



POLITECNICO DI TORINO

Master's Degree in Computer engineering

Master Thesis

Towards optimal orchestration of security controls in software networks

Supervisor

Prof. Cataldo Basile
Prof. Antonio Lioy
Dr. Fulvio Valenza

Candidate

Igor FERRETTI

ACADEMIC YEAR 2019-2020

*To my father and
To my mother*

Summary

In the last few years we saw how far technology succeeded, especially in the IT world. New paradigms as SDN and NFV have started to overthrow the traditional way to do networking and this gives the possibility to grow up to all the new computation environment, especially the ones regarding the “Cloud”.

Electronic devices are increasingly widespread and used by anyone, from all generations, who are always connected to Internet exchanging a big amount of data. They are so helpful to stay in contact with family and friends often far from us, but also to work remotely, especially in these few past months we saw the real potential of these devices.

New paradigms were born around these technologies as *Software Defined Networking* (SDN) and *Network Function Virtualization* (NFV) which have their natural application in data centers, where the “Cloud computing” has started to gain more and more relevance. The management cost of the network infrastructure decreases using the combination of both technologies, which carry features such as flexibility and resiliency. Those paradigms have some limitations, though, for instance, they do not guarantee any performance.

The continuous expansion of those new technologies and connected smart-devices lead to new vulnerabilities to be exploited by cyber attacks. The security request in cloud environment has begun to be a critical issue to be faced by Cloud Providers, which find in SDN and NFV technologies the solutions they are looking for. Customers are not only interested in security matters, but also they want to get the best quality for the services they pay for, this turns in high performance requirements, in terms of latency and bandwidth that network devices need to be achieved, without lost in security. Usually high level protection and high level performance are conflicts requirements, which get the network providers to choose service quality over security functions for business reasons.

Management of security controls still suffer the lack of dynamism. This is due to manual operations and device configurations that the network administrator has to face every time a change happens in the cloud infrastructure.

In view of the highlighted problems, this thesis gives its contribution in the design and in the implementation of a framework which aims to provide an *automatic policy-driven security* system. The main goal is to support the network administrator in the definition of security policies, which are customizable for each users, analysing anomalies and conflicts to avoid misconfigurations. The outcome of the tool is a distribution of the filtering rules in such a way the overall infrastructure gains the best performance without any leakage in security protection.

The major contribution provided by this thesis has been the modeling of the security policy and the cloud infrastructure, represented as XSD schemas and the definition of the optimization model. The optimization problem has been modeled as a binary programming model, which, starting from the traffic pattern analysis, understands what are the filtering points crossed by specific packets. The model allows the choice of the filtering control to be configured to achieve the better performance, minimized delay and maximized bandwidth.

Performance tests have been designed and executed to evaluate scalability features of each network node based on number of enforced rules. The results of these tests have been used to determine the coefficients and weights of each filtering point that will be used in the optimization model.

The problem that came out is a MaxSMT problem, which is an NP-complete problem. For this reason the implementation was done through the integration of the *z3 Prover* inside the framework, which is the standard *de facto* between SAT solvers. It provides operations of “pre-processing” and “cube&conquer”, associated to heuristics, to discard suboptimal solutions to avoid the computational time to explode.

The effectiveness and the validation of the optimization model has been proved by the implementation of performance tests. They show great improvements for what regards latency reduction, larger bandwidth and CPU consumption if compared.

The testing phase highlights that the goal defined at the beginning have been successfully achieved.

The results have highlighted the fact that the distribution can take advantage by the resources already present in the cloud infrastructure, like the built-in filtering modules of OpenStack, instead of instantiating new ones. This saves hardware resources and improves performances.

Acknowledgements

This thesis summarizes my university journey, which gives me the possibility to improve my background knowledge and also to seek goals accordingly to my personal interests.

First and foremost, I would like to acknowledge my thesis' supervisors. I thank the Professor Basile and Professor Liroy who believed in me and gave me the possibility to work on research matters, and I thank the Professor Valenza for his incessant help during the thesis work and for his advice, not only regarding technical issues, but also concerning his experience.

The most important acknowledge goes to my parents that undoubtedly always supported me during these five years, with their sacrifices they allowed me to study and to make easier any problems I faced. They have always been there if I needed help and they always listened to me talking about things they do not understand and giving reasonable meanings to my words.

Then I want to acknowledge all my friends, no one excluded, who closer, who further, but anyone gives his contribution to my achievements, unfortunately I cannot mention them one by one. I want to thank my room mates Dario e Federico. Our friendship became stronger during the first years of university.

As well Alessandro, Riccardo and Andrea who make these years easy and funny, with them I shared a lot of nice moments of my university life, including my thesis work on which Andrea gave his contribution.

This work means the end of this unique five-year path which is an experience I would suggest to anyone. Hopefully what I achieved will open new possibilities to improve myself, because I learned that *the moment you give up your dreams, you are lost*.

Contents

1	Introduction	10
2	Background	13
2.1	Computing Virtualization and Cloud Computing	13
2.1.1	Computing Virtualization	13
2.1.2	Cloud Computing	15
2.1.3	Cloud toolkits	17
2.2	Cloud Security and Security as a Service	18
2.2.1	Cloud Security	18
2.2.2	Security as a Service (SECaaS)	18
2.3	Networking in a Virtualized Environment	19
2.3.1	Network Virtualization	19
2.4	Filtering Policy	21
2.4.1	Firewall	21
2.4.2	Firewall Policy Anomalies	21
2.4.3	Optimization of Packet Matching	22
3	State of the art	23
3.1	Software Defined Networking and Network Function Virtualization	23
3.1.1	Software Defined Networking (SDN)	23
3.1.2	Network Function Virtualization - (NFV)	24
3.2	Network Security Function and Interface to NSF	27
3.2.1	Network Security Function	27
3.2.2	Interface to NSF (I2NSF)	27
3.3	Network Security Policies	28
3.3.1	Network Security Policies (NSP)	28
3.3.2	Policy Analysis, Conflicts and Transformation	29
4	Goals	33
4.1	Goals definition	33
4.2	Use-case definition	34

5	Infrastructure analysis	36
5.1	Cloud infrastructure	36
5.1.1	OpenStack introduction	36
5.1.2	Filtering points	36
5.1.3	Packet flows	37
5.2	Performance analysis	38
5.2.1	Iptables	40
5.2.2	Open Virtual Switch	43
5.2.3	Analysis conclusion	46
6	Solution design	47
6.1	High level design	47
6.2	Workflow	49
6.3	Policy Services	50
6.4	Path model design	51
6.4.1	Tree structure	51
6.4.2	Forwarding models	52
6.4.3	UML representation	52
6.5	Optimal distribution model design	54
6.5.1	Propositional equation	54
6.5.2	Hard constraints	55
6.5.3	Integer equation	56
7	Implementation	57
7.1	Policy conflict analysis Implementation	57
7.1.1	Library introduction	57
7.1.2	Policy - XSD schema	58
7.2	Path finder Implementation	61
7.2.1	Graph traversal	61
7.2.2	Depth-first search	61
7.2.3	Breadth-first search	62
7.2.4	Pruning approach	62
7.2.5	Topology Graph - XSD schema	65
7.3	z3 Prover	69
7.3.1	Introduction to z3	69
7.3.2	SAT problem	69
7.3.3	SMT problem	70
7.3.4	MaxSMT problem	70
7.4	Optimization Development	72
7.4.1	Model implementation	72
7.4.2	Binary equation	72
7.4.3	Hard constraints computing	73
7.4.4	Cost function	73

8	Testing	74
8.1	Scalability test	74
8.1.1	Entities test	75
8.1.2	Infrastructure test	76
8.1.3	Rules test	78
8.2	Comparison test	79
8.3	Use-case test	81
8.4	Test conclusions	82
9	Conclusion and Future work	83
9.1	Conclusion	83
9.2	Future work	84
9.2.1	VIM and lightweight virtualization	84
9.2.2	Different metric analysis	85
9.2.3	New hard and soft constrains	85
A	Programmer manual	86
A.1	Performance tool	86
A.1.1	Tool architecture	87
A.1.2	Modify the tool	89
A.2	Prover tool	89
A.2.1	Tool architecture	89
A.2.2	Modify the tool	91
A.3	Optimal Distribution tool	91
A.3.1	Tool architecture	91
A.3.2	Modify the tool	94
B	Path models	95
	Bibliography	98

Chapter 1

Introduction

In the last few years we saw how far technology has succeeded, especially in the IT world. New paradigms as SDN and NFV have started to overthrow the traditional way to do networking and this gives the possibility to grow up to all the new computation environment, especially the ones regarding the “Cloud”.

All of us use a huge number of devices which are always connected to the network and they steadily send great amount of data around Internet. Generally these smart-devices are well integrated with the paradigms previous mentioned. Keywords as Cloud computing has been used to mean technological progress and future hi-tech. They are so helpful to stay in contact with family and friends often far from us, but also work remotely, especially in these few past months we saw the real potential of these devices.

Alongside the continuous expansion of these connected devices, it grows the areas to be exploited which could be victim of cyber attacks due to new vulnerabilities introduced. NIST statistics [1] show that in last three years the discovered vulnerabilities grow from 14,086 in 2017, to 17,308 in 2018 until they reach 20,362 in 2019 and this trend will not stop.

For those reasons a lot of security issues rise each day, they are not coming only from big enterprises, but also from the common users, who start to question themselves on the security of their own data and the protection of their privacy.

Pre-existing infrastructures suffer the growing demand of bigger bandwidth and high-speed connections for all the “Real Time” applications, which puts under pressure the network devices. The Network Service Providers prefer to get the best quality of their services to their paying customers and this leads to a less care on the network security to save hardware and software resources to spend on own services.

The amount of data and information produced by smart-devices have become huge and they cannot be handled and stored by companies, which have started to move into cloud solutions. For this purpose nowadays the cyber security for those infrastructures is a daily topic to be studied and a lot of works is being done in this direction to make the cloud environment completely protected. The justification, to spend efforts by companies and by the Research world, it is to seek in the impact of a cyber attack on the global economy, for example the “Indian Banks data breach” in 2016 3.2 millions of credit cards were stolen.

The annual study examination of the financial impact, by IBM [2] on data breaches of organization. It highlights the cost of data breach has risen 12% over the last past five years and now it costs \$3.92 million on average. The picture depicted below (Figure 1.1) shows the trend of cost damages to organizations from 2005 to 2019.

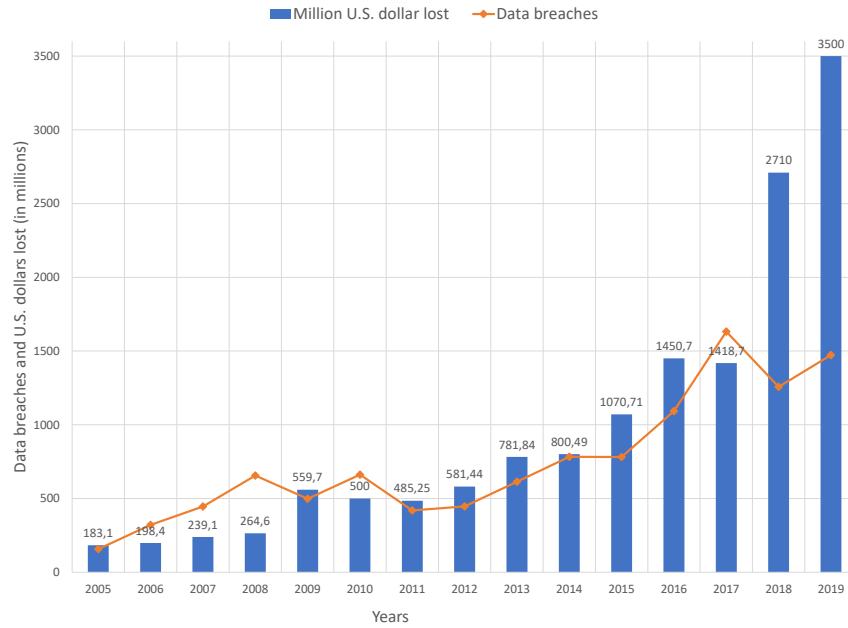


Figure 1.1. Data breaches and

These protection leakages are fed by networking administrators that have to face security issues without strong tools and suitable systems, which lead in discrepancies in the security policy definition that brings misconfigurations and incorrect operations.

Management of security controls still suffer the lack of dynamism in management functions and security controls. This is due to manual operations and device configurations that the network administrator has to face every time a change happens in the cloud infrastructure. In a cloud environment creating new instances and moving them from a server to another it is an ordinary operation. Due to the frequency of changes in software network, handling policy migration could be an impossible manual task. In addition, the manual configuration of security devices is prone to errors.

Enforcement of security requirements could limit the power and effectiveness of software networks, therefore, it is an important research goal supporting security administrators when enforcing security policies in software networks.

Current technologies as Software Defined Networking and Network Function Virtualization are already used to provide security, but these technologies are exploited to provide functionalities in the same way that are provided in a traditional network. For this reason further activities can be carried out regarding security management and resource optimization.

The main objective of this thesis is to extend elasticity and easy management exploited in the networking also to security controls. The aim of the project is to develop a framework which gets support to the security administrator during the policy definition and the infrastructure management. Moreover it is focused on the optimization of security policies thorough the conflict analysis, which finds anomalies, and the optimization of resources utilization to get better performance as expressed as latency, bandwidth and CPU consumption.

In the workflow entailed by the tool, high-level security requirements are specified by the security administrator. However, the actual device configuration is automatically performed by the tool, hence avoiding human errors and misconfigurations. Any topology change triggers an automatic reconfiguration of all filtering controls, which are building blocks of firewall, with rules that enforce the desired security requirements.

The main contribution of this thesis is the optimized distribution model which, starting from the traffic pattern analysis, understands what are the filtering points crossed by specific packets. The model allows the choice of the filtering control to configure to achieve the better performance, minimized delay and maximized bandwidth. This gets a first optimization goal that is to reduce the number of rules in each filtering point, which provides a second optimization result to get a better use of the available resources.

The framework takes advantage of the virtualization to optimize the location of security rules in distinct point over the infrastructure providing dynamism and flexibility in security controls as well as they are provided in the networking.

A *Proof of Concept* is developed as support for this thesis work to evaluate the system previous defined and to confirm the feasibility of it. This objective is achieved through the analysis of a cloud infrastructure (e.g. OpenStack, Kubernetes) that relies on SDN and NFV technologies, which are used to perform an optimal distribution. The optimization model is proved by the implementation of tests for performance and validation of it, which highlight excellent scalability features.

From this thesis work it comes out that the same networking flexibility can be reached also for security controls, without losing in performance if compared to physical appliances. Moreover, the automation of security and management operations get to better usage of resources and to an increase of the security level provided.

The thesis is structured in the following chapters and topics.

- Chapter 1 briefly introduces the problems and the idea of this thesis;
- Chapter 2 describes all the background knowledge studied to develop this thesis. It starts from the virtualization and cloud computing, to talk about virtualized networking and firewalls;
- Chapter 3 presents the new networking paradigms and the network security policies principles. The former topic explains *Software Defined Networking* (SDN) and *Network Function Virtualization* (NFV) architectures and what are the advantages of these technologies, the latter is about security analysis and conflicts;
- Chapter 4 defines the thesis' goal, moreover it provides a use-case definition to understand the problem to be solved;
- Chapter 5 analyses the infrastructure on which the thesis is developed and it performs performance tests on it;
- Chapter 6 provides a complete description of the solution design and the system workflow. It defines the proposed model for the optimization and distribution;
- Chapter 7 shows what choices were taken to implement the security system and what libraries were exploited to achieve the tasks;
- Chapter 8 presents all the tests to validate and to evaluate the solution proposed;
- Chapter 9 analyses the results obtained from the performance tests and it summarizes the goals achieved. The chapter exposes additional research direction that may be followed to extend and to improve the system functionalities;
- Appendix A is the manual for future developers which will maintain and develop additional functions to understand the code structure and how the different modules interact between them;
- Appendix B presents forwarding models used to design the distribution model.

Chapter 2

Background

This chapter describes the background on which this thesis has been developed, especially it will introduce the cloud computing paradigm and its relation with cyber security.

2.1 Computing Virtualization and Cloud Computing

2.1.1 Computing Virtualization

At first, servers have been placed across the whole company and then the enterprise moved all the hardware in data centers. The consequence is to get an easier management and a better data preservation. Only after this shift, enterprises noticed that there were a lot of servers and mostly of them were in idle state, but at the same time they could not consolidate multiple applications on a single server, because of the *One Application per Server* rule.

The “One Application per Server” rule means that each server could run only one application, because it has strict dependencies due to shared libraries or kernel version that may cannot be compatible with another one, which has different requirements.

This is due to the incapability of Operating Systems (OSs) to provide:

- configuration components full isolation: this is about kernel versions, OS’ patches and libraries;
- temporal isolation: this refers to the performance predictability, what if an application A could affect the performance of an application B in term of CPU load or device ownership?;
- space isolation: this means what if an application crashes, it may compromise other applications running on the same physical system;
- security isolation: this refers to internal communication, an application A may contact another application B without passing through any firewall.

Those dependencies lead a company to have a lot of servers, mostly under used, with a low workload and, obviously, it is a losing of money and resources. In this contest the *Computing Virtualization* (Figure 2.1) is born as an easy way to share hardware resources (CPU, Memory, I/O devices) between different OSs.

Computing Virtualization is not a new technology, it was used in the past for other reasons, as to share expensive hardware mainframes, but with the diffusion of mini and personal PCs that are cheaper, this approach fade out. Nowadays hardware virtualization has a new purpose and it is focused on the reliability and consolidation of business servers.

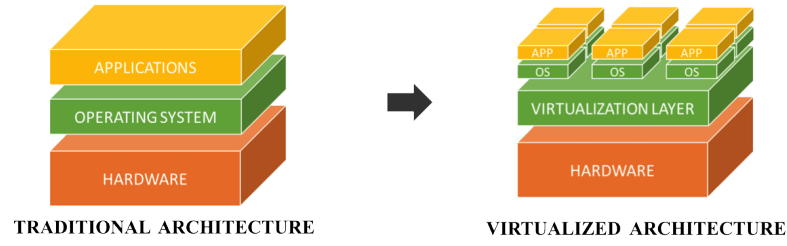


Figure 2.1. From traditional architecture to a computing virtualized architecture solutions.

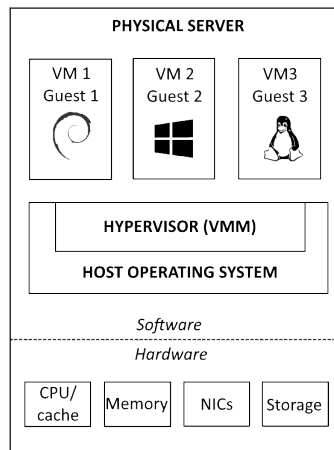
The main advantages of virtualization are the following.

- **Isolation:** several services could run in different isolated OSs that are deployed in the same physical host. Each Virtual Machine (VM) could have assigned its own CPUs with an improved degree of isolation.
- **Consolidation:** the same physical machine can run different OSs at the same time, saving hardware resources and reducing the energy consumption exploiting multi-core CPUs.
- **Flexibility and agility:** the host has the complete control on the VMs running on it, so it has the possibility to start, to stop and to restart instances. Moreover the *VM migration* became a crucial point to create a dynamic virtual environment, adding the possibility to duplicate VM instances to handle peak workload and make a disaster recovery easier to be implemented. This feature is the one which is more interesting in this virtual world, because it provides the creation of *autonomic* systems, which can automatic react to changes and difference situations.
- **Cost saving:** the virtualized layer provides all the elements to create dynamically the requested features for a virtual server, so it is no more significant have a physical server with a specific set of hardware elements. This leads company to buy huge amount of general and equivalent servers called Common Off The Shelf (COTS) at a low price and aggregates them on in a single data center.

At the same time this technology has some limitations as:

- overhead, each VM has a different OS running on it and it is isolated from the other OSs, hence more resources are required in term of memory (RAM) and disk;
- application, each Application must have his Operating System, they could not share underlying layers also if they required the same OS and kernel version;
- heterogeneous hardware, a few applications could have some requirements for special hardware and it is an hard challenge to provide them with heterogeneous devices.

In Computing Virtualization architecture there are four key elements:



- **host OS,** it is the Operating System running on the physical machine, it handles the hardware and the I/O devices;
- **Virtual Machine (VM),** it is the emulation of a physical machine running both OS and applications;
- **guest OS,** it is the OS running on the VM and it is in charge of hosts applications, this is like the Host OS but it runs on a virtual environment;
- **hypervisor (VMM),** it is the software running on the Host OS that is in charge to virtualize the environment, hardware and devices.

Moreover, a virtualized environment could be set up in different ways, because there are three distinct types of virtualization.

The first one is *Dynamic Binary Translation (DBT)*, it is the oldest, developed by VMware in the latest 90'. This technique allows to run the Guest OS unmodified on the Host OS, this means it does not know that it is running on a virtual environment.

The second one is called *Paravirtualization*, the Guest OS is modified to provide better performance for the virtualization execution. Not all the Operating Systems could follow this solution, but just the Linux/Unix based.

The last technique, came out in 2006, is *Hybrid Virtualization* and it was developed in order to sum up all the advantages of the previous techniques. It adds in the Host OS new primitives that could be exploited by the Hypervisor.

2.1.2 Cloud Computing

When Enterprises started to move from their virtualized servers to public Data Centers (DC) the *Cloud Computing* appeared, in particular it was born in 2006 with Amazon Web Service (AWS) that provides mostly computing and storage virtualization. In the Data Center there are a lot of equal COTS, so all the hardware is virtualized and there is no difference where a VMs is allocated, because the performance are the same.

There is an *Orchestrator* which is a middle-software whose purpose is to check and to control all the physical servers in the Data Center. It decides where a certain VM's instance will started, based on some metrics as the workload of the CPU, the amount of free memory or the number of VMs already present. This approach provides new features and advantages to company as:

- elasticity, where the Cloud Computing approach brings an easy way to scale up or down, because the Data Center has unlimited resources, as CPU but also storage, database or load balancing, to be provided on-demand if needed;
- always available, this virtualized environment supplies business continuity that means services are always on, anytime and anywhere, with the guarantees of 5-nines availability;
- separation of concerns, this is the main key of the success of this technology, because it allows the users to focus on their business and they do not need to know anything about the cloud infrastructure or any implementation details.

Companies began to use solution based on the Cloud Computing also for side effects that this technology has, for instance the management of software updates is instant and it could happen automatically, moreover customers has unlimited resources available and they could ask more CPUs, memory or storage if they need.

Cloud operator, besides, can provides *redundancy* and *data replication* to increased data reliability and other services as *Geo-localized* one in which they run client applications in the Data Center closer to the user to have better performance.

Service Models

At the beginning, Cloud Computing offers only a way to host VMs with applications of other companies, but many different services can exploit this environment using the paradigm of "*as a Service*" (Figure 2.2).

- Hardware as a Service (HaaS): in this case the Cloud provider gives hardware where VMs could run, a location where placed these physical servers and all the facilities to make all this stuff running as conditioning, power and network connectivity. The customer has full control of physical structures, so it is mostly like to have servers in the company. This solution is chosen if there are strictly security requirements, unless it is not so spread.

- **Infrastructure as a Service (IaaS):** it is the most used case, it provides the virtualized environment in which customers can create virtual machines, networks and storage. This solution is suitable for legacy applications and users that need the full control over the infrastructure. However, a lot of work needs to be done to install OSs, to update all the VMs and to handle all the security stuffs as firewalls.
- **Platform as a Service (PaaS):** this solution works for new services and application development. It offers a basic environment where new software can take advantage of it for deploy other services as Databases or Web Services. The tenant, which is interested in PaaS, does not need to know or to manage any underlying details that concerns Operating System, updates or network connectivity.
All the management, related to the service provisioning, like load balancing, number of instances as so on, is handled by the Cloud Provider. Here, it comes out the problem of the PaaS provider lock-in, this is due to a non standardization of the interfaces between different providers and this leads to a limited or impossible portability of the software developed.
- **Software as a Service (SaaS):** users use directly the software provided by the cloud infrastructure without worrying about anything of the implementation, virtualization or connectivity. It is easy and fast to give the service and to be used by customers.
This solution it is mostly suitable for standard, often used applications and this means that it is hard to create customized services and also not so interesting from the cloud provider point of view.

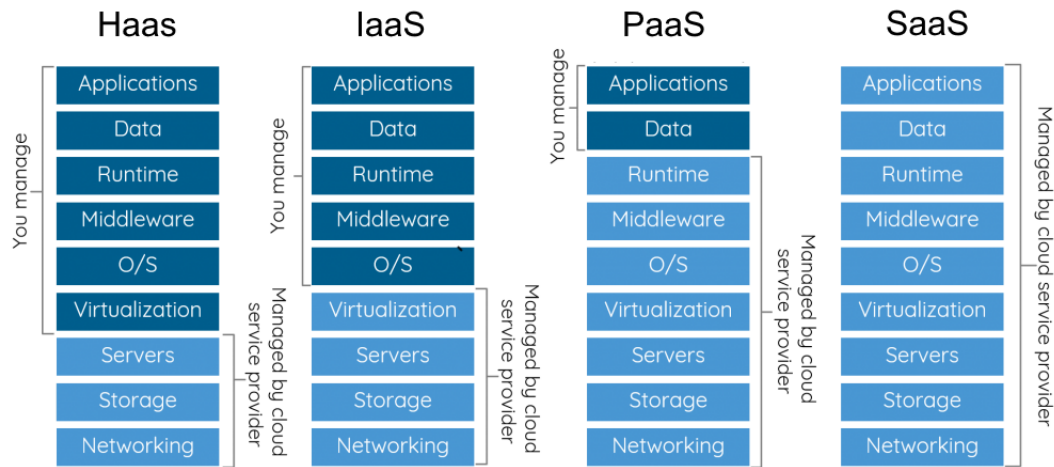


Figure 2.2. Cloud Service Models: Haas, IaaS, PaaS and SaaS.

Deployment Models

Cloud Computing grows up thanks to these types of *Service Model* and this brings at new type of *Deployment Models* in which virtual environments could be instantiated:

- **private**, the company has its own cloud infrastructure that must be managed and maintained by it self. This solution is the most expensive, but it provides the best solution for strictly security requirements;
- **dedicated hosting**, it is the HaaS solution mentioned in the Service Models in which the customer rents physical server from the cloud provider;

- public, it is like the dedicated hosting, but the customers rent virtual machines instead of physical servers. This deployment fits well the *pay-per-use* approach where the client pays for what he effectively used in terms of time or resources;
- hybrid, it is a mix of Public and Private Deployment, the customer has his own cloud but with limited resources, so in case of peak load he can move some of his VMs on the Public Cloud to reach better performance and to scaling up faster.

Limitation of Cloud

As we seen Cloud Computing spreads so fast for all the features that allows company to grow up faster and, at the same time, save a lot of money. This new technology is not a win-win solution, because it has different limitations.

- Network connection: the network connectivity is the most critical point here, because Cloud Computing requires not only a reliable Internet connection, because without it all the resources cannot be accessible outside, but also we need to consider some metrics as the latency and delay time. Indeed critical real-time applications not fit well this solution and they moved to the new conception of *Frog Computing*;
- Available features: it means that some Service Model has less flexibility than others as SaaS, in which the software offered by the provider has some characteristics, but it may not have some others that are more interesting. Paas has the same limitation, due to proprietary interfaces provided to host companies' applications that limit their portability;
- Data: users has no more a full control over the data, where it is generated and where it is stored. This has to face a lot of privacy and ownership problems that before Cloud Computing does not have to be handled.

