgType" is the type of message provided in numerical format, "Sender" is the sender of the message (could be the hypervisor or any MigApp instance), "MigVM" is the identification of the migrating VM (usually the IP), "Src" is the IP

of the MigApp source, "Dst" is the IP of the MigApp destination, and "Data " is either empty or contains the migrated rules. In the following, we describe the most important messages:

- *Migration_Request* (type = 100): It is send by the hypervisor to request security policy migration for a particular VM. The message is destined to the MigApp related to the controller supervising the identified switch to which the migrating VM is attached at the source location.

- *Migration_Acknowledge* (type = 101): It is sent from MigApp at source to the hypervisor to notify about the successful completion of firewall rules transfer. In case of failure, message type 109 is sent instead.

- *Firewall_Rules* (type = 200): This message includes the firewall rules matching with the migrating VM to be sent to the destination MigApp from the source MigApp.

- *Firewall_Acknowledge* (type = 201): It is sent from MigApp at the destination to the one at source indicating that the firewall rules have been received. At this time MigApp at destination can apply the received rules. In case of failure, message type 209 is sent instead.

- *Detached* (type = 300): It is used by the hypervisor to inform the MigApp at source that the execution of the migrating VM has been successfully suspended at the source location. This will trigger the deletion of the ACL at source that will not be used any more.

- *Detached_Acknowledge* (type= 301): It is sent from source MigApp to the hypervisor at source location to acknowledge the successful deletion of unused firewall rules. VM migration triggers upon receiving this message. In case of failure, message type 309 is sent instead.

- *Attach* (type = 400): This message is sent from destination MigApp to the hypervisor at the destination location to inform that firewall rules of the migrating VM have been safely handled, and that the VM can be securely attached to the destination switch.

As a natural choice of a data structure to transfer the aforementioned messages, we selected the JavaScript Object Notation (JSON) [9]. The latter is a text-based language-independent open standard used to transmit structured data over network connection. JSON can be built using two structures: unordered set of name/value pairs (objects) or an ordered list of values (arrays), separated by commas.

As mentioned earlier, we defined at the RabbitMQ broker side a queue per controller and a queue per hypervisor such that each entity may publish in any queue but may only receive from its own queue the messages that were specifically intended to it.

3.4.2 Implementation

We implemented MigApp using Python. The main program implements the core message processing function, which is responsible for identifying the received message and performing a set of proper actions, based on the message type. The algorithm corresponding to ProcessMessages is listed in Algorithm 1. It makes use of various functions dedicated for specific purposes. MigApp may receive messages from the

hypervisor through a dedicated exchange binding that triggers policy rules migration. It may also receive from other MigApp instances messages to coordinate the transfer of firewall rules. Thus, function *R*eceiveMsg is an implementation of the RabbitMQ client that allows receiving messages from a specific exchange binding. A dual function, namely *S*endMsg, takes a message and the destination exchange binding as parameters and then sends the message to its destination through the exchange binding.

Algorithm 1 Message Processing

```
 1: procedure ProcessMessages
 2:     msg ← ReceiveMsg()                          ▷ message reception
 3:     type ← GetType(msg)                         ▷ message type
 4:     src ← GetSrc(msg)                           ▷ source MigApp
 5:     dst ← GetDst(msg)                           ▷ destination MigApp
 6:     VM_IP ← GetVM(msg)                          ▷ migrating VM IP
 7:     if type == 100 then
 8:         migrules ← MatchRules(VM_IP)
 9:         req ← BuildMsg(200,VM_IP,src,dst,migrules)
10:         SendMsg(req,dst)
11:     else if type == 200 then
12:         rcvdrules ← GetData(msg)                ▷ extract received rules
13:         WriteRules(rcvdrules)
14:         req ← BuildMsg(201,VM_IP,src,dst,'')
          SendMsg(req,src)                          ▷ send migack to src
15:     else if type == 201 then
16:         req ← BuildMsg(101,VM_IP,src,dst,'')
17:         SendMsg(req, hyper)
18:     else if type == 300 then
19:         DelRules(VM_IP)
20:         req ← BuildMsg(301,VM_IP,src,dst,'')
21:         SendMsg(req, hyper)
22:     end if
23: end procedure
```

We also implemented specific functions in order to interact with the firewall module via the REST API. The first function, namely *M*atchRules, takes as parameter the IP of the migrating VM and reads the rules from the Floodlight storage at the source and returns a set of matched rules to be migrated to the destination. In order to find the actual migrating rules, we match the host IP (or MAC) with source and destination (IP or MAC) fields in the rules. For the case of an exact match, we migrate the rule as it is. For instance, the following rule will migrate:

$swid = 1, src = \text{VM\_IP}, dst = \text{otherVM} \rightarrow \text{allow}$, where *VM_IP* is the IP value of the migrating VM and *otherVM* is the IP of another VM.

With respect to rules with wildcards (or a subnet) used in one of the IP fields, if the VM IP is included within the mentioned subnet, we only migrate an instance of the rule by replacing the wildcard (or the intersecting subnet) with the VM IP value of the migrating VM. For instance, the following rule such that $VM\_IP \in Sub1$:

$swid = 1, src = \text{Sub1}, dst = \text{otherVM} \rightarrow \text{allow}$, will be transformed into $swid = 1, src = \text{VM\_IP}, dst = \text{otherVM} \rightarrow \text{allow}$ and then migrated. Finally, a rule with a wildcards or a subnet at the source (or destination) field and the exact IP of the migrating VM at the destination (or source) field will be migrated as it is.

The second function, namely *W*riteRules, takes the incoming rules and update the Floodlight storage at the destination. In order to apply the received firewall rules at the destination Floodlight storage, we need to precess them before actually writing them. Once the rules are received, we keep their relative order as defined by their priority levels, but modify these priorities so that we avoid conflicting with existent rules. We refer the reader to a broad range of research works on the detection and resolution of firewall rules conflicts [27, 26]. We also modify in the rules the old values

Algorithm 2 Migrate Host

```
1: procedure MigrateHost(VM_IP,switch)
2:     set(TimeOut)                                   ▷ set reception timeout
3:     type ← 100
4:     req ← BuildMsg(type,VM_IP,src,dst,'')          ▷ build message
5:     SendMsg(req,src)
6:     msg ← ReceiveMsg()
7:     while msg ≠ Null or ¬ is(TimeOut) do
8:         if msg ≠ Null then
9:             type ← GetType(msg)                    ▷ message's type
10:            if type = 101 then
11:                type ← 300
12:                msg ← BuildMsg(type,VM_IP,src,dst,'')
13:                DetachHost(VM_IP)
14:                SendMsg(msg, src)
15:            else if type = 301 then
16:                AttachHost(VM_IP,switch)
17:                Break
18:            end if
19:            msg ← Null
20:        end if
21:        msg ← Receive
22:    end while
23: end procedure
```

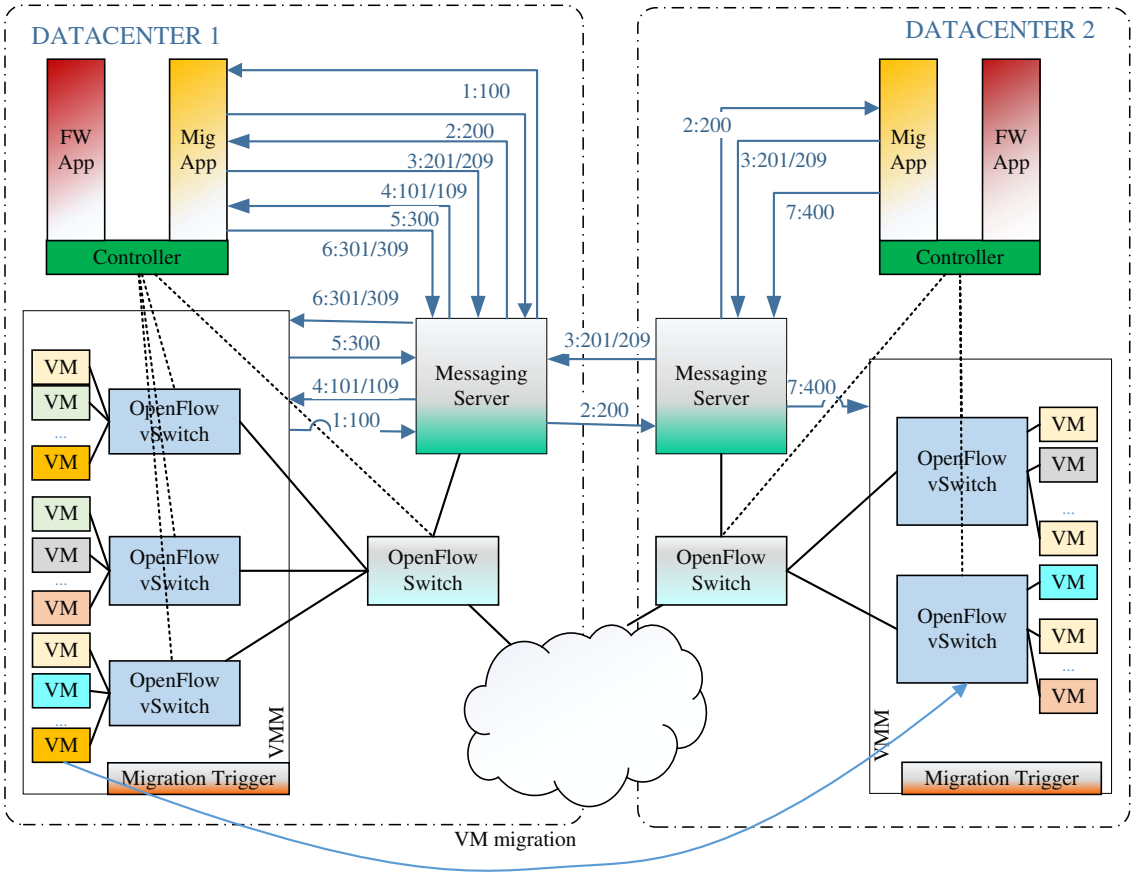

Figure 3.2: *Deployment of the Proposed Solution*

of switchid and src-inport with the new values that correspond to the configuration at the destination. Finally, *D*elRules is used to delete the rules from the firewall if the rules do not concern any host under the supervision of the source controller. In order to get such an information, MigApp interacts with the Floodlight device manager module through the REST API. Furthermore, a Python program named BuildMsg is responsible of preparing the content of the messages in JSON format for being sent to the receiver exchange binding. Finally, we implemented some utility functions such as *G*etType, *G*etSrc, *G*etDst, etc. in order to read specific parts of the messages and returns the needed information in the needed format.

Algorithm 2 lists a pseudocode of the function that should be implemented by the hypervisor (or the VMM) in order to interact with our MigApp. Note that *A*ttachHost and *D*etachHost functions refer to the function of attaching the migrating VM to and detaching it from the switch port, respectively.

3.4.3 Deployment

We proposed a novel firewall rule migration framework, namely MigApp, that we believe can be used with any SDN controller that offers a firewall module and a REST API, with minor changes. Our framework uses REST API to communicate with the controller security applications and transfers the rules and policies that match with the migrated VM into the destination controller. This controller can be local or remote depending on the migration process. The entire process is triggered by the hypervisor or any higher privileged software or system that is responsible for VM migration. Hence, our solution requires to add into it a module to communicate using the RabbitMQ server. Figure 3.2 illustrates the deployment of our solution.

It also illustrates the messages exchanged between different entities involved in the migration of the security rules. A message is denoted by a directed arrow labeled with the order of precedence of the message and the type of message, separated by a colon.

3.5 Experiments

In the following, we describe the performed experiments using MigApp and the obtained results.

3.5.1 Setup Environment

In order to test our framework, we used a custom Mininet [66] forked from the main source code by Big Switch Networks and deployed in an Ubuntu virtual machine. Mininet is a SDN emulator that we use for simulating the data plane. As it is shown in Figure 3.3, we created a topology in Mininet 1.0.0 that consists of a set of Open vSwitch [13] instances that are supporting OpenFlow 1.1.0 [77]. We omit the RabbitMQ servers as they were explained earlier. The used Mininet version has two added functions that enable attach and detach of a host; therefore mobility of a host is simulated by detaching from source switch and attaching it to the destination switch. To do so, we modified mininet code in order to play the role of hypervisors by implementing Algorithm 2 as a Python function named *M*igrateHost. Each simulated host is a process that is created in a different name space [66]. We set up the entire test environment on a PC with an Intel Core i7 3.4 GHz processor with 16 Gbytes of RAM. We also used Eclipse Juno Service Release 2 for running different Floodlight

controllers on different ports. For firewall testing, we used Nmap 6.25, a port scanner that shows status of each port from the view of outsider. We also used Netcat 1.10, a great tool for many network testing purposes like opening TCP or UDP ports. Finally, Wireshark 1.4.6.is a packet analyzer used for capturing packets and studying packets in different switch ports.

