POLITECNICO
DI TORINO

Master of Science in Computer Engineering

Master Degree Thesis

Optimized Configuration of Network Security Policies

Supervisors
prof. Riccardo Sisto
prof. Guido Marchetto
prof. Fulvio Valenza
dott. Daniele Bringhenti

Candidate
Luigi La Mattina

Academic Year 2020-2021

Summary

The emergence of new networking technologies, Network Functions Virtualization (NFV) and Software Defined Network (SDN) has brought significant improvements and greater flexibility in building the functions that make up a Service Function Chain. Thanks to NFV technology, each specific function such as NAT or proxy server is no longer constituted by an ad hoc hardware device, but is implemented on a standard server that can host multiple network functions at the same time. This allows you to make the most of the resources of the system and of the device itself, in fact to add a new device it is no longer necessary to purchase a new physical box but to implement it directly on the server, which will use the resources of the device by sharing them with other devices installed inside it.
Thanks to SDN technology, on the other hand, the possibility has been added to create specific routes for each type of traffic or user, so as to add greater flexibility in the construction of a Service Function Chain. In fact, it is possible to modify and customize the path performed by a packet, through the programming of a controller, which manages in real time the actual crossing of the packet between the various network devices.

A not negligible problem that arises in the creation of a Service Function Chain is given by the manual configuration of the network devices since an incorrect configuration can lead to very serious errors such as violating security or accepting unwanted traffic. This configuration can also cause significant latency due to the time required for updating or maintaining the security system. One solution is provided by Network Automation, as automating the configuration of network security devices would drastically reduce human errors and the latency of any device configuration changes.

An example of a framework that carries out a Network Automation work is given by VEREFOO (VErified REFinement and Optimized Orchestration), which given in inputs a Service Graph and some Network Security Requirements, generates a Service Graph in the output solution, which identifies a physical network, with the presence of some network security functions automatically allocated and configured in order to best meet the Network Security Requirements passed in input.

The main contribution of this thesis work concerned the allocation and automatic configuration of the Network Security Functions. In particular, the main contribution was the expansion and improvement of VEREFOO, a framework already able to satisfy the network security requirements by managing the allocation and configuration of Firewall and Channel Protection System separately, but not able to allocate and configure them simultaneously in the same iteration. Along with this innovation, aspects of the previous version of the framework have been

improved and all possible conflicts that may occur in the configuration of the two network security functions have been resolved. Finally, a translator was written to allow the configuration of the output VPN Gateways to be transformed into an input configuration for the StorngSwan platform so as to directly create Ipsec Channels.

At the end, performance tests were performed to verify the correctness and scalability of the framework in order to verify the improvements and highlight the factors that have the greatest impact on the scalability of the system. These tests were carried out by varying two input values, the Allocation Places and the Network Security Requirements and it was verified which of the two values has the greatest impact on the scalability of the framework.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Thesis objective

In recent years, the emergence of new network technologies, Network Functions Virtualization (NFV) and Software Defined Network (SDN) has brought significant improvements and greater flexibility in the construction of the functions that make up a Service Function Chain. In fact, thanks to the introduction of NFV technology, the ability to perform various network functions in standard hardware has been added, in order to obtain greater efficiency in the use of resources. Instead, thanks to SDN technology, the possibility of creating specific routes for each type of traffic or user has been added, so as to add greater flexibility in the construction of a Service Function Chain. Thanks to these improvements, each specific function such as NAT or the proxy server is no longer constituted by an ad hoc hardware device, but is implemented on a standard server that can simultaneously host multiple network functions. This allows you to make the most of the resources of the system and of the device itself, in fact to add a new device it is no longer necessary to purchase a new physical box but to implement it directly on the server, which will use the device's resources by sharing them with other devices installed inside it.