2.1.3 Cloud toolkits

Virtualization and Cloud Computing have a great potential that is exploited in Data Centers where the number of physical server is high and there is a huge number of running VM instances. In this context it is hard to coordinate the location of VMs across the all infrastructure, because to choose the best physical server needs to take into account many aspects as resource consumption or free hardware, besides dependencies and requirements. Others operations related to the live-cycles (start/stop/migrate) of VMs could be done by hand, but it is not feasible in huge Data Centers.

Cloud providers are using a software framework, called *Cloud toolkit*, which gives a simple way to the administrator to control the entire infrastructure (e.g: OpenStack, Kubernetes, Apache CloudStack, etc). It provides an high level abstraction of the hardware resources and facilities, it is able to interacts with heterogeneous technologies as:

- the network resources, it is important that VMs can interact between them;
- the storage systems, it is used for store images of instances and the generated data;
- the hypervisor, that means it is able to handle the live cycles of the VMs.

Moreover, the Cloud Toolkit chooses where the orchestrator has to run. It decides the nodes that will hosts the operating system for the virtualization infrastructures and it exports North-bound API, and usually also a web interface, to interact with it.

2.2 Cloud Security and Security as a Service

2.2.1 Cloud Security

Nowadays all the companies realized that Security issues have a big role and they improved their infrastructures with a lot security devices, not only for the IT department, but also for general security. Cloud Computing is not exempted from all this, so *Cloud Security* was born.

Cloud Service Provider (CSP) must take into account how its infrastructure works and ensure security for it, not just taking in consideration all the virtual threads correlated to computing, but also CSP has to be protected by physical access.

The customer is not excluded from Cloud Security and it must sign a document, called *Service Level Agreement* (SLA), signed also by the CPS, where there are all the security matters that both parties must take care and who is responsible for. In this terms, the customer has a role in the cloud security, because it needs to ensure security for its application and data, but also make sure that CSP did all the possible security measure to provide and to protect the services.

Virtualization and Cloud systems have been changed the way “to do computing”, creating a new environment that it is a field where new cloud threads could came out. They are all the events that can cause some problem, that demotes *CIA* principles, which can alter sensitive data or delete some files. Possible list of threads are:

- Abuse and nefarious use of cloud computing
- Insecure interface and APIs
- Malicious insiders
- Data loss and leakage
- Service hijacking
- Account hijacking
- Risk profiling
- Identity theft

Cloud Service Provider must know the new threads to make reliable and secure its services and new vulnerabilities must be addressed by it to avoid weakness in the security architecture that may leads to loss the control of the network and the resource infrastructure. For instance, virtual layer must be protected to prevent any internal attack as *VM-to-VM* or *VM-to-VIM*, where who is attacking can gain the control of the resources or the entire cloud environment. *Sanitization* and *Anonymization* must be performed by CPS to ensure the complete destruction of user’s information and to avoid any data leakage.

2.2.2 Security as a Service (SECaaS)

Companies are moving into the cloud environment, but at the beginning, the general feeling was that virtual infrastructures was not safe as the private ones, this is due to losing the direct control over their facilities and data. Like all new technologies, also Cloud Computing has his *threads*, but if it is designated appropriately it causes no more than traditional environment does.

Cloud providers have started to offer another functionality “*as a Service*” for their customers called *Security as a Service* (SECaaS). This is a package of security functions that gives the possibility to the clients to no more concern about security issues and it moves the responsibility from the enterprise to the service provider.

It has the the same features of any others services provided in that way, so it is always available with a fast provisioning, it is a cost-saving solution because it fits well the *pay-per-use* approach and the provider is responsible instead of the enterprise.

Cloud provider offers different types of security services for its clients, which could be summed up into the following categories:

- web security
- e-mail security
- security assessments
- Intrusion Detection System
- encryption
- network security

2.3 Networking in a Virtualized Environment

2.3.1 Network Virtualization

Computing Virtualization gives the capability to run multiple VMs and containers on the same physical server and this introduced additional issues and requirements for the network communication. The first challenge is at Layer 2 (Ethernet connectivity) in which the VMs or containers must:

- exchange traffic between them and from/to the physical network, which means to have the possibility to reach, or to be reached by, external destinations;
- exchange traffic between them in the same physical host.

Additional requirements must be provided to achieve this communication pattern with Layer 3 connectivity:

- all the VMs/containers must have an IP address that needs to be assigned to them;
- all the physical network devices, as firewall, load balancing or NAT, should be provided as well to instances in the virtual environment.

The main idea is to give the same view from the physical network to the VMs and vice versa, so the virtual network must work as the physical one. There are three options, given the architecture depicted in Figure 2.3, to solve these requirements that have different characteristics.

The former is called *Host-based switching* that uses a software, called “virtual switch”, which behaves like a physical switch device. Usually it runs in the host OS or in the Hypervisor of the machine and for this reason it provides high bandwidth reducing the traffic flowing in the network. This solution introduces additional overhead to the virtualization, because some CPU cycles are spent to running network processes. In this case is not so clear who is in control of this switch and this part of network. It is a problem to understand which people have to configure it and monitoring it.

The latter is *Hairpin switching* that solves the problem of the additional overhead, because it uses the hardware already present. This means that the traffic exits the server, it goes to the physical network and arrives on the switch that puts back the packets to the same server. There are two problems related to this solution, one is that the traffic needs to pass on the physical link so the bandwidth is reduced, the other problem is that the physical switch must change his configuration to forward packets on the same interface where the traffic comes from.

The last solution is *NIC switching* and is the trade off between the others options. It solves the problem of “who is in charge of the edge network”, because the interfaces are under the network management, and the bandwidth is mostly like the host-based solution.

Nowadays the *Host-based switching* is the most spread solution for its simplicity and, in this way it easy to distinguish traffic coming from different VMs, because each virtual switch knows exactly which VM generates each frame. This is more hard to reach for the physical solutions, may be they need some way to tag the traffic before they send it back on the network.

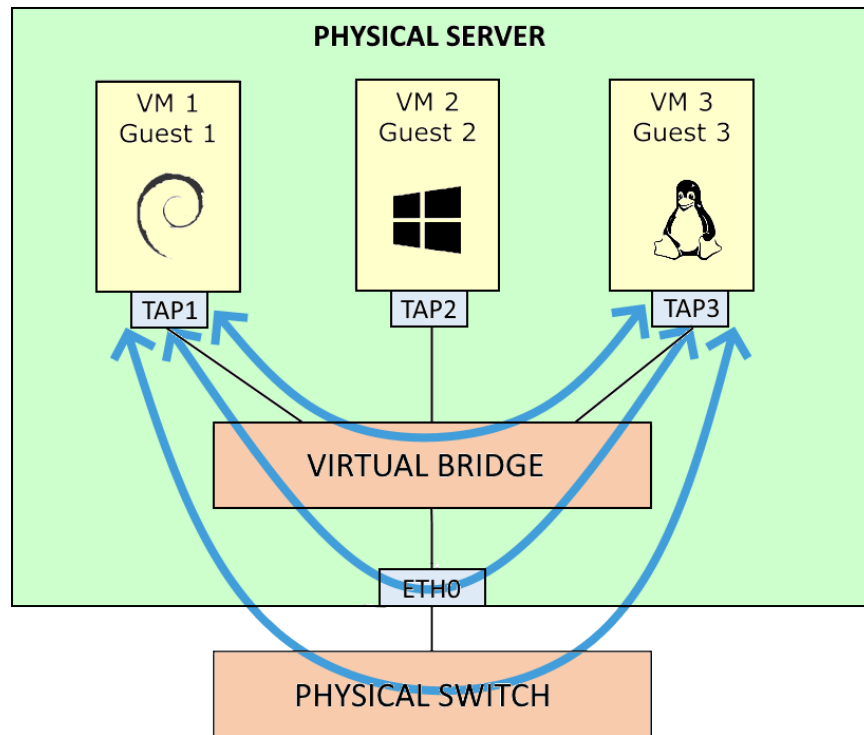


Figure 2.3. Virtual network architecture.

Virtual Networks has an impact on the networking in physical world, because level two connectivity must be provided also on a level 3 network that links all the servers, which introduces segmentation. Solutions based on VLAN-tag are used but there are more problems correlated to the L2 networks such as the limited number of VLAN IDs and the loss of scalability of the LANs.

The last problem, but not least, the *policy migration* is not available, this means that if a VM is migrated to another server that may be is on a different L3 network, all the policies correlated to this VM are not following this change.

The main solution to offer a transparent connectivity to the virtual environment is *tunnelling*, this is done using GRE protocol or VxLAN tagging. Both provide a L2 network over different IP subnets.

2.4 Filtering Policy

2.4.1 Firewall

In the last few years Internet communication became essential for the every day life, as at the same time network attacks and threats grow up. Therefore network security gained high attention by, not only researcher, but also by companies and enterprises. This make *Firewalls* a crucial security point inside the network.

A Firewall is a security network element deployed at the edge of a network that controls the traversal flows of packets in or out of the boundaries, based on a specific *Filtering Policy*. A Filtering policy is an ordered list of filtering rules which defines an action that must be performed on matching packets [6]. The filtering actions are either *accept*, which allow the packet to transit the firewall to or from the network, or to *deny*, which causes the packet to be dropped.

There are different types of Filtering Policy.

- Packet filtering, it makes a packet inspection at network level, usually it checks IP headers and transport headers [7], in particular five header fields of the packet are analyzed which are destination and source IP address, protocol ports and protocol type. It is stateless, that means each packet is processed as it is, without any additional information.
- Stateful filtering [7], it improves the previous packet filtering functionality by maintaining an history of the connection states and sessions for the transport or application level. It can distinguish new connections from those already open using a state table for open connections and packets matching one row in the tables are passed without any further control. This type of filtering has an additional action called *reject* that has the same effect of drop action, but it sends back an error message the the sending host.
- Application-level gateway, it is more powerful than stateless and stateful filtering, because it can read nested information inspecting the packet payload at application level [7]. It could be used as part of a firewall, usually performs also peer authentication and the rules are more fine-grained and simple than those of a packet filter.
- Content filtering, the last one is used to block traffic for some resources based on what the packet contains, which could be a text, photos or videos.

Mostly widespread filtering are the firewall stateless, which are the frontier defense for networks against attacks and unauthorized traffic [6].

2.4.2 Firewall Policy Anomalies

Firewall configuration is done manually by the network administrator, who usually has the support of some automatic tools with limited functionalities for the security management.

Misconfiguration is not a rare scenario, because it is easy to introduce human errors and this has a no negligible impact on the effectiveness of security controls.

As we mentioned before a firewall has an ordered set of filtering rules, a default action, in case no rules are matched and a resolution strategy. The latter is the element that solve some conflicts between the rules.

Firewall anomaly is defined as the presence of multiple rules in the filtering policy that match the same packet. This leads to have different types of anomaly as *Shadowing anomaly*, *Correlation anomaly*, *Generalization anomaly* and *Redundancy anomaly*, some of these are considered warnings potential conflicts or errors for specific conflicts.

It is not enough to have no conflicts in a policy, because especially in large companies there are more than one policy and one firewall. Usually different policies are deployed in different firewalls.

This could lead to *Inter-Firewall Anomaly* that is defined as the presence, on a network path, of two firewalls, the former called *upstream firewall* and the latter is the *downstream firewall*, that performed different actions on the same packet flows. Inter-Firewall Anomaly could be classified with the same anomaly types mentioned before, plus a new one called *Supriusness Anomaly*.

2.4.3 Optimization of Packet Matching

Nowadays network links reach high speed transmission and this brings with it the requirement of high speed network devices, security ones are not excluded from it.

Firewalls need to perform optimized algorithms for the packet classification that is a critical component for performance evaluation, because a packet is considered correctly handled if an action is performed on it. This is done by ordered sliding the list of the rules until the matching one or till the end with the default action, so the search is proportional to the length of the rule list $O(N)$, where N is the number of the rules [8].

Hardware-based solutions use specialized memory called Content Addressable Memory (CAM) or Ternary CAM (TCAM), because they take advantages of hardware parallelism to match more than one rule in parallel. This gets a high speed lookup, but those memory types are limited in the number of entries.

Other solution follows the *Aggregated Bit Vector* (ABV) approach which is able to perform n parallel and independent lookups on one dimension, or the *geometric algorithm*.

The delay, or *skew*, introduced by the middle-boxes, depends on the the searching time before a matching rule is found, so the ordering of these rules became important. Besides the lookup algorithms, *Dynamic Rule-Ordering Optimization technique* [9] improves firewalls performance. This new approach is made up of two layers, the upper layer contains the *active rules*, which means all the most frequently matching rules ordered on the matching rate base. The second layer has all the *inactive rules*, the rules that has no matching or performs less matching.

Chapter 3

State of the art

This chapter describes the state of art of new technologies as *SDN* and *NFV* which are used to design and to develop this thesis. Moreover the concept of a policy will be introduced within the conflicts analysis.

3.1 Software Defined Networking and Network Function Virtualization

3.1.1 Software Defined Networking (SDN)

Software Defined Networking (SDN) is a new paradigm in which the forwarding hardware is decoupled from control decisions [12]. In this architecture there is a logical centralized controller and multiple forwarding devices that are configured via an interface (e.g OpenFlow).

The former has a view of the whole network, it runs the control plane that is in charge of handle flows and forwarding rules, the latter are simple network devices (e.g switches) which manage the forwarding process of the packets.

Due to its decoupled nature, SDN is believed to be a new networking technology that simplifies today's network operation and management, also it enables networking innovations and new infrastructure design [13]. An SDN instance has three main parts as it is shown in Figure 3.1:

- application, it takes advantage of the decoupled control and data plane to gain some specific goals as security rules or network measurement. It communicates with the control plane through the northbound interface used by the applications to give it commands;
- control plane, it is the entity called SND controller which manipulates forwarding devices independently from the protocol adopted. It uses its southbound to talk with the data plane and to push flow-rules on physical appliance;
- data plane, this layer is connected to the controller via shared protocol (OpenFlow is the most used) and it handles the packets based on the rules and configuration given by the control plane.

This programmability, given by separation between controller and forwarding devices, allows to have some features that the traditional networking has been hard to reach:

- flexibility: the user could change flow rules and device configuration on demand;
- availability: the network topology could change, some middle-boxes could be add or be removed without loss of connectivity;

- programmability: it easy to program the network, because the user needs to focus only on the controller and not on the forwarding functions;
- open interfaces: SDN devices use open source interface, both for north and south bounds, to communicate to the controller;
- centralized control: it is useful to have one point that configures all the devices, because it has a wide-view of the network, which simplifies trouble shooting operations.

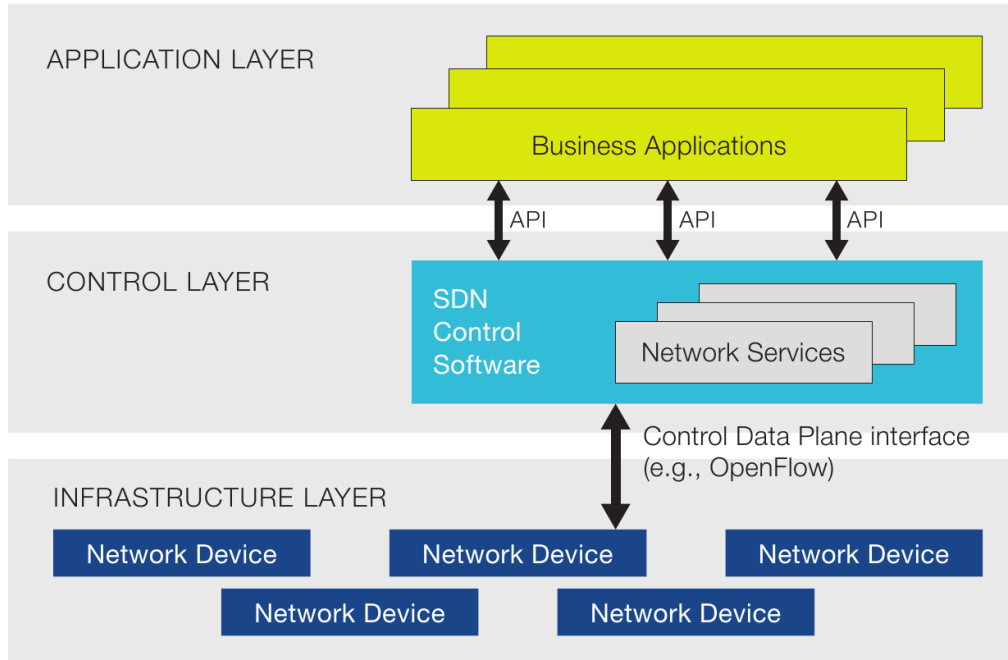


Figure 3.1. From network functions hardware-based to software-based NFV solutions [14].

OpenFlow

The main protocol used in SDN network architecture is OpenFlow [15] standardized by Open Networking Foundation. OpenFlow is a protocol that exposes an API used for the communication between the controller and switches. It enables to control the network-wide forwarding behaviour under an SDN concept [13]. OpenFlow switch can act as a router, a bridge, a NAT, a firewall, or exhibit similar functions that depend on packet-handling rules [13]. OpenFlow protocol allows the switches to setup flows not based only on IP addresses, protocol types or port, but the controller could push some forwarding rules following high-level logic as the forwarding based on the input port of the switch where the traffic is coming from.

SDN paradigm could be more powerful if it is used combined with NFV (Network Function Virtualization), because both these technologies gave more flexibility and scalability that the current hardware-centric Internet infrastructure suffers from several shortcomings [13].

3.1.2 Network Function Virtualization - (NFV)

Service provision, within the telecommunications industry, has usually been based on network operators deploying physical proprietary devices and equipment for each function that is part of a given service. In addition, network function components have strict chaining and/or ordering that must be followed in the network topology and in the localization of service elements.

These specifications, coupled with requirements for stability, availability, high quality and stringent protocol adherence, have led to long product cycles, very low service agility and heavy dependence on specialized hardware. NFV [17], has been proposed as a way to address these challenges by leveraging virtualization technology to offer a new way to design, to deploy and to manage networking services. The main idea of Network Function Virtualization is the decoupling of physical network equipment from the functions that run on them [16]. This leads to a consolidation of network infrastructure due to the possibility for Telecommunications Service Provider (TSP) to locate them into data centers or at the edge of the network, next to the user's premises.

The evolution of network elements is no more reliant on both hardware and software development, but they have different and independent time progress timelines and maintenance. NFV paradigm exploits the decoupling software from hardware to gain more flexibility and agility, which means TSPs could open up their network capabilities and network controls to the users and other functionalities such as the ability to deploy or to support new network services faster and cheaper as well as to provide better service agility [16]. The main benefits of NFV architecture are the following:

- flexibility, this architecture can scale (up or down) easily in number of instances or resources;
- portability, VMs can run on different physical servers due to the decoupling of software from hardware;
- resiliency and stability, VMs can be deployed in one or more instances at the same time;
- orchestration, the NFV infrastructure can be managed by an hypervisor (MANO);
- energy efficiency, the TSPs can use efficiently this technology to reduce energy consumption;
- service continuity and availability, the network services are no more related to a physical devices, in case of any hardware problems the network functions can run on a different physical infrastructure;
- security, the functions providing security can be deployed as NFV service.

According to ETSI, the NFV Architecture is composed of three key elements: Network Function Virtualization Infrastructure (NFVI), VNFs and NFV MANO [16].

- NFV Infrastructure (NFVI): it is made up of both hardware and software resources that set up the environment in which VNFs are deployed. Commercial-off-the-shelf (COTS) are the physical resources exploited for the computing, network stuffs (as link or nodes) for the connectivity and storage. The virtual resources are the abstraction of the hardware cited before and they are achieved using an hypervisors that provides a virtualization layer. In this way VMs can share the underlying hardware with a guaranteed isolation level.
- Virtual Network Functions (VNFs): this layer represents all the instances of the network functions running as software products. There are several objects, which are called Element Managers that are used to control and monitor all these instances.
- NFV Management and Orchestration (NFV MANO): it offers all the functionalities required for running, coordinating the NFV infrastructure and also managing all the related operations. It is made up of three subsystems deployed in one single set-up, the VIM (Virtualized Infrastructure Manager) manages the physical infrastructure, the NFV Orchestrator (NFVO) provides all the functions for the file-cycle of VNFs and the VNF Manager (VNFM) is in charge of the VNFs instances.

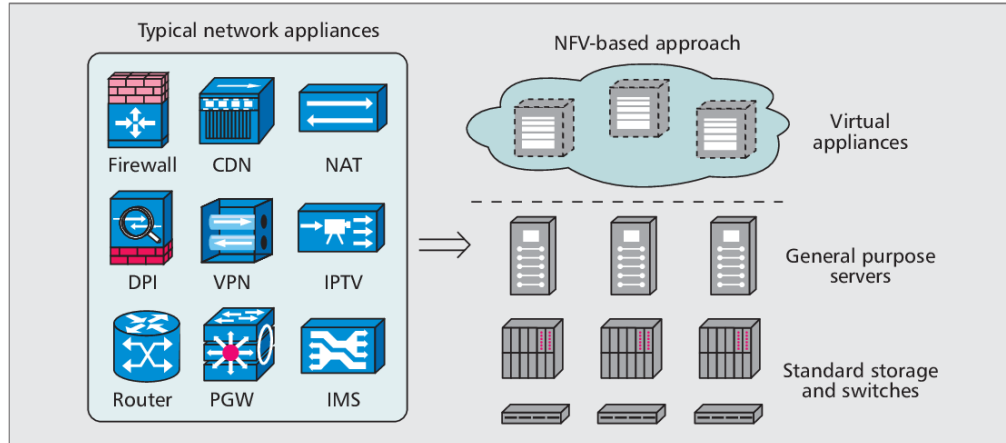


Figure 3.2. From network functions hardware-based to software-based NFV solutions [17].

Virtual Network Function

A Network Function (NF) is the implementation of a function block deployed in a network infrastructure with well-defined interfaces and behaviour. Nowadays a Physical Network Function (PNF) runs on a network appliance [18].

Virtual Network Function (VNF) is the software implementation of a Network Function which is deployed on virtual resources. A single VNF could be made up of more than one component that could be run on a single or multiple Virtual Machines (VMs). VMs could be implemented not in a single place, but across different Data Center of the same TSP and the Network point of Presence (N-PoP) provide the information of where these VMs are located [19].

VNF features are agrees with the NFV characteristics such as:

- energy efficiency, this technology demonstrates its potential to reduce the energy consumption of network equipment, due to the possibility of turn off physical servers or devices during off-peak hours in which the workload is low;
- reliability, it represents the system capability not to be affected by unexpected or hostile situations. The challenge here is to make VNFs more or equal reliable then the traditional hardware-based functions;
- migration, it is referred to change the location of VNF instance depending on the actual necessity and this must be done without data loss or discontinuity service;
- performance, it is a crucial point of virtualized environment because traditional network functions run on specialized hardware-devices that provide a guaranteed efficiency. The software developed must follows all the feature of micro-service idea to provide the same level of quality, measured in terms of latency and throughput using efficient algorithms;
- security, since the virtualization technologies used in cloud environments are also applied in NFV to provide a virtualized system, all the challenges caused by these virtualization technologies are also faced by NFV [19]. The Security Expert Group (SEG) created by ETSI lists all possible problems and threads that VFN technology could be affected as multi-tenant isolation, secure boot or crash, user authentication, authorization and accounting. The list of potential security issues investigated by SEG could be found in the ETSI deliver “Network Functions Virtualization (NFV) Security” [20].

The notion of Service Function Chain (SFC) came out with the spread of VNF technology, which means more than one network function can be interconnected. This is easily accessible due to the standard interfaces of VNF instances. SFCs are used by the network operators and the TSPs to offer custom services to users and, thanks to Cloud Computing techniques, they could re-designed applications and provide redundancy of the services.

3.2 Network Security Function and Interface to NSF

3.2.1 Network Security Function

Network Security Function (NSF) can be used to achieve security goals such as integrity, confidentiality and availability to protect a network system by detecting malicious traffic and/or reducing the impact of cyber attacks on the network system [21].

NSF can take advantage of NFV features, first it could support the automatic enforcement of security policies and second NSF instances could provide the capability of dynamic adaptation to network changes [22]. The TSPs, similarly to the provisioning of NVF services, allow the customers to consume network security services, which are developed by different vendors or they are open source technologies.

A Network Security Function could work on a different level of OSI-stack, inspecting the header or the payload of the packets. Some use cases are the following:

- packet filter, it is the lowest layer of a NSF;
- firewall, it is a security network element that controls the traversal flows of packets in or out the boundaries;
- Deep Packet Inspection (DPI), it is a way to monitor the packets flow on the network;
- DDoS mitigation, to avoid external or internal Distributed Denial of Service;
- proxy TLS
- Data Loss Prevention (DLP)

The challenge of I2NSF is to implement and to define a list of interfaces to be used to configure, to monitor and to control both NSF devices and instances, if the second is developed in a virtual environment.

3.2.2 Interface to NSF (I2NSF)

Interface to Network Security Function (I2NSF) is a Working Group (WG) of IETF, which is the primary mechanism for development of specifications and guidelines, many of which are intended to be standards or recommendations [23]. The purpose of I2NSF is to implement a framework and a set of interfaces for the interaction with Network Security Functions (NSFs). The I2NSF framework allows heterogeneous NSFs developed by different security solution vendors to be used in the Network Functions Virtualization (NFV) environment by exploiting the capabilities of such NSFs through I2NSF interfaces such as Customer-Facing Interface and NSF-Facing Interface [24].

Without standard interfaces to monitor and control the behaviour of NSFs, it has become really impossible for providers, who offers security services, to automate their infrastructure using different NSF implementation from multiple vendors [25]. I2NSF defines two different functional levels of interfaces, both for configure and control NSFs.

- The I2NSF Capability Layer: it handles all the control and monitoring operations at a functional implementation level. This means that all NSF controllers must implement a set of standard interfaces by which it can operate, invoke and monitor NSFs.

- The I2NSF Service Layer: it provides the definition of how clients security policies may be expressed to a security control [24] at high level. Then the NSF controller puts into effect its policies using the available capabilities provided by the I2NSF Capability Layer, which makes it to contact one or more NSF's instances through it self.

The customer can use both the former, in which he may leverage the controller, or the latter, in which the client interacts directly with one ore more NSF's to express its security policies.

3.3 Network Security Policies

3.3.1 Network Security Policies (NSP)

A *security policy enforcement point* is a network element that controls the packet flows across network segments based on a given *network security policy* [3]. These could be both host's element or middle-boxes as firewalls, VPN gateway or IPSec devices.

The Security Policy determinates the level of the packets protection on the network, each packet is usually identified by header fields of Level 3 and 4 of the ISO/OSI protocol stack, the so called *5-tuple*. it is made up of source and destination IP addresses, IP protocol (usually TCP and UDP) and at the end source and destination ports. The actual policy rule that matches the packets headers, decides the action to be performed on the packet.

There are two modes for the composition of a Network Security Policy for filtering packets.

- Access list, it is a ordered list of rules that are sequentially slided until a match is found with the current packet. The order is important and it leads to behave that a specific flow triggers the same rule. The matching rule determines what security action must be performed between:
 - bypass: that means the packet is allowed to be forwarded and it is transmitted as it is;
 - discard: this action drops the packet;
 - protect: this last option is actuated by IPSec devices and the task is securely transmit the packet on the network.
- Map list, it is a set of rules that determinates how the packets selected from the access list, must be protected. Each rule in the set has a priority and the mapping is based on the high priority, this leads to applying the security transformation on the traffic.

Firewalls performs filtering actions on packets as *bypass* which allows the traffic to enter or to leave the security network, or as *discard* that blocks the traffic and drops all the packet related to that flow.