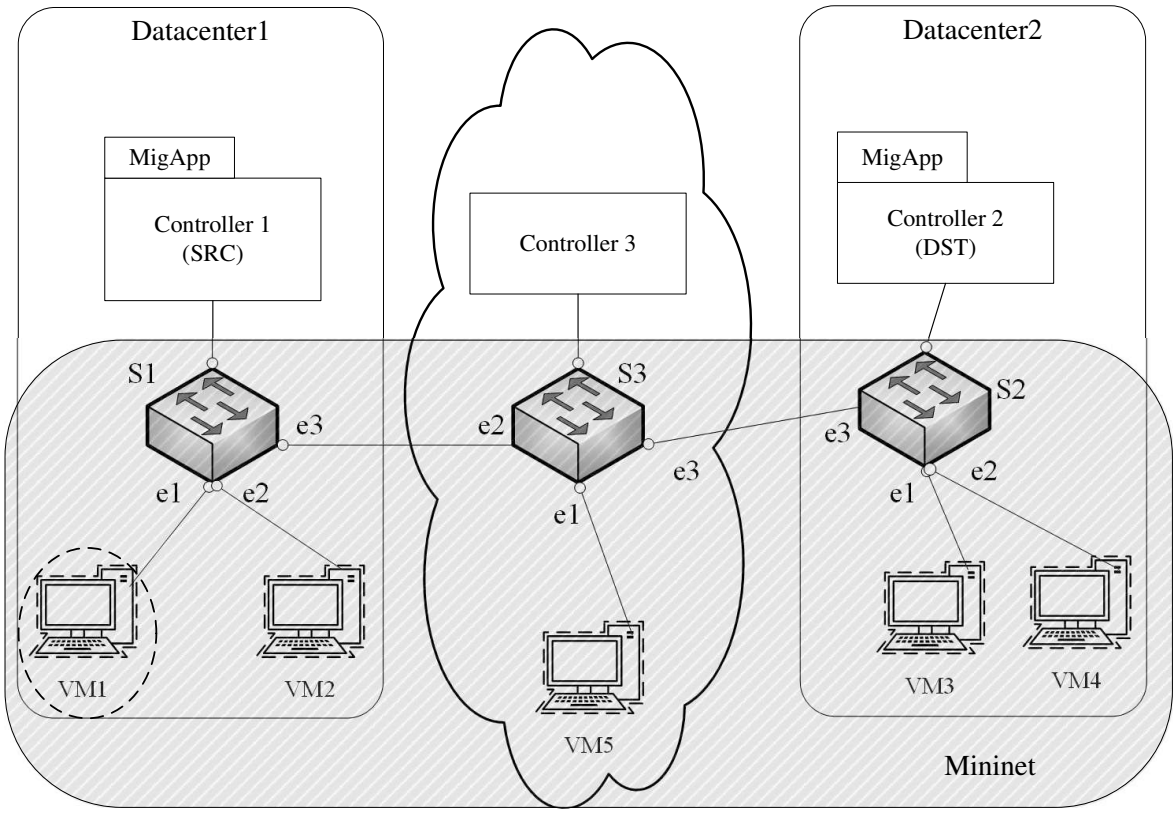

Figure 3.3: *Testing Environment (VM1 is the Migrating VM)*

3.5.2 Running Scenario

Our test-bed has three separate domains such that each of them is controlled by at least a single controller. We assume two data centers, each of which has a Floodlight

controller and a MigApp installed on the same server. In the simplest scenario, we considered a single VM migration (VM1) from data center 1 to data center 2. As it is shown in Figure 3.3, access to VM1 can happen from each of the three domains. VM2 is in the same data center as VM1 before migration; VM3 and VM4 are in the destination data center. Thus, after migration VM1 would be in the same location as VM3 and VM4. We consider VM5, a host that is neither in source nor in destination data centers that can be considered as a host connected to the Internet. All the VMs and switches are simulated by Mininet who has a migration module and switches are connected to the Floodlight controllers, which are running on different ports. Each controller has a separate MigApp connected to it through REST API. Both MigApps use a single messaging server on the same machine. Each MigApp has its own messaging queue within RabbitMQ server. Running scenario starts by a Python script that has the customized topology of Figure 3.3. Then, we set ACL rules on all firewalls using REST API and enable them. For testing firewalls, we leverage the capability offered by Mininet to run other programs to test services like a web service on VM1. We used Netcat to reach this service from other VMs. Wireshark has been used on port e1 of switch S1, where VM1 is attached, and on each port connecting the other VMs. This is for the purpose of capturing all packets that are sent from other VMs and received by VM1. This firewall testing is performed both before and after migration. Firewall rules and Wireshark usage are discussed in details in the next section. Migration is triggered from Mininet CLI by user intervention by passing IP of the migrating VM and the destination switch as parameters to our implemented CLI function *M*igrateHost.

3.5.3 Experimental Results

Testing Security Policy Migration

We validated the correctness of our migration application MigApp by testing the firewall rules before and after migration. To this end, we used Nmap port scanner to scanned VM1 ports from each of the other VMs before migration and then we saved the obtained results. We repeated the same test after migrating VM1 and compared both scans. We achieved completely identical results, implying that all VM1 ports that were open before migration for a specific VM, remained open after migration, same as filtered or closed ones. For example, VM2 was supposed to have a SSH access in addition to a HTTP access (which all VMs had) to VM1; Nmap results show that VM2 has SSH as well as HTTP access to VM1 after migration. This shows that MigApp correctly migrated the firewall rules. To have a stronger proof, we went one step further and performed a more detailed assessment using Netcat and Wireshark. We opened ten important ports namely FTP, SSH, Telnet, SMTP, DNS, HTTP, POP3, HTTPS, IMAP, HTTP-proxy on VM1 using Netcat and tried to connect and send arbitrary data to VM1 from each of other VMs. We did not have any control on data but we made sure that each VM sends data to all ten ports. In order to have a better understanding, we mention a subset of firewall rules and briefly explain the granted access to each VM in order to communicate with VM1.

All VMs should have access to port 80 (HTTP). VM2, VM3, VM4 and VM5 respectively should only have access to port 22 (SSH), port 443 (HTTPS), port 21 (FTP) and port 8080 (HTTP-proxy). All other ports on VM1 have to be filtered for other VMs; therefore, each VM can have HTTP access other than one port access

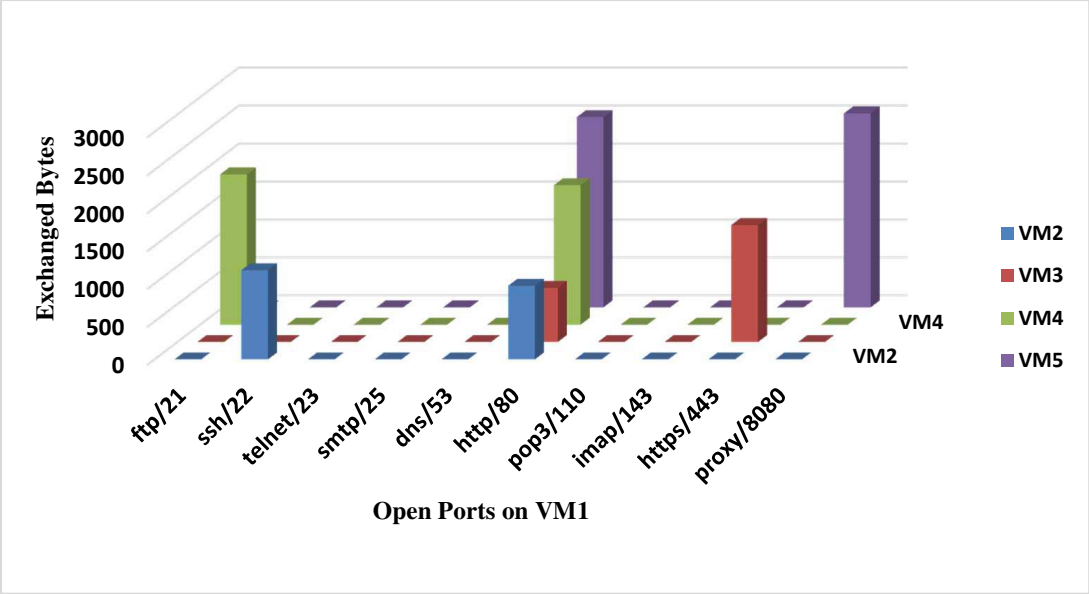

Figure 3.4: *Communications between the VMs Before Migration*

specific to that VM. Figure 3.4 and Figure 3.5 demonstrate all TCP communications with VM1 respectively before and after migration. The X axes in the charts indicate open ports on VM1 to which we sent random data from the other VMs and Y axes indicates the number of exchanged bytes between VM1 and the contacting VMs. Since we sent random data from each VM, the number of bytes received in VM1 differs for each of them but as it was expected, only those who were allowed reached to VM1. Comparing the charts before and after migration shows that in either cases, the same ports have received data from a specific VM. This results testify that all previous access grants and filtration of banned ports are preserved after migration.

Measuring and Evaluating MigApp Overhead

The major time-consuming part of the framework is transferring the policies from source to destination of migration. This constraint can be affected by network band-

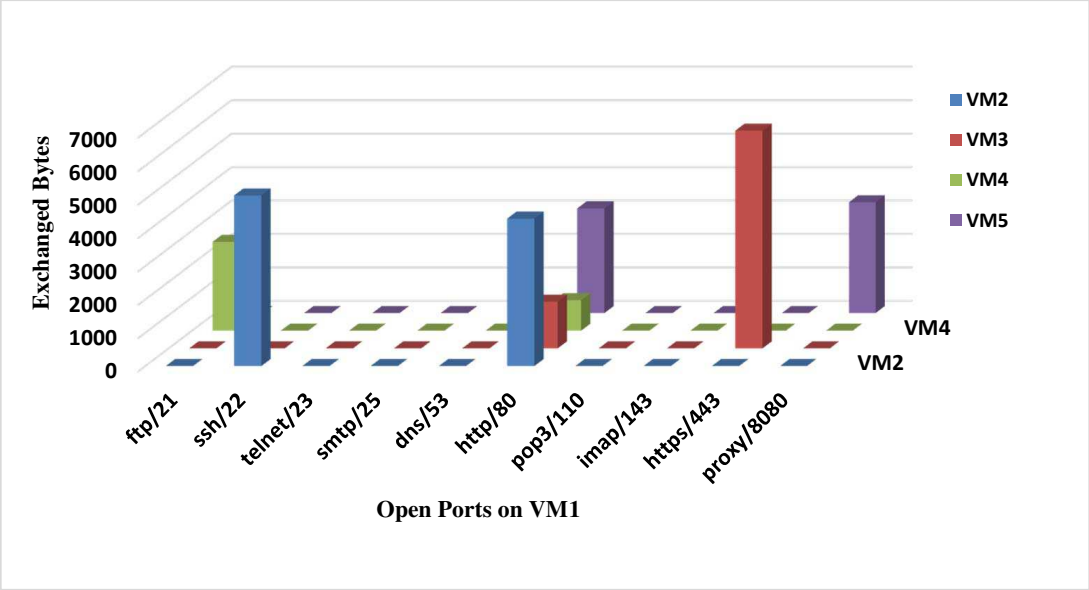

Figure 3.5: *Communications between the VMs After Migration*

width, messaging server response time and some other factors that are out of the scope of this dissertation. Reading the firewall rules and matching them with the migrating VM (i.e. Read/Match) are the most time-consuming functions at the source of migration, whereas, processing the received rules and applying them (i.e. Process/Write) are the functions that adds most of the overhead at the destination. Thus, we measured and evaluated the overhead of these two functions using our test-bed after increasing the number of transferred rules. Figure 3.6 reports on the performance results. The latter illustrates the execution time of the mentioned functions for an increased number of rules. As indicated in Figure 3.6, for 500 rules, Read/Match requires less than 0.14 seconds, whereas Process/Write needs less than 1.2 seconds, which brings the total overhead to less than 1.29 seconds. Finally, one can notice that the trend of the execution time grows smoothly.