Having assimilated the new technologies, a not insignificant problem that occurs in the creation of a Service Function Chain is given by the manual configuration of the network devices and above all of the network security devices, since an incorrect configuration can lead to very errors serious as breaching security or accepting unwanted traffic. This configuration, in addition to the aforementioned problems, can also cause significant latency due to the time required for updating or maintaining the security system. A solution to these problems is provided by Network Automation, since automating the configuration of network security devices would drastically reduce human errors and the latency of any device configuration changes.

Starting from these innovations, the objective of the thesis was the expansion and refinement of the implementation of VEREFOO (VErified REfinement and Optimized Orchestration), a framework whose main purpose is the allocation and configuration of Network Security Functions, to fulfill some network security requirements. Specifically, in this thesis a module has been developed that automatically allocates the following functions:

- **Firewall:** to allow unwanted traffic to be discarded from the network respecting the Network Security Requirements, this action is carried out through a Packet Filter, a type of Firewall whose main purpose, after a preliminary control phase, is to allow or discard the unwanted traffic.
- **Channel Protection System:** to allow two Hosts to communicate in a protected manner respecting the Network Security Requirements, this is done through specific security functions called Gateway VPN, which have the task of opening and closing the secure communication channel.

These Network Security Functions are automatically allocated by the framework on a Service Graph, a graph that identifies the real composition of the network in question in which all the network devices are present except the security functions. Such an allocation, must not only satisfy the input network security requirements but also be the optimal total solution.

1.2 Thesis description

In **Chapter 1** starting from new technologies, the objectives to be achieved in this thesis have been introduced, these objectives are described in the rest of this thesis which is structured as follows:

- **Chapter 2:** in the initial part of the chapter the concept of Service Function Chain (SFC) is introduced, that is a chain composed of several network functions that provide different services, which represents the theory behind the creation of a Service Graph. Subsequently, after having highlighted the main limitations of an SFC, to overcome them the two new technologies Software Defined Network (SDN) and Network Function Virtualization (NFV) are introduced, in which the main innovations are described and the advantages that these technologies bring to the construction of a Service Function Chain (SFC).
- **Chapter 3:** in the initial part of this chapter VEREFOO is introduced, an example of a framework that implements the Service Function Chain approach improved by SDN and NFV technologies, for which this thesis has contributed to the development. After providing an overview of the framework, the second part of the chapter introduces the two graphs used by VEREFOO, Service Graph and Allocation Graph. Finally, the last part of the chapter focuses on the description of the Network Security Requirements used in this thesis work, describing the model and diversity through implementation examples in the XML schema.
- **Chapter 4:** the configurations of the two security functions used in this thesis work, namely the Firewalls and the Channel Portection Systems, are described. The first part of the chapter describes the problems due to a

manual configuration of the two security devices with any solutions due to an automatic configuration. Subsequently, a configuration example is presented in the XML schema for each of the two security functions. Finally, in the final part of the chapter the possible conflicts that can occur between the two functions are introduced, so that network security administrators can manage them in order to eliminate any network vulnerabilities.

- **Chapter 5:** this chapter introduces the main objective of this thesis work and how it was achieved. The first part of the chapter explains how the main objective has been divided into minor objectives, the achievement of which is essential for achieving the overall objective of the thesis. Finally, the three objectives are described in detail, presenting the possible approaches followed to achieve them.

- **Chapter 6:** this chapter presents the implementation of the first part of this thesis work, that is the reformulation of three constraints of the tool that performs the allocation and configuration of the VPN Gateway. In the first part of the chapter a brief description of the objective to be achieved is made and the constraints to be remodeled are described. Subsequently, the reformulated constraints are presented with the respective formulas and finally for each constraint, the operation performed is shown by means of examples.

- **Chapter 7:** this chapter describes how the second objective of this thesis work was achieved, namely the expansion of the VEREFOO ADP module, making it able, at the same time, to perform an automatic Firewall allocation and configuration and Channel Protection System, after having received in input a series of requirements to be respected, following optimality criteria through the resolution of a MaxSMT problem. The first part of the chapter describes and explains the choices made to achieve the main objective. Finally, the final part of the chapter describes how the configuration conflicts of the Network Security Functions have been overcome due to the presence in the network of a different Security Function or to the presence of a NAT function.