IPSec [4] is a Security Protocol (Figure 3.3) used to provide a secure method to transmit packets on the open Internet. It could be implemented in two different sub protocols that could be combined, based on what kind of protection we want to reach:

- Authentication Header (AH), this protocol provides data origin authentication, integrity for connectionless traffic and, as an option, protection on reply attacks;
- Encapsulation Security Payload (ESP), this other implementation provides all the CIA paradigms, Confidentiality using cryptography, Integrity and Authentication of data origin. It also has the same option of AH protocol on anti replay function.

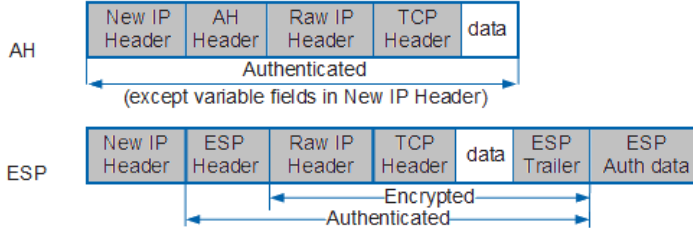


Figure 3.3. IPSec Protocol.

The Security Policy relies on what and where the traffic has to be protected, at the edge or at an intermediate point and, for both of them, IPSec offers two *operation modes* (Figure 3.4). The former is *Transport Mode* where the security protocol operates on source of the traffic and also on the destination entity, this allows to protect the end-to-end communication. This mode is going to work for ISO/OSI Layer upper then 3-level, so IP header is not changed. If we want to ensure a network-based or domain-based security, we need to adopt *Tunnel Mode* that is performed on intermediate devices called Security Gateways. Unlike the previous mode, this one changes all the headers, because the protection is applied on the entire packet, so the output is to have a new packet with IP source and destination address changed.

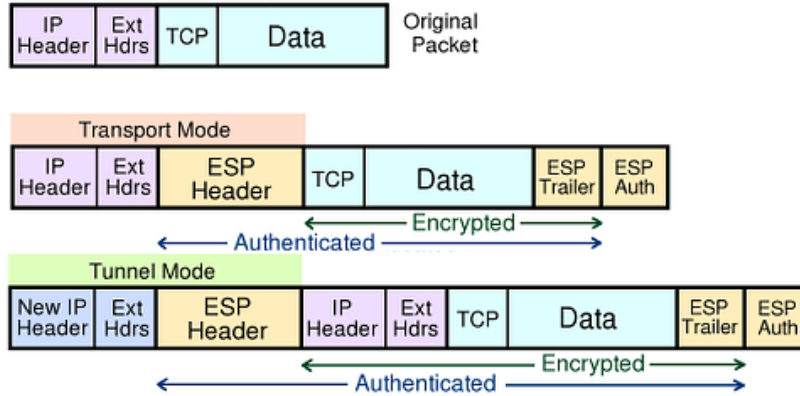


Figure 3.4. IPSec protocol: Transport and Tunnel Mode [5].

3.3.2 Policy Analysis, Conflicts and Transformation

Models of Rule Relations

A Network Security Policy is a set of rules with a specific order and in most of the cases the number of rules are huge. Therefore it becomes hard to understand if the rules have dependencies and how they are related. The relations between filtering rules are due to misconfiguration or maybe the security admin is aware of them. They could be summed up as follow [6]:

- exactly matching, if one rule R_1 for each field has an equal correspondence to another rule R_2 , they are exactly matching ($R_1 = R_2$);
- inclusively matching, if in rule R_1 , called *subset match*, every fields are a subset of a second *superset match* rule R_2 ($R_1 \subset R_2$);
- correlated rules, neither of the two rules is a subset or superset of the other one, but their fields space is partially overlapped and the rest of the fields are equal. ($R_1 \bowtie R_2$);

- disjoint rules, two rules (R_1 and R_2) are *completely disjoint* if every fields in the first rule is not equal, not a subset or superset to the corresponding field in the second rule ($R_1 \cap R_2 = \emptyset$). Eventually there is also the term *partially disjoint* that identify the condition when the R_1 has at least, one field that is equal, subset or superset to the corresponding field of the R_2 , but at the same time the first rule has at least one field that is not equal, subset or superset of the same field in the second rule.

Policy Analysis

Policy Analysis performs an inspection of the rules in a policy, or more than one, in order to detect anomalies, which can be threads that are evidence of a misconfiguration of the system and thus leads to security problems.[10] These conflicts may exist between just a policy, or between different policies and security devices, so they lead to the following classification:

- intra-policy anomalies, this class represent all the conflicts that arise between rules of the same policy;
- inter-policy anomalies, which are the anomalies that come out between different policies, but with the same capability;
- inter-function anomalies, it is the class that represent the anomalies between two function owned by different policies having different capabilities;
- inter-actor anomalies, which are the anomalies between two rules of different policies, but defined by different actors.

Policy Conflicts

Moreover, each anomaly class can have different rules conflicts, the first one looks into the Intra-Policy Anomalies:

- Shadowing anomaly (Figure 3.5(a)): a rule is called shadowed if every packet that could be matched by this rule, it is matched by another rules that is preceding or it has higher priority then the first rule. In others words two rules, R_1 and R_2 , which performs different action, make a shadowing anomaly when R_1 matches, at least, the same packets of R_2 (the condition clause of R_1 includes all the condition clause of R_2) and R_1 has a greater priority then R_2 or it is preceding the second rule.
- Correlation anomaly (Figure 3.5(b)): it occurs when there is some traffic that matches both R_1 and R_2 rules, which means the two rules are partially, and not completely, overlapped for some fields' values. They perform different filtering action on the matching condition, so the result action applied on the traffic depends on the ordering of the rules. This dependency leads to ambiguity in the definition of the security policy.
- Exception anomaly (Figure 3.5(c)): it is also called Generalization anomaly and it expresses the situation in which, R_1 and R_2 perform different actions, the first rule, with the higher priority, match all the packets of the second one that has lower priority, so this last one is a superset of the matching condition of R_1 . This is not an error, but it is used to allow or to exclude a particular pattern traffic, while the general one performs the opposite action.
- Redundancy anomaly (Figure 3.5(d)): it occurs when R_1 matches all the traffic that could be matched by R_2 and R_1 has a better priority then R_2 , but they perform the same action. This means that if R_2 is removed the behaviour of the security policy is unchanged.
- Irrelevance anomaly: it refers to a rule that cannot match any packets flows on the devices where it is enforced.

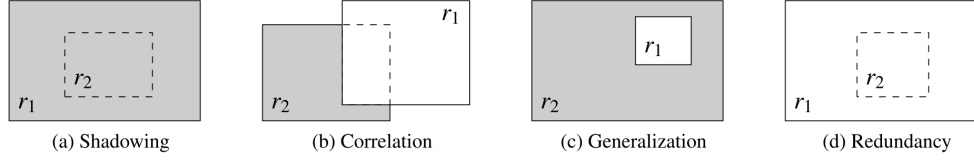


Figure 3.5. Geometrical representation of rules anomalies.

Inter-Policy Anomalies class has some of the previous conflicts as *Shadowing anomaly*, *Redundancy anomaly* and *Correlation anomaly*, which are similar in the definition and in the behaviour. There is another anomaly called *Spuriousness anomaly* that has seriously impact on the network security, because the traffic is not properly handled and it flows in the network. This occurs when the upstream security device, having a define set of filtering capability, allows some packets to be forwarded, but the downstream filtering device that has the same capabilities of the first one, blocks this traffic. Obviously the traffic will not reach the destination, but it could be dropped before in the upstream firewall, avoiding the unnecessary flows through the network.

Moreover, the *Policy Analysis* does not takes into account two particular cases. The first one is what is the behaviour of the system when no rules are matched, and the second case is what the system will do if conflicts rules match the same traffic [11].

It is necessary defines and chooses a policy action that is applied to all the packets when the traffic do not match any rules, this is called *default action* $d \in \mathcal{A}$. The security system, on the contrary, when more than one rule is matched, takes the decision according to the *Resolution Strategy* function that decides which action must be performed. At first instance all the matching rules are picked out and then the resolution strategy chooses the action that will be performed between the matching ones, so it could be represented in a formal way as: $\mathfrak{R} : 2^{\mathcal{R}} \rightarrow \mathcal{A}$ in which the resolution strategy function maps all the possible groups of rules into an action $a \in \mathcal{A}$.

Each Security Policy could define its own custom resolution strategy, it depends on the contexts of the policy deployment, commons functions are the following:

- First Matching Rule (FMR): the policy performs the first rule between the set of the matching ones;
- Allow Takes Precedence (ATP): if two matching rules have different actions, the one that allows the forwarding of the traffic is chosen;
- Deny Takes Precedence (DTP): that means, if two matching rules have different actions, the one that drops the packet is chosen;
- Most Specific Takes Precedence (MSTP): the resolution policy chooses the rule which has more matching fields;
- Least Specific Takes Precedence (LSTP): the resolution policy chooses the rule which has less matching fields, so it is more generic.

Policy Transformation

Usually Security Administrators specify policy at high level, maybe in a human readable syntax for debugging and they make the security policy more readable to be understandable. This set of rules needs to be translated into a format that real devices can read, accept and enforce.

This policy manipulation task needs the concept of *Semantics-Preserving Policy Morphism* that is a transformation of the policy representation, but it guarantees that it keeps the policy unchanged [11].

This translation of policy representation allows analyzing anomalies independently from the resolution chosen by the administrator, but leaving its semantics unchanged.

This is done following two steps:

- first, it is a process to translate the policy in a generic intermediate representation called *Canonical Form*;
- second, it is a process that takes advantage of the first step to translate the canonical form into any target resolution strategy.

The *Canonical Form* is a policy representation, useful for policy manipulation and based on the operation set to solve conflicts. It is formally equivalent to the original one, but it has more rules, this permits an easier processing, because all the possible combination of conflict cases are precomputed with the resolution strategy \mathfrak{R} . Now all the algorithms that work on the Canonical Form of a policy do not have to consider the original \mathfrak{R} , because all the differences between the resolution strategies are removed.

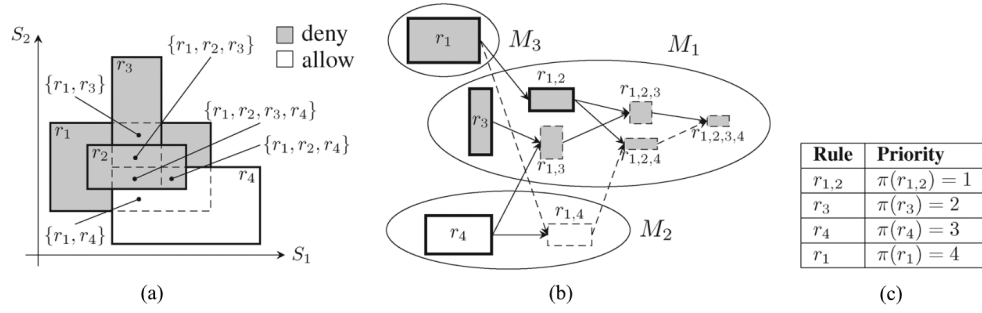


Figure 3.6. Graphical geometrical representation of the match function (a), the cover graph (b) and, finally, the exported rules (c).

Chapter 4

Goals

This chapter describes what are the thesis' goals, how they are achieved and the final purpose of this thesis work.

4.1 Goals definition

The previous Section 3.1.1 and Section 3.1.2 show how Software Defined Networking (SDN) and Network Function Virtualization (NFV) have changed the network management and the underlying infrastructure. Network Providers have begun to take advantage of those technologies for their features, such as flexibility and availability, to provide services where and when they are needed, to obtain redundancy and load balancing which brings to better services.

The management cost of the network infrastructure decreases using the combination of both technologies, which carry features such as flexibility and resiliency. Those paradigms have some limitations, though, for instance, they do not guarantee any performance.

Management of security controls still suffer the lack of dynamicity. This is due to manual operations and device configurations that the network administrator has to face every time a change happens in the cloud infrastructure. What happens if something changes in the network topology? Does a security policy follow the migration of a machine from a subnet to another one?

The main goal of this thesis is to extend elasticity and easy management exploited in the networking also to security controls. The aim of the project is to develop a framework which gets support to the security administrator during the policy definition and the infrastructure management.

In the workflow entailed by the tool, high-level security requirements are specified by the security administrator. However, the actual device configuration is automatically performed by the tool, hence avoiding human errors and misconfigurations. Any topology change triggers an automatic reconfiguration of all filtering controls, which are building blocks of firewall, with rules that enforce the desired security requirements.

To reach the proposed objectives, several research tasks need to be completed. First, abstract models need to be defined (Chapter 6) for policy representation and distribution on a cloud infrastructure. They define how the filtering rules are located in different firewall. The optimization problem has been modeled as a binary programming model, which, starting from the traffic pattern analysis, understands what are the filtering points crossed by specific packets. The model allows the choice of the filtering control to be configured to achieve the better performance, minimized delay and maximized bandwidth.

Those models need to be integrated into an orchestration framework (Chapter 7), whose purpose is to split the policies over the filtering controls already present in a virtual environment. This approach should help administrator to avoid complex device configurations and to reduce the overall costs avoiding new devices with filtering capabilities.

Security controls are critical devices from the performance point of view. The resource consumption such as CPU usage and memory occupation, could be bottleneck when these controls are implemented in software. Performance degradation is often associated to large rule sets. For this reason, optimization models may help in reducing the resource utilization and to get better performance.

The effectiveness and the validation of the optimization model has been proved by the implementation of performance tests (Chapter 8).

The test's results and the final considerations are presented in the last Chapter 9, which describes the goals achieved and what are the possible future developments that may be undertaken.

4.2 Use-case definition

The definition of use-cases provides support to the main purpose of the thesis and offers a simple way to understand what is the problem and how the solution is developed.

The Section 2.1.2 presents different deployment models in which a cloud environment could be instantiated and, between them, *private* and *public* cloud are the most popular and used in modern infrastructures.

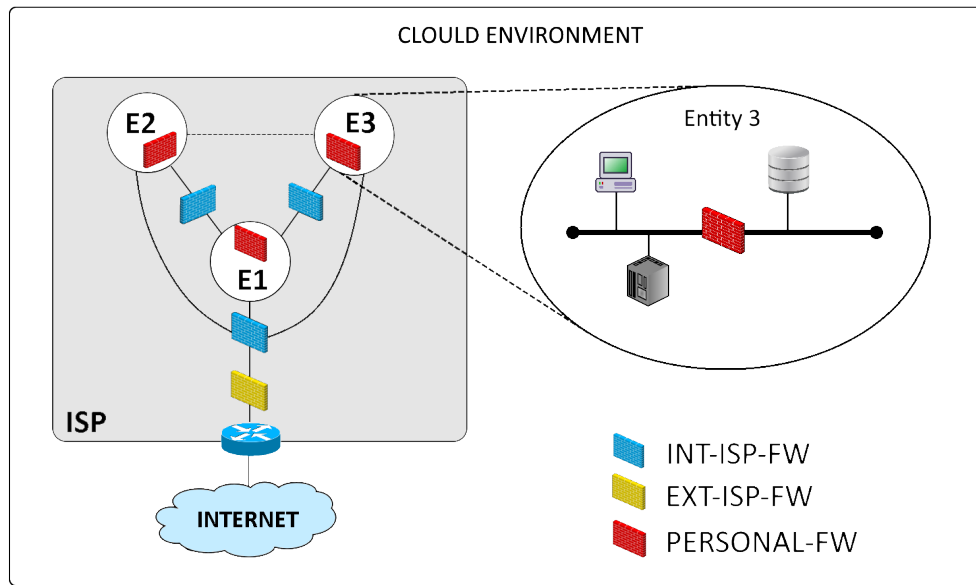


Figure 4.1. ISP and enterprises communication with logical security filtering points.

The following use-cases are examined in a business context.

- Private cloud, it is the first way to deploy a cloud infrastructure and that means a company has its own data center used only by themselves. In an organization there are many internal departures that concern disparate aspects and they deal with distinct security issues. For these reasons it is fair that each departures have different security requirements and security policies to be enforced.

It is understandable that different policies means distinct filtering rules, which could be also conflicting with each other, despite they are policies of the same company.

The main security requirements are enforced by the company policy and those rules are meant to protect the own company from the external attacks, but some departures could have sensibilities information to be protected or are essential for the company.

So it is important to provide a way to protect those departments from internal attacks or to avoid that a weakness, of some lowest security department policy, is exploited to start an exterior attack.

In a traditional security infrastructure the enterprise will enforce its security with one or few physical appliances, which presumably will be overloaded and they will represent a single point of failure. In case of any cyber attack, the possibility to exploit different filtering points allows to put several levels of protection implementing the so called *defence in depth*.

- Public cloud, it is nowadays the most chosen way to approach cloud computing and there are a lot of companies that make this solution their own main business. The context presented is different from the previous one, because there is a company in charge to deploy and to manage the cloud infrastructure which provides services to other enterprises, which are independent from each other.

This is the most complex situation in which is prone to have a lot of different uncorrelated policies and the number of conflicts between them is expected to be higher than the first use-case. The cloud provider company needs to enforce a basic security policy from the external world, because if it applies too strict conditions the risks to block some useful service or communication of the guest enterprises it is further high.

Otherwise each guest entity has strict security policy to protect themselves from the other companies in the data center, but also from the external attacks. For these reasons they need to have the possibility to enforce they own rules.

The implementation of a traditional security infrastructure is impossible, because machines of different enterprises could be hosted on the same physical server or in the same slice of network. How can the protection be provided? How many policy conflicts will come out? The optimal distribution will provide a simple solution to those problems, through the correct rule allocation, only on the filtering points which require those rules.

Chapter 5

Infrastructure analysis

5.1 Cloud infrastructure

The orchestrator chosen as test-bed for this thesis work is OpenStack cloud tool kit.

5.1.1 OpenStack introduction

OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds [26]. It is used to deploy compute instances, storage devices and networking communication in a datacenter, so it can be defined as Infrastructure-as-a-Service (IaaS) discussed in Section 2.1.2. All the software tools that compose this open source project can be controlled in different ways, from a command line interface (CLI) or via a web dashboard provided to be the most intuitive mode to interact and control the infrastructure. Both of them lean on the REST API, which provides the best and complete way to access to all the resources.

The “core” of OpenStack is made up of seven key components, *Nova* for computation, *Neutron* for the networking, *Horizon* to provide a web dashboard and *Cinder*, *Keystone*, *Glance*, *Heat* that offer additional services for storage and access.

Security

OpenStack suite provides a basic security system called *security group* which is a list of stateless filtering rules that could be added to the instance creation process to ensure security protection according to the rules specified in the security group.

This security mechanism is implemented through the Host Iptables tool, the orchestration tool builds table chains from the FORWARD chain, through the `neutron-openvswi-sg-chain` that redirects all the traffic to the specific INPUT/FORWARD/OUTPUT table chain of the instance.

OpenStack creates one Iptables chain for each Virtual Machine that belongs to a security group and injects all the rules into these tables.

The following section shows how the traffic flows into the network, through the virtual switches and different physical hosts.

5.1.2 Filtering points

From the thesis analysis provided by my colleague Andrea Della Chiesa, OpenStack presents one solution to enforce security on its architecture, but there are many different filtering points already present in this cloud infrastructure over the Host Iptables. He found that Guest instance Iptables could be another point that could provides security operations and, the virtual switches controlled by OpenFlow could behave as well as a firewall.

The OpenStack network architecture and how the packets flows between VMs and hosts are represented in the Figure 5.1.

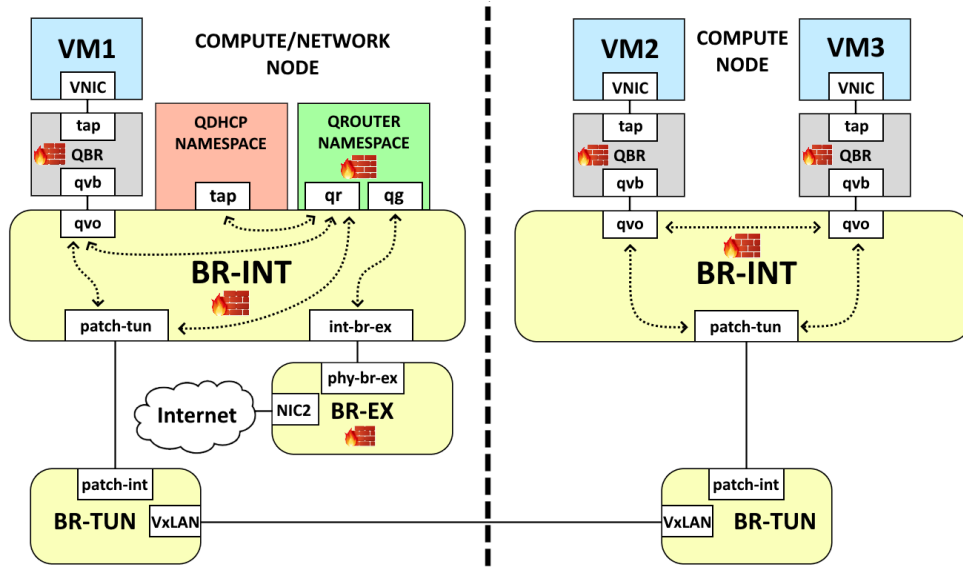


Figure 5.1. OpenStack internal architecture

The following chapter analyzes the performance of the filtering points just mentioned taking into account distinct metrics to understand the characteristics of each one.

5.1.3 Packet flows

First and foremost it is fundamental to figure out all the possible communication scenarios that could take place to know where are the best filtering points. Understanding all the possible paths, that the traffic pattern can get, allows the security service to be performed efficiently.

The following cases represents all the possible types to exchange traffic.

- *Service to service* communication: in this scenario two services inside the same Virtual Machine communicate each other. It does not matter if they are process or containers, but their packets do not leave the network stack of the instance and all the internal traffic is not visible from the outside environment. This involves that the instance itself is the only filtering point available to enforce security rules.
- *VM to VM* communication: in this case the VM traffic leaves the instance to land in a Linux Bridge called QBR which is connected, through a virtual interface, to an OpenFlow switch. This allows to exploit other two filtering points, one represented by the Linux Bridge which leans on the Host OS Iptables, and the other one is represented by the virtual switch that behaves as a firewall.
- *Subnet to subnet* communication: two VMs on the same subnet can communicate directly, the packets coming from a virtual interface are forwarded to the destination virtual interface. Otherwise is different the situation in which two VMs have to exchange traffic between them, but they belong to different virtual L2 networks. The packets cannot reach directly the destination, but the flow is modified by the OVS to reach before a virtual router which is named QROUTER in the Figure 5.1.
- *Host to host* communication: it is frequent that in a multi-node deployment two VMs have to communicate, but they are hosted in different physical machines. In this case OpenStack provides an additional virtual bridge called br-tun which is an end-point of a tunnel created

with the other physical hosts. The packets come from the `br-int` switch and they are forwarded to this bridge through `path-tun` virtual interface. The packets are enveloped in a VxLAN protocol and they are sent on the physical network. Once they are received, the tunnel header is deleted and the packets are forwarded into the receiver `br-int` switch.

- *Internet* communication: this last scenario represents the traffic pattern that covers the communication between the instances running on the cloud infrastructure and the external destinations. The packets have to reach the *network node* mentioned before, which is directly connected to the exterior network, than the traffic goes to the virtual router that is the real default gateway.

The IP header is modified, because it has a private IP address, the `QROUTER` sets, as source address, the IP address of the router interface that is exposed outside the cloud deployment or the “floating IP” associated to the VM, if it has one.

After this operation, the packet is forwarder to `br-ext` switch and it is sent outside through the second NIC of the physical machine. This last one bridge is another filtering point that could be used to enforce the rules that concern the traffic coming from and the one which goes to the external networks.

Once understood how packets flows inside and outside the cloud infrastructure, it is now possible to design a distributed firewall that optimizes where the rules have to be enforced, which exploits the filtering points previous described.

5.2 Performance analysis

This section exposes the analysis on the security controls offered by the cloud infrastructure, that leads to future choices that weigh on the design implementation.

The main purpose is to find how the behaviour of different technologies changes based on the number of filtering rules that are enforced. OpenStack infrastructure bases its security on *Iptables*, as mentioned before, but there are different filtering points that could be exploited, for this reason it is needed to find the best way to enforce security actions.

Security controls are critical devices from the performance point of view. The resource consumption such as CPU usage and memory occupation, could be bottleneck when these controls are implemented in software. Performance degradation is often associated to large rule sets.

For this reason the most interesting and significant metrics are taking in accounts, especially the ones regarding network devices and security controls.

The metrics that are evaluated are the following:

- latency
- bandwidth
- CPU consumption

The performance evaluation are done on an OpenStack deployment running on two machines whose technical specifics are described:

- **TYPE:** Intel NUC NUC5i5MYHE
- **CPU:** Intel Core i5-5300U, dual-core, 2.30 GHz
- **RAM:** 16 GB, DDR3
- **OS:** Ubuntu Server 18.04 LTS

The three metrics measures different aspects of a communication between two virtual machines and for each metric different tools are used to evaluate the performances.

- **ping** and **nping**: these tools are useful to compute the delay and the Round Trip Time between two end-points, Nping is an open source tool for network packet generation, response analysis and response time measurement. Nping can generate network packets for a wide range of protocols, allowing users full control over protocol headers [27]. The following commands are the ones launched to evaluate the performance.

```
$ sudo ping -c 1000 -i 0.005 -q -s 1000 destination-IP
$ sudo nping -H --no-crypto -c 1000 --send-eth --data-len 1000 --delay
5ms --tcp -p 80 destination-IP
```

Both of them send a packet of *1000 Bytes* every *5 ms* to the *destination-IP* address.

- **iperf3**: iperf3 is a tool to perform network throughput measurements. It can test either TCP or UDP throughput. To perform an iperf3 test the user must establish both a server and a client [28].

```
$ iperf3 -s -f K
$ iperf3 -c server-IP -f K
```

The former command starts the server on one virtual machine and the latter starts the client on the second virtual machine, which starts a TCP connection to *server-IP* address and gives the output in KBytes per second.

- **top**: this tool provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel [29].

```
$ top

$ top - 18:53:02 up 0 min, 0 users, load average: 0.52, 0.58, 0.59
$ Tasks: 4 total, 1 running, 3 sleeping, 0 stopped, 0 zombie
$ %Cpu(s): 0.3 us, 0.8 sy, 0.0 ni, 98.8 id, 0.0 wa, 0.1 hi, 0.0 si, 0 st
$ KiB Mem: 16663516 total, 10118300 free, 6315864 used, 229352 buff/cache
$ KiB Swap: 29221116 total, 29221116 free, 0 used. 10213920 avail Mem

$ PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
$ 1 root 20 0 8892 312 272 S 0.0 0.0 0:00.07 init
$ 7 user 20 0 16776 3440 3328 S 0.0 0.0 0:00.08 bash
$ 39 user 20 0 17624 2048 1516 R 0.0 0.0 0:00.00 top
```

This command displays the % of CPU used by the system or specific of a program.

Each filtering controls is implemented with a different technology. For these reasons different tested were developed. The following subsections exposes the results of the performance tests implemented on each technology used.

5.2.1 Iptables

Definition

Iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules [30]. The rules are organized in tables and, for each one, there are a number of built-in chains or maybe user-defined chains.

Each rule has fields to be matched in the packet and an action that is performed if the traffic has a match. If the packet does not match any rule, the following rule is examined, until the default action will be performed if no rules has matched.