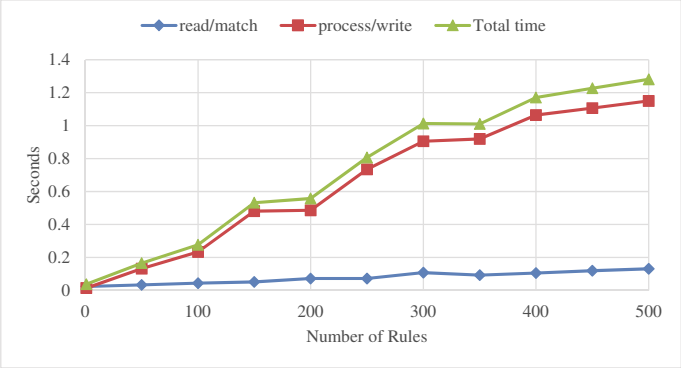

Figure 3.6: *Time for Execution as a Function of the Number of Rules*

3.6 Summary

In this chapter, we presented a novel and flexible framework for migrating firewall policies with the corresponding VMs in order to support the elasticity of cloud services in an SDN context. Our framework coordinates the migration of the policies based on a reliable open source distributed messaging system, namely RabbitMQ. The latter is based on AMQP, a highly scalable publish and subscribe message protocol that is increasingly used in cloud architectures (e.g. VMware vCenter Orchestrator [2]).

Our framework enables cooperation between SDN security applications that run on top of the controller and hypervisor responsible of the VM migration. This provisioning improves the security of the migrating VM within the same or between different data centers. This framework requires a minor modification of the hypervisor responsible for the VM migration so that it triggers policy migration through the RabbitMQ messaging system. We designed and implemented our framework as a distributed REST application called MIGAPP, on top of the Floodlight controller

[2]http://www.vmware.com/support/orchestrator/doc/amqp-plugin-102-release-notes.html

and we tested our prototype in a simulation and programming environment. We designed and performed experiments and showed that our approach is performing as expected since the same network services were accessible before and after migration. We also measured and evaluated the overhead introduced by our MigApp and concluded that the process introduces acceptable delays when the number of rules are increased. Future work can be developing a comprehensive application for various SDN controllers; however this is not achievable until the controller is not equipped with a security module or application. As an extension to this work, we can consider migration of QoS or even load-balancing rules with VM.

Chapter 4

Customized Test-bed for Testing Migration Security in Cloud

In Chapter 3, we presented a framework and a prototype application to migrate security policies in SDN context. In order to test the viability of our prototype application, we used Mininet which is a well-known simulator for SDN. Although there are many simulators for traditional networks and clouds, most of them are focused on performance and cost assessments. In this chapter, we strive to prepare a testing environment that supports VM migration and focuses on security assessments.

4.1 Overview

Cloud computing is widely deployed all over the globe and its popularity is growing due to the benefits that it offers to both service providers and users. As the rate of adaption to the cloud increases day by day, cloud security is growing more important. Multi-tenancy is one of the main points of concern in cloud. Migrations are essential for cloud elasticity and security of data center and VMs should be preserved during and after migrations. There are many other examples that highlights the importance of security researches in cloud.

In order to conduct a research, a test environment is a must for researchers. Benchmarking an application performance, testing the compatibility of a new protocol

or analyzing the security of a new feature, all are examples that need a test-bed for evaluation. On the other hand, testing security on real world cloud environments is not a good idea. First of all, a real cloud needs a huge amount of money and time to deploy and it may not be safe to conduct a security testing on a production network. Further, running multiple tests may need reconfiguration of the entire network that apparently takes more time in a real network. Thus, simulation environments are a good alternative for real employments, because they are cost effective, safe and flexible.

To model a real network behavior, there are two ways that are known as *Simulation* and *Emulation* and each one has pros and cons. A network simulator is usually a piece of software that models network entities and the interactions between them, by using mathematical formulas. Simulators are typically used in research for studying and predicting network behavior and performance analysis. Most of the simulators model network devices, links between them and generate network traffic within the same program. Discrete-event simulation that models system operations as a sequence of events in time, is widely used in network simulators. Another way of simulation is using Markov chain which is less precise but faster than discrete-event simulations. There are many commercial and open-source network simulators with various features. For instance OPNET [14] is a commercial simulator with GUI, NS2/NS3 [47][12] are open-source simulators that accepts scripts as input for network parameters and NetSim [11] is another example. [1].

A network emulators is a piece of software or hardware to test and study a network that imitates the behavior of a production network. Emulators normally do

[1]A list of popular network simulators can be found at: http://www.idsia.ch/ andrea/sim/simnet.html

	Net. Simulator	Net. Emulator	Actual Network
Modeling/deployment Time	very fast	fast	slow
Deployment Difficulty	very easy	moderately easy	hard
Cost	cheap	moderately expensive	expensive
Real Services	No	Yes	Yes

Table 4.1: *Comparison between Network Simulation,Emulation and Actual deployment*

not simulate endpoints such as computers; and therefore, computers or any type of traffic generator can be attached to emulated network. Normally, in emulation actual firmware is running on general purpose hardware. As a result, it is possible to run live applications and services on an emulated network which usually is not feasible in a simulation. Hardware-based network emulators are more expensive and more accurate than software-based ones and are commonly used by service providers and network equipment manufacturers. Dynamips [49] is a free emulator for routers and QEMU [29] is an open-source hypervisor that can be used as a machine emulator.

Although, both simulators and emulators are applied for testing network performance, they are used for different purposes based on the capabilities that each of them offers. For example, simulators are good for scalability and performance tests while emulators can be used to test network applications and real services. Nevertheless, both simulators and emulators are crucial in network research. See Table 4.1 for a comparison between using network simulators, network emulators and actual deployment.

Network and cloud simulation has been around for a while. However, most of the network simulators are not capable of cloud modeling. On the other hand, most of the existing cloud simulators focus on performance benchmarking, cost effectiveness evaluations and power consumption assessments. Hence, majority of them lack in

modeling security boxes such as firewall, IPS and security services like VPN. Furthermore, in some experiments a real running VM and actual services who imitate the behavior of a real network are necessary. At the time of writing this thesis, there is no free cloud simulator available, who mimics middleboxes and real services in simulations. Hence, we decided to prepare a distributed test-bed based on GNS3 that is mainly a network simulator. In order to use GNS3 for cloud, we introduce an architecture that models the deployment of standard data centers in a small scale but with real running services and security features. We also equip the test-bed with a set of free network and testing utilities that facilitate many experiments. In addition, we focus on VM migration in cloud and first design a migration framework and then improve it to a security preserving migration framework. In summary, contributions of this chapter are:

- Design and deployment of an architecture that introduces the way to use GNS3 for simulating multiple data center on multiple host machines.
- Design and implementation of a framework for migrating VMs on the test-bed by improving the features in VirtualBox.
- Design and implementation of a framework to migrate firewall rules in the case of migration.

The remainder of this chapter is organized as follows: Section 4.2 describes the related work, other projects that have been done with GNS3 and a comparison of well-known simulation solutions. In section 4.3 we elaborate on the preliminaries of this work and introduce main software programs that are used in test-bed preparation. In section 4.4 we describe the test-bed components and setup environment.

Section 4.5 explains the migration framework design and its implementation on VirtualBox. Section 4.6 and section 4.7 discusses some use cases and clarifies the limitations and shortages of the test-bed, respectively. Finally, in section 4.8 we summarize the chapter and talk about future work.

4.2 Related Work

Simulations have been widely used in computer world and there is a large number of open-source and commercial network simulators available. NS-2 [47] and OPNET [14] are well-known discreet-event simulators that are used both in academia and industry. First one is open-source and the latter is a commercial simulator. Both are capable of simulating stateless firewalls and neither of them can simulate IPS. OPNET is not for cloud simulation and NS-2 need an extension for that purpose.

Cisco Packet Tracer [5] and Boson NetSim [2] are proprietary network simulators that their main objective is exam preparation and to facilitate the training with Cisco devices. Packet Tracer does not support firewall and other middleboxes however, Boson Netsim is capable of simulating firewall, IPS and some other advanced devices.

Kliazovich et al. [63] presented Greencloud which is packet-level simulator, as an extension for NS-2. However, Greencloud focus is on energy consumption in cloud communications and can not be used for security experiments. Nez et al. [75] presented iCanCloud that is a cloud simulator, developed on OMNeT++ [95] platform. The main focus of iCanCloud is on predicting the trade-offs between cost and performance of a given set of applications executed in a specific hardware, and then provide information about costs to the users.

CloudSim [36] is a mature JAVA-based simulator that supports modeling and simulation of large scale cloud computing data centers. CloudSim adds a layer on top of GridSim [34] in order to offer the ability to simulate clouds; however, the latest version is re-implemented the kernel and it does not rely on GridSim anymore. CloudSim is a generalized simulation framework but it does not simulate middleboxes at the time of writing this thesis.

SupernaNET [16] and EstiNet Network Simulation Cloud [7] are proprietary cloud and network simulators. SupernaNET is built as a VMware appliance and its focus is on addressing interoperability, management and network security challenges in cloud. EstiNet Network Simulation Cloud runs on top of the Openstack system software and can be used to construct a private network simulation cloud located in the laboratory for research purpose, or an enterprise to execute the simulation tasks.

GNS3 [8] is an open-source simulator that is similar to Boson NetSim and its main use is for training and certification preparation. Although, GNS3 does not have limitations of Boson NetSim; for instance, GNS3 supports Virtualbox to run desktop and server operating systems and also is able to simulate devices from multiple vendors. The fact that its source-code is available is another advantage which makes it possible for researchers to add features on top of it. A summary of different simulation solution is depicted in Table 4.2.

*G*NS3 WorkBench [99] and *L*ive Raizo [94] are two ready-to-use testing environments that are prepared based on GNS3. The first one is a Linux VM with GNS3 that includes installed VPCS and a collection of exercises and labs. The goal of this appliance is preparing an easy-to-use environment for beginners who wants to discover GNS3 and its possibilities. The latter is a Live Linux Debian that simu-

	Usability					Availability		Security		
	U1: Network-Simulation	U2: Cloud-Simulation	U3: GUI	U4: Open-Source	U5: Easy-to-Learn	A1: Free	A2: Multi-Platform	S1: Firewall	S2: VPN	S3: IPS
Packet-Tracer [5]	●		●		●	○	◐			
OPNET [14]	●		●		◐		◐	○	○	
SupernaNET [16]	●	●	–		–		○	–	–	–
CloudSim + EMUSIM [36][35]	○	●		●	○	●	◐			
NS-2 + GreenCloud [47][63]	●	●		●	○	●	○	○	○	
Boson NetSim [2]	●		●		◐			●	●	●
EstiNet [7]	●	●	–		–			–	–	–
GNS3 [8]	●	○	●	●	◐	●	●	●	●	●

Table 4.2: *A comparison of Network and Cloud Simulators. Key:* ● *(offers the benefit);* ◐ *(almost offers the benefit);* ○ *(offers partial benefit); blank (benefit not offered);* – *(No Information Available).*

lates networks and system administration experiments. Live Raizo contains GNS3, VirtualBox, QEmu, VPCS and multiple Linux VirtualBox guests. It also includes minicom, Wireshark, as well as DHCP, DNS, FTP, TFTP and SSH servers. This training appliance is used by the CFA UTEC training centre in Seine et Marne in France for networking and Linux administration classes.

4.3 Preliminaries

In this section we introduce different software programs that are used to make the test bed. Hardware that was used in setup environment is explained in 4.4.

4.3.1 GNS3

Graphical Network Simulator (GNS3) [8] is an open-source software that prepares the environment for emulating Cisco and Juniper routers in a simulated network. Dynamips, Oracle VirtualBox [98] and Qemu are three main back-end components that GNS3 use them to emulate different network OSes [2]. GNS3 acts as a front-end system with a GUI that enables the user to make arbitrary network. GNS3 is mainly used for training purpose and network labs. We use GNS3 as the base layer of our cloud test-bed.

4.3.2 VirtualBox

VirtualBox [98] is a free x86 and AMD64/Intel64 virtualization software. It is used in GNS3 for emulating Juniper JunOS. We use VirtualBox to make VMs in our cloud environment and benefit from features that VirtualBox offers for migration to make a test-bed for VM migration in a simulated data center environment.

4.3.3 Wireshark

Using a tool to capture and analyze packets is a must in network and security researches. Wireshark [42] is an open-source, multi platform network packet analyzer that can perform a live capture of network traffic. Captured traffic can be saved and analyzed offline. Wireshark can be used with GNS3 to analyze the packets that pass through each interface. Interface can be a network device port or even a host NIC. Wireshark offers various filters that we use them frequently in our tests.

[2]Visit the following URL for a list of network OSes that can be emulated by GNS3: http://www.gns3.net/hardware-emulated/

4.3.4 TFTP/SFTP Server

Trivial File Transfer Protocol (TFTP)[3] [86] is a protocol to transfer files and is supported by many commercial routers and firewalls in order to read/write configuration data from/to their flash memory. We use a TFTP server in source, to read firewall configuration and extract the rules that has to be migrated and another server in destination to insert the rules into firewall.

4.4 Test-Bed Description

In this section we explain our test-bed architecture, environment setup and network design for two data center with 3-tier deployment model. We also introduce some tools as well as optional software applications that can facilitate network configuration and tests.