- **Chapter 8:** this chapter describes how the third objective of this thesis work has been achieved, namely the writing of a translator who performs the translation from the VEREFOO output language to the language accessible from the StrongSwan platform. The first part of the chapter gives an overview of the most important parameters present in the ipsec.conf file, a fundamental file for configuring Channel Ipsec using StrongSwan. Finally, in the final part of the chapter, through examples, it is described how the translator, starting from the configuration of a VPN Gateway in the VEREFOO output, creates the ipsec.conf configuration file.

- **Chapter 9:** in this chapter the real functionality of the framework is verified and the computational performance is tested as the various input parameters vary. In the first part of the chapter the correctness of the implementation is verified by means of specific tests where given the inputs the obtained

outputs are verified. Finally, in the second part of the chapter, the computational performance of the framework is tested after the changes made in the implementation of this thesis work and the parameters that have the greatest impact on the scalability of the system are identified.

- **Chapter 10:** this chapter is dedicated to the conclusions of this thesis work which summarizes the objectives achieved and the future work to be carried out in order to improve the functionality of the framework.

Chapter 2

Software Networking

This chapter introduces the concept of Service Function Chain, that is a chain composed of several network functions aimed at providing end-to-end services in a given communication; this concept, as we will see later, represents the theory that underlies the creation of a Service Graph, a graph necessary for this thesis work. Subsequently, after describing the functioning, the main limitations will be highlighted including the lack of agility and flexibility due to the use mainly of hardware compared to software.

Finally, to overcome these limitations, two new technologies are introduced: Software Defined Network (SDN) and Network Function Virtualization (NFV). Thanks to the first technology it is possible to easily overcome the limits due to the poor agility and simplicity of the SFC, in fact in the forwarding process it allows, thanks to the help of the software, to dynamically decide the path that the package will have to follow. Thanks to the second technology, it is possible to overcome the limitations due to the lack of flexibility of SFC devices, by running the images of network functions on standard hardware devices.

2.1 Service Function Chain

The provision of end-to-end services requires the use of multiple interlinked functions, so that traffic flows in a certain order to achieve a result that meets the needs of the service designer. A formal definition of Service Function (SF) and Service Function Chain (SFC) was presented in RFC 7665 description [1] from which the following definitions derive:

- **Service Function (SF):** It is a service function that takes care of managing packets received from the network, whose work is done on various OSI layers of the protocol stack. This feature can be implemented through a physical device or through a virtual device.
- **Service Function Chain (SFC):** It consists of a set of functions and constraints that must be applied to packets in order to differentiate the paths between the various traffic flows. The path followed may not be linear since the SFC is flexible in choosing the order in which the functions are performed.

The chaining of service functions (SF) allows the creation of small network compounds that must be applied to packets or flows, so that you can manipulate the traffic to your liking. In most networks, the services are managed by an abstract sequence of SF which represent an SFC that from a higher level manages the whole and the order of execution. For example, Figure 2.1 shows an end to end service consisting of two Web Clients, a Web Server and other network functions allocated between the three end-Hosts. In this situation, the Web Client1 would like to communicate in a protected manner with the Web Server, given that there is an untrusted node in the network, therefore two VPN gateways are placed, devices able to put and remove the protection of the traffic that passes through them based on of the policies configured in advance. The Server in turn would not want to receive traffic from Web Client 2, for this reason a Firewall is allocated, a device capable of eliminating unwanted traffic based on its configuration (whitelisting or blacklist); the last function is represented by an Intrusion Detection System, a device capable of detecting attacks on the basis of known or known a priori schemes after an initial monitoring phase. This example highlights the less than optimal use of the