The test is built to be a stress-test, which simulates different situations, from the one with lowest security requirements that implements few filtering rules, to the highest critical security environment. To pretend these circumstances the performance estimation is computing with the following number of rules: [0, 10, 100, 250, 500, 750, 1000, 1500, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]. This is done exploiting the following bash script that receive the number of rules as parameter in input:

```
1  #!/bin/bash
2
3  # clear the perf-test table
4  iptables -F perf-test
5
6  for j in seq 1 $1; do
7
8      if [[ $j -ne 22 ]] && [[ $j -ne 5201 ]] && [[ $j -ne 80 ]];then
9          iptables -A perf-test -p tcp --destination-port $j -j DROP
10     fi
11
12 done
13
14 exit
```

OpenStack offers two possibilities to use Iptables as security system, one is using it directly on the Operating System of the Hypervisor, or the second choice is insert rules using the Virtual Machine's Iptables. The performance tests are developed to evaluate both of them.

Latency test

The following graph shows that the Iptables latency performance degrades as a straight line with the number increasing of rules. From zero to a hundred of rules both host OS and the guest OS have the same performance, with little fluctuations due to a non complete system isolation. The differences between them starts to diverge after 1500 rules when the two lines diverge until reach 450 ms of difference at ten thousand rules.

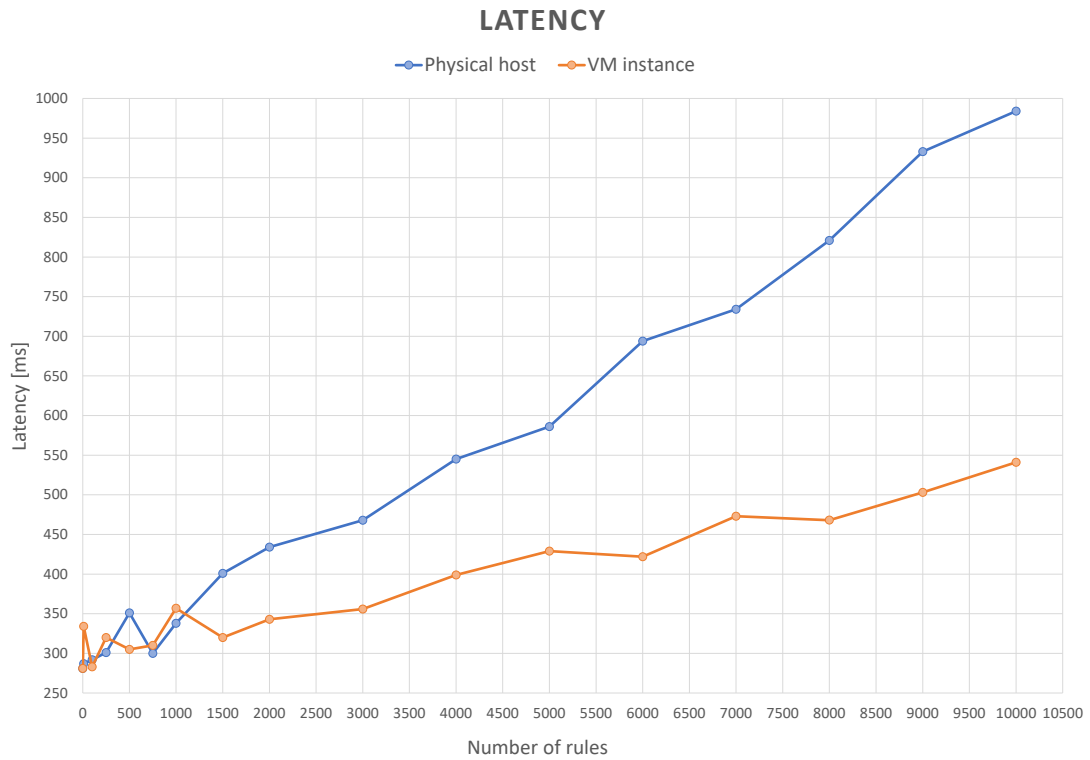


Figure 5.2. Latency test report for Iptables

This behaviour proves that guest Iptables has a better response then host Iptables when high numbers of filtering rules are injected. This result leads to prefers the Virtual Machines over the physical host if the latency is the metric to be optimized.

Bandwidth test

The following graph shows that the Iptables bandwidth performance degrades as a negative exponential line with the number increasing of rules. It displays that the bandwidth goes down quickly with a little amount of entry in the Iptables data structure for both of the OSs. The differences between them starts to be marked after 250 rules when the two lines diverge until reach more than 250 MB/s of difference at ten thousand rules.

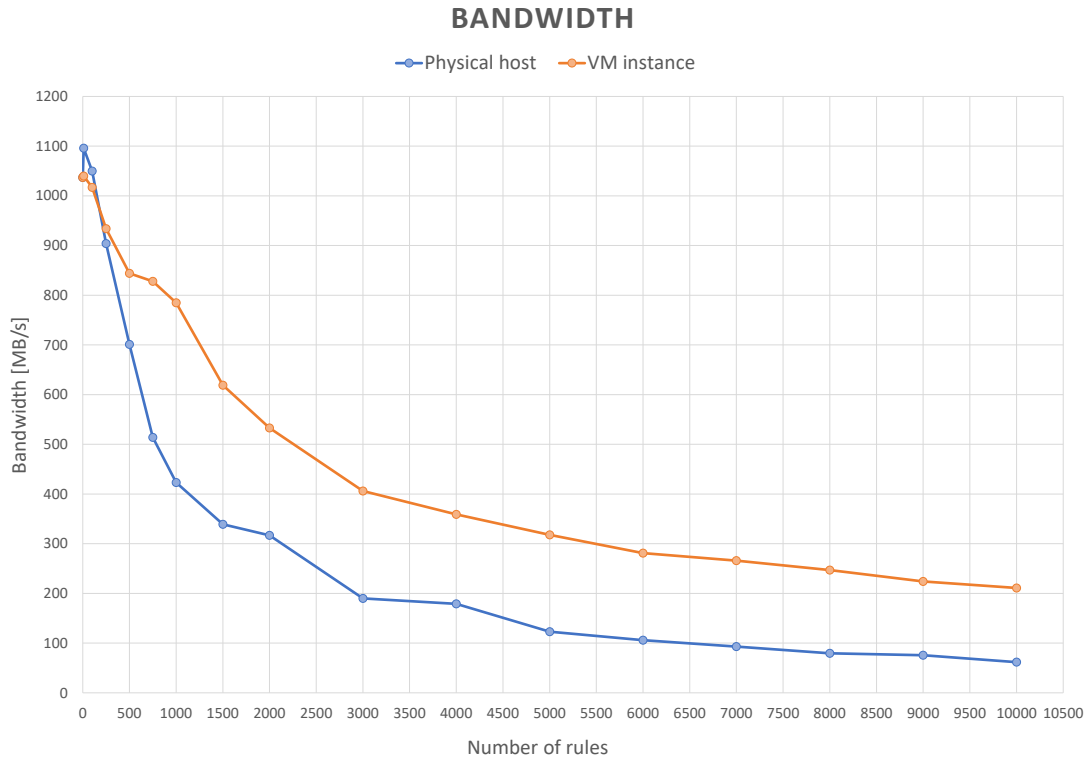


Figure 5.3. Latency test report for Iptables

This behaviour proves that guest Iptables has a better response then host Iptables when high numbers of filtering rules are injected. This result leads to prefers the Virtual Machines over the physical host if the bandwidth is the metric to be optimized.

CPU consumption

The following chart shows that the Iptables CPU consumption performance degrades as a straight line with the number increasing of rules, but unlike the latency trend, this one highlight that OSs has different behaviour also with a little number of rules.

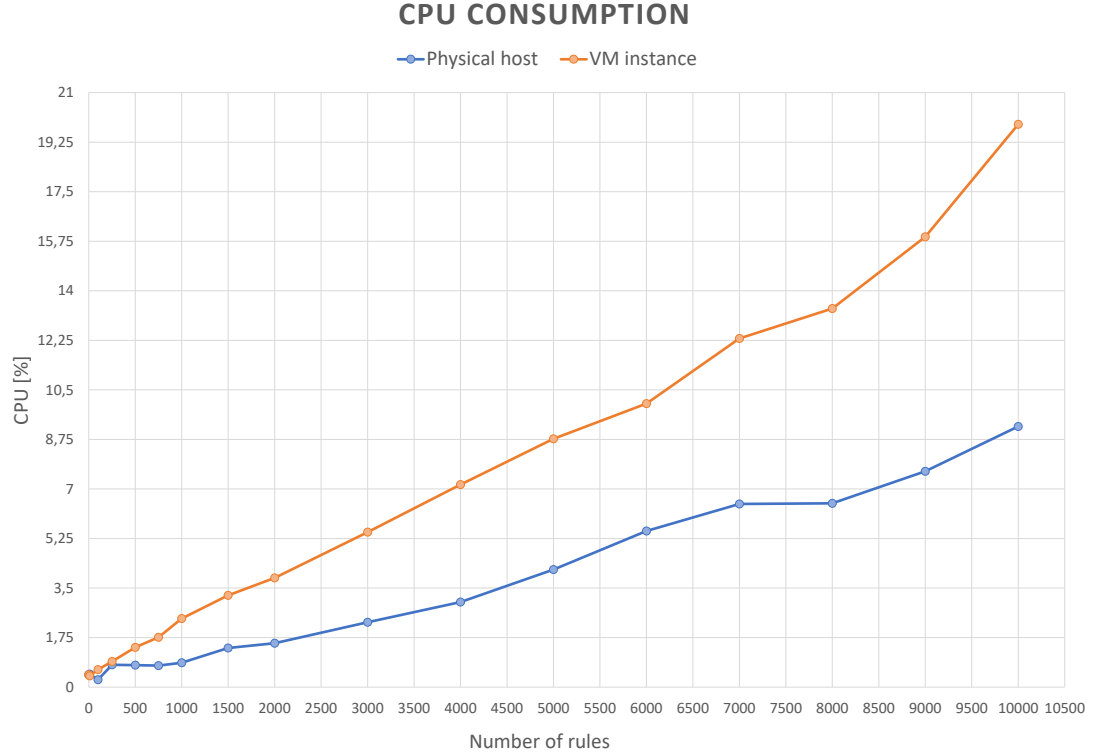


Figure 5.4. Latency test report for Iptables

This behaviour proves that guest Iptables has a better response then host Iptables when high numbers of filtering rules are injected. This result leads to prefers the Virtual Machines over the physical host if the CPU consumption is the metric to be optimized.

5.2.2 Open Virtual Switch

Definition

Open Virtual Switch, sometimes abbreviated as OVS, is an open-source implementation of a distributed virtual multilayer switch. The main purpose of Open Virtual Switch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks [31]. OVS was born to be a forwarding device, but its data representation is similar to Iptables' one, in which each entry has a list of fields to be match and an action to be performed on the packet, so the idea is to use this feature to transform the OVS in a stateless packet filtering.

The test is built to be a stress-test, that simulates different situations, from the one with lowest security requirements, that implements few filtering rules, to the highest critical security environment. To pretend these circumstances the performance estimation is computing with the following number of rules: [0, 10, 100, 250, 500, 750, 1000, 1500, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000].

This is done exploiting the following bash script that receive the number of rules as parameter in input and it sets “\$1” number of rules on “ovs-ID” virtual switch, where the rules drops all the tcp packets with “\$i” destination port and coming from “port-ID” switch port.

```
1  #!/bin/bash
2
3  num=$((1+39999))
4
5  for i in seq 40000 $num; do
6      if [[ $i -ne 22 ]] && [[ $i -ne 5201 ]] && [[ $i -ne 80 ]];then
7          docker exec "ovs-ID" ovs-ofctl add-flow br-int in_port="port-ID",
              dl_type=0x0800, nw_proto=6, tcp_dst=$i, actions=drop
8      fi
9  done
10
11 exit
```

Latency test

The following graph shows that the ovs latency performance is more or less constant with little perturbations due to the non complete isolation of the system.

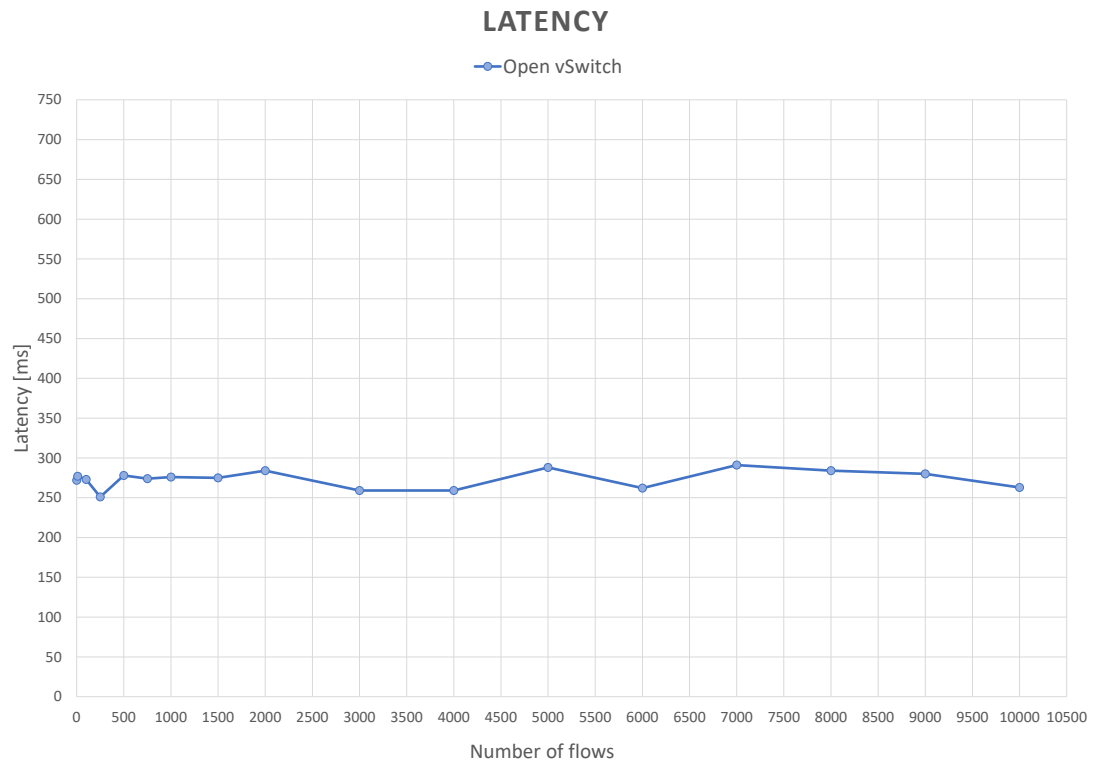


Figure 5.5. Latency test report for Iptables

Bandwidth test

The following graph shows that the ovs bandwidth performance is more or less constant with little perturbations due to the non complete isolation of the system.

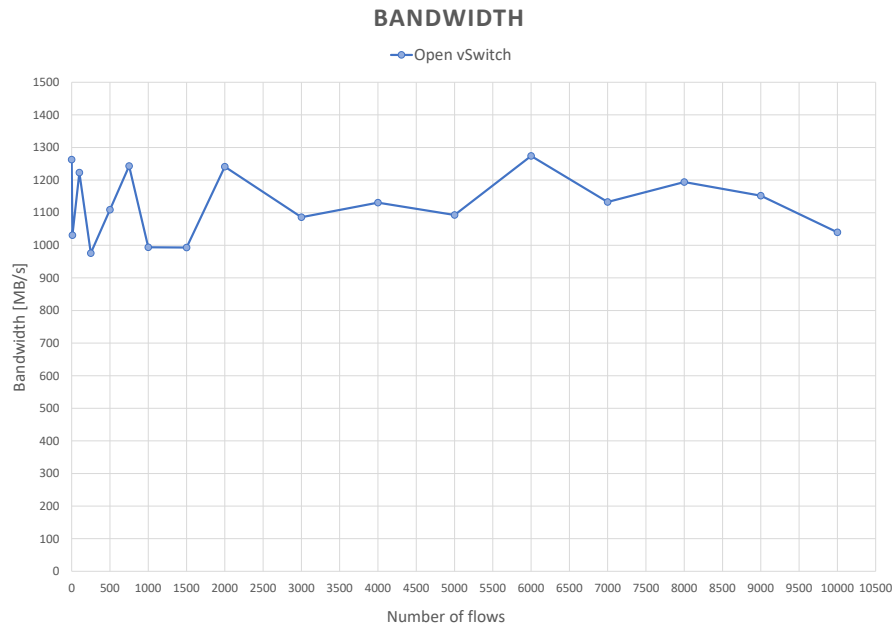


Figure 5.6. Latency test report for Iptables

CPU consumption

The following graph shows that the ovs CPU consumption performance is more or less constant with little perturbations due to the non complete isolation of the system.

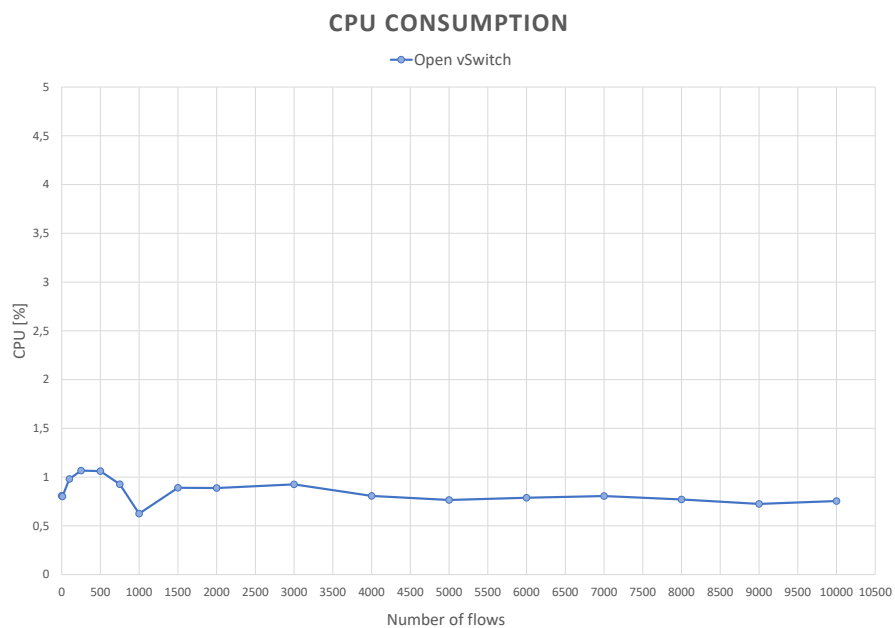


Figure 5.7. Latency test report for Iptables

5.2.3 Analysis conclusion

The tests show us that Open Virtual Switch technology has better scale-up feature based on the number of rules then Iptables. This behaviour is due to the different internal data structure of the two systems, the former implements that *exact-match cache (EMC)* (Figure 5.8) that is a fast mechanism the Open Virtual Switch uses to determine the action to be performed on an incoming packet. This is like how the CPUs checks the memory caches when accessing data. Otherwise Iptables has a linear data structure with no cache or hash table, that means a packet needs to scan all the filtering rule list until the one that matches. The implementation of the data structure has a big impact on the computational cost that in the first case, with OVS the trend is $\sim O(1)$, contrarily Iptables has a computational cost of $O(n)$.

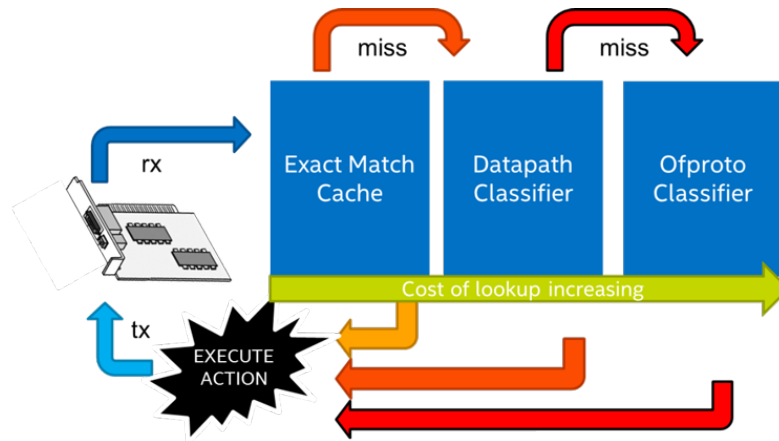


Figure 5.8. The Open vSwitch table hierarchy.

Chapter 6

Solution design

This chapter highlights the project design choices taken during the development of the framework.

6.1 High level design

The results of the performance and virtual infrastructure analysis highlights the possibility to design a solution which allows distributing filtering rules among the whole cloud environment. This gives the opportunity to avoid the exploitation of standard firewalls on which all the rules are concentrated and it represents the the well-known problem of *One Point of Failure* (OPF).

To reach the proposed objectives, several research tasks need to be completed. First, abstract models need to be defined for policy representation and distribution on a cloud infrastructure. They define how the filtering rules, are located in different firewall.

Those models need to be integrated into an orchestration framework, which purpose is to split the policies over the filtering controls already present in a virtual environment.

To design a security infrastructure, working on a cloud data center, different aspects need to be taken into account, starting from the modelling of the physical infrastructure, defining the entities at stake and what are the policies to be enforced.

Security policies are handled by the *Policy Tool Library*, which is integrated in the framework. It is used to perform operations on those security policies as inter-policy and intra-policy conflict analysis.

Then, an optimization problem has been modeled as a binary programming model. This model is represented by a set of propositional equations made of binary variables and a cost function, that should be minimized. The optimization model produces two outcomes, the former is the assignment of an ordered list of rules to each filtering point, the latter is a list of unnecessary rules that are notified to the administrator. Finally a distribution process injects the filtering rules based on the output of the optimization process.

The thesis work is focused on the abstract model design for policy representation and distribution on a cloud infrastructure.

The distribution controller and the distribution process is developed by Andrea Della Chiesa, who studied the different filtering technologies. His work is related to the definition of distinct commands, one for each system involved, based on the output produced by the optimization model.

The framework is composed of several modules:

- the policy module, which handles anomalies and conflicts, thus optimizing the security requirements. It removes redundancies and contradicting rules;
- the optimization process, which is used to determine the security controls to be configured and the best place to assign the rules that enforce the security requirements;
- the distribution model, which injects the rules into the filtering points.

In the following Figure 6.1 it is shown the model representation of the framework.

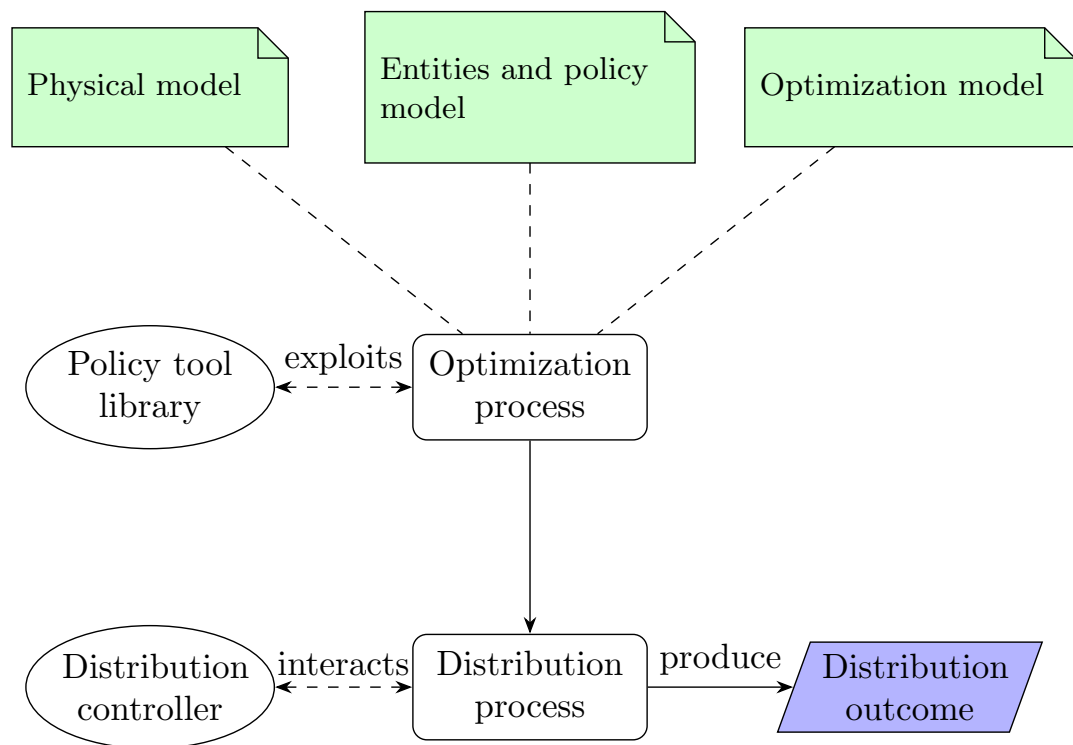


Figure 6.1. Security system high level design.

6.2 Workflow

This section has the purpose to give a graphical representation of how the workflow should be and it focuses on the distribution and optimization model developed in this thesis work.

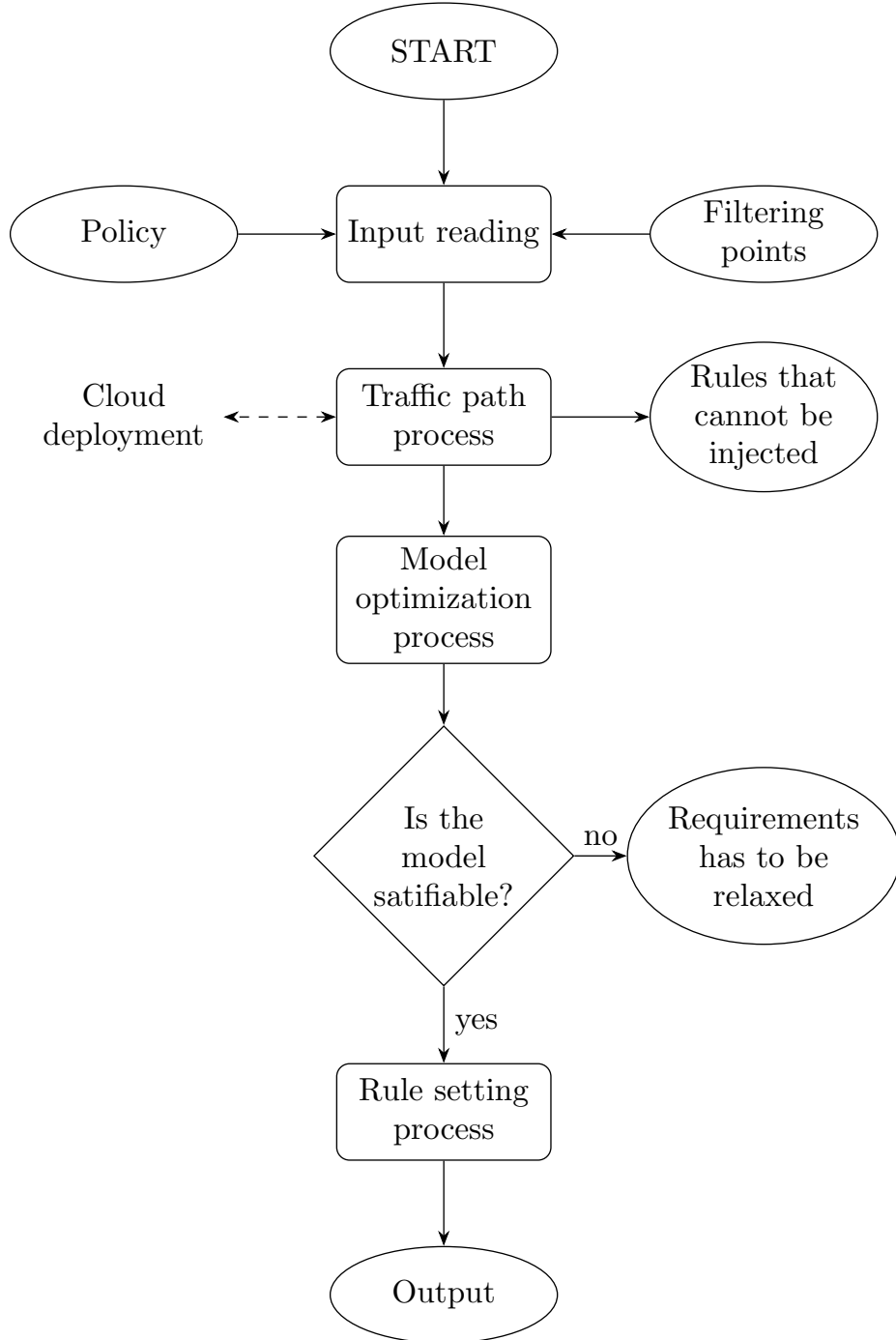


Figure 6.2. Optimization model workflow.