4.4.1 Test-bed Architecture

Nowadays, cloud management systems are common to manage different data centers. The main concept behind such software, is having a global real-time view of all the network and resources in every data center which is owned by one entity (one or group of collaborating organizations). With such software, global administrative tasks can be done with a clear vision on the entire cloud system. In addition, the resources can be distributed more efficiently and evenly between VMs. As it mentioned earlier in 4.1, there is no simulator available (at the time of writing this thesis) which provides such view and administrative features. The reason is clear! Because these systems

[3]SSH File Transfer Protocol(SFTP)[50] offers the security option for file transfer protocol.

consist of various services and in order to imitate a service we need to emulate things. At the time of writing this dissertation, no such emulator is available publicly. Hence, we used some open-source projects to build up the cloud test-bed architecture and implemented a sample base topology for performing cloud related experiments. Some examples of possible experiments that can be done on this test-bed are introduced in 4.6.

In our test-bed architecture (depicted in Figure 4.1), GNS3 plays the role of cloud management system who has a global view of the network. All of the network devices and the links between them, are provided by GNS3. VirtualBox, which is a virtualization software, plays the role of hypervisor in cloud systems. As long as we target security testings in our research and this test-bed is actually prepared to host security experiments, we need another element which is missing in VirtualBox. In data centers, multi-tenancy is a matter of concern and this means that there should exist a firewall between different VMs. Moreover, some type of VMs contain more sensitive information (i.e. data-base VMs) than the others (i.e. web-server VM). Thus, a distributed firewall should be placed between VMs. We use a physical firewall/router to perform this task (since virtual firewalls are not supported in GNS3) and steer all the traffic between VMs to the firewall. However, using a firewall/router instead of a switch raise some questions that need elaboration. First , Can we use firewall/routers instead of switches? Actually, switches in data center are normally working in network layer or even higher in application layer. Packet inspections and firewalling are very common feature in most of today commercial switches. So, the

answer is absolutely yes [4]. Second question that may come to the mind of reader is, Why a physical device is used for firewalling between VMs? Isn't it better to use a virtual firewall? Virtual and physical firewall do exactly the same thing. One is inside a box with specialized hardware and does exactly the same job, as the other one (virtual firewall); although, when it is hardware we can expect faster process. Data center level hypervisors, usually come with a security package that contains a virtual firewall in it but since our hypervisor does not have this package, we use a physical firewall to safeguard inter-VM communications. Final question that a reader may ask is, What about few number of available ports in a physical firewall? They seem not to be sufficient for connecting all VMs! That is true. Physical firewalls have a limited number of ports; however, three internal port would be enough for connecting each group of VMs to the firewall. We do not need to show all of the VMs in test-bed (since security is the issue). So, based on the three tier design model that consists of *W*eb, *A*pplication and *D*atabase servers, one VM considered a representative for each type of virtual server (See Figure 4.2).

As shown in Figure 4.1, Test-bed architecture has three tiers: 1)*Physical* layer, 2)*Simulation* layer and 3)*Testing* layer. *Physical* layer consists of *Host* machines and includes the hardware resources. Each Host plays the role of one data center in cloud and VMs are residing on storage device of Host machines. Qemuwrapper, vboxwrapper and dynamips are files that enable the connection of GNS3 with distributed remote machines. Thus, simulated routers, firewalls and VMs can exist on different machines. Next layer (*S*imulation layer) is GNS3 that is the main tier in

[4]There are features in switches that are not available in firewalls and there are performance differences which we considered them as negligible for this special test-bed that is meant to handle security tests

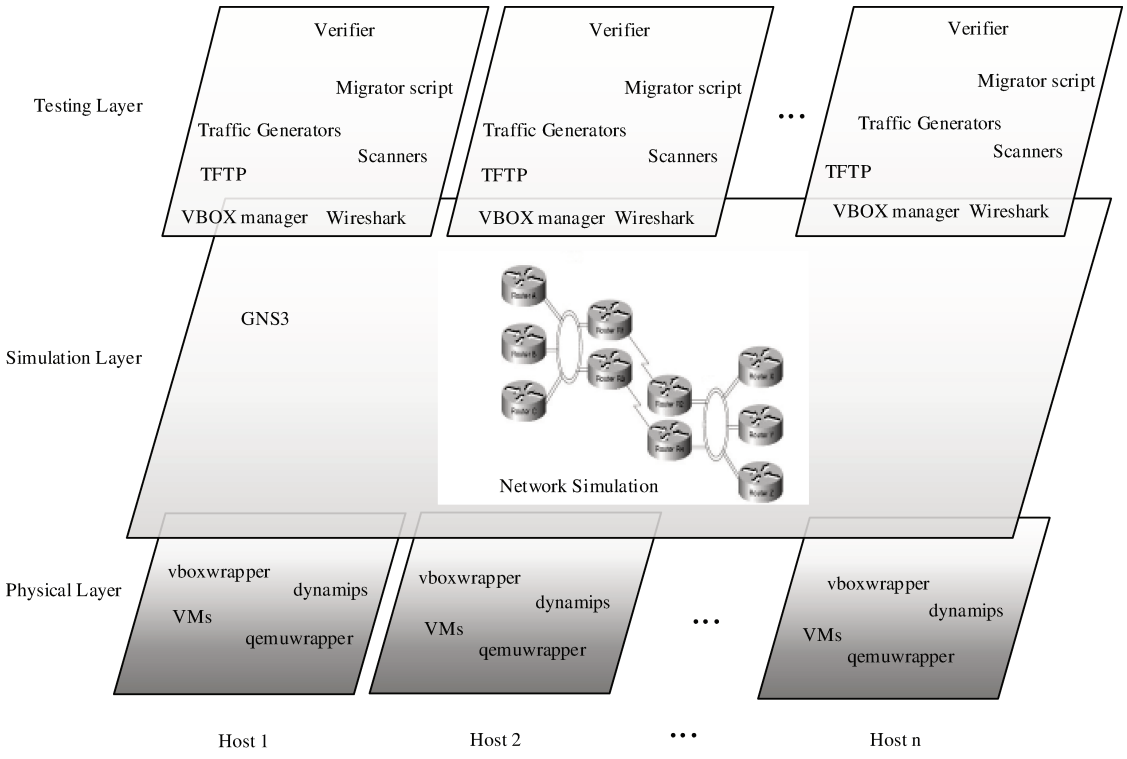

Figure 4.1: *Test-bed Architecture*

simulation procedure; GNS3 is able to communicate with all hosts and, like a cloud management system, has a global view of entire network topology. A simulation can be carried out completely with the two mentioned layers. Top layer (*Testing* layer), whose name can clearly reveal its task, is for experimenting and consists of various tools such as traffic-generators, packet-sniffers, scanners and other network testing tools that can be used in conducting various experiments. *TFTP* server in testing layer is for communicating with some firewalls and routers in order to access (and edit) their configurations. *VBOX manager* and *Wireshark* were introduced in Section 4.3. Many other software applications can also be implemented to conduct a research or ease the test procedure. For example, a *Migrator* script can automate the migration process by detaching the VM from source location and attaching it to the destination network with respect to all migration procedure conditions (Migration framework is explained in 4.5). Another application could be an automatic *Verifier* application that validates and verifies the correctness of migration with respect to security preservation aspect in VM migration.

4.4.2 Environment Setup and Network Design

Figure 4.2 demonstrates the network topology of our test-bed implementation. This test-bed is mainly prepared to do experiments on VM migration scenarios between two data centers. GNS3 0.8.5 is used for simulating network devices. VMs are created by VirtualBox 4.2.16. GNS3; VMs of `Datacenter_1` are located on Host 1, and VMs of `Datacenter_2` are placed on Host 2. Host 1 and Host 2 are Windows 7 machines with Intel Core i7 3.4 GHz processor, 16 Gbytes of RAM and Intel Core 2 duo 3 GHz processor, 4 Gbytes RAM, respectively; they are connected to each other by a CAT

6 cross cable. Internet in this Setup is a a group of simulated routers; however, it is possible to connect the network to real internet through a physical router which is out of the scope of this test-bed implementation. `Corp_User` and `Internet_User` are VMs who are located on other Hosts (with same configurations as Host 2) and are connected to Host 1 through a campus LAN network.

Each `Datacenter` has *Core*, *Aggregation* and *Access* layers; which is the well-known way of data center deployment, nowadays. Each layer has firewall-enabled routers that are deployed with the identical redundant component for failover cases. Aggregation and Access firewalls have VPN modules that support site-to-site IPsec VPN connections. VMS are defined with respect to the well-known 3-tier WEB, APPLICATION and DATABASE service model. Each firewall in access layer, controls the inter-VM traffic. `Corp_Network` is a corporation network that has special privileged access to VMs and `Internet_User`, represents a general user on the internet. VMs can have arbitrary operating systems.

4.4.3 Optional software and useful tools

Various optional software applications are installed on each Host and VM, that can be categorized in four groups: 1)Packet generators and network tools 2)Network scanners 3)Graphical device managers 4)CPU optimizers. A brief introduction of each application comes as follows:

1. **Packet Generators and Network Tools:**

    - **Netcat [53]:** A well-known free Unix-based, network utility program that can read and write data across network connections.

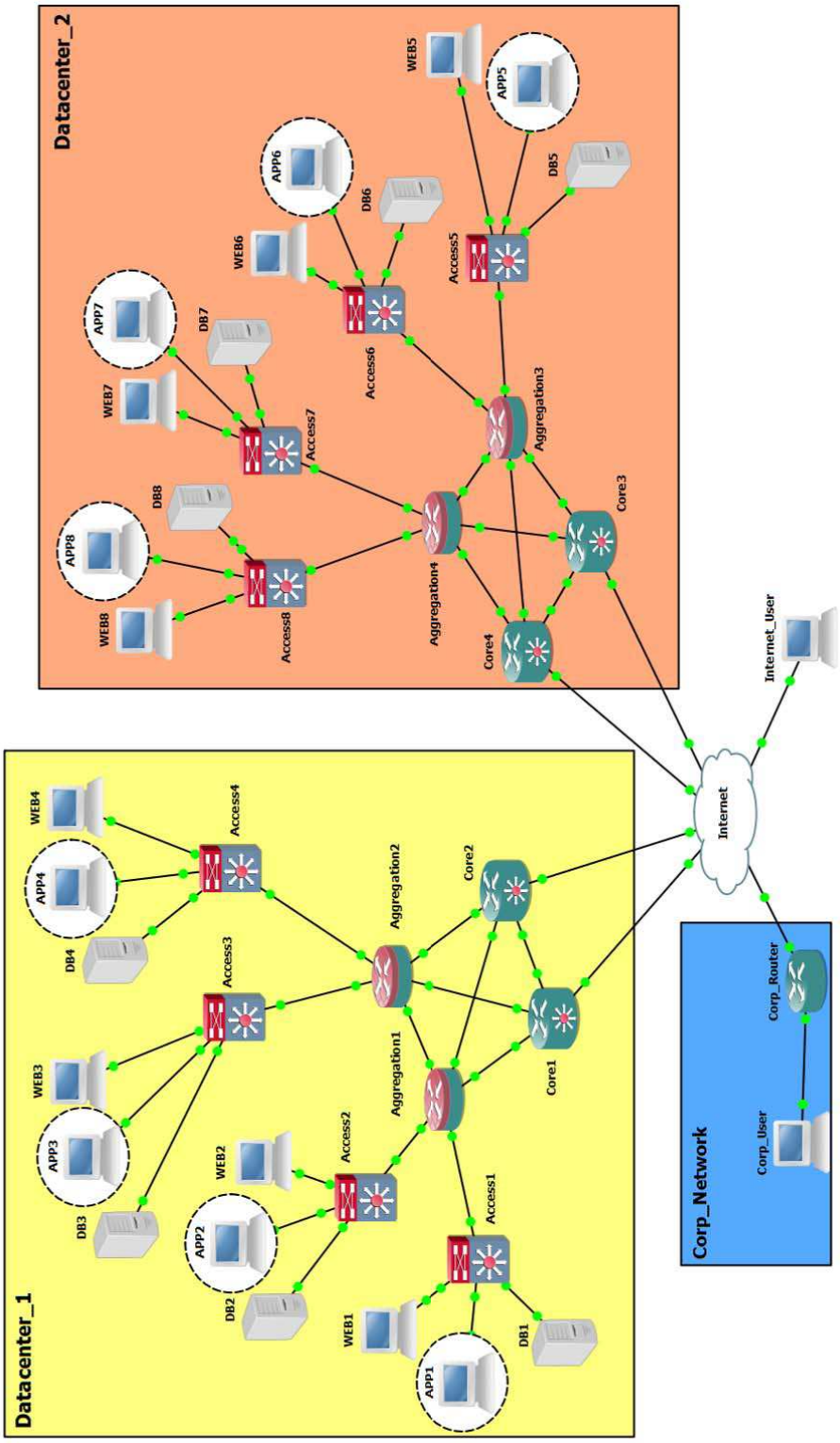

Figure 4.2: *Test-bed Topology*

- **hping [82]:** A free network security tool that is able to send custom TCP/IP packets. hping can be used for testing firewall rules and port scanning.
- **Scapy [30]:** An open-source packet manipulation program written in Python. It can be used for crafting packets, scanning, probing, attacks and some other purposes in order to test network.