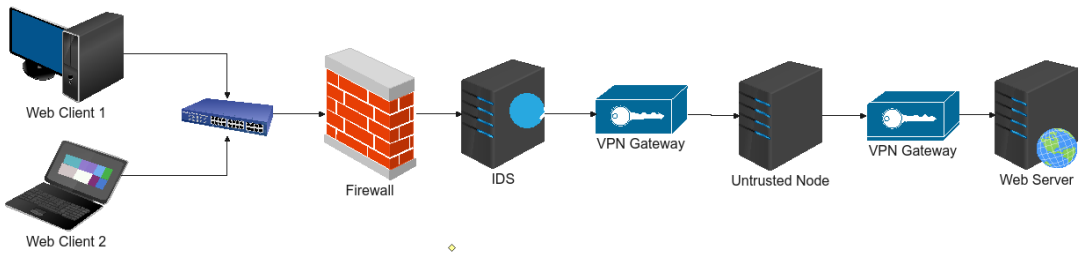

Figure 2.1. Example of SFC

network and computing resources, since the entire service chain must be traversed even when this is not necessary. In the previous example, the traffic coming from Web Client 1 and destined for the Web Server must cross all the physical boxes, even if not necessary, so as to create delays and decrease the Throughput of the entire network. Since network functions are essentially physical boxes dedicated to a specific purpose, a chain of network functions built in this way suffers from severe limitations including:

- Between the physical devices it is not possible to let the traffic flow at will sing in the IP world the traffic is forwarded through a route for the network address;
- Inability to share network resources; Each network function is represented by a dedicated physical box, so the various devices are unable to exchange and share resources, so in certain situations you can have devices that carry a heavy load and others that are almost discharged, so as to waste unnecessarily Network resources.
- Constant maintenance of the devices by the SFC supplier.

- Difficulty in differentiating services; Usually each user requires the use of different services, but with an SFC it is not possible to provide due to the inflexibility to manage these differences since the connections between the various devices are fixed.
- Service interruption during a failure / modification of the SFC; For any modification due to a possible failure or improvement, the service is temporarily interrupted so that the new piece of the chain can be modified or added.

2.2 Software-Defined Networks

Software-defined networking technology represents a new cloud computing approach to network architectures that facilitates their configuration and administration to improve performance. SDN suggests decoupling the packet forwarding process (data plane), responsible for packet forwarding and low-level management, from the routing process (control plane), responsible for coordinating forwarding actions using shared data structures, in so as to centralize the network intelligence in the logical or physical control apparatus. It is worth noting that the first option would avoid scalability issues and have a single point of failure. The SDN controller communicates with the forwarding infrastructure via the southbound interface and, thanks to the northbound interface, is able to interact with the application at the user level or with a higher level controller. While centralizing network intelligence can overcome most of the limitations of SFCs, this approach contains disadvantages with regard to security, scalability and elasticity, which represent the major flaw of SDN technology.

2.2.1 SDN model Architecture

The ideas and approaches for the implementation of SDN services vary depending on the operator, supplier or communication standard adopted. In fact it is not possible to draw a clear line of demarcation between the single models, so that different elements coming from different approaches can belong to an SDN network.

Figure 2.2 shows the architecture of an SDN network that respects all the principles mentioned. At the lowest level, there is a set of elements that allow traffic to be forwarded quickly, not necessarily expensive routers since their main task, given by forwarding packets to the output ports, is quite trivial. This simplicity in forwarding traffic increases the efficiency of the SDN technology. This speed of forwarding of packets is favored by the presence in each device of a forwarding table where there are the rules to be respected in order to forward packets in the fastest possible way. Each rule in the table is divided as follows:

- **Fields for comparison:** which allow you to filter a subset of packets that satisfy them;
- **Action field:** describing the actions that are performed on packets by the switch (e.g. forwarding at a specific rate, forwarding to a specific port, packet discard, push or pop of a field)