The workflow has two inputs that are the policies, with their rules, and the filtering points, which means all the points with filtering capabilities. In the Figure 6.2 there are three outputs, the first one shows the rules that cannot be injected, the second one is a bad outcome, because the requirements are too strict and there is no solution to the optimization problem. The last output is the list of filtering point with their ordered list of rules.

6.3 Policy Services

Before explaining how the distribution and the optimization modelling is done, it is essential to underline the role of the “*Policy tool library*”, as it shown in the high level design 6.1. In the thesis economy it provides functionalities to manage security policies and filtering rules.

It is integrated to solve all the issues introduced in Section 4.2, where it explains that policy inconsistency and rule anomalies represent the reasons why a policy refinement is necessary. The security problems faced by the design are *intra-policy anomalies*, which means there are misconfiguration in the same security domain, and *inter-policy anomalies* that represents a more complex issue, because conflicting rules between different security domains need to be enforced side by side may be on the same filtering points (e.g the entity A wants to deny all the tcp incoming traffic from 8080 port and the entity B wants to allow that traffic pattern).

For these reasons the steps depicted below represent the process flow of the security policies.

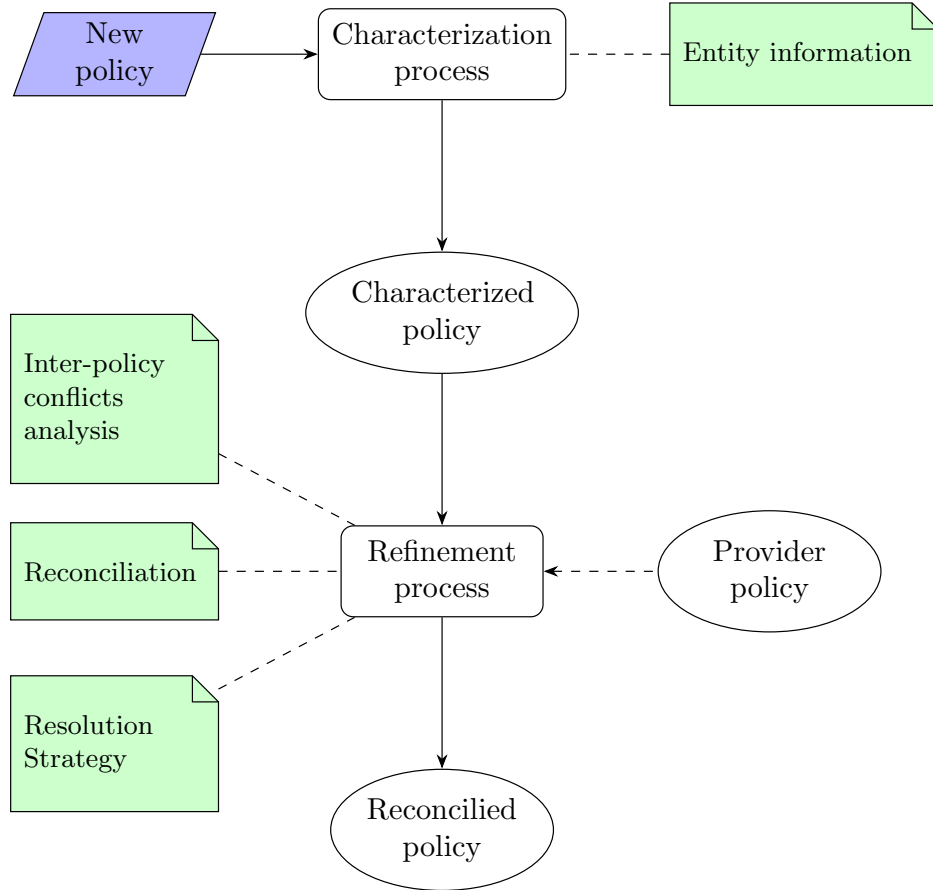


Figure 6.3. Policy service design.

- Policy characterization: it solves the ambiguity in rules definition between different security policies. In other words it is fair that a filtering rule, defined by an entity, provides security protection for the entity itself, but in a share infrastructure could interfere with others entities present on it. When a new policy is given in input, all the rules that compose the policy it self are analyzed and it is checked if they are well-formed, this means each one is related to the entity that requires it.

The possible scenarios to understand if they are well-formed are the following:

- no IP address clause, this rule has not relationship with the entity, so it will be duplicated and the entity network address will be added as condition clause, one as source IP and the second one as destination IP network;
- one IP address clause, this rule specifies an IP address in its condition clauses, if it is a subset of the entity network the rule is not changed, otherwise the network IP address of the related entity is added;
- two IP address clauses, this rule has both IP source and destination addresses and, at least, one of them has to be a subset of the entity network. If neither the source and the destination IP addresses are related to the own entity, the rule is marked as not well-formed and it is discarded.

This will not change the semantic of the rule, because the process add informations that are implicitly described in the rule, but it expresses them explicitly, adding the condition clauses.

- Inter-policy conflicts analysis: it solves all the conflicts between the rules of the same policy, in other words this step analyses the policy, it detects all the anomalies and the conflicts and it deletes them.
- Policy reconciliation: it generates a unique reconciled policy between the ISP security policy and an entity. The former has higher priority then the second one. It will have no effect doing policy reconciliation between entity policies, because there is no conflicts due to the policy characterization.

6.4 Path model design

6.4.1 Tree structure

The Section 5.1 shows all the possible pattern of traffic and where the packets flow inside the cloud infrastructure. The *Path model* has the purpose to represent these flows into an high level view.

First of all, the representation of the nodes involved into packet forwarding process is defined. These points comes out from the previous Della Chiesa infrastructure analysis. The result is a Tree data structure view of the deployment. The nodes into a tree structure are divided into tree types:

- root element
- leaf element
- intermediate element

Each forwarding point is mapped into one of these categories, all the physical servers and the virtual switches are labeled as *intermediate* elements, because they are neither the source or the destination of the traffic. The *leaf* element are defined both as Virtual Machines and Services, because they are the basic element that could generate traffic or the one which receives packets. The *root* element is fake, there is no item which could be mapped with it, but it represent the possibility to reach each node directly, instead of passing through another intermediate element.

6.4.2 Forwarding models

From this starting point, the “Path model” pulls out a set of “submodels” which are useful to describe the forwarding role into the tree structure, based on the traffic path. These models are defined in the Appendix B, each one with the description of how the traffic flows into them and with a graphic representation.

The definition of these models allow mapping in the following schema all the entities at stake:

- virtual machines: they could be mapped as A, B, C or D model;
- virtual switches: they could be mapped as all the model from E to L model;
- physical servers: they could be mapped as E and F model.

To achieve the goal to find all the filtering points for each traffic path, it is necessary to find an algorithm to explore the structure represented in the Figure 6.4. This could be a very expensive computational task to be done, if standard graphs exploring algorithms are used to seek from which node the traffic starts and on which nodes the traffic lends.

To improve computing performance and to have an efficient search, the tree nodes has additional informations about what nodes are above them and what nodes are children of them.

The path model exploits the concept of “Condition Clause”, taken from the *TORSEC* work during the development of *policy tool library*, which is used here as a formal representation of an identifier for leaf elements.

6.4.3 UML representation

The UML diagram class depicted in this Figure 6.4 is a formal representation of a tree structure, in which the main attributes and methods are highlighted.

This classes allows to:

- efficiently find a node in the deployment structure without a big effort in computational power;
- effective realize an exploration of the nodes that are crossed by a specific traffic pattern.

The relationships described between the UML-classes are an important aspect, because they described the association structure and the direction of it.

UML class diagram

The diagram depicted below shows the Tree data structure representing the physical and virtual topology with pruning information and the main methods.

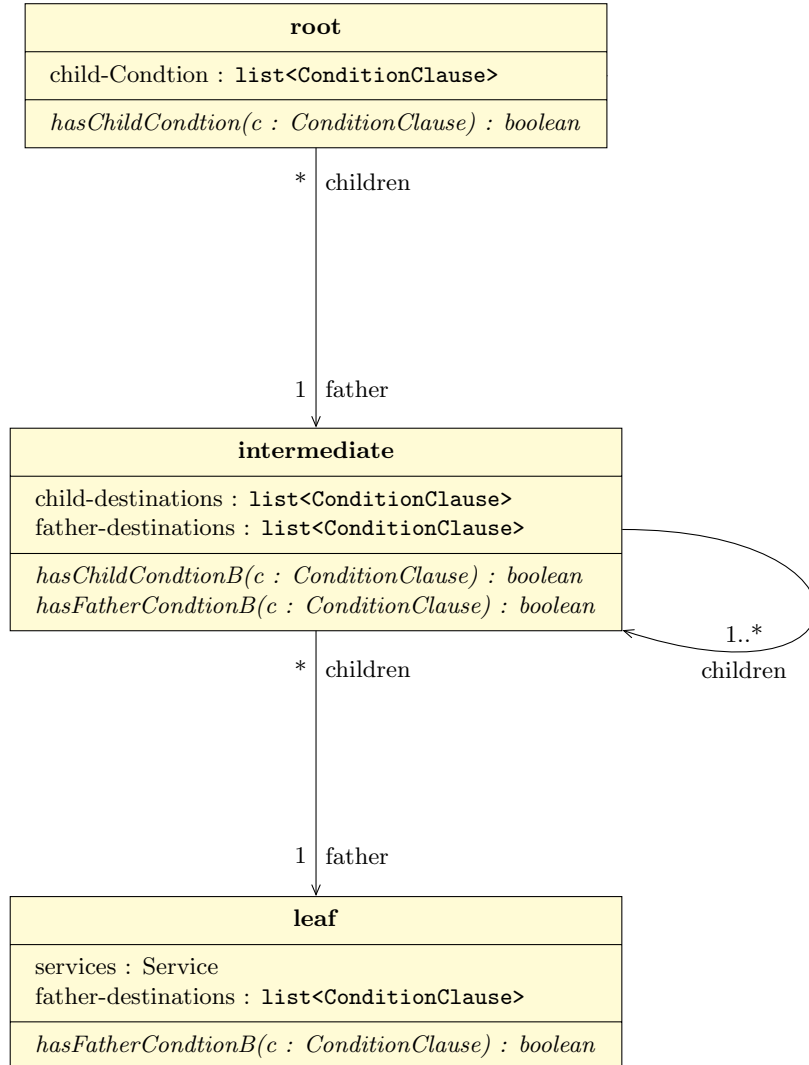


Figure 6.4. Tree model representation.

6.5 Optimal distribution model design

The optimal distribution model, starting from the traffic pattern analysis, understands what are the filtering points crossed by specific packets. The model allows the choice of the filtering control to configure to achieve the better performance, minimized delay and maximized bandwidth. This reduces the number of rules in each filtering point and to get a better use of the available resources.

The distribution model opens to a new security problem that rises the same anomalies listed in Section 3.3.2, because this model, from an equivalent firewall, builds a distributed firewall in which the same filtering point has a list of heterogeneous rules coming from different policies. The Section 6.3 presents how the conflicts are handled and removed, but in this case the task is to find an ordered list of rules without any anomalies between them to place in a filtering point, avoiding any further policy refinement.

The optimization is due to two different aspects, the former is to enforce the same security level of the equivalent firewall with the minimum number of rules and the latter is to maximize the performance in terms of latency, bandwidth and resource consumptions.

The distribution is modelled as SAT problem (Boolean Satisfiability problem), which is the problem of determining if exists an interpretation that satisfies a given boolean expression. In other words, it tries to set the variables of a boolean formula with the values *true* or *false* in such a way that the given expression is evaluated as *true*, in this case the boolean formula is called satisfiable, otherwise *false* if it is unsatisfiable.

This optimization model is made of a list of boolean expressions, each one has a distinct task.

6.5.1 Propositional equation

It expresses a boolean propositional equation that means it is composed by binary variables.

- vertical equation: it represents the choice to place a specific rule in a specific filtering point. This is done exploring the rules that could be enforced in this point.

$$P_{n,x}^\alpha = \text{generalization}() \wedge \neg \text{precedingSuperSet}() \wedge S_{n,x}$$

- $P_{n,x}^\alpha$: it expresses the boolean variable which is true if the n -rule, with the α action, on the x filtering point is present, otherwise false if it is not enforced;
- $\text{generalization}()$: it is a boolean function that expresses if the n -rule falls under the Generalization anomaly (Figure 3.5) that is not an error or a misconfiguration. This is done searching a *correlation anomaly* and, if it is found, making sure that there is no rules that falls under the *redundancy anomaly* between the current rule and the “correlated” rule;

$$\text{generalization}(P_{n,x}^\alpha) = \bigvee_{i=n+1}^{R-1} \left[P_{i,x}^{\alpha, \cap} \wedge \neg \text{redundant}(P_{n,x}^\alpha) \right]$$

- $\text{redundant}()$: it is a boolean function that expresses if the n -rule falls under the Redundancy anomaly, if this expression returns true, the rules must be not enforced. To check if the n -rule is redundant it is needed to search any rules, between the following rules and the current one, which is a super set or equivalent one and it has the same action to be performed.;

$$\text{redundant}(P_{n,x}^\alpha) = \bigvee_{j=n+1}^{i-1} \left(P_{j,x}^{\alpha, \supseteq} \right)$$

- *precedingSuperSet()*: it is a boolean function that expresses if the n -rule falls under the shadowing anomaly (Figure 3.5) and in this case the rule must be deleted. This function searches any rule, with any action, preceding the n -rule which is a superset or an equivalent set, if it is found at least one rule, the function returns true;

$$\text{precedingSuperSet}(P_{n,x}^\alpha) = \bigvee_{k=0}^{n-1} \left(P_{k,x}^{\alpha, \supseteq} \right)$$

- $S_{n,x}$: it is a boolean variable that represents the freedom degree of the system.

6.5.2 Hard constraints

The hard constraints express the clauses that must be satisfied by the model.

- horizontal equation: it represents the choice to place a specific rule at maximum in one filtering point. This equation is built only for the rules whose action is *deny*, independently from the default action.

$$\mathcal{D}_{n,p}^\alpha = \text{setInOnePoint}() \vee \text{!necessary}() = \text{True}$$

- $\mathcal{D}_{n,p}^\alpha$: it is the name of the equation that represents the n -rule in the p traffic path;
- *setInOnePoint()*: it is the boolean function that values true if the n -rule is placed in one filtering point, or false if it is placed in multiple instances or not placed at all;

$$\text{setInOnePoint}() = \bigvee_{i=0}^M \left[S_{n,i} \wedge P_{n,i} \wedge \text{noOtherPlaces}() \right]$$

- *noOtherPlaces()*: it is the boolean function that checks if no other filtering point has the n -rule present variable set to true;

$$\text{noOtherPlaces}() = \bigwedge_{j=0, j \neq i}^M \left(\text{!}S_{n,j} \right)$$

- *necessary()*: it is the boolean function that returns true if the n -rule is neither *shadowed* or *redundant* in at least one filtering point.

$$\text{necessary}() = \bigvee_{i=0}^M \left(P_{n,i} \right)$$

- freedom degree equation: it represents a check on the degree of freedom that the model has, this equation adds a constraint such that limits this flexibility. It imposes that at least one $S_{n,i}$ has to be set to true on the list of filtering points, otherwise the optimization engine puts all $S_{n,i}$ to false.

$$\mathcal{F}_{n,p}^\alpha = \bigvee_{i=0}^M \left(S_{n,i} \right) = \text{True}$$

- $\mathcal{F}_{n,p}^\alpha$: it is the name of the equation that represents the n -rule in the p traffic path;
- $\bigvee_{i=0}^M$: it builds an expression in which the boolean variables are placed in *OR* condition with the others. This expression slides all the M filtering points, given a specific p traffic path.

- number delimiter equation: it represents a check on the number of rules that a filtering point has enforced on itself, because some security controls has strictly performance requirements or implementation requirements that cannot allow more than a certain number of rules.

$$\sum_{n=0}^R \text{countOneIfTrue}(P_{n,i}) \leq \mathcal{M}_i$$

- \mathcal{M}_i : is identifies the maximum value of number of rules that the i filtering point can enforce that is give in input;
- $\text{countOneIfTrue}(P_{n,i})$: it is the boolean function that returns the value 1 if the n -rule is enforced in this filtering point, otherwise returns 0;

$$\text{countOneIfTrue}(P_{n,i}) = \begin{cases} 1, & \text{if } P_{n,i} = \text{true} \\ 0, & \text{if } P_{n,i} = \text{false} \end{cases}$$

- $\sum_{n=0}^R$: it counts how many rules are settled in the filtering point.

6.5.3 Integer equation

The integer equation made this model a SMT problem, because it takes advantages of, not only boolean logic, but others theories like integers or real numbers. Moreover there is the clause that this value has to be minimize and this leads this problem to be classified as MaxSMT.

- cost equation: it represents the evaluation of the performance based on the number of rules injected for each filtering point. It sums all the costs of the M points and it finds the minimal value.

$$\mathcal{C} = \min \left\{ \sum_{i=0}^M \mathcal{P}_i(n) \right\}$$

- \min : is the arithmetical function that finds the minimum value given an expression;
- $\mathcal{P}_i(n)$: it is the representation of the performance evaluation based on n -number of rules in the i filtering point;
- $\sum_{i=0}^M$: it sums all the performance values of all the filtering points available for the distribution;
- n : it is the number of rules which are enforced on the i filtering point.

Chapter 7

Implementation

This chapter has the purpose to show the tools exploited for the optimization process and the techniques used to implement a framework which gets support to the security administrator during the policy definition and the infrastructure management.

7.1 Policy conflict analysis Implementation

7.1.1 Library introduction

As already introduced in the previous Section 6.3, this security system is strongly integrated with the project *policy tool library* developed by “TORSEC” security group in Politecnico di Torino for the past EC-funded project SECURED.

This library is developed to handle and to manage security policies, to resolve rules conflicts and to analyse reachability properties in a given landscape with servers, services and firewalls. The core of the library is the module that implements what the Section 3.3.2 explains, which is to say it bases its policy representation through the *Canonical Form*.

The library also provides all the tools to generate a *Semilattice* representation and the possibility to translate a given policy into a *Morphism* form.

Different modules of the library were exploited to solve anomalies and conflicts of the input policies.

- Intra-policy conflict analysis: each policy is represented by the Java object `PolicyImpl` that offers a “resolution strategy”, a “default action” and a list of “filtering rules”. Starting from this class and following the procedure described in the section “Policy Analysis, Conflicts and Transformation” (3.3.2), the correspondent closure is generated and, from that results, the canonical form is created to have an easy processing of the policy itself.

After this operation the class `SemiLatticeGenerator` allows the library to analyse the policy and to discover all the rule anomalies which are solved by the `FMRMorphism` object. It generates a morphism representation of the policy from which a new policy could be defined with no conflicts.

- Inter-policy conflict analysis: this process is called “reconciliation”, because it is done between policies. The first operation is to create a `ComposedPolicy` that contains an ordered list of the policies to be reconciliated.

After this step, the procedure to find all the anomalies and to remove the conflicts is the same of the “intra-policy conflict analysis” described in the previous point.

7.1.2 Policy - XSD schema

The XSD schema is defined to model the security policies of different organizations and it is an extension of the DTD schema used in the policy library. The implementation and the modelling were developed to make them full compatible with the previous schema in order to reuse library's modules. Moreover XSD language was chosen instead of DTD one, because it is more expressive and it is possible to define and to establish more specific constraints and references.

Additional work is done to make it more compliant with the input model described in Section “High level design” (6.1). In particular all the policies are nested in a more general node named *EntityElement*, which could express further information.

This node is characterized by:

- *ISP*: it is a boolean variable that identifies if this is the owner entity of the infrastructure. This variables is introduced because it identifies the entity with the higher priority among all the entities;
- *Label*: it is the uniquely identifier of the entity among all the entities;
- *Subnet*: it is the network linked to this entity, on which its nodes could be instantiated;
- *PolicyElement*: it is the entity's policy.

```
1 <xsd:complexType name="EntityElement">
2   <xsd:sequence>
3     <xsd:element name="Policy" type="PolicyElement" />
4   </xsd:sequence>
5
6   <xsd:attribute name="ISP" type="xsd:boolean" use="required" />
7   <xsd:attribute name="Label" type="xsd:string" use="required" />
8   <xsd:attribute name="Subnet" type="SubnetElement" use="required" />
9 </xsd:complexType>
```

Each policy is modeled as a “PolicyElement”, it expresses the security behavior that must be followed by all its instances. It is characterized by:

- *PolicyName*: it is the name the policy;
- *PolicyType*: it is the type of the policy, in this thesis work the only useful policy type is *PolicyTypeElement* = {FILTERING}, but this feature is kept for compatibility and for additional future work;
- *DefAction*: it is the default action performed by this policy;
- *rule list*: it is the ordered filtering rule list.

```
1 <xsd:complexType name="PolicyElement">
2   <xsd:sequence>
3     <xsd:element name="PolicyName" type="xsd:string" />
4     <xsd:element name="PolicyType" type="PolicyTypeElement" />
5     <xsd:element name="DefAction" type="xsd:string" />
6     <xsd:element name="Rule" type="RuleElement" minOccurs="0"
7       maxOccurs="unbounded" />
8   </xsd:sequence>
9 </xsd:complexType>
```

The “rule” item is the last one modelled, it has the task to describe what traffic pattern needs to be matched by this filtering rules and what action has to be performed on it.

It is defined as:

- *Priority*: it sets the rule priority in the policy;
- *Selector list*: it is the list of selectors, which means the fields in a packet that need to be checked to be matched with the rules;
- *Transformation*: it is ignored, it is present for compatibility, but it may be used in future for extensions;
- *Action*: it represents the action to be performed on the matched traffic;
- *Label*: it is the uniquely identifier of the rule.

```
1 <xsd:complexType name="RuleElement">
2   <xsd:sequence>
3     <xsd:element name="Priority" type="PriorityElement" />
4     <xsd:element name="Selector" type="SelectorElement" minOccurs="1"
5       maxOccurs="unbounded" />
6     <xsd:element name="Transformation" type="TransformationElement"
7       minOccurs="0" maxOccurs="1" />
8   </xsd:sequence>
9   <xsd:attribute name="Action" use="required">
10     <xsd:simpleType>
11       <xsd:restriction base="xsd:string">
12         <xsd:enumeration value="ALLOW" />
13         <xsd:enumeration value="DENY" />
14       </xsd:restriction>
15     </xsd:simpleType>
16   </xsd:attribute>
17   <xsd:attribute name="Label" type="xsd:string" />
18 </xsd:complexType>
```

The following Listing 7.1 shows a complete XML file, which defines an example of policies definition.

```

1  <Entities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml_policy.xsd">
2
3      <Entity Label="ISP" ISP="true" Subnet="10.0.0.0/24">
4          <Policy>
5              <PolicyName>policy1</PolicyName>
6              <PolicyType>FILTERING</PolicyType>
7              <DefAction>DENY</DefAction>
8
9              <Rule Action="ALLOW" Label="policy1.R1">
10                 <Priority>1</Priority>
11                 <Selector Label="Source Port">11</Selector>
12                 <Selector Label="Source Address">10.0.0.56</Selector>
13             </Rule>
14
15             <Rule Action="ALLOW" Label="policy1.R2">
16                 <Priority>1</Priority>
17                 <Selector Label="Destination Port">80</Selector>
18                 <Selector Label="Destination Address">10.0.0.56</Selector>
19             </Rule>
20         </Policy>
21     </Entity>
22
23     <Entity Label="entity2" ISP="false" Subnet="20.0.0.0/24">
24         <Policy>
25             <PolicyName>policy2</PolicyName>
26             <PolicyType>FILTERING</PolicyType>
27             <DefAction>DENY</DefAction>
28
29             <Rule Action="ALLOW" Label="policy2.R1">
30                 <Priority>1</Priority>
31                 <Selector Label="Source Address">20.0.0.38</Selector>
32                 <Selector Label="Destination
33                     Address">192.168.0.0-192.168.0.255</Selector>
34             </Rule>
35         </Policy>
36     </Entity>
37 </Entities>

```

Figure 7.1. XML file describing the entity's policies.

7.2 Path finder Implementation

This module has the goal to solve the hard challenge to find, in a complex graph structure, all the filtering points crossed by each pattern traffic that has a match with one or more security rules.

7.2.1 Graph traversal

In computer science, graph traversal, also known as graph search, refers to the process of visiting (checking and/or updating) each vertex in a graph [32].

The graph model is represented as a connected graph $G = (V, E)$ with directional edges, the algorithm starts from a specific vertex and it explores all the the incident outgoing edges and vertices at the end of them. If the node is not found, the previous steps are repeated starting from a different vertex until all the nodes are visited.

This process has a significant computational cost expressed as $O(V(E + V))$, which could be reduced to $O(E + V)$ if the *Kosaraju's Strongly Connected Component Algorithm* is used.

7.2.2 Depth-first search

The *Depth-first search* (DFS) is a traversing algorithm for graph and tree data structures. The process starts from the root vertex, which is chosen randomly in case of a graph, and for all the branches it explores as far as possible along the chosen one until backtracking.

The pseudo-code example of a recursive implementation is reported below.

Algorithm 1: Depth-first search - DFS(G, v)

```

Input:  $G$  ▷ the graph
Input:  $v$  ▷ a vertex of the graph

begin
  label  $v$  as “discovered”
  if  $v$  is the goal then
    | return  $v$ ;
  end

  ▷ exploring this branch
  forall directed edges from  $v$  to  $w$  that are in  $G.adjacentEdges(v)$  do
    | if  $w$  is the goal then
    | | return  $w$ ;
    | end

    | if vertex  $w$  is not labeled as discovered then
    | | recursively call DFS( $G, w$ )
    | end
  end
end

```

7.2.3 Breadth-first search

The Breadth-first search (BFS) is the other searching algorithm for the tree and graph data structures. Its approach is the opposite of the previous one, because, instead of exploring completely a branch before backtracking, it explores all the neighbour nodes at the current depth and then passes to the following level of depth.

The pseudo-code example of a recursive implementation is reported below:

Algorithm 2: Breadth-first search - BFS(G, v)

```

Input:  $G$  ▷ the graph
Input:  $v$  ▷ a vertex of the graph

begin
  define  $Q$  as a queue
  label  $v$  as "discovered"

   $Q.enqueue(v)$ 

  if  $v$  is the goal then
    | return  $v$ ;
  end ▷ exploring this depth level

  forall  $q$  in the  $Q$  queue do
     $w \leftarrow Q.dequeue()$ 

    if  $w$  is the goal then
      | return  $w$ ;
    end

    ▷ adding the following level vertex to the queue
    forall directed edges from  $w$  to  $k$  that are in  $G.adjacentEdges(w)$  do
      | if vertex  $w$  is not labeled as discovered then
        | | label  $k$  as discovered
        | |  $k.parent \leftarrow w$ 
        | |  $Q.enqueue(k)$ 
      | end
    end
  end
end

```

7.2.4 Pruning approach

The analysis of the previous algorithms leads to prefer the *Depth-first search* as the one to be implemented. This chosen is done because of the intrinsic data structure that is going to represent the cloud deployment.