2. **Network Scanners:**

- **Nmap [68]:** An Open-source tool for exploring networks and security auditing with a GUI for Windows. Nmap supports many features such as OS fingerprinting and ping sweeps.

3. **Graphical Device Managers:** [5]

- **Cisco ASDM [3]:** Cisco Adaptive Security Device Manager (ASDM) is a GUI appliance management tool that provides means for network administrator to configure, monitor and troubleshoot Cisco firewalls.
- **Cisco SDM [6]:** Cisco Router and Security Device Manager (SDM) is a Web-based management tool for Cisco routers.
- **Cisco IPS Manager Express [4]:** Cisco IPS Manager Express is a GUI-based tool for Cisco IPS sensor management.

4. **CPU Optimizers:**

[5]All Cisco appliance management tools are only available for Cisco users who have the corresponding hardware and license

- **BES [1]:** Battle Encoder Shirase (BES) is a free CPU control tool that provides the capability of limiting CPU usage of any selected process.

4.5 Migration Framework

Migration of a VM from one physical server to another, can be done in two different ways. These methods are based on the features that are provided by VirtualBox. below, each method with its features is explained:

1. **Live Migration by Teleporting** : *Teleporting* is a feature of VirtualBox that enables live migration of a VM state. However, it has some conditions and limitations that has to be considered. First limitation is that the migration can be done between two VMs with the same configuration. An important condition for migration is that both of the VMs has to be running at the migration moment; this means that a VM in destination of the migration should be running and ready to receive the running state from source VM. Sending the state is through a TCP/IP port and it starts from destination side by listening to the port. Then source VM sends its running state to the listening VM and stops that process(VM is still running). Whereas teleporting supports multi platforms, some errors may occur when CPU of the source and the destination machines have different architectures. Last and the most important condition for teleporting is a shared storage that keeps both source and destination VMs. It implies that even though the source and the destination can be two separate physical machines, they need to have access to a shared storage and both of the mentioned VMs have to reside on that shared storage device. Access to the

shared storage can be achieved by implementing Network File System (NFS) or Server Message Block (SMB)/Common Internet File System (CIFS)[6].

Although in many real world data centers, live migration is similar to what offered by teleporting in VirtualBox, a shared storage between data centers is not always available. Normally, shared storage is available between servers that are located within the same data center. Hence, we propose another way to migrate a VM without having access to a shared storage media by using snapshots.

2. **Migration by Moving Snapshots** : *Snapshot* is another feature that VirtualBox offers for saving and restoring the current state of a running VM. By default, this feature is not meant to use for VM migration, however we utilize it in our migration framework. Snapshot is the preservation of the current running state of a VM. This state can be reverted in future or immediately after taking the snapshot. When a snapshot is taken, three things are saved to the disk:

   - **All VM Configurations:** A small XML file that contains complete copy of the VM settings and configurations.
   - **Hard Disk Image:** A file that contains the state of all virtual disks that are attached to the VM. Note that this file is a *differencing image*, and is not the entire virtual hard disk image.
   - **Memory State:** An image from the memory of running VM that can be as large as the size of VM memory.

[6]Samba is a free implementation of SMB that can be used for sharing storage in UNIX systems.

There are two ways for migrating the VM and its state by using snapshots. First way is freezing the VM in the source location and migrating it to the destination in one shot. Apparently, it takes quite a long time (depends on the entire VM size) which may not be acceptable for a VM migration process. In our migration framework, we leverage snapshot features and move the VM and its state from source to destination in two steps (See Figure 4.3). In the first step, the source machine sends a copy of VM virtual hard disk to the destination place. Apparently, sending the virtual hard disk can take a long time but this is not a problem, because meanwhile, VM is running in the source location. When VM virtual hard disk is completely received by the destination, the source machine freezes the VM state by taking a snapshot from the running VM. As it mentioned earlier, snapshot creates a file that contains differencing image from the virtual hard disk in addition to two other files that contain VM configurations and memory state, respectively. In the second step, source machine sends the snapshot to the destination machine. When the VM snapshot has been received in the destination location and VM connection to the destination network established properly, VM starts running and resumes all the processes that were stopped before migration. At this point, the source storage is allowed to wipe out the VM. Although the VM transfer is not a live migration, it has a short downtime and more importantly, it does not need a shared storage between source and destination machines. All the steps in the migration process can be verified by a tool that checks the correctness of each step, and either pass or revert the migration process. An extension to this framework is a security preserved migration framework that is explained in 4.6.

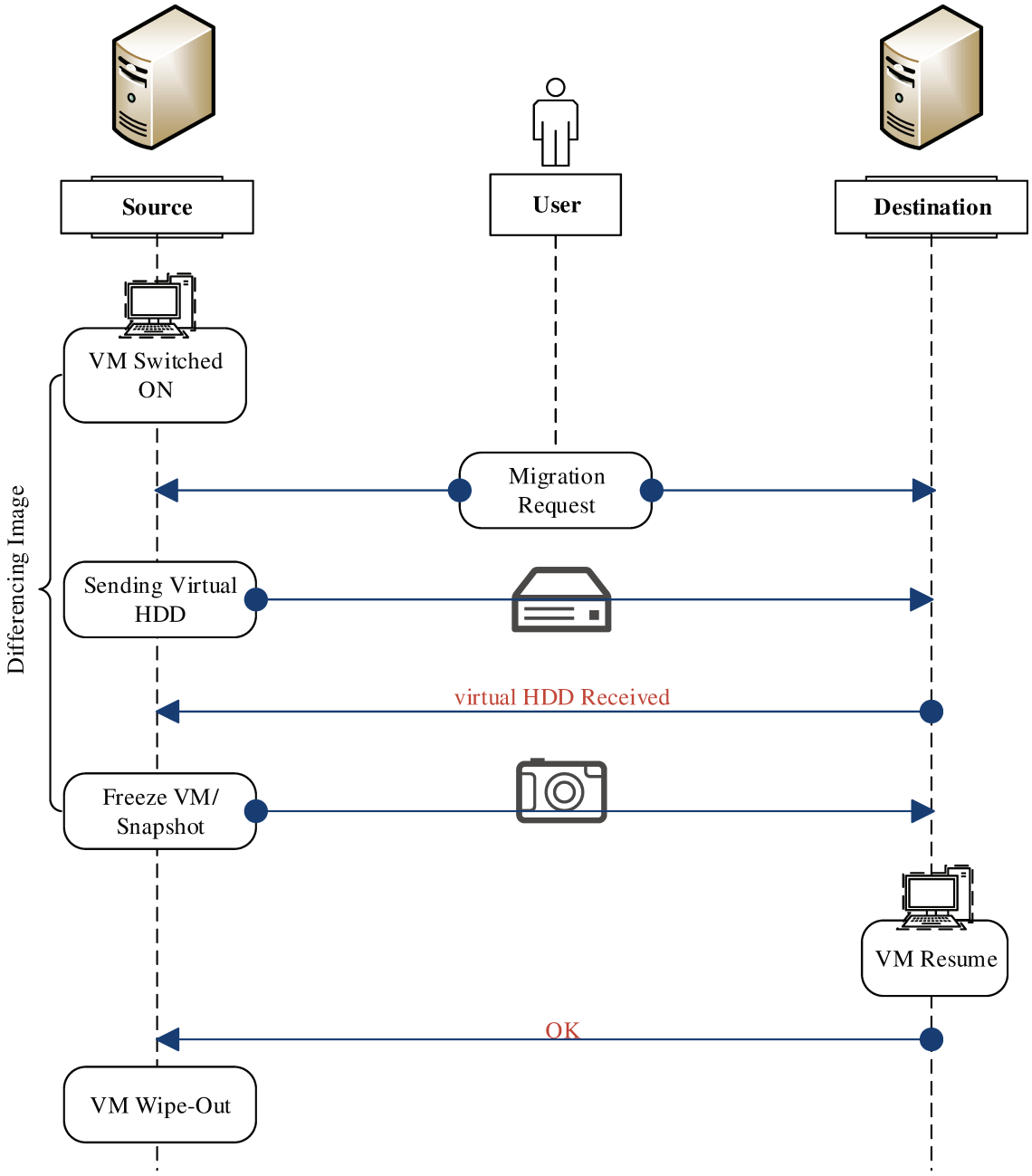

Figure 4.3: *VM Migration Framework*

4.6 A Case study and other Use Cases

In this section we first discuss about some use cases of our cloud test-bed and then we study a VM migration case from one data center to another.

Our testing environment has a flexible architecture that enables the user to test various types of security scenarios for different purposes. For instance, one of the features that can be added to the test-bed is Network-based IPS [7] devices. IPS is a security appliance that monitors the network activities and block/stop malicious activities after detecting them. Many researches can be conducted to evaluate the impact of using IPS on the performance of data centers and finding methods which can optimize it. Another possible use case could be testing Site-to-Site or Client-to-Site VPN from different locations and then investigate how the IPsec connection can be affected by VM migration or firewall rule changes. Moreover, various scenarios can be designed to test the compliance of SSL VPN, which is also supported by this test-bed. It is worth mentioning that, although VPN can significantly improve the security of a remote connection, misconfiguration of that can backfire and turn VPN into an attack vector.

Principally, this test-bed architecture can be used as a testing environment in network and cloud security researches or as a training appliance for security labs and online classes.

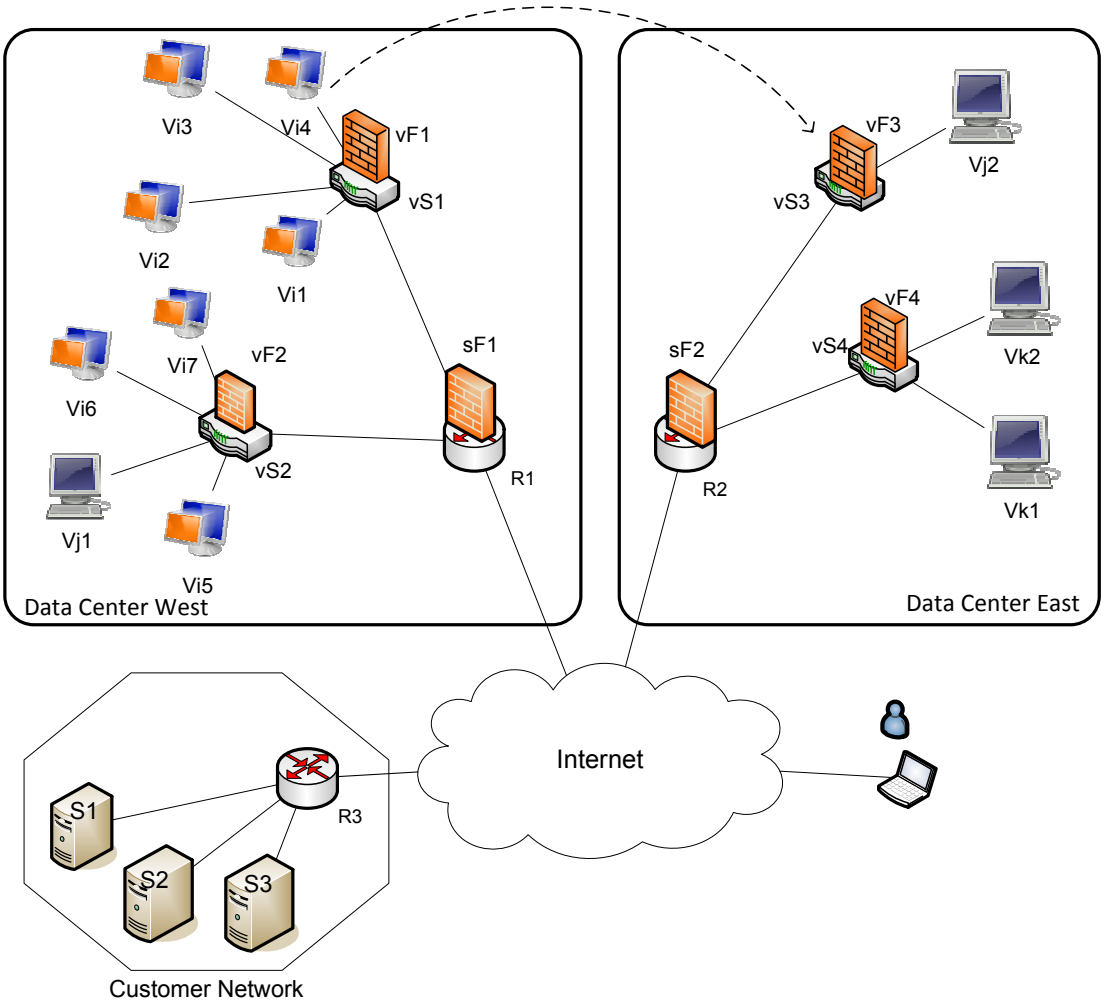

Figure 4.4: *Case Study Topology*

4.6.1 Case-Study: Firewall Rule Migration and Verification

We now explain the migration case-study (See Figure 4.4). We assume that the migration can be initiated and controlled by a network admin or an automated program. This Migrator not only moves the VM, but also is responsible for migrating the firewall rules. Thus, the security of VM and data centers can be preserved after migration.In this case study we deploy a typical three-tier cloud of web applications: A web tier that implements the presentation logic, application server tier that implements the presentation logic, application server tier that implements the business logic, and a back-end database tier. A possible implementation would be to deploy a virtual machine per each tier. The case study consists of two data centers (west and east) as depicted in Figure 4.4. We suppose that a tenant i has VM instances $\{\text{Vi}_l\}_{1 \leq l \leq 7}$ such that Vi_1 is a database, $\{\text{Vi}_2, \text{Vi}_3\}$ are application services, and $\{\text{Vi}_4, \text{Vi}_5, \text{Vi}_6, \text{Vi}_7\}$ are web services. The tenant has also specified a firewall policy for each VM group as follows

1. Web group allow any host to connect on ports 80 (http) and 443 (https),
2. Application group allow only web services to connect to port 8000,
3. Database group allow only application services to connect to port 3306,
4. All the above groups allow the corporation network (CorpNet) to connect on port 22 (ssh).