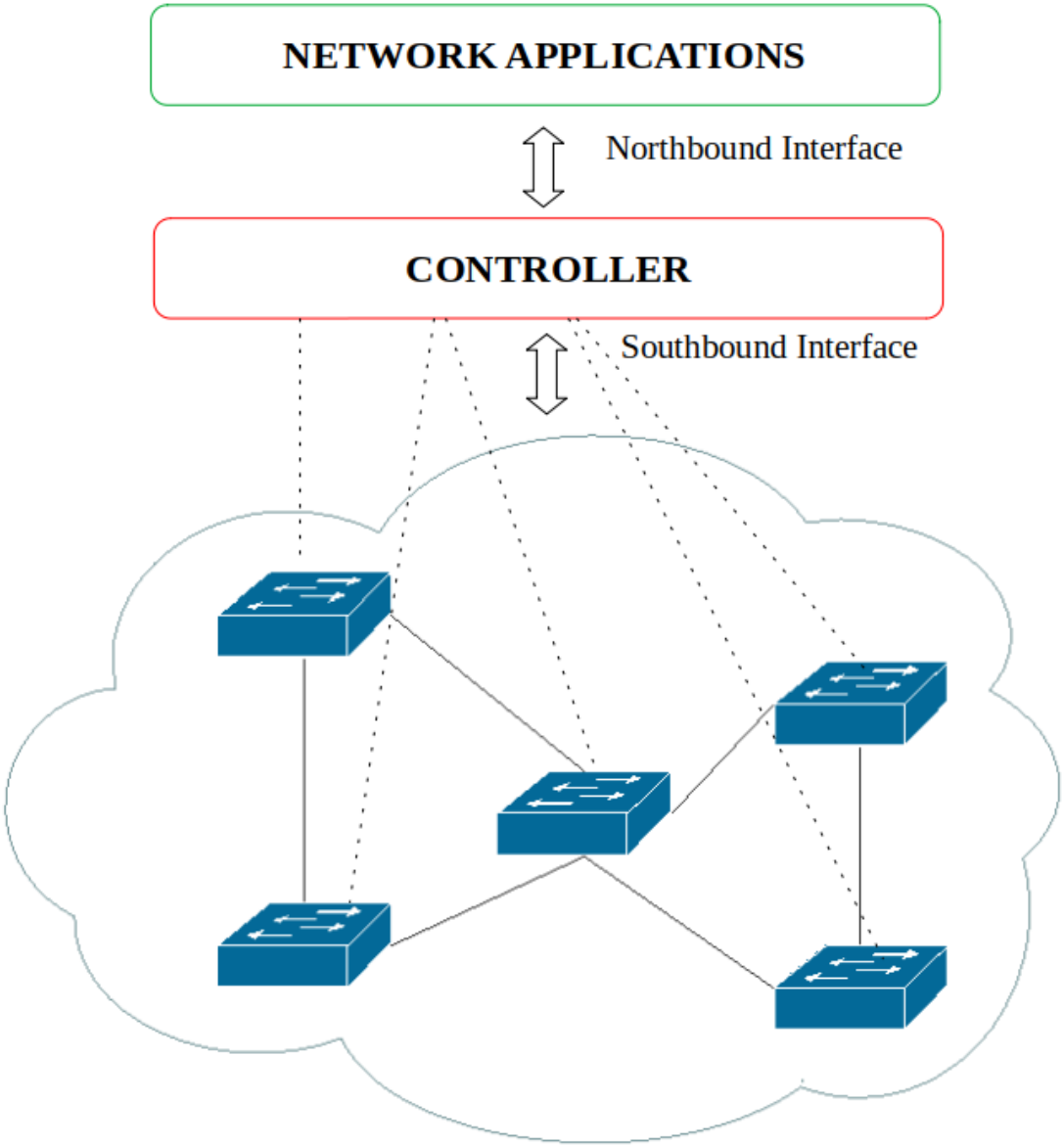

Figure 2.2. SDN model

The simplicity of these forwarding devices is due to the separation of the control plan from the forwarding plan and therefore all the decisional part, such as the calculation of the forwarding tables, is entrusted to the SDN controller. Through an open southbound interface, the SDN controller receives information from the network for monitoring or statistics or to create and modify forwarding rules based on user requests. When a packet reaches a switch, the headers are compared to the rules in the flow table. If no match is found, the packet will be encapsulated and forwarded to the controller. The latter will have to decide how to manage the data packet, communicating to the affected switch whether to discard it or forward it by entering a new rule in the table. One protocol used to perform these functions is the Open Flow protocol. It is implemented both on the switches and