In the networking world the tree structure is preferred over a graph representation to avoid loop cycle and black holes and, as is explained in Section 6.4, the physical topology on which this thesis work is based is structured as a tree. Moreover not all the vertexes of the tree could be the goal of the traversal process, because only leafs are a potential target for the traffic and for this reason also for the algorithm. The DFS is in most of the cases faster than the BFS algorithm because, the last one explores the potential vertex only at the end of process, instead of the former algorithm.

Nevertheless the DFS is the best choice in this environment, the computational cost still remain $O(E+V)$ and it is too much time expensive. The decision taken is to follow the *pruning* approach for the algorithm to have a faster searching process and to reduce the size of decision trees by removing some sections of the data structure.

The pruning allows the algorithm to avoid the exploration of complete branches of the tree, this means that the step of backtracking is not more contemplated in the algorithm, but it starts to examine a section if there is the target node on that branch. The optimization of the searching time has a side effect, because each vertex has to store additional informations, in particular it has to provide what nodes are its children and what are reachable from the parent node. In this way each node is able to know the right direction to take to get the goal vertex.

The process to create a traffic path is divided into several steps:

- to find the starting points: the process starts from the *root* element and search all the nodes where a specific traffic pattern could be generated;
- to find the ending points: the process, as the previous step, starts from the *root* element, and it explores with the pruning DFS algorithm the tree structure to retrieve all the nodes where the packets could be intended;
- to find the filtering points: starting from one of the nodes, which could generate the traffic, the process begins looking for how to reach any end points. If the current node is a potential destination node the path process ends, returning as the only filtering point the node where the procedure starts. Otherwise the algorithm searches between the children nodes and the parent destinations, any nodes that could match with this type of traffic.

The number of leafs in a cloud environment could be huge, and a linear search $O(n)$ to find a match for each vertex is too time consuming. For this reason the *condition clause* notation is introduced to have a constant $O(1)$ matching. It is used to refer, in security context, to the rule fields forming the clause that match the packet.

Here it has a new utilization, because each node has two condition clause, except the leafs vertex and the root element that they just have one, which are used to check in a unique operation the children nodes and the parent's one.

In this way each intermediate node has a *child* condition clause, which is the union of the children condition clauses, and another one called *father* condition clause that has all the condition clauses coming from the parent node.

Finally the algorithm behaves in two ways based on the $a \in \mathcal{A}$ action performed by the current rule and the $d \in \mathcal{A}$ default action enforced.

- deny action: in case the rule action is equivalent to a *deny* action, this filtering rule has to be enforced at least in one place from the node that generates the traffic and the one that receives the packets. The path finder process (Algorithm 3) starts from the first nodes and it continues until the ending point, unless it found a fork in the path. This means the traffic could takes different paths from this point forward, here the process stops and return all the point found, include this last one;
- allow action: if the rule action is *allow* action, the behave of the Algorithm 4 is quite different. It acts like the previous one until it reaches a fork in the path, here recursively explores all the branches of the fork found as far as it reaches all the nodes that matches. The reason is to maintain reachability between nodes that the other way could not be achieved.

Algorithm 3: Pruning search - deny action $PS(G, v, start, end)$

Input: G ▷ the graph
Input: v ▷ a vertex of the graph
Input: $start$ ▷ condition clause that identify the starting point
Input: end ▷ condition clause that identify the ending point
begin
 define L as the list of node that matches the “end” condition clauses
 if $v.matchConditionClause(end)$ **then**
 | return v ; ▷ v is a leaf node
 end

 $L.addAll(v.childrenNodesMatching(end));$
 if $L.size() > 1$ **then**
 | return v ; ▷ a fork is found - process stop
 end
 if $L.size() = 1$ **then**
 | return $v, PS(G, L.get(0), start, end);$ ▷ recursively call $PS()$
 end

 $L.addAll(v.fatherNodesMatching(end));$
 if $L.size() > 1$ **then**
 | return v ; ▷ a fork is found - process stop
 end
 if $L.size() = 1$ **then**
 | return $v, PS(G, L.get(0), start, end);$ ▷ recursively call $PS()$
 end
end

Algorithm 4: Pruning search - allow action $PS(G, v, start, end)$

Input: G ▷ the graph
Input: v ▷ a vertex of the graph
Input: $start$ ▷ condition clause that identify the starting point
Input: end ▷ condition clause that identify the ending point
begin
 define L as the list of node that matches the “end” condition clauses define R as the
 list of node to return
 if $v.matchConditionClause(end)$ **then**
 | return v ; ▷ v is a leaf node
 end

 $L.addAll(v.childrenNodesMatching(end));$
 if $L.size() > 0$ **then**
 | **forall** w in the L list **do**
 | $R.addAll(PS(G, w, start, end))$
 end
 $R.add(v);$
 return R ;
 end

 $L.addAll(v.fatherNodesMatching(end));$
 if $L.size() > 0$ **then**
 | **forall** w in the L list **do**
 | $R.addAll(PS(G, w, start, end))$
 end
 $R.add(v);$
 return R ;
 end
end

7.2.5 Topology Graph - XSD schema

The physical topology is represented as graph, in particular as previous specified, it is a tree structure. The XSD schema is used to define the structure of each element that takes part in this representation and to delineate constraints.

The XML file allows to specify the type of VIM = {OPENSTACK, KUBERNETES} which is running on the machines, the metric type to be optimized `performanceType = {LATENCY, BANDWIDTH, CPU_CONSUMPTION}` then a list of nodes.

```
1 <xsd:element name="Deployment">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element name="Host" type="HostElement" minOccurs="1"
5         maxOccurs="unbounded"/>
6     </xsd:sequence>
7     <xsd:attribute name="VIM" type="VIMElement" use="required"/>
8     <xsd:attribute name="PerformanceType" type="PerformanceTypeElement"
9       default="LATENCY"/>
10  </xsd:complexType>
11 </xsd:element>
```

Each graph element is called “Deployment” and it is composed by a set of *node* elements, each one represents a physical machines in the cluster. Each *node* is characterized by:

- *IP*: it is the IP address of the server;
- *SecurityUserName*: it is the the username of the network admin manager;
- *NetworkNode*: it is a boolean variable in which the *true* value means that this node is responsible for the networking, otherwise it is only a compute node;
- *coefficient*, *offset*, *maxNumRules*: they are parameters for the optimization model;
- *switch list*: it is a list of virtual switches.

```
1 <xsd:complexType name="HostElement">
2   <xsd:sequence>
3     <xsd:element name="vSwitch" type="vSwitchElement" minOccurs="1"
4       maxOccurs="unbounded"/>
5   </xsd:sequence>
6   <xsd:attribute name="SecurityUserName" type="xsd:string" use="required"/>
7   <xsd:attribute name="IP" type="IPElement" use="required"/>
8   <xsd:attribute name="NetworkNode" type="xsd:boolean" use="required"/>
9   <xsd:attribute name="coefficient" type="xsd:decimal"/>
10  <xsd:attribute name="offset" type="xsd:decimal"/>
11  <xsd:attribute name="maxNumRules" type="xsd:integer"/>
12 </xsd:complexType>
```

Each *vSwitchElement* it is a node of the graph that is characterized by the same elements of the node *HostElement*, but it contains a list of different node types, *vRouterElement* and *NodeElement*, furthermore it has a type that could be chosen from the enumeration `vSWTypeElement = {OVS, LINUXBRIDGE}`.

The former is a virtual router that could be instantiated only in a “NetworkNode”, it is characterized by:

- *Label*: it is the identifier of the router, which must be unique;
- *interface list*: a list of interfaces, each one with an IP address.

```
1 <xsd:complexType name="vRouterElement">
2   <xsd:sequence>
3     <xsd:element name="Interface" type="InterfaceElement" minOccurs="1"
4       maxOccurs="unbounded"/>
5   </xsd:sequence>
6   <xsd:attribute name="Label" type="xsd:string" use="required"/>
7 </xsd:complexType>
```

The latter is called “NodeElement”, which is the basic element of the topology, it is characterized by its type `NodeTypeElement = {VM, POD}` and it represents the unit where the services will be deployed on.

Each *NodeElement* is characterized by:

- *NodeType*: it represents the type of the service node;
- *IP*: it is the IP address of the server;
- *SecurityUserName*: it is the the username of the network admin manager;
- *coefficient*, *offset*, *maxNumRules*: they are parameters for the optimization model;

```
1 <xsd:complexType name="NodeElement">
2   <xsd:attribute name="NodeType" type="NodeTypeElement" use="required"/>
3   <xsd:attribute name="SecurityUserName" type="xsd:string" use="required"/>
4   <xsd:attribute name="IP" type="IPElement" use="required"/>
5   <xsd:attribute name="coefficient" type="xsd:decimal"/>
6   <xsd:attribute name="offset" type="xsd:decimal"/>
7   <xsd:attribute name="maxNumRules" type="xsd:integer"/>
8 </xsd:complexType>
```

A complete example of an XML file representing a physical topology is provided in the Listing 7.2 and the picture (Figure 7.3) shows the graphical representation.

```

1  <Deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml_physical_network.xsd" VIM="OPENSTACK">
2
3    <Host NetworkNode="true" IP="192.168.0.102" coefficient="6.87"
        offset="275" SecurityUserName="admin">
4
5        <vSwitch External="false" Label="SW1" vSWType="OVS" coefficient="0.5"
            offset="100">
6
7            <Node NodeType="VM" IP="10.0.0.56" SecurityUserName="admin1"
                coefficient="7" offset="300"/>
8            <Node NodeType="VM" IP="20.0.0.38" SecurityUserName="admin2"/>
9
10           <Router Label="R1">
11               <Interface Name="eth0" IP="10.0.0.1"/>
12               <Interface Name="eth0" IP="20.0.0.1"/>
13           </Router>
14
15           <Router Label="R2">
16               <Interface Name="eth0" IP="30.0.0.1"/>
17               <Interface Name="eth0" IP="40.0.0.1"/>
18
19           </Router>
20       </vSwitch>
21
22       <vSwitch External="true" Label="SW-EXT" vSWType="LINUXBRIDGE"
            maxNumRules="3000"></vSwitch>
23 </Host>
24
25 <Host NetworkNode="false" IP="192.168.0.103" SecurityUserName="admin">
26
27     <vSwitch External="false" Label="SW2" vSWType="OVS">
28
29         <Node NodeType="VM" IP="30.0.0.100" SecurityUserName="admin3"/>
30         <Node NodeType="VM" IP="40.0.0.3" SecurityUserName="admin4"/>
31
32     </vSwitch>
33 </Host>
34
35 </Deployment>

```

Figure 7.2. XML file describing a physical topology.

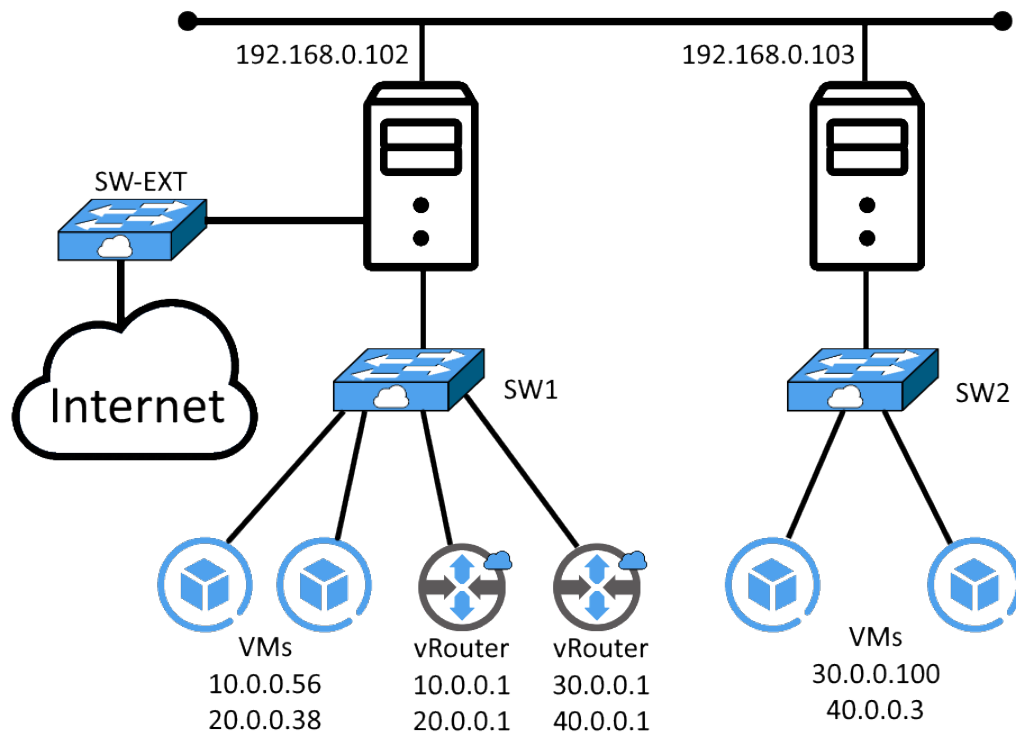


Figure 7.3. Graphical representation of the physical topology described in XML file.

7.3 z3 Prover

7.3.1 Introduction to z3

The z3 tool is a *Satisfiability Modulo Theories* (SMT) prover [33] developed by Microsoft Research and released as an open source project. It is useful to solve all the problems that fall under the *Satisfiability Modulo Theories* (SMT) problem which are a subsection of the *Boolean Satisfiability* (SAT) problem. It integrates different decision procedures and additional theories as equality reasoning, arithmetic, fixed-size bit-vector and quantifier.

It exposes different APIs in several programming language like C, C++ and Java. In this thesis work the version 4.8.7 for 64-bit machines is implemented.

The framework z3 receives a set of boolean and arithmetic formulas through its interfaces and it translates them into a SMTLIB2 file, in this way it provides a common background for all the SMT theories described in the Figure 7.4. The first operation is apply tactics like “Preprocessing” or “Cube & Conquer”, this avoid the computational time to explode and to discard suboptimal solutions. After this operation, z3 chooses and exploits a specific solver (e.g. SAT, SMT) to get a solution, or the best solution if the problem asks for an optimal result.

For the optimization phase, this solver provide the optimizer engine called *z3Opt*.

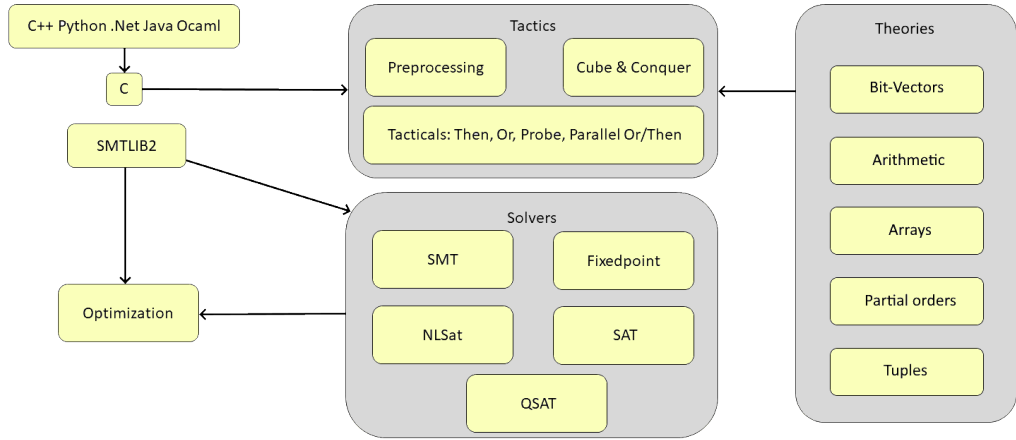


Figure 7.4. Z3 architecture.

7.3.2 SAT problem

The SAT (propositional satisfiability) problem, is a well-known problem in computer science. The task of a SAT problem is to decide if a specific combination of boolean variables that compose a set of propositional formulas exists to assurance the satisfiability of the formulas. This means that a problem could be solved in different ways and it could exist more then one solution. The SAT problem finds any solution that allows the problem to be satisfiable.

This type of problem is *non deterministic polynomial time*, which means, despite the correctness of the formulas could be evaluated in a polynomial time, the time to find a solution increases with a non polynomial shape with the size of the problem. So it can be classified as *NP-complete*.

7.3.3 SMT problem

The SAT problem is limited to boolean logic, so to represent others theories as integers, real numbers, lists and others, a generalization is done with the SMT problem. *Satisfiability Modulo Theories* is an instance of a Boolean satisfiability problem in which some the binary variables are replaced by predicates over a set of non-binary variables [34].

This allows to express more complex problems that can exploit, instead of boolean propositional formulas, a set of binary predicates made of not boolean variables. This made the SMT language more expressive then the SAT boolean language.

The structure depicted in Figure 7.4 shows that z3 tool offers SMT solver with different efficient search methods exploiting pruning algorithms and also heuristic combinations of methods called tactics.

7.3.4 MaxSMT problem

In computational complexity theory the *Maximum Satisfiability Modulo Theories* (MaxSMT) problem is defined as an extension of the SMT problem, where the goal is not just to find a solution, which guarantees the problem to be satisfiable, but the best solution which optimizes the variable values to achieve the maximum satisfiability of the clauses.

The main difference between the simple SMT problem is that each clause has a weight, so the consequence is to find a solution which, among all the satisfiable ones found, maximizes the number of satisfied clauses with the minimal cost.

In the the context of optimization the MaxSMT problem allows different clauses to be expressed as:

- *soft constraints*, they are relaxable clauses ($s_i \in S$), which means they could be not satisfied but with a “penalty”;

$$\text{maximize } \sum_{i=1}^S w_i * s_i$$

where w_i is the weight of the clauses and the s_i is the soft constraint to be satisfied;

- *hard constraints*, they are clauses that must be satisfied.

$$\text{subject to } h_j, \text{ with } \forall j \in [1, H]$$

The z3 tool could be used as library in a Java project as the following source code (Figure 7.5), given as output the values of the binary variables as shown in Figure 7.6.

z3 source code

```
1 Context ctx = new Context();
2 Optimize opt = ctx.mkOptimize();
3
4 // boolean variables creation
5 BoolExpr b1 = ctx.mkBoolConst("b1");
6 BoolExpr b2 = ctx.mkBoolConst("b2");
7 BoolExpr b3 = ctx.mkBoolConst("b3");
8 BoolExpr b4 = ctx.mkBoolConst("b4");
9
10 BoolExpr expr1 = ctx.mkAnd(ctx.mkOr(b1, b2), b3);
11 BoolExpr expr2 = ctx.mkNot(b4);
12
13 // soft constraints
14 opt.AssertSoft(b1, 1, "group");
15 opt.AssertSoft(b2, 3, "group");
16 opt.AssertSoft(expr2, 5, "group");
17
18 // hard constraints
19 opt.Assert(expr1);
20 opt.Assert(expr2);
21
22 // model computationg
23 opt.Check();
24 Model model = opt.getModel();
```

Figure 7.5. z3 used as library in a Java project.

z3 output

```
(define-fun b1 () Bool true)
(define-fun b2 () Bool true)
(define-fun b4 () Bool true)
(define-fun b3 () Bool true)
```

Figure 7.6. z3 outcome after the computation of the optimizazion model.

7.4 Optimization Development

7.4.1 Model implementation

The optimization model described in the Section 6.5 represent a SAT Problem, build of boolean variables and equations. Since the SAT problem is NP-complete, only algorithms with exponential complexity are known for it and a lot of efficient software were developed to solve these types of problems. In this thesis the *z3 Prover* it is chosen to be the engine of the optimization process.

The optimization process is made of different steps, each one has a representation in the model defined in Section 6.5. The process is implemented in `it.polito.policyorchestration.impl.optimization.distribution` package.

7.4.2 Binary equation

- The first step is the creation of boolean variables which values represent if a specific rule will be present in a filtering point or not.
- The second step is the creation of the boolean equation that defines all the necessary rules that have to be enforced. The task of this part is to find the rules and policy anomalies and to delete them.

Algorithm 5: optimization model - vertical equation

```

loop1:
forall fp in filtering points list do
    loop2:                                     ▷ find generalization rule
    forall rule1 in rule list do
        rule2 ← findIntersectingRule(rule1);

        if rule1.action ≠ rule2.action then
            for rule3 in rule list between rule1 and rule2 do
                rule3 ← findSuperSetRule(rule1);

                if rule1.action = rule2.action then
                    mark: rule1 as “redundant”;
                    go to loop2
                end
            end
        end

        for rule4 in rule list between firstRule and rule1 do           ▷ find preceding superset
            if rule1 ⊆ rule3 then
                mark: rule1 as “shadowed”;
                go to loop2
            end
        end

        mark: rule1 as “necessary”;
    end
end
end

```

7.4.3 Hard constraints computing

- The third step is about adding the constraints to the problem, which is to say make sure that the number of rules injected in each filtering point, does not exceeded the hard constraints given.

Algorithm 6: optimization model - maximum number of rules

```

begin
  forall fp in filtering points list do
    counter ← 0;
    forall rule1 in rule list do
      if rule1 is marked as "necessary" then
        counter ← counter + 1;
      end
    end
    max ← maxNumberOfRulesSupported(fp);
    Set constrain: counter ≤ max;
  end
end

```

7.4.4 Cost function

- The fourth step is the building of the cost function, which exploits the **performance tool** to retrieve the performance cost of each filtering point. At this point the STM problem is defined and complete.

Algorithm 7: optimization model - cost function

```

begin
  costSum ← 0;
  forall fp in filtering points list do
    currentCost ← getPerformanceCost(fp)
    costSum ← costSum + currentCost
  end
  Set: costFunction ← costSum
end

```

- The last step has two tasks, the first one checks if the model is *SATISFIABLE*, if it is not the administration will be notified. The second one is to compute the optimization by minimizing the cost function and set for each filtering point the security rules.

At the end of this process each filtering point has an minimal ordered list of rules that will be injected on the real or virtual devices behaving as firewall.

Chapter 8

Testing

This section develops the testing phase of the implemented solution, it is focused on one of the most important and critical aspects regarding security services and, in general, any network services, the time.

The tests have been carried out in different scenarios to detect how the tool execution time change due to different parameters given in input. Scalability is a crucial characteristic to understand the usability of the framework. In particular tests' scope is focused on the evaluation of the optimal distribution model to highlight the best features and to understand the most critical parameter that effects the scalability. Moreover this model, as it is discussed in Section 6.5, has a great role in the inter-policy analysis, for this reason some comparison tests are done to compare the policy services exploited by the *policytoollib* and the optimization model.

Four test cases with different metrics have been identified as critical and interesting for the scalability evaluation:

- number of entities hosted in the cloud environment;
- number of physical hosts which make the cloud infrastructure;
- number of virtual machines instantiated and running;
- number of rules to be enforced.

All the performance evaluations are done on a physical machines with the following software and hardware characteristics:

- **TYPE:** x64 architecture
- **CPU:** Intel(R) Core(TM) i7-4720HQ, Quad-core @ 2.60GHz
- **RAM:** 2x8 GB, DDR3 @ 1600 MHz
- **OS:** Windows 10

8.1 Scalability test

The purpose of the following tests is to progressively to increase one of the previous metrics, fixing the others and evaluate the execution time.

The tool tested has several components, from the input reading to the rule injection on the devices, the optimization process is the scope of these tests. This means the following results and measures refers to the execution time used to create the distribution model and to optimize it.

The goal is understand how the different inputs and scenarios, that the tool may face, affects on the scalability and usability of the system itself.

The scalability test is divided into three subsection:

- *entities test*: it evaluates the first metric previous described;
- *infrastructure test*: it evaluates the second and the third metrics, about physical host and virtual machines;
- *rules test*: it measures the last metric, the one which evaluate the performances based on the number of filtering rules.

8.1.1 Entities test

This test evaluate the scalability of the optimization model when the number of entities increase from one entity to five hundred entities.

The test is developed with three virtual machines instantiated for each entity, which defines five rules in its security policy and the cloud infrastructure is composed by three physical hosts. The figure 8.1 shows that the execution time is almost linear and the documented values are very low, approximately from $\sim 50ms$ to $\sim 275ms$.

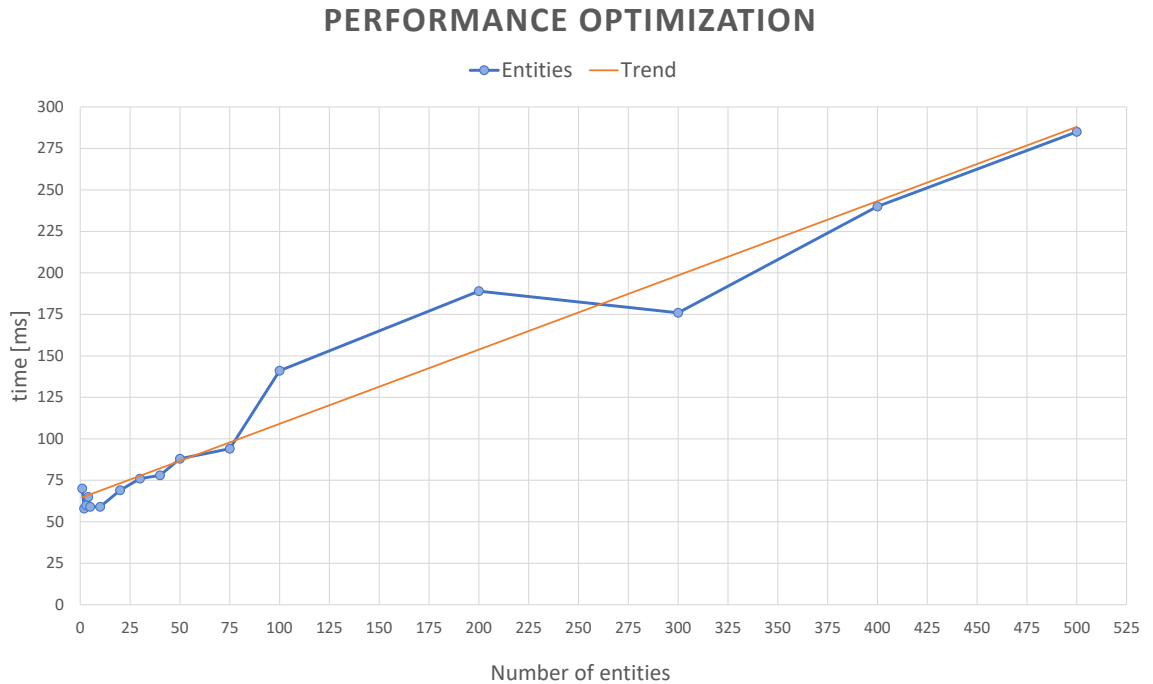


Figure 8.1. Scalability based on number of entities

This test highlights excellent framework scalability as expressed in the number of organizations. This is an important result, because cloud infrastructure are extremely growing environment, in which each day new companies move their business in this solution.

8.1.2 Infrastructure test

The second test points to analyse the execution time under the cloud infrastructure point of view. The test wants to highlight:

- horizontal scalability, which relates to the number of physical hosts;
- vertical scalability, which refers to the number of virtual machines.

These two tests are developed fixing three entities, with three rules defined in their security policy. The other two parameters expressed as number of virtual machines and number of physical hosts will change.