[7] A tutorial for how to install and use IPS in GNS3, is available at: http://www.brainbump.net/how-to-emulate-cisco-ips/

The firewall rules before migration are given in Table 4.3. We consider the case where the virtual machine Vi_4 has to be migrated from data center west (DW) to data center east (DE).

The firewall configurations after a correct migration are given in Table 4.4.

We consider the following three scenarios in order to conduct the test:

- *Scenario 1 - Migration Error 1.* Rule 2 in vF1 before is deleted from vF1 after migration but is not added to sF1 after migration.

- *Scenario 2 - Migration Error 2.* Rule 3 in sF1 that allows access to *ssh* service on Vi_4 from customer network before is not added to sF2 after migration.

- *Scenario 3 - Correct Migration.* The configurations of all the firewalls are updated correctly.

In order to verify access control preservation after VM migration, we use the approach that presented by Jarraya et al. [60]. See Appendix A for firewall paths and a subset of generated CSP constraints.

We test each scenario by scanning the ports of a target VM from all other VMs and capturing the traffic on the sending as well as on the receiving interfaces. The goal of this test is to make sure that the new configuration after the VM migration preserves the security policy. To this end, we used Zenmap[8], the windows GUI for Nmap security scanner, to scan all ports. We also used the Wireshark packet analyzer in order to compare captured packets from the originated VM/network with respect to the packets that successfully passed through the firewall and received at the destination VM.

[8]http://nmap.org/zenmap/

Table 4.3: *Firewall Policy Before Migration*

sF1							
1.	TCP	*	*	**Vi4**,Vi5, Vi6,Vi7 Vj1	80	NEW, ESTAB	Allow
2.	TCP	*	*	**Vi4**,Vi5, Vi6,Vi7 Vj1	443	NEW, ESTAB	Allow
3.	TCP	CorpN	*	Vi1,Vi2, Vi3,**Vi4**, Vi5,Vi6, Vi7	22	NEW, ESTAB	Allow
4.	TCP	Vi5,Vi6, Vi7	*	Vi2,Vi3	8000	NEW, ESTAB	Allow
5.	TCP	I_DW	*	O_DW	*	NEW, ESTAB	Allow
sF2							
1.	TCP	*	*	Vk1,Vk2	*	NEW, ESTAB	Allow
2.	TCP	*	*	Vj2	80	NEW, ESTAB	Allow
3.	TCP	*	*	Vj2	443	NEW, ESTAB	Allow
4.	TCP	I_DE	*	O_DE	*	NEW, ESTAB	Allow
vF1							
1.	TCP	Vi2,Vi3	*	Vi1	3306	NEW, ESTAB	Allow
2.	**TCP**	**Vi4**	*	**Vi2,Vi3**	**8000**	**NEW, ESTAB**	**Allow**
3.	**TCP**	*****	*****	**Vi4**	**80**	**NEW, ESTAB**	**Allow**
4.	**TCP**	*****	*****	**Vi4**	**443**	**NEW, ESTAB**	**Allow**
vF3							
1.	TCP	*	*	Vj2	80	NEW, ESTAB	Allow
2.	TCP	*	*	Vj2	443	NEW, ESTAB	Allow

Table 4.4: *Firewall Policy After Migration*

sF1							
1.	TCP	*	*	Vi5,Vi6, Vi7,Vj1	80	NEW, ESTAB	Allow
2.	TCP	*	*	Vi5,Vi6, Vi7,Vj1	443	NEW, ESTAB	Allow
3.	TCP	CorpNet	*	Vi1,Vi2, Vi3,Vi5 Vi6,Vi7	22	NEW, ESTAB	Allow
4.	TCP	Vi5,Vi6, Vi7,**Vi4**	*	Vi2,Vi3	8000	NEW, ESTAB	Allow
5.	TCP	I_DW	*	O_DW	*	NEW, ESTAB	Allow
sF2							
1.	TCP	*	*	Vk1,Vk2	*	NEW, ESTAB	Allow
2.	TCP	*	*	Vj2,**Vi4**	80	NEW, ESTAB	Allow
3.	TCP	*	*	Vj2,**Vi4**	443	NEW, ESTAB	Allow
4.	**TCP**	**CorpN**	*	**Vi4**	**22**	**NEW, ESTAB**	**Allow**
5.	TCP	I_DE	*	O_DE	*	NEW, ESTAB	Allow
vF1							
1.	TCP	Vi2,Vi3	*	Vi1	3306	NEW, ESTAB	Allow
vF3							
1.	TCP	*	*	Vj2,**Vi4**	80	NEW, ESTAB	Allow
2.	TCP	*	*	Vj2,**Vi4**	443	NEW, ESTAB	Allow

Figure 4.5, Figure 4.6, and Figure 4.7 illustrate charts that we acquired from Wireshark captures at Vi_4 after running Nmap port scan from various locations, with firewalls configuration before migration, after correct migration, and scenario 2, respectively. The X-axis is represents the type of packets who were sent by Zenmap, where we typically show packets corresponding to the tcp 3-way handshake (SYN,SYN-ACK,ACK) for the HTTP, HTTPs, and SSH as well as the ones for ICMP requests. However, we group all the other type of packets under the label "others". The Y-axis represents the percentage of packets received by Vi4 per source and per packet type.

The results of the experiments on the accessible services over the different paths to Vi4 matches the satisfiability results obtained by verification approach. With respect to Figure 4.5 and Fig. 4.6, we can observe that the same type of packets is accepted before and after a correct migration of the rules. The percentage of packets is different as we did not have control over the number of packets sent by Zenmap. However, we can compare the verification results and the testing results, by the type of packets. With respect to chart of Figure 4.7, one can notice that packets of SSH service that were supposed to be accepted by Vi4 from customer network are being filtered by the firewall configuration in the erroneous scenario (Scenario 2). In Figure 4.7, the percentage of received packets which ssh+flag(syn)=1, is 0% (hashed circle area) compared to the percentage of the same type of packets in Figure 4.5. This confirms the verification results for scenario 2.

In order to prevent the error cases that mentioned before, we can integrate the CSP verification method into migration framework. Figure 4.8 shows the VM and Firewall Migration framework that also verifies the correctness of rule migration. Migration

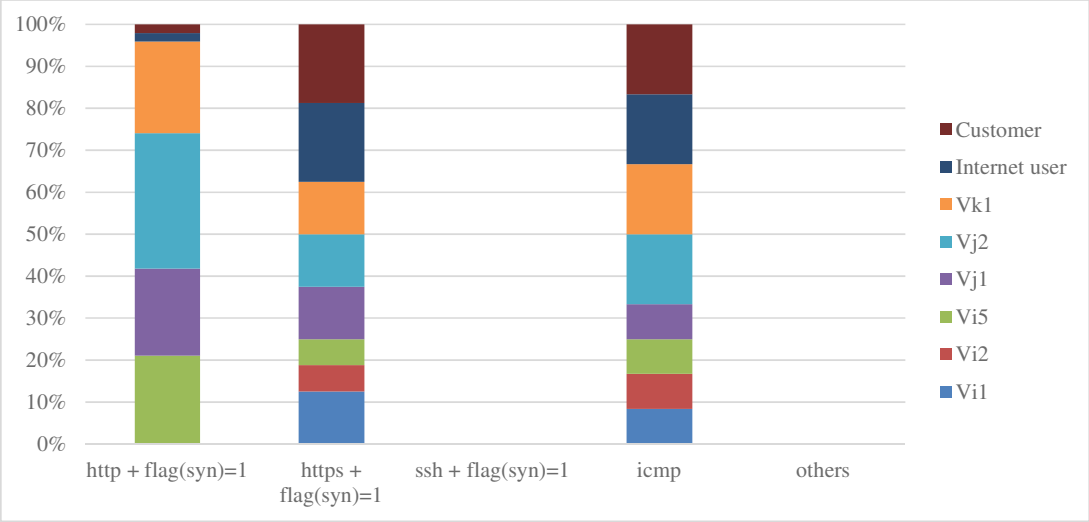

Figure 4.5: *Percentage of Packets Received by Vi4, per Packet Type and per Source - Before Migration*

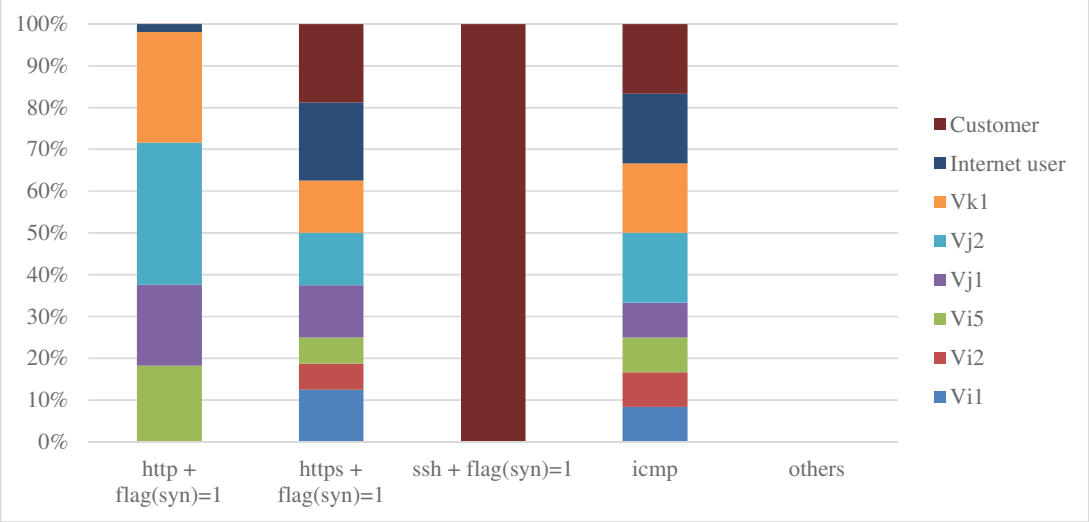

Figure 4.6: *Percentage of Packets Received by Vi4, per Packet Type and per Source - After Correct Migration*

can only be completed if the it can pass the verification procedure. Migrator can be an automated program or the procedure can be done manually by a network

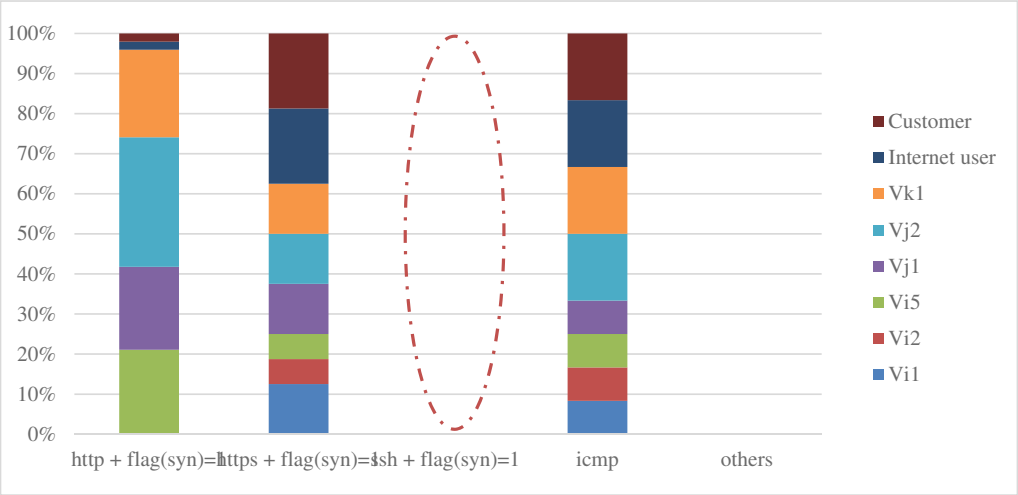

Figure 4.7: *Percentage of Packets Received by Vi4, per Packet Type and per Source - the Erroneous Migration Scenario 2*

administrator. A TFTP server can be used in order to read and write the security configuration from/to Cisco firewalls.