on the controller, establishing a secure end-to-end communication channel thanks to cryptographic protocols such as TLS or SSL. This represents one of the critical points of this protocol, since for devices where it is not supported it would require either a replacement or a software patch. As a result, other protocols already supported by network technology such as Simple Network Management Protocol (SNMP), Network Configuration Protocol (NETCONF) and (RESTCONF) have spread. Finally, the Northbound Interfaces (NBI) represent the programming interfaces indispensable for the communication between the SDN controllers and the control application software, which are typically based on REST.

This technology represents the advantages due to the flexibility in the creation of rules and the single control point capable of managing and manipulating the entire network, but has security and performance limits. Security because each switch has direct access to the controller and can manipulate it at will and performance due to the non-negligible latency of the responses to the switches. Finally, we must consider that the crash of an application would put the controller itself in crisis with serious repercussions on the network.

2.2.2 Application of the SDN to a SFC

Thanks to the SDN technology, many of the limitations of the service function chains described in the previous paragraph can be overcome. In fact, the various service functions that make up an SFC correspond to hardware appliances which can be connected to each other via an SDN switch completely governed by the controller. The structure of Figure 2.1 can be remodeled by inserting an SDN switch in the center of the network and connecting all the physical devices to it. In this way the controller would have the power to determine the flow of packets and the order of crossing of the middleboxes.

Although an SFC implemented in this way has various advantages including agility in the provision of new services, maintenance, reliability and the ability to differentiate different service chains for each user, it is still difficult to partition the physical appliance since the devices are still based on hardware and therefore it is not yet possible to share resources between the devices and the installation of a new function remains a very slow operation.

2.3 Network Functions Virtualization

A next step in solving the limitations of the SFC is given by the possibility of being able to virtualize the network and devices. Network Function Virtualization (NFV) takes advantage of virtualization by changing the way the network is created by defining virtualized structures for network functions so that software can be decoupled from hardware. The images of these functions are performed on standard hardware, so that they can be added and removed very quickly at the request of the user. So this technology requires particularly fast hardware, software-based

network functions, computer virtualization and standard APIs. The advantages of NFVs are:

- **Flexibility:** thanks to the decoupling of the software from the hardware it is possible to carry out different activities on the same hardware device. Providers can choose devices from different vendors so they have more flexibility in selecting the optimal hardware for their network architecture.
- **Scalability:** the network scales according to the needs of the moment, in fact the devices are assigned greater quantities of CPU, memory or band, so as to reduce the implementation times of the service.
- **Cost:** running functions in devices with standard hardware presents a reduction in network costs. This greatly simplifies the choice of network administrators who can create hardware devices as efficiently as possible so as to best meet the needs of the network.

2.3.1 Characteristics of Network Functions Virtualization

As mentioned earlier, a fundamental point of this technology is that network functions are nothing more than software images running on standard hardware. Traditional virtual machines (VMs) were the first technologies used to implement these functions. Each function is implemented on a different VM, which shares the same resources as the server and enjoys excellent isolation. The excellent level of security due to the total isolation between the various VMs, makes this an excellent solution for scenarios such as multi-tenant data centers where the protection of your services is essential. The disadvantage is due to the excessive number of resources needed to create a VM, in fact, creating a virtual machine requires a lot of memory and disk space. Even the real-time migration of a VM is very complex and requires the use of very sophisticated algorithms.

To counter these limitations, we moved to the use of lightweight virtualization with Linux Containers (LXC). This technology is based on the sharing of some parts of the kernel thanks to the creation of the namespaces of the Linux world. In this way, the hypervisor kernel can be partitioned and shared between the various Containers. For example, by creating an appropriate namespace, an LXC can share the network stack, while all other parts of the kernel are separated and duplicated. Although this technology has the advantage of low memory allocation compared to VMs, it has not been very successful because it has the limit of portability, in fact an LXC created on one machine cannot be moved to another machine.