Horizontal scalability

The first test is done thinking about horizontal scalability in which a datacenter presents from one server to five hundred of physical machines, which represents a middle size cluster.

The trend displayed into the graph (Figure 8.2) presents a perfect scalability, because its trend is a straight line with a small slope. Performance degrades linearly with the increase of the number of physical hosts, starting from $\sim 100ms$, it reaches $\sim 600ms$ with a half thousand of machines.

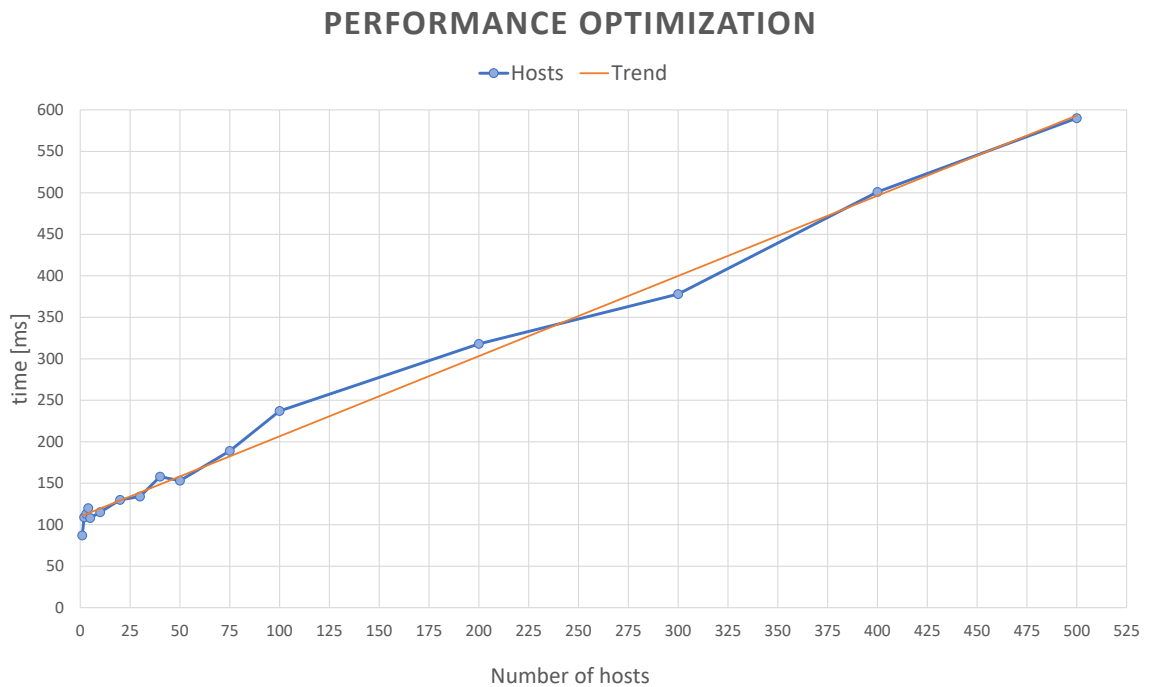


Figure 8.2. Scalability based on number of physical hosts

The trend of cloud clusters is to increase the number of physical servers that compose the cluster itself. The graph shows a ideal flexibility of the optimal distribution model which is feasible also of big datacenters with thousand and thousand of servers.

Vertical scalability

Otherwise the second test is about vertical scalability, that means the number of physical servers are fixed to three and the number of virtual instances for each entities increase from one to six hundred. The picture depicted below (Figure 8.3) displays that the scalability trend is a bit different from the previous ones. Its line is between a linear shape and an exponential shape, though it is not so evident.

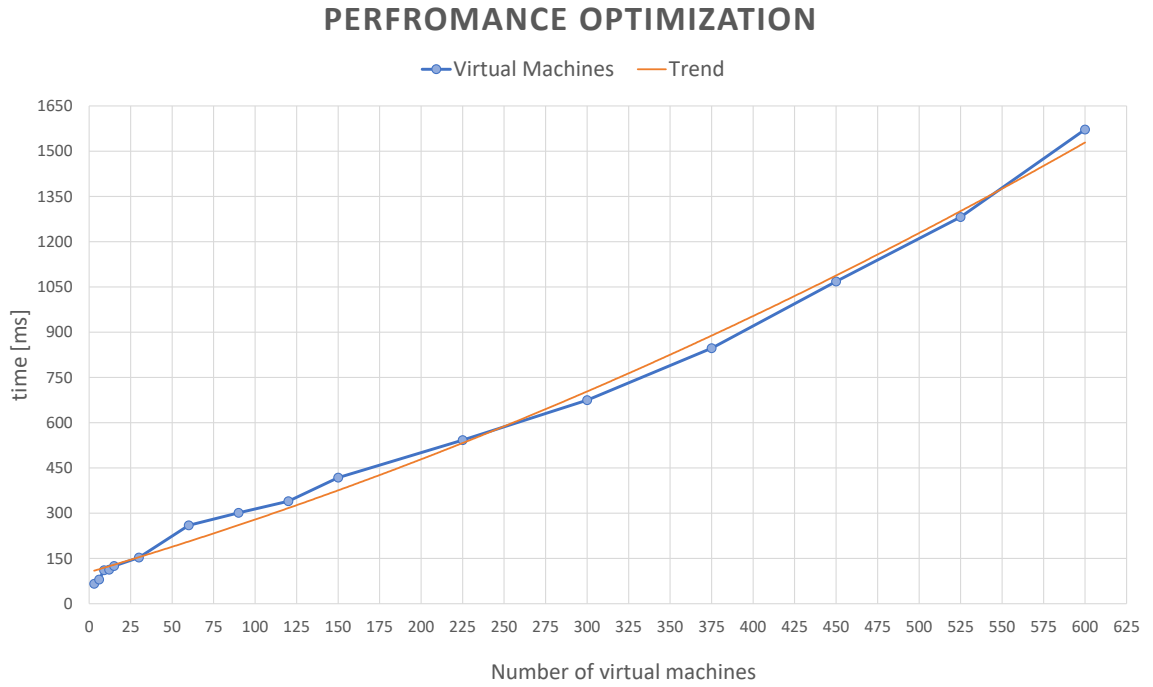


Figure 8.3. Scalability based on number of virtual machines

Cloud solutions as OpenStack, the one used as testbed, are strongly based on the dynamism of the environment. This means that to create and to destruct virtual machines are high frequent operations performed everyday.

The figure shows a great efficiency if the number of virtual machines grow. This result is meaningful for the scalability of the framework, because the test highlights that in a use-case scenario the optimization model is able to analyse in a limited amount of time and to locate rules also when the number of destination in the infrastructure is huge.

8.1.3 Rules test

The last scalability test wants to analyse the tool scalability based on the number of rules enforced.

It is developed enforcing an increasing number of rules, from one filtering rule to one thousand and five hundred rules. They are defined in such a way that the intra-policy and inter-policy analysis cannot delete any of those rules. Each filtering rule is a subset of the previous one, but it performs the opposite action, in this way they fall under the generalization anomaly.

The rules are injected in a cloud deployment made of three entities, three virtual machines for each entity and three physical hosts for the infrastructure.

The evaluation shown by the picture (Figure 8.4) highlights that the intrinsic computational cost of the MaxSMT problem is exponential in function of the number of rules to be injected. The computational is not negligible, because at a hundred and five thousand rules, remembering that the rules are divided among three entities, the computational time is $\sim 625s$.

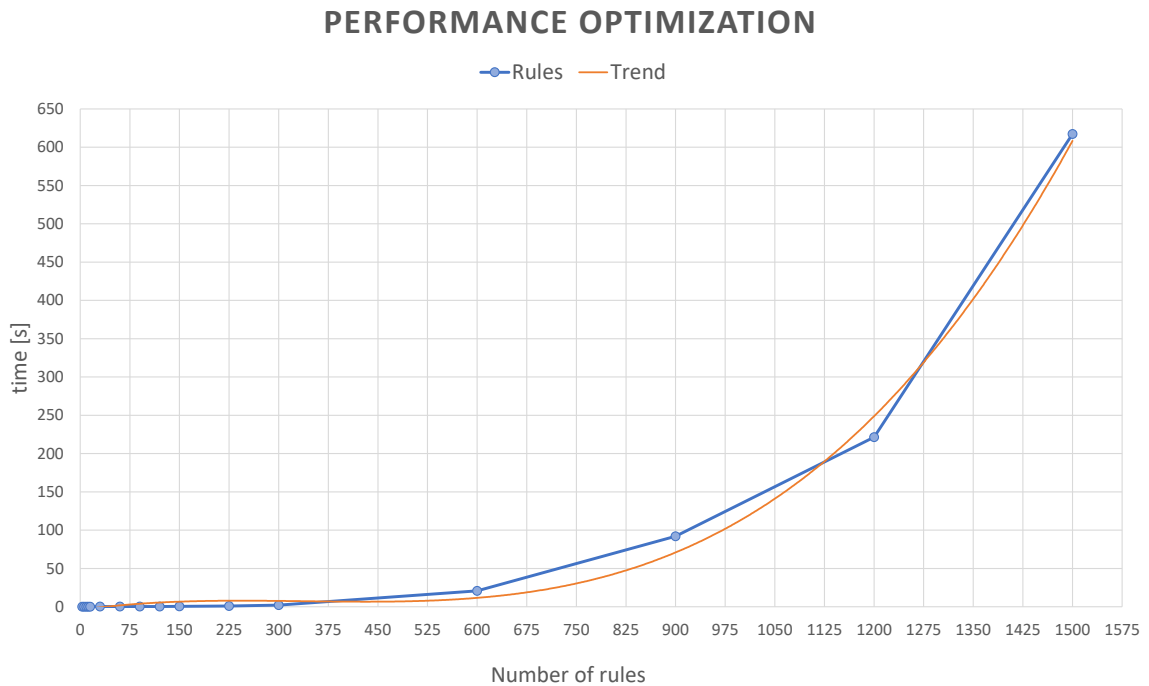


Figure 8.4. Scalability based on number of rules

This result is fundamental, because it highlights what is the most critical scalability aspect. The optimization model has increased its execution time of one order of magnitude compared to the computational time due to the previous tests.

It is also necessary to underline that in a real use case it unusual to have that huge number of rules for each entities, which cannot be optimized by the policy analysis.

8.2 Comparison test

This section regards the comparison tests between the inter-policy and intra-policy analysis, done by the *Policy Tool Library* and the distribution process developed by this tool.

First of all scalability tests are done on the inter-policy and intra-policy conflicts analysis process, the former present three entities and for each entity the number of rules change as in the previous “Rules test” 8.1.3. The latter is following the structure of the “Entities test” 8.1.1 on which all the parameters are fixed and the number of entities change.

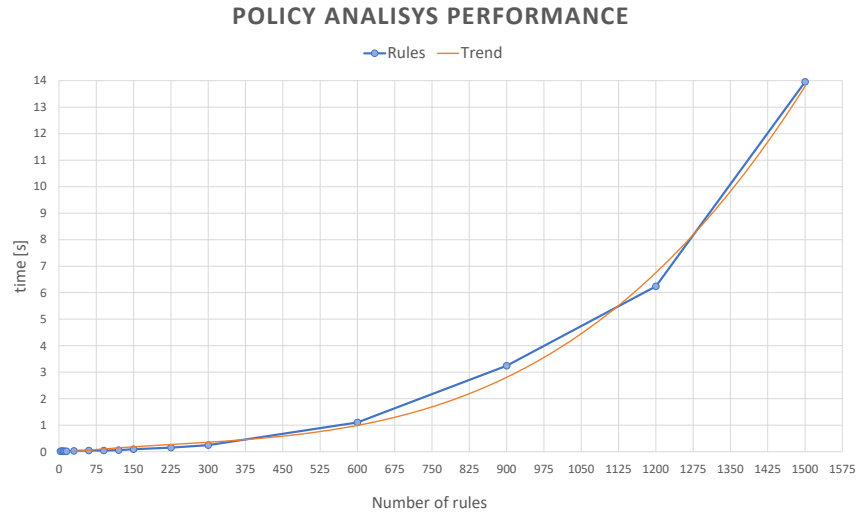


Figure 8.5. Scalability based on number of rules

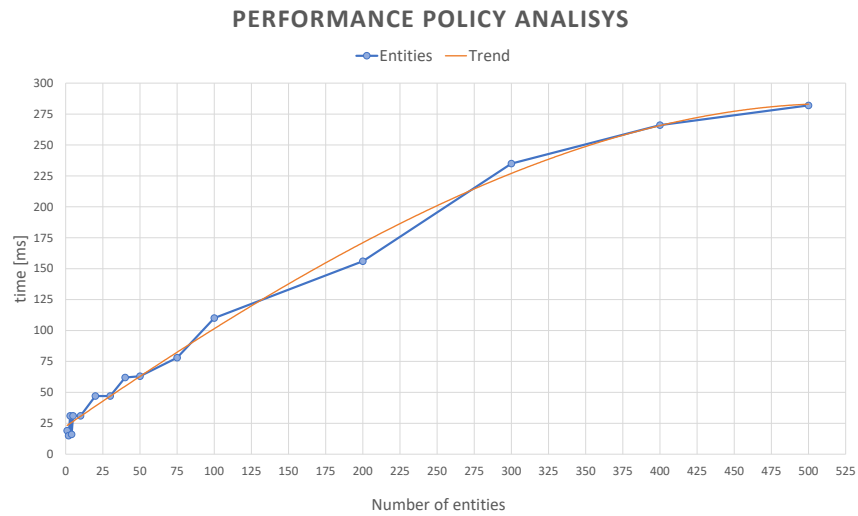


Figure 8.6. Scalability based on number of entities

These two tests are developed to highlight the scalability characteristics of first, the intra-policy analysis and second, the inter-policy analysis.

The first graph (Figure 8.5) points out that a huge number of filtering rules in the same policy can be challenging for the inter-policy analysis process, because the exponential shape expresses the same trend of the “Rules test” developed for the optimization process.

Otherwise the second test shows that the inter-policy conflicts analysis has a great scalability, because it seem to be a logarithmic shape in function of the number of entities.

This result is significant in a cloud context, because there is more dynamism regarding the organizations than the number of rules which are defined by them.

Ratio test

This last chart has the goal to compare the execution time of the optimization process related to how the policy conflicts analysis impacts on the overall execution.

As the picture depicted below (Figure 8.7) displays if the number of rules is low, the policy services exploited from the *policy tool library* has a significant weight on the computation cost, which it reduces its impact with the increase of the number of filtering rules.

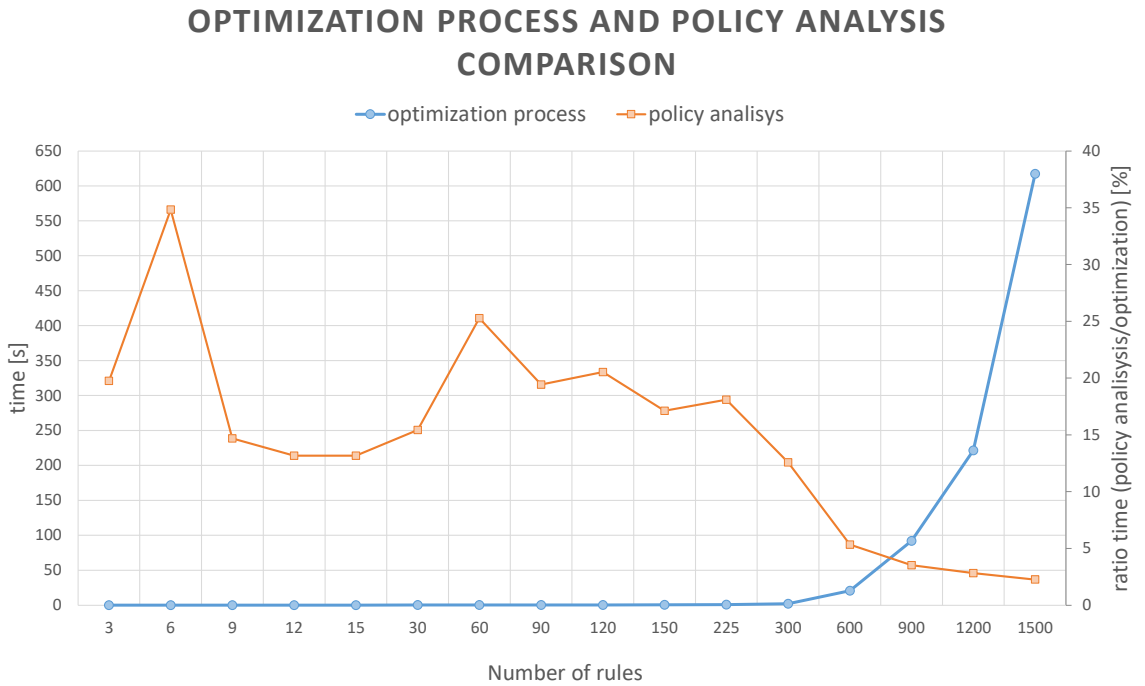


Figure 8.7. Scalability based on number of rules

This result leads to consider that for a limited number of rules (< 300), which corresponds to $< 2s$ of execution time, the external library is useless, because it burdens $\sim 20\%$ on the computation time. The same actions could be done by the optimization process. Otherwise when the number of rules increase until they reach > 1200 rules, it is displayed that the actions performed by the policy analysis are extremely efficient respect to the downstream process.

The previous graph (Figure 8.5) shows that the external library takes $\sim 14s$ to compute one hundred and five thousand rules, which is a better result than the one get by the optimization process, which spends $\sim 600s$ for the same task.

8.3 Use-case test

The previous test deeply analyses the scalability of the framework and highlights what parameters are critical for the computational time. One of the goal of this thesis was to be a practical work and not only has a theoretical approach, for this purpose this test is performed to show a real-case use of this tool to understand its feasibility and usability in a production environment. The evaluation is developed following with the guide lines described at the beginning of this chapter, which means to developed a stress test based on different parameters that affect the complexity of the problem.

A little time slot was spent to retrieve informations about the big cloud organizations, their infrastructure and their policies. The following test-bad was define for this performance test:

- *number of entities*: from 50 to 150 organizations;
- *number of physical servers*: from 10,000 to 20,000 units;
- *number of virtual machines*: from 500 to 30,000 instances;
- *number of security rules*: from 250 to 15,000 filtering rules.

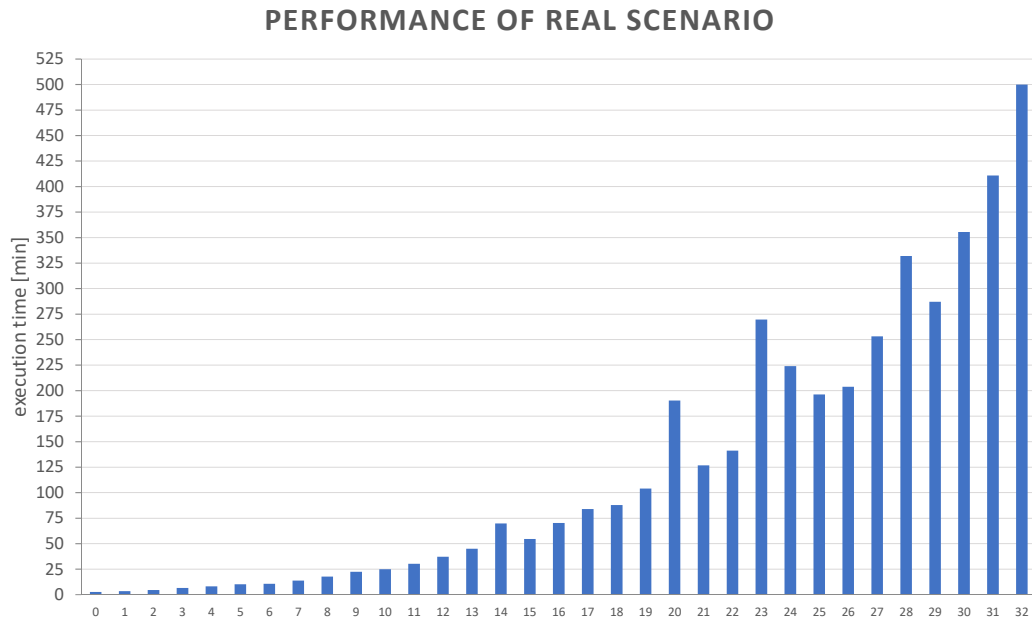


Figure 8.8. Scalability based on real case scenario

8.4 Test conclusions

The tests development highlights how the optimization process scales based on different inputs. The external integration of the policy library with the distribution framework gets excellent results, in term of performance and optimization.

Both of them, the optimization process and the policy tool library, are tested independently and in particular the first optimization performed by the “policy tool library” is necessary each time a huge number of rules are defined in a single policy. This avoids the optimization model to explode in the definition of condition clauses for the MaxSMT problem, which gets a lot of time to be solved.

Then the framework and in particular the optimization model have great scale features in a flexible environment as represented by a cloud infrastructure. Linear scalability with a small slope is a perfect characteristic for those environment, in which the dynamism is the main pillar.

Finally the test phase marks the number of rules as the main parameter on which both the external library and the optimization model scale worst. In this context future analysis and improvements can be done.

Chapter 9

Conclusion and Future work

9.1 Conclusion

The development of this thesis has required big commitment to achieve the goals, which are outlined in the Chapter 4. Nevertheless it was satisfactory to overcome the hard challenges faced and to get the results of this work, how they were reached is really rewarding.

The first phase was spent to improve in-depth theoretical notions, by reading literature and papers, about networks, virtualization and security policies, which give lots of information and useful knowledge.

A second phase was used to study and to model all the components that take some roles in a cloud environment using the XML schemas. This is a meta-language for the definition of markup languages and this allows to define object models and controls them through the description of constraints. Moreover the Java language supports XML language providing JAXB framework, which offers marshal and unmarshal functions between files and Java classes.

The design and the implementation of a security system, for the rule orchestration, makes this thesis practical, besides it deals with theoretical aspects due to the Research approach. This framework allows to express all the work done during the process of studying the technologies used, analyzing and understanding how they work with the purpose to build a system that is able to manage and to run in symbiosis with them.

The hard challenge of this work was to define the “optimal distribution model”, which means to formulate a set of propositional equations, made of boolean variables. A set of soft and hard constraints that has to be satisfied was established, in such way the model described the optimal rules allocation of a distributed firewall. The optimal model is a the MaxSMT problem, which is *NP-complete* problem. For this reason the *z3 Opt. Prover* project was integrated and exploited to solve the model in a limited amount of time.

The effectiveness and the validation of the optimization model has been proved by the implementation of performance tests. They show great improvements in terms of latency reduction, larger bandwidth and CPU consumption. Several JUnit tests were developed to validate the REST APIs and the interaction with the framework.

The testing phase highlights that the goal defined at the beginning have been successfully achieved. The tests have highlighted the fact that the distribution can take advantage by the resources already present in the cloud infrastructure, like the built-in filtering modules of OpenStack, instead of instantiating new ones. This saves hardware resources and improves performances.

Moreover, the integration with the policy tool library produces excellent results and it gives additional power to the system, not only from a security point of view, but also under the management perspective.

From this thesis work it comes out that the same networking flexibility can be reached also for security controls, without losing in performance if compared to physical appliances. Moreover, the automation of security and management operations get to better usage of resources and to an increase of the security level provided.

9.2 Future work

In the previous Chapter 5 and Chapter 6 the definition of the cloud infrastructure and the technologies used characterizes how the traffic flows and what are the filtering points and how they could be used.

The system is designed in different modules, each one is designed to be independent from the other ones and it is implemented in different levels. This opens towards the possibility to be extended or to be modified for future works.

9.2.1 VIM and lightweight virtualization

This thesis work is focused on OpenStack [26] toolkit as test-bed on which all the performance analysis are performed. This leads to characterize all the choices done during the design and implementation of the system, because the cloud environment available has a pre-defined structure in terms of communication, packet forwarding and virtual instances of switches and machines. Moreover all the efforts are spent on the Virtual Machines without the possibility to explore different virtualization techniques.

In particular it will be interesting to explore other types of Virtual Infrastructure Managers (VIMs) as Kubernetes, which does not present a fixed internal virtual network implementation, which can change based on the project chosen.

The picture depicted below (Figure 9.1) shows a Kubernetes project for network implementation and it is clear that the communication is different compared to the one implemented by OpenStack in Section 5.1.

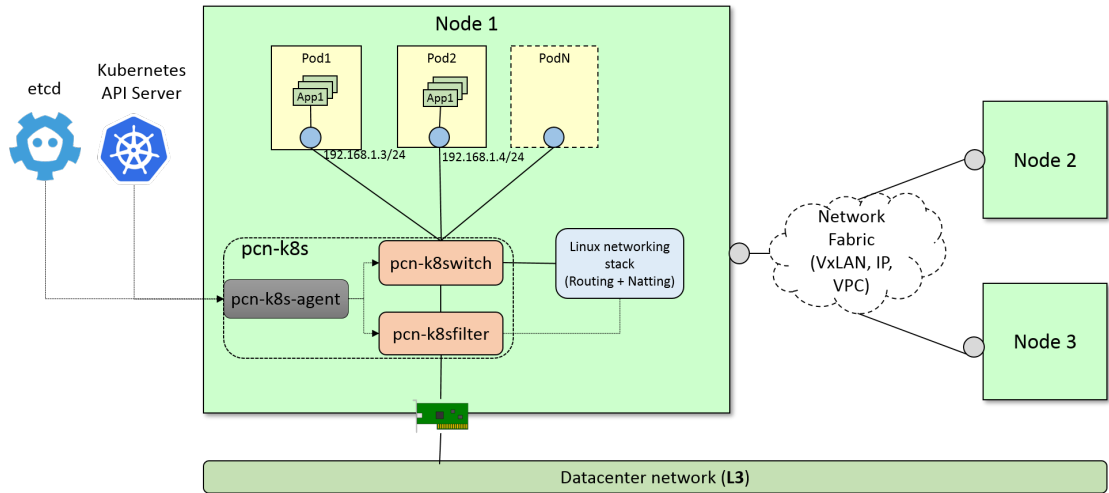


Figure 9.1. PolyCube network implementation of Kubernetes [35].

In addition OpenStack provides only Virtual Machines instances, unlike others VIM that runs lightweight virtualization as containers and dockers. These last two techniques are already supported in the physical model given in input to the system. The additional work is to study how the packets flow into the virtual network, to find all the new filtering points and to analyse them under a performance point of view.

9.2.2 Different metric analysis

The Section 5.2 exposes all the analysis performed on the cloud infrastructure to design a model of performance and optimization. Nowadays, with the increasing of real-time applications, the main metric, on which all the efforts will be spent, is the *delay*, because the Round Trip Time is critical in a networking and security field. On the other side in the last decade media applications spreads and the traffic load grows on network device. For these reasons also others metric as the *bandwidth* and the *CPU consumption* were taken into account.

Obviously the goal of this work was not to exploit all the metrics that could be optimized in a datacenter, but the efforts fell on the most relevant.

Future works may compute performance analysis on the *RAM consumption*, or in general on any *Hardware Device consumption* that, in this work, has not been evaluated.

Additional works could be done using specific forwarding hardware integrated into the NICs of the physical machines, this could improve the performance and so the previous evaluation will change.

9.2.3 New hard and soft constrains

The optimal distribution model defined in the Section 6.5 presents a list of hard constraints that must be achieved to make the model *satisfiable*, such as to minimize the number of rules. This hard constraint could be relaxed and it will be transformed into a soft constraint. In this way more than one instance of the same rule could be injected in different filtering points, which make a sub-optimal performance situation but it improves the security.