4.7 Limitations

Major limitations include:

(*a*) The test-bed is using a wide range of applications that should run simultaneously. Even though, it is possible to balance the load to some extent by distributing network devices and VMs on different hosts, the router simulation is a CPU-intensive procedure which limits the number of simulated router per machine. This number is directly dependent on the machine CPU and RAM. So as it mentioned before, data centers can be simulated in a very small scale but capable of running real services. **(*b*)** The migration framework, does not apply to a live migration. Always, there is a delay between freezing the VM in source and resuming it in destination data center.

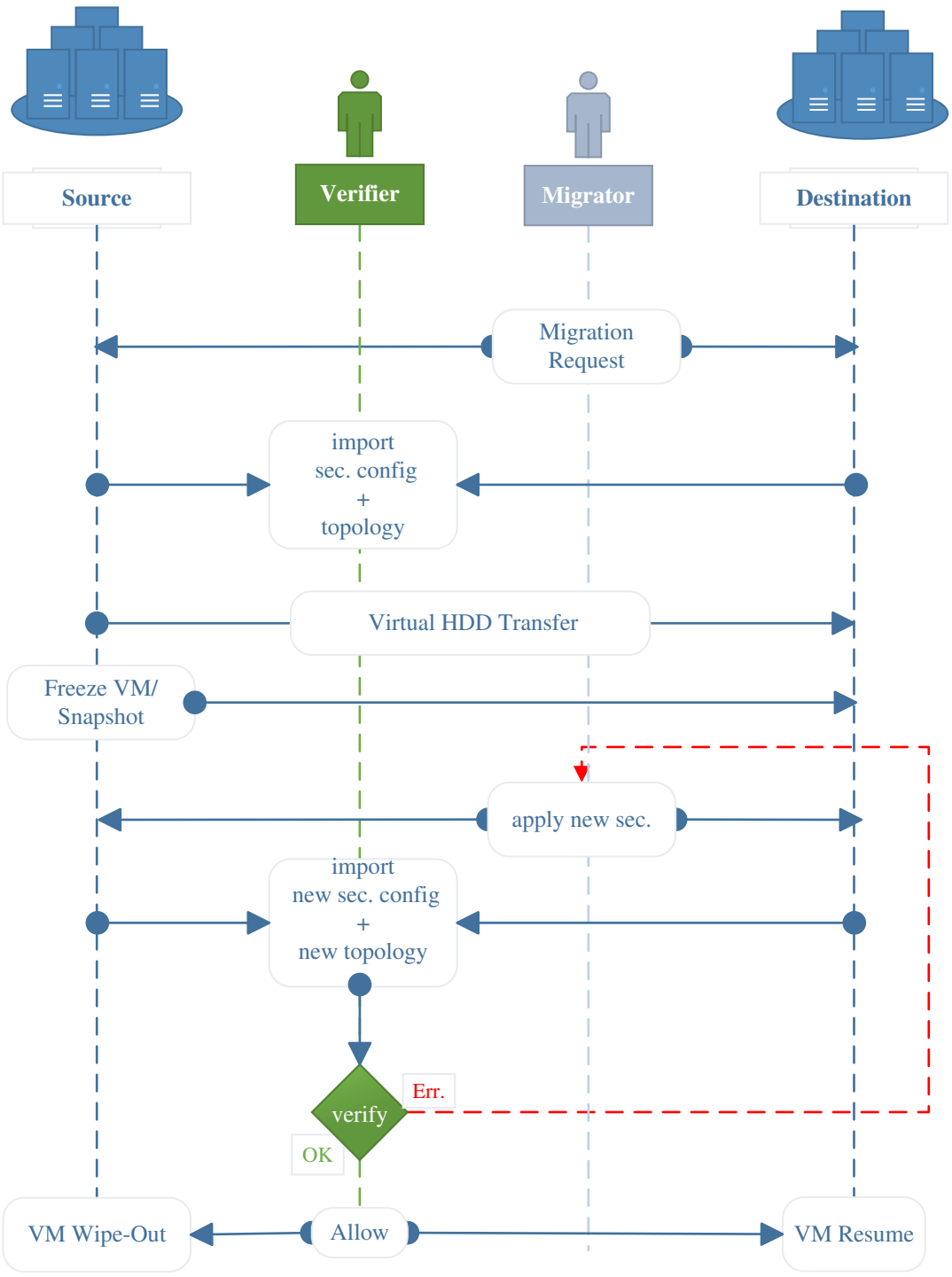

Figure 4.8: *VM and Firewall rules Migration Framework*

This delay is directly proportional to the size of VM virtual disk and inversely proportional to the source-destination communication speed. A large virtual hard disk, needs longer time to migrate from the source to the destination. This longer time creates a bigger differencing image and a bigger differencing image, causes more delay for resuming VM in the destination.

A possible solution to improve the framework is to send the hard disk differencing image from the source to destination, after the completion of hard disk transfer without stopping the VM in source location. Another differencing image is being created from the starting time of sending first differencing image; this cycle can be repeated a few times. The iterations, decrease the size of differencing image and as a result, the delay between stopping the VM in the source and resuming it in the destination, shortens significantly. However, this will increase the total time of the migration from the moment that migration has been requested by the User.

4.8 Summary

In computer networking researches, a testing environment can help to examine the correctness and the soundness of a solution or a finding. Simulators and emulators are affordable software or hardware solutions for modeling a real network and there are many of them available in open-source and commercial versions. Simulators have some limitations; for example running a real service is not possible on most of them, because they are using mathematical formulas to model a network and predict the behavior and the future state of that network. Thus, simulators are not suitable for the majority of security testings because security breaches normally exist in application layer. Moreover, security services and protocols cannot properly be implemented

on simulators. Hence, emulators have the advantage of offering real services on a general purpose hardware in a small scale. There are a few number of mature cloud simulators that most of them are focused on performance evaluations and energy/cost assessment. In this chapter, we presented a test-bed for testing security and VM migration, based on GNS3 network emulator. To this end, we proposed a conceptual three layer architecture that defines a new way of using GNS3, as a distributed cloud simulating and testing environment with multiple host machines on the underlying tier and various network security testing tools on the top layer. We also presented a framework for migration of VM between hosts who play the role of data centers in the simulation. Then as a case study, we designed a security-aware extension in order to migrate the firewall rules that are related to the migrating VM and verified it by CSP formulas. Finally, we introduced other possible use cases of our test-bed. As a future work, we are going to use multiple Amazon AWS accounts, as remote infrastructure through connecting the test-bed to the real internet. By doing this for each simulated data center, we can distribute the entire network and VMs among instances in one AWS account. As a result, we would expect a more realistic simulation and a better performance in larger scales.

Chapter 5

Conclusion and Future Work

Cloud computing is a fast-developing area that relies on sharing of resources over a network. While more companies are adapting to the cloud computing and data centers are growing rapidly, data and network security gain more importance and still firewalls are the most common means to safeguard the networks for any size. Whereas today data centers are distributed around the world, VM migration within and between data centers is inevitable for an elastic cloud. In order to keep the VM and data centers secure after migration, the VM specific security policies should move along with the VM as well.

In this thesis, we provided background on different domains such as cloud computing, Software-Defined Networks and firewalls in cloud environment and we discussed the VM migration security issues. We investigated the problem in both traditional and software-defined networks. We presented a novel framework for migration of the security policies along with virtual machine migration in SDN context. Our framework migrated the security context through coordinating hypervisor with SDN security applications that run on top of the controller in a VM migration process. This provisioning preserved the security of data centers and VM after VM migration. We also designed and developed MIGAPP, a distributed prototype application for floodlight SDN controller. This application interacted with the controller through a REST API and communicated to its peers as well the hypervisors, through Rab-

bitMQ distributed messaging system. Then we evaluated our application in a SDN simulation and programming environment. Moreover, in order to simulate migration we added this functionality to the simulator program. Finally, we measured the efficiency and the scalability of our application and the results showed an acceptable trend in time increment that is caused by the number of the rules increase.

In addition, we presented a distributed cloud test-bed, consisting of a GNS3 emulator, VirtualBox and a wide range of security tools that can be used in various test cases in security labs, online classes and research projects. We also designed a framework for migrating VirtualBox VMs, between two different machines without a shared storage.

Future work can be developing a comprehensive application for various SDN controllers. As an extension to this work, we can consider migration of QoS or even load-balancing rules with VM.

Bibliography

[1] Battle encoder shirase. http://mion.faireal.net/BES/. [Online; accessed 30-September-2013].

[2] Boson netsim 9.0. Technical report, Boson.

[3] Cisco adaptive security device manager. http://www.cisco.com/en/US/products/ps6121/.

[4] Cisco ips manager express. http://www.cisco.com/en/US/products/ps9610/index.html.

[5] Cisco packet tracer. https://www.netacad.com/web/about-us/cisco-packet-tracer.

[6] Cisco router and security device manager. http://www.cisco.com/en/US/products/sw/secursw/ps5318/.

[7] Estinet network simulation cloud. http://www.estinet.com/products.php?lv1=12&sn=10.

[8] Gns3 graphical network simulator. http://www.gns3.net/.

[9] Introducing json. http://www.json.org/.

[10] Junos sdk. http://www.juniper.net/as/en/products-services/junos-developer/junos-sdk/.

[11] Netsim. http://tetcos.com/.

[12] Ns-3 network simulator. http://www.nsnam.org/.

[13] Open vswitch. http://openvswitch.org/.

[14] Opnet. http://www.opnet.com/.

[15] Security appliance. ttp://en.wikipedia.org/wiki/Security_appliance. [Online; accessed 17-September-2013].

[16] Supernanet cloud simulator. Technical report, Superna. [Online; accessed 20-October-2013].

[17] Virtualization. http://www.vmware.com/virtualization.html.

[18] Securing the cloud: A review of cloud computing, security implications and best practices. Technical report, VMware and Savvis, 2009.

[19] Vmware fault tolerance. Technical report, 2009.

[20] Vmware high availability. Technical report, 2009.

[21] Vmware vmotion. Technical report, 2009.

[22] Global environment for network innovations (geni). http://www.geni.net/, September 2012.

[23] Data sheet: Scalable website/application how-to: Enable and manage gogrids firewall service. Technical report, GoGRID, 2013.

[24] Onf. https://www.opennetworking.org/sdn-resources/sdn-definition, 2013.

[25] Project floodlight. http://www.projectfloodlight.org/floodlight/, 2013. [Online; accessed 10-July-2013].

[26] Muhammad Abedin, Syeda Nessa, Latifur Khan, and Bhavani Thuraisingham. Detection and resolution of anomalies in firewall policy rules. In *Data and Applications Security XX*, pages 15–29. Springer, 2006.

[27] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *Selected Areas in Communications, IEEE Journal on*, 23(10):2069–2084, 2005.

[28] Amazon.com. Amazon web services: Overview of security processes. http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, May 2011.

[29] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[30] Philippe Biondi. Scapy. *Retrieved September*, 3:2011, 2011.

[31] David Bolton. Definition of virtualization. http://cplus.about.com/od/glossar1/g/virtualization.htm.

[32] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui. OpenFlow supporting inter-domain virtual machine migration. In *the Proceedings of the 8th International Conference on Wireless and Optical Communications Networks (WOCN)*, pages 1–7, 2011.

[33] Jon Brodkin. Gartner: Seven cloud-computing security risks. http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853, July 2008.

[34] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid

computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[35] Rodrigo N Calheiros, Marco AS Netto, César AF De Rose, and Rajkumar Buyya. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, pages 00–00, 2012.

[36] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[37] Andrew T. Campbell and Irene Katzela. Open signaling for atm, internet and mobile networks (opensig'98), 1999.

[38] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues, 2002.

[39] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[40] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. Sane: A protection architecture for enterprise networks. In *USENIX Security Symposium*, 2006.

[41] Cisco. Cisco Open Network Environment: Network Programmability and Virtual Network Overlays. Technical report, Cisco and/or its affiliates, June 2012. White paper C11-707978-00 06/12.

[42] Gerald Combs et al. Wireshark. *Web page: http://www. wireshark. org/last modified*, pages 12–02, 2007.

[43] CSA. Security guidance for critical areas of focus in cloud computing v3.0. Technical report, 2011.

[44] Florence de Borja. Idc report: It cloud services market. http://cloudtimes.org/2012/11/30/idc-report-it-cloud-services-market-2016/, November 2012.

[45] Avri Doria, Ram Gopal, Hormuzd Khosravi, Ligang Dong, Jamal Salim, and Weiming Wang. Forwarding and control element separation (forces) protocol specification. 2010.

[46] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. Netconf configuration protocol. *Network*, 2011.

[47] Kevin Fall and Kannan Varadhan. The network simulator ns-2. *URL: http://www. isi. edu/nsnam/ns*, 2007.

[48] S Fayazbakhsh, Vyas Sekar, Minlan Yu, and J Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *the Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN) August 2013*, 2013.

[49] Christophe Fillot. Dynamips. http://www.gns3.net/dynamips/.

[50] J Galbraith. Ssh file transfer protocol. 2007.

[51] Aaron Gember, Robert Grandl, Junaid Khalid, Shan-Hsiang Shen, and Aditya Akella. Design and Implementation of a Framework for Software-Defined Middlebox Networking. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2013. TR1794.

[52] Soudeh Ghorbani and Matthew Caesar. Walk the Line: Consistent Network Updates with Bandwidth Guarantees. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[53] Giovanni Giacobbi. The gnu netcat proje ct. *URL http://netcat. sourceforge. net*, 2013.

[54] Nelson Gonzalez1, Charles Miers, Fernando Redgolo, Marcos Simplcio, Tereza Carvalho, Mats Nslund, and Makan Pourzandi. A quantitative analysis of current security concerns and solutions for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications 2012*, 1, 2012.

[55] Vijay K Gurbani, Michael Scharf, TV Lakshman, Volker Hilt, and Enrico Marocco. Abstracting network state in software defined networks (sdn) for rendezvous services. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6627–6632. IEEE, 2012.

[56] Mohammad Hajjat, Xin Sun, Yu wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *the Pro-*

ceedings of the ACM SIGCOMM 2010 conference, pages 243–254, New York, NY, USA, 2010. ACM.

[57] HP. Prepare for software-defined networking: Build the foundation for SDN with OpenFlow. Technical report, Hewlett-Packard Development Company, L.P, February 2013. Business white paper 4AA3-8562ENW.

[58] IBM Systems and Technology. IBM Software Defined Network for Virtual Environments. Technical report, IBM Corporation, June 2013. Thought Leadership White Paper.

[59] IDC Corporate USA. SDN Shakes Up the Status Quo in Datacenter Networking, IDC Says . http://www.idc.com/getdoc.jsp?containerId=prUS23888012, 2012. published 19 Dec 2012. Last accessed June 2013.

[60] Yosr Jarraya, Arash Eghtesadi, Mourad Debbabi, Ying Zhang, and Makan Pourzandi. Formal verification of security preservation for migrating virtual machines in the cloud. In *Stabilization, Safety, and Security of Distributed Systems*, pages 111–125. Springer, 2012.

[61] Paulus Kampert. A taxonomy of virtualization technologies. Master's thesis, Delft University of Technology, August 2010.

[62] Eric Keller, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live migration of an entire network (and its hosts). In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 109–114, New York, NY, USA, 2012. ACM.

[63] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[64] Tommy Koorevaar. Dynamic enforcement of security policies in multi-tenant cloud networks. Master's thesis, Ecole Polytechnique de Montreal, November 2012.

[65] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a Distributed Control Platform for Large-Scale Production Networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[66] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[67] Pingping Lin, Jun Bi, Hongyu Hu, Tao Feng, and Xiaoke Jiang. A quick survey on selected approaches for preparing programmable networks. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pages 160–163, New York, NY, USA, 2011. ACM.

[68] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, 2009.

[69] Neil MacDonald. Five myths and realities of virtualization security. http://blogs.gartner.com/neil_macdonald/2012/09/06/five-myths-and-realities-of-virtualization-security/, September 2012.

[70] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman. Crossroads: Seamless vm mobility across data centers through software defined networking. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 88–96, 2012.

[71] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[72] Peter Mell and Tim Grance. The NIST definition of cloud computing. Technical report, September 2011.

[73] Marc Mendonça, Bruno Nunes Astuto, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, June 2013. In Submission In Submission.

[74] A Noordam and R van Loenen. Securing the virtual world, 2010. Presentation VMware Forum 2010.

[75] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. icancloud: A flexible and

scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[76] ONF. Software-Defined Networking: The New Norm for Networks. Technical report, Open Networking Foundation, April 2013. ONF white paper.

[77] Open Networking Foundation. *OpenFlow Switch Specification*, February 2011. Version 1.1.0 Implemented (Wire Protocol 0x02).

[78] Open Networking Foundation. *OpenFlow Switch Specification*, September 2012. Version 1.3.1 (Wire Protocol 0x04).

[79] Pivotal. Rabbitmq. http://www.rabbitmq.com/.

[80] PLEXXI,Lightspeed venture partners, SDNCentral. Sdn market sizing. http://cdn.sdncentral.com/wp-content/uploads/2013/04/sdn-market-sizing-report-0413.pdf. Market Report (Apr. 2013).

[81] Leonard Richardson and Sam Ruby. *RESTful Web Services: Web services for the real world*. O'Reilly Media, May 2007.

[82] Salvatore Sanfilippo. Hping–active network security tool, 2008.

[83] Karen Scarfone and Paul Hoffman. Guidelines on firewalls and firewall policy. Technical report, September 2009.

[84] Karen Scarfone, Murugiah Souppaya, and Paul Hoffman. Guide to security for full virtualization technologies. Technical report, January 2011.

[85] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation

challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):–, 2013.

[86] K Sollins. The tftp protocol (revision 2). 1992.

[87] Greg Stabler, Aaron Rosen, Sebastien Goasguen, and Kuang-Ching Wang. Elastic ip and security groups implementation using openflow. In *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date*, VTDC '12, pages 53–60, New York, NY, USA, 2012. ACM.

[88] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. System description of a sat-based csp solver sugar. *Proceedings of the Third International CSP Solver Competition*, pages 71–75, 2008.

[89] Zahra Tavakoli, Sebastian Meier, and Alexander Vensmer. A Framework for Security Context Migration in a Firewall Secured Virtual Machine Environment. In *the Proceedings of the 18th EUNICE/IFIP WG 6.2, 6.6 International Conference on Information and Communication Technologies, Budapest, Hungary, Aug. 29-31*, volume 7479 of *LNCS*, pages 41–51. Springer, 2012.

[90] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86, 1997.

[91] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3. USENIX Association, 2010.

[92] Amin Tootoonchian and Yashar Ganjali. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 Internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[93] Urs Hoelzle. Openflow @ google. http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, April 2012. Talk in the Open Networking Summit. Last visited June 2013.

[94] CFA UTEC. Live raizo. http://www.utec-tic.org/index.php?option=com_content&view=article&id=33&Itemid=118. [Online; accessed 20-October-2013].

[95] András Varga et al. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM2001)*, volume 9, page 185. sn, 2001.

[96] Steven Wallace. Network development and deployment initiative (nddi). http://www.canscouncil.net/presentations/CANS2011/NDDI-presentation-steven%20wallace.pdf, 2011. Talk presented in CANS: Chinese-American Networking Synposium.

[97] Kuang-Ching Wang. http://docs.projectfloodlight.org/display/floodlightcontroller/Architecture, 2012. [Online; accessed 10-July-2013].

[98] Jon Watson. Virtualbox: bits and bytes masquerading as machines. *Linux J.*, 2008(166), February 2008.

[99] Chris Welsh. Gns3 workbench 5.8. http://rednectar.net/gns3-workbench/. [Online; accessed 30-September-2013].

[100] Dan Williams, Hani Jamjoom, Zhefu Jiang, and Hakim Weatherspoon. Virtualwires for live migrating virtual networks acrosss clouds. Technical report, IBM Corporation, April 2013. IBM Researsh Report RC25378(WATI1304-079).

[101] Chen Xianqin, Wan Han, Wang Sumei, and Long Xiang. Seamless Virtual Machine Live Migration on Network Security Enhanced Hypervisor. In *the Proceedings of the 2nd IEEE International Conference on Broadband Network Multimedia Technology, IC-BNMT'09*, pages 847 –853, Oct. 2009.

[102] SLA Information Zone. Typical service level agreement contents. http://www.sla-zone.co.uk/, 2007.

Appendix A

Firewall paths and CSP constraints

Table A.1 shows the firewall paths before and after migration that has to be traversed from one zone to reach the migrating VM in the destination location.

In order to verify access control preservation, we first encode each firewall configuration before and after migration, for each scenario, in Sugar syntax. Then, we generate the needed CSP constraints for being solved with Sugar [88]. Table A.2 depicts a subset of the generated constraints.

Table A.1: *Firewall Paths Before and After Migration*

Before Mig.	After Mig.
Affected Paths	
$vF1$	$vF1 \odot sF1 \odot sF2 \odot vF3$
$vF1 \odot sF1$	$vF3 \odot sF2$
$vF1 \odot sF1 \odot vF2$	$vF3 \odot sF2 \odot sF1 \odot vF2$
$vF1 \odot sF1 \odot sF2 \odot vF3$	$vF3$
$vF1 \odot sF1 \odot sF2 \odot vF4$	$vF3 \odot sF2 \odot vF4$
Non Affected Paths	
$vF2$	$vF2$
$vF4$	$vF4$
$vF2 \odot sF1$	$vF2 \odot sF1$
$vF4 \odot sF2$	$vF4 \odot sF2$

Table A.2: *A Subset of the Generated CSP Constraints*

Firewall Paths	CSP Constraints
$vF1$	$\mathcal{C}_1 = (vF1^b \wedge p_{ij} \wedge \neg p) \wedge \neg(vF1^a \wedge p_{ij})$
	$\mathcal{C}_2 = (vF1^a \wedge p_{ij}) \wedge \neg(vF1^b \wedge p_{ij} \wedge \neg p)$
$vF1 \odot sF1$	$\mathcal{C}_3 = (vF1^b \wedge sF1^b \wedge sF2^b \wedge vF3^b \wedge p_{ij} \wedge \neg p) \wedge \neg(vF1^a \wedge sF1^a \wedge sF2^a \wedge vF3^a \wedge p_{ij} \wedge \neg p)$
$\odot sF2 \odot vF3$	$\mathcal{C}_4 = (vF1^a \wedge sF1^a \wedge sF2^a \wedge vF3^a \wedge p_{ij} \wedge \neg p) \wedge \neg(vF1^b \wedge sF1^b \wedge sF2^b \wedge vF3^b \wedge p_{ij} \wedge \neg p)$
	$\mathcal{C}_5 = (vF1^b \wedge p_{kl} \wedge p) \wedge \neg(vF1^a \wedge sF1^a \wedge sF2^a \wedge vF3^a \wedge p_{ij} \wedge p)$
	$\mathcal{C}_6 = (vF1^a \wedge sF1^a \wedge sF2^a \wedge vF3^a \wedge p_{ij} \wedge p) \wedge \neg(vF1^b \wedge p_{kl} \wedge p)$
$vF1 \odot sF1$	$\mathcal{C}_7 = (vF1^b \wedge sF1^b \wedge p_{ij} \wedge \neg p) \wedge \neg(vF1^a \wedge sF1^a \wedge p_{ij})$
	$\mathcal{C}_8 = (vF1^a \wedge sF1^a \wedge p_{ij}) \wedge \neg(vF1^b \wedge sF1^b \wedge p_{ij} \wedge \neg p)$
$vF3 \odot sF2$	$\mathcal{C}_9 = (vF3^b \wedge sF2^b \wedge p_{ij} \wedge \neg p) \wedge \neg(vF3^a \wedge sF2^a \wedge p_{ij} \wedge \neg p)$
	$\mathcal{C}_{10} = (vF3^a \wedge sF2^a \wedge p_{ij} \wedge \neg p) \wedge \neg(vF3^b \wedge sF2^b \wedge p_{ij} \wedge \neg p)$
	$\mathcal{C}_{11} = (vF1^b \wedge sF1^b \wedge p_{kl} \wedge p) \wedge \neg(vF3^a \wedge sF2^a \wedge p_{ij} \wedge p)$
	$\mathcal{C}_{12} = (vF3^a \wedge sF2^a \wedge p_{ij} \wedge p) \wedge \neg(vF1^b \wedge sF1^b \wedge p_{kl} \wedge p)$