Physical and virtual network devices has a limited amount of resources, for this reason a “maximum number of rules” hard constraint is added for each filtering point, also this limitation could be relaxed as a soft constraint. Filtering devices are great resource consumption equipment, in terms of CPU cycles, RAM and electrical power that may be wanted to be limited. Future works could focus on the relation between the number of rules and the derived hardware consumption, this leads to add new constraints as resource limitation requirements that will be translated into maximum number of rules that a specific filtering point could be performed.

Appendix A

Programmer manual

In this section the tool architecture and all its feature are explained in details, with a code function presentation for each part of the tool. The code language chosen for the orchestration tool is Java and also all the additional tools are written in the same language, in this way they could be well integrated. The input and the output data are XML files and validated by XSD schemas, they are handled by API provided by JAVA for the XML Processing (JAXB). The architecture presents REST API to accept the input and to allow the administrator to retrieve the output of the rule distribution. In this tool there are external libraries from different projects:

- policy managment: *it.polito.policytoollib*, to use directly the source code;
- virtual switch managment: *floodlight.jar*;
- server: *tomcat-dbcp.jar*, *tomcat-embed-core.jar*, *tomcat-embed-el.jar*, *tomcat-embed-jasper.jar*, *tomcat-embed-websocket.jar*.
- REST DataBase management: *hk2-api-2.6.1.jar*, *hk2-core-2.6.1.jar*, *hk2-locator-2.6.1.jar*, *hk2-utils-2.6.1.jar*;
- Jersey and JAXB: *jaxrs-ri-2.30.1.jar*;
- SAT Solver: *com.microsoft.z3.jar*

A.1 Performance tool

The performance model tool is developed in the package `it.polito.policyorchestration.impl.optimization.performance` and it allows representing the performance of a filtering point in the deploy architecture. In the thesis' economy this class is useful to map each filtering point with a specific value that could be used to optimize the distribution of the rules all over the deployment.

Performance tool produce the output as the following workflow:

- reads the filtering point parameters;
- setups of the internal parameters (e.g. `coefficient`, `maxRules`);
- for each filtering point saves the number of the rules injected;
- creates the function of the performance based on the previous parameters
- computes the performance function through `getPerformance()` method and gives a number or an expression as output.

A.1.1 Tool architecture

The tool architecture is made up of an interface and an abstract class that implements the previous interface. It is extended by other three classes and there is an object factory to get the instances, the structure of the classes are the following:

- **class PerformanceToolFactory:** it is the class that handles the creation of the **PerformanceTool** instances, based on the metric chosen in the input files. It has no instance variables except the one that identify the metric used, but it defines an enum **public enum deviceType** that allows identifying what kind of filtering point the instance refers to. The Object Factory has the following methods:
 - **public PerformanceToolFactory(PerformanceTypeElement metric)**
constructor method that save inside the Object the metric used to evaluate the performance;
 - **public Performance getInstance(deviceType device, double offset, double coefficient, int nMaxRules)**
this method allows to create and to get a Performance instance in which it needs to be specified the type of the device, the maximum number of rules that are allowed to be injected, the offset and the coefficient. These last two variables are related to the performance function, the former is the value of the performance when there are no rules and the second one is the rate between the performance after 100 rules and the performance with no rules.
- **interface PerformanceTool:** it is the interface implemented by all the Performance Objects, the following methods are available:
 - **double getPerformance()**
it returns the number that represents the current performance of the filtering point;
 - **double getPerformance(int numRules)**
it returns the number that represents the future performance of the filtering points after additional **numRules** rules;
 - **ArithExpr getPerformance(Context ctx, ArithExpr numRules)**
it returns the arithmetic expression that is compute based on the arithmetic expression in input;
 - **void addNumRules(int numRules)**
it adds **numRules** rules at the Performance instance;
 - **boolean isFull()**
it returns a boolean value that represents if the filtering point cannot accept any additional rule;
 - **int remainsNumRules()**
it returns the possible number of rules that could be hosted by the Performance Object;
 - **deviceType getDeviceType()**
it returns the type of the filtering point;
 - **int compareDevice(Performance p)**
it compares the current instance to the **p** instance;
 - **int getMaxNumRules()**
it returns the maximum number of rules that the device could hosts.
- **abstract class PerformanceType:** this class is an abstract class, it defines all the instance variables in common between all the specific performance type. In this class are implemented all the function that allows the framework to interact with the Performance Object. The instance variables are:
 - **protected deviceType device**
it identifies the type of the filtering point, this allow filling with default value the following variables when they are not set in input files;

- **protected double coefficient**
it represents how fast the performance degrade based on the number of rules;
- **protected double offset**
it represents the initial performance of the filtering point when there are no rules;
- **protected int maxRules**
it is the maximum number of rules that is possible to host;
- **protected int currentRules**
it is the current number of rules that are hosted in the filtering point;
- **private static int MAXRULES**
it is a static variable that fix the maximum default number to 100000 rules.

This class has no possibility to represent a specific metric, because it handles only shared methods expressed in the interface, so it is needed to extend this class to implements the performance function.

- **class PerformanceLatency**: this class it represent an evaluation of the performance based on the latency in the communication. It defines default values for coefficient and offset, one for each **deviceType** and a linear function expressed in the two following methods:

```
* @Override public double getPerformance()
    P = coefficient/100 * currentRules + offset;
* @Override public double getPerformance(int numRules)
    P = coefficient/100 * (numRules + currentRules) + offset;
* @Override public ArithExpr getPerformance(Context ctx, ArithExpr
    numRules)
```

- **class PerformanceCPU**: this class it represent an evaluation of the performance based on the CPU consumption related to the number of the rules injected. As the previous class the performance is represented by a linear function that rises with the increase of number of rules. These are the methods implemented:

```
* @Override public double getPerformance()
    P = coefficient/100 * currentRules + offset;
* @Override public double getPerformance(int numRules)
    P = coefficient/100 * (numRules + currentRules) + offset;
* @Override public ArithExpr getPerformance(Context ctx, ArithExpr
    numRules)
```

- **class PerformanceBandwidth**: this class it represent the performance evaluation based on the bandwidth in the communication. Unlike the previous models it expresses an exponential function to represent the correct variations and so it needs another instance variable **bound** that is the infinite value that the function trend to reach with an infinite number of rules. The following methods are implemented:

```
* @Override public double getPerformance()
    P = (offset - bound) * exp(-coefficient/10000 * currentRules) + bound;
* @Override public double getPerformance(int numRules)
    P = (offset - bound) * exp(-coefficient/10000 * (numRules + currentRules)) +
    bound;
* @Override public ArithExpr getPerformance(Context ctx, ArithExpr
    numRules)
```


A.1.2 Modify the tool

It is possible to add different Performance evaluation, besides the ones considered for this work. Currently it supports the latency evaluation, the CPU consumption and the bandwidth, but it is possible adding classes that could evaluate others metrics as RAM and electricity consumption and so on. This is possible by the implementation of a class that extends `PerformanceType` class, adding to the XML schema a new element in the enum `PerformanceTypeElement` and adding in `PerformanceFactory`. `getInstance` the return case in the switch that choose the performance instance as

```
case RAM_CONSUMPTION:
return new PerformanceRAM(device, offset, coefficient, nMaxRules);
break;
```

A.2 Prover tool

The model tool is represented by the package `it.polito.policyorchestration.impl.optimization.solver` and it allows representing the model of distribution by a set of boolean variables and equations, moreover it provides an easy way to create boolean variables, equation, disequations and constraints. In others words it is an API to use the SAT Solver Optimize. The workflow of the Optimize tool is the following:

- creation of a set of boolean variable;
- creation of a set of equation, expressed as OR, AND and NOT operators;
- creation of a set of constrains as assertions;
- creation of a cost function and its value is minimized;
- computation of an optimized model and setting a value for each boolean variable.

A.2.1 Tool architecture

The tool architecture is made up of two classes, the first one is an handler for Z3 SAT Solver and the second one is a class that exposes useful functions to handles Z3 data types. The structure of the tool is the following:

- **class OptimizeTool:** this class is a wrapper for the solver and its purpose is to manage all the useful functions to create boolean variables, equation and cost functions. Moreover is in charge of minimize the cost function and create a model which set all the value to the boolean variables.

The instance variables are:

- **private Context ctx**
it is the context in which all the boolean variables and the equations are stored;
- **private Optimize opt**
it represents the Optimize Solver which stored the constrains and minimize the cost function;
- **private Model model** it the model that contains all the setted boolean variables after the optimization;
- **private Status status**
it is the status of the model, it could be `SATISFIABLE`, `UNKNOWN` or `UNSATISFIABLE`;

- `private HashMap<String, BoolExpr> equationMap`
`private HashMap<String, BoolExpr> boolVariablesMap`
`private HashMap<String, ArithExpr> realVariablesMap`
they are maps to associate a “name” to boolean variables and to equations, instead of using directly the Z3 Objects. In this way who is using the Optimize tool need to manage only `String` variables.

The following methods are the wrappers for the Z3 SAT solver functions:

- `public OptimizeTool()`
it is the constructor of the class, it initialized all the instance variables;
 - `public void createBooleanVariable(String bool)`
it allows to create a new boolean variable named `bool` in `Context ctx` that will be stored in `boolVariablesMap`;
 - `public void addConstraints(String... constraints)`
it allows to add an assertion to the `Optimize opt` variable;
 - `public void or(String equation, String... args)`
`public void and(String equation, String... args)`
`public void not(String equation, String arg)`
they are wrappers for the operators of the Z3 Solver, each method check if there is an equation already store in `equationMap`, if there is one the method add the `args` to the previous equation, if the method could not found it, it will create a new one;
 - `public void Eq(String equation, String left, String right)`
it creates an equation of the form `left = right`;
 - `public void Le(String equation, ArithExpr left, ArithExpr right)`
it creates an disequation of the form `left < right`;
 - `public void addReal(String realvariable, ArithExpr... args)`
it creates an arithmetic expression that represents a real variable. This method is used to create a cost function;
 - `public void minimizeRealVariable(String variable)`
it is the method that forces the Solver to optimize the variable passed in input;
 - `public void calculateModel()`
it checks the `Status` of the model and if it is `SATISFIABLE` it sets the model;
 - `public boolean getBooleanVariableResult(String booleanName)`
it retrieves the values of the boolean variable given in input its name.
- `class OptimizeFunctionHandler`: this class is not instantiatable and it provides additional functions useful for the tool, so it has no instance variables.
It structures as follow:

- `public static ArithExpr countTrue(OptimizeTool opt, List<BoolExpr> listBool)`
it create an arithmetic expression that represents the sum of the element in `listBool` list. Each `BoolExpr` has the value of 1 if it is `true` or the value of 0 if it is `false`;
- `public static List<BoolExpr> getMatchingBooleanVariable(OptimizeTool opt, String... matchingSubstring)`
it returns all the boolean expressions with their names contains all the strings passed as parameters;
- `public static boolean getBooleanValue(OptimizeTool opt, BoolExpr bool)`
it returns the value of the boolean expression;
- `public static ArithExpr convertIntToArithExpr(OptimizeTool opt, int n)`
it returns the arithmetic expression of the value passed as parameter;

- `public static String getPbooleanName(String ruleName, String filterinPointName)`
`public static String getSbooleanName(String ruleName, String filterinPointName)`
they creates the name for the boolean variables used in the tool;
- `private static ArithExpr countOneIfIsTrue(Context ctx, BoolExpr bool)`
it return an arithmetic expression that represent if the `bool` value is *true* it counts 1 or else it counts 0.

A.2.2 Modify the tool

Both of the classes could be extended, in the class `OptimizeTool` new wrapper methods could be added to represents new equation or disequation (e.g. $>$, \leq or \neq) that in this tool are not needed. The class `OptimizeFunctionHandler` is a class that exposes functions to manage the arithmetic and the bool expression, so any additional useful function could be placed in this class.

A.3 Optimal Distribution tool

The model tool is represented by the package `it.polito.policyorchestration.impl.optimization.distribution` and it allows the framework to distribute one equivalent firewall into many different distributed firewall in a physical and virtual deployment. It distributes the rules only on the nodes where the traffic could flows and it minimizes the number of rules injected using the Performance tool (A.1). Distribution tool produces the output follows the workflow:

- reads the rules to be injected;
- reads the available filtering points where the rules could be injected;
- for each rule compute all the possible path from each destination to each hosts that could generate packets;
- for each node on the path set that is a possible candidate to host the rule;
- create all the equation to be passed to a SAT solver;
- distribute the rule on each chosen filtering point.

A.3.1 Tool architecture

The tool architecture is made up of a class that handles the distribution of the rules. It exploits the features of another class used to represent the deployment architecture through an interface and the performance tool to optimize the distribution. The main class structure is the following.

- **RuleDistributorTool**: it is the class that handles the rule distribution, it knows only the set of rules and the set of filtering points associated to each rule. The class present the following methods:
 - `public RuleDistributorTool(InfrastructureManager tool)`
it is the constructor that has an `InfrastructureManager` as parameter, in which there are the rules and all the informations used to optimized the distribution;
 - `public void startOptimization()`
it is the method that gets all the available filtering points, it builds the paths for each rules, it creates the distribution model and than it optimizes that distribution.
 - `public void injectRules()`
it is the method that, for each filtering point available, calls the methods which inject the rules associated into it self;

- `private void findAllTheRulesForEachFilteringPoint()`
it is a method that for each rule find a path, which means finding all the filtering points, between the receiving point and the transmission point. In other words a rule could be matched by a traffic pattern, this method finds all the destination points for this type of traffic and for each one it finds where the rule must be placed or could be placed in a such way that the number of filtering point chosen is the minimum;
 - `private void createBooleanVerticalEquationsForTheModel()`
it is a method that builds a boolean equation for each filtering point. The purpose of this equation is to set a variable $\mathcal{P}_{n,i}$, which could be *true* or *false*, and it represents if the rule n must be injected on the filtering point i ;
 - `private void createBooleanHorizontalEquationsForTheModel()`
it is a method that builds a boolean equation for each rule. The purpose is to impose that if a rule, which as the same action of the default action, could be injected must be injected but at least one time;
 - `private void optimizeModel()`
it is a method that creates a real cost equation by adding all the performance costs of each filtering point. At the end this function minimize the model and the optimization is computed;
 - `private void distributeRulesAsModel()`
it is a method that reads the result of the computation of the optimizer and it sets for each filtering points their rules.
- **DeploymentStructure**: it is the interface for the deployment structure that is independent from the environment. It has two methods:
 - `LinkedList<DeploymentNode> getAllFilteringPoints()`
it returns all the filtering point of the deployment structure
 - `public LinkedList<TreeNode> getConditionClausePoints(ConditionClause c)`
it is a method that is useful to retrieve one or more filtering points and end-points from the tree structure.
 - **TreeStructure**: it is the class that handles the structure of the deployment and it implements the **DeploymentStructure** interface. It is in charge of modelling the physical and the virtual nodes and creating a tree where those nodes are placed. This class has the following methods:
 - `public TreeStructure(InfrastructureManager tool, LinkedList<IpSelector> internalSubnetList)`
it is the constructor of the class, the parameters are the **InfrastructureManager** tool used to retrieve the physical and virtual environment and the `LinkedList<IpSelector> internalSubnetList` that is a list of all the subnet present inside the deployment. This last parameter is useful to understand if the traffic pattern is from/to external points or internal ones;
 - `private void populateTree()` it is the method that builds the tree with the information of the **InfrastructureManager** tool;
 - **DeploymentNode**: it is the interface for the deployment node that is independent from the deployment structure used. It has two important methods:
 - `LinkedList<String> getORPointPath(ConditionClause arrival, ConditionClause departure, IpSelector subnet)`
it is the method that finds the minimal set of possible filtering points from the end-point arrival to any departure points. The last parameter `subnet` identifies the entity that owns the rule for which the method tries to find a path for the corresponding traffic pattern. This method is based on ten models represented in [Appendix B](#) that the single node could be, it chooses what kind of model it is;

- `public LinkedList<String> getANDPointsPath(ConditionClause arrival, ConditionClause departure, IpSelector subnet)`
it is the method that finds the minimal set of necessary filtering points from the end-point `arrival` to any `departure` points. The last parameter `subnet` identifies the entity that own the rule for which the method tries to find a path for the corresponding traffic pattern. This method is based on ten models represented in Appendix B that the single node could be, it chooses what kind of model it is;

- **TreeNode**: it is the class that represents a single node inside the tree structure and implements the **DeploymentNode** interface. All the **TreeNode** that are internal nodes are filtering points, on the contrary the leaf nodes are the services with an **IpAddress** and a **Port** associated. The class has the following instance variables:

- `private String label`
it identifies the node;
- `private TreeNode parent`
it is the variable pointer to the parent node, this is always not `null` except for the root element;
- `private List<TreeNode> children`
it is a list of the pointers to all the children of the current node, this list is always `notEmpty` except for the leaf elements;
- `private FilteringPoint filteringPoint`
it is the pointer to the physical or virtual filtering point of the **InfrastructureManager**;
- `private LinkedList<ConditionClause> childConditionClauseList`
it is a list of all the end-points under this node, which means they are all the end-points that are reachable from this node passing through its children;
- `private LinkedList<ConditionClause> fatherConditionClauseList`
it is the list of all the end-points above this node, which means all the end-points that are reachable from this node passing through its parent.
- `public TreeNode(FilteringPoint filteringPoint, boolean isGateway, ConditionClause externalConditionClause)`
it is the first constructor of the class, `filteringPoint` is the element of the deployment, `isGateway` identify if this node reaches directly the exterior of the infrastructure and `externalConditionClause` represents all the destination that are external to the deployment;
- `public TreeNode(String label, boolean isService, boolean isRoot, ConditionClause externalConditionClause)`
it is the second constructor of the class, it is useful because both root element and leaf elements are not related to any `filteringPoint` or could not be gateway;
- `public void addChildConditionClause(ConditionClause c)`
`public void addChildConditionClauseList(LinkedList<ConditionClause> cl)`
`public LinkedList<ConditionClause> getChildConditionClauseList()`
`private boolean hasChildrenConditionClause(ConditionClause c)`
they are methods that manage the instance variable `childConditionClauseList`;
- `public void addFatherConditionClauseList(LinkedList<ConditionClause> cl)`
`public LinkedList<ConditionClause> getFatherConditionClauseList()`
`private boolean hasFatherConditionClause(ConditionClause c)`
they are methods that manage the instance variable `fatherConditionClauseList`;

A.3.2 Modify the tool

It is possible to change the model tree of the deployment with other model, as chain or graph. Currently it supports only the tree model because it is the mostly used physical and logical topology that avoids possible loops, besides that the work related to this thesis is based on an OpenStack deployment. To change the model representation must add a new class that substitutes the `TreeStructure` class and implements the `deploymentStructure` interface. Then a new class is needed to substitute the `TreeNode` class that implements the `deploymentNode` interface. All the interface's methods must be implemented following the previous definition.

Appendix B

Path models

The distribution tool assigns a model to each deployment nodes, based on how the traffic passes through, it ends on or it starts from the node.

- model A: the node is an end point for the traffic, but it could not generate that type of traffic.

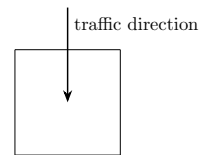


Figure B.1. Model A

- model B: the node is a start point for the traffic, but it could not be a destination for that type of traffic.

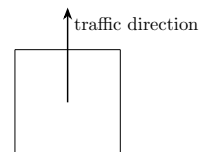


Figure B.2. Model B

- model C: the node is both a start point and an end point for the traffic.

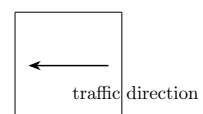


Figure B.3. Model C

- model D: the node is like “model C”, but the traffic could come from its self or from outside.

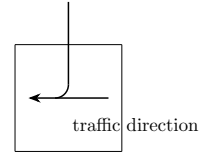


Figure B.4. Model D

- model E: the node is neither a start or an end point for the traffic, but the traffic comes from a node that is nearest to the root element and it goes to a node that is farther from the root element.

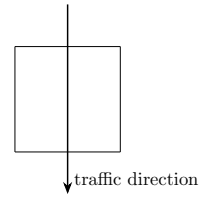


Figure B.5. Model E

- model F: the node is like “model E”, but the direction of the traffic is opposite, it comes from a sub element and goes to an above node.

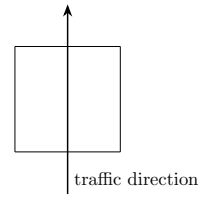


Figure B.6. Model F

- model G: the node can receive traffic from the nodes under it or above it and the destination is one of the child nodes.

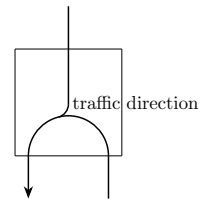


Figure B.7. Model G

- model H: the node is like “model F”, but it has two different sub element that could generate that type of traffic.

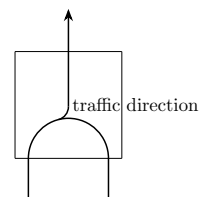


Figure B.8. Model H

- model I: the node receives the traffic from a sub element and the destination for the traffic is also for a node that is under this node, there is no destination above it.

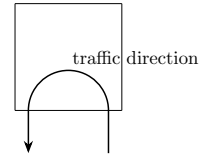


Figure B.9. Model I

- model L: the node is like model D", but it has two different sub element that could generate that type of traffic.

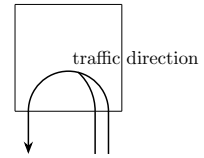


Figure B.10. Model L

Bibliography

- [1] NIST statistics, <https://nvd.nist.gov/vuln-metrics/cvss>
- [2] IBM Data Breach Costs, <https://newsroom.ibm.com/2019-07-23-IBM-Study-Shows-Data-Breach-Costs-on-the-Rise-Financial-Impact-Felt-for-Years>
- [3] S. Kent, R. Atkinson, "Taxonomy of Conflicts in Network Security Policies", IEEE Communications Magazine, Vol. 44, No. 3, March 2006, pp. 134-141, DOI [10.1109/MCOM.2006.1607877](https://doi.org/10.1109/MCOM.2006.1607877)
- [4] H Hamed, E. Al-Shaer, "Security Architecture for the Internet Protocol", RFC-2401 November 1998, DOI [10.17487/RFC2401](https://doi.org/10.17487/RFC2401)
- [5] IPv6Now, <http://www.ipv6now.com.au/primers/IPv6PacketSecurity.php>
- [6] E. S. Al-Shaer and H. H. Hamed, "Firewall Policy Advisor for anomaly discovery and rule editing", IEEE Eighth International Symposium on Integrated Network Management, April 2003, pp. 17-30, DOI [10.1109/INM.2003.1194157](https://doi.org/10.1109/INM.2003.1194157)
- [7] Firewall technologies definitions, https://security.polito.it/~lioy/02krq/firewall_en.pdf
- [8] H. Hamed, A. El-Atawy and E. Al-Shaer, "On Dynamic Optimization of Packet Matching in High-Speed Firewalls", IEEE Journal on Selected Areas in Communications, Vol. 24, No. 10, October 2006, pp. 1817-1830, DOI [10.1109/JSAC.2006.877140](https://doi.org/10.1109/JSAC.2006.877140)
- [9] Hazem H. Hamed, Ehab S. Al-Shaer, "Dynamic Rule-Ordering Optimization for High-Speed Firewall Filtering", Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, Taipei (Taiwan), March 2006, pp. 332-342, DOI <https://doi.org/10.1145/1128817.1128867>
- [10] Cataldo Basile, "Policy analysis and reconciliation", Report of SECURED, March 2016
- [11] C. Basile, A. Cappadonia, A. Lioy, "Network-Level Access Control Policy Analysis and Transformation", IEEE/ACM Transactions on Networking, Vol. 20, No. 4, August 2012, pp. 985-998, DOI [10.1109/TNET.2011.2178431](https://doi.org/10.1109/TNET.2011.2178431)
- [12] A. Bruno, M. Mendonca, X. Nguyen, K. Obraczka, T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", IEEE Communications Surveys & Tutorials, Vol. 16, No. 3, February 2014, DOI [10.1109/SURV.2014.012214.00180](https://doi.org/10.1109/SURV.2014.012214.00180)
- [13] H. Farhady, H. Lee, A. Nakao, "Software-Defined Networking: A Survey", Computer Networks, Vol. 81, April 2015, pp. 79-95, DOI [10.1016/j.comnet.2015.02.014](https://doi.org/10.1016/j.comnet.2015.02.014)
- [14] Software-Defined Networking: The New Norm for Networks, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>
- [15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus Networks", SIGCOMM Computer Communication Review, Vol. 38, No. 2, pp. 69-74, 2008.
- [16] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges", IEEE Communications Surveys & Tutorials, Vol. 18, No. 1, Firstquarter 2016, pp. 236-262, DOI [10.1109/COMST.2015.2477041](https://doi.org/10.1109/COMST.2015.2477041)
- [17] B. Han, V. Gopalakrishnan, L. Ji and S. Lee, "Network function virtualization: Challenges and opportunities for innovations", IEEE Communications Magazine, Vol. 53, No. 2, February 2015, pp. 90-97, DOI [10.1109/MCOM.2015.7045396](https://doi.org/10.1109/MCOM.2015.7045396)
- [18] Network Functions Virtualization (NFV), https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01_02.01_60/gs_NFV003v010201p.pdf

- [19] B. Yi, X. Wang, K. Li, S. k. Das, M. Huang, “A comprehensive survey of Network Function Virtualization”, *Computer Networks*, Vol. 133, 14 March 2018, pp. 212-262, DOI [10.1016/j.comnet.2018.01.021](https://doi.org/10.1016/j.comnet.2018.01.021)
- [20] Network Functions Virtualization (NFV) Security, https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/003/01.01.01_60/gs_NFV-SEC003v010101p.pdf
- [21] E. Kim, K. Kim, S. Lee, J. P. Jeong, H. Kim “A Framework for Managing User-defined Security Policies to Support Network Security Functions”, *IMCOM*, No. 85, January 2018, pp. 1-8, DOI [10.1145/3164541.3164569](https://doi.org/10.1145/3164541.3164569)
- [22] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, A. P.r Perales, “Adding Support for Automatic Enforcement of Security Policies in NFV Networks”,
- [23] IETF Working group, <https://www.ietf.org/how/wgs/>
- [24] J. Jeong, S. Hyun, T. Ahn, S. Hares, D. Lopez, “Applicability of Interfaces to Network Security Functions to Network- Based Security Services”, *I2NSF Working Group*, 18 March 2020, https://datatracker.ietf.org/doc/draft-ietf-i2nsf-applicability/?include_text=1
- [25] Interface to Network Security Functions (i2nsf), <https://datatracker.ietf.org/wg/i2nsf/about/>
- [26] Open Stack project <https://opensource.com/resources/what-is-openstack>
- [27] Nping project <https://nmap.org/nping/>
- [28] iperf3 - perform network throughput tests <https://www.systutorials.com/docs/linux/man/1-iperf3/>
- [29] top - display Linux processes <http://man7.org/linux/man-pages/man1/top.1.html>
- [30] iptables wikipedia definition <https://en.wikipedia.org/wiki/Iptables>
- [31] OVS wikipedia definition https://en.wikipedia.org/wiki/Open_vSwitch#cite_note-ibm-developerworks-3
- [32] Graph traversal definition https://en.wikipedia.org/wiki/Graph_traversal
- [33] z3 Prover source code <https://github.com/Z3Prover/z3>
- [34] SMT definition https://en.wikipedia.org/wiki/Satisfiability_modulo_theories
- [35] PolyCube project <https://polycube-network.readthedocs.io/en/latest/components/k8s/pcn-kubernetes.html>