Another solution for light virtualization is given by Dockers which are based on the same principles as LXC but manage to overcome the limits of portability. A Docker can be transferred to another machine via an intermediate repository (Docker Hub) or can be reproduced via a Docker File that contains the same operations that must be performed to obtain the same Docker. The problem of the separate file system is also overcome, this is solved by sharing parts of the file system with the hypervisor. The main problem is represented by poor isolation between the various containers, in fact the use of Dockers in containers belonging

to different tenants could cause problems since it would be difficult to prevent all access attempts.

2.3.2 Application of the NFV to a SFC

Thanks to the improvements made by network functions virtualization technology, many of the limitations of SFC are overcome. In fact, thanks to this technology and the SDNs, the service function chains are much more flexible because each function does not need to be a physical box but can be a software function installed on a server. An example is represented in Figure 2.3 which takes up the model of Figure 2.2, where the functions are not represented by specific hardware but by virtual machines that can be installed on the same server.

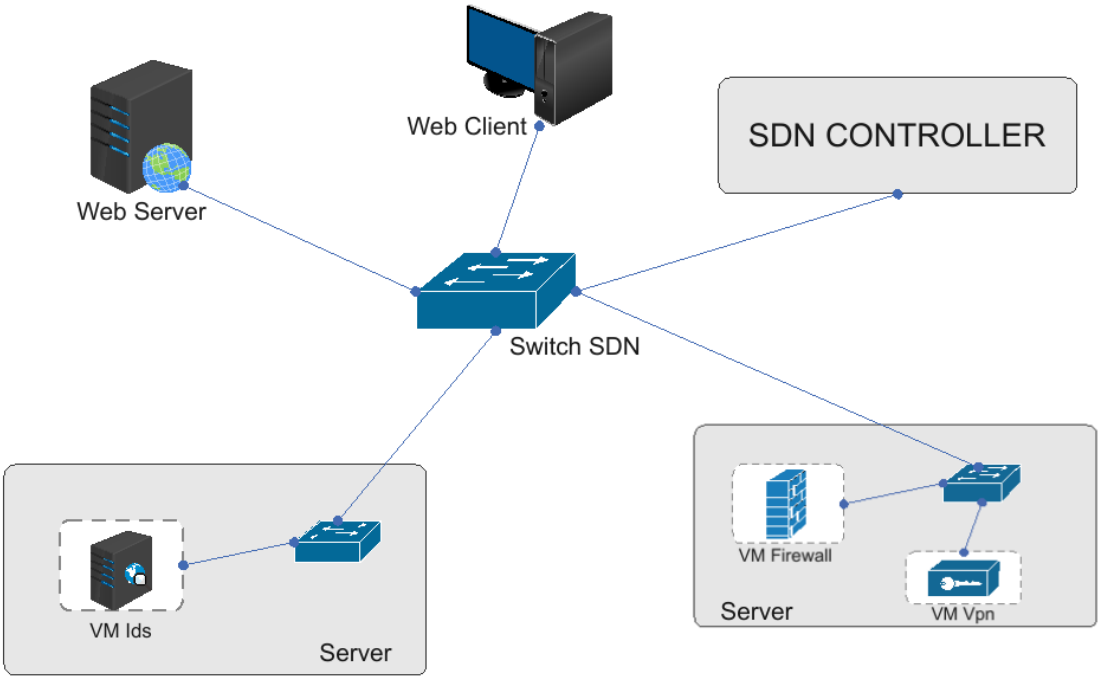

Figure 2.3. Example of Service Function Chain with a NFV-SDN architecture

In the figure it can be seen how the packet flow is managed by an SDN controller that determines and sets the rules not only to the switches that connect the external servers but also to the internal switches created by the Operating System of each server that allow communication between the functions implemented. In addition to the advantages introduced by SDN, NFV adds a lot of flexibility in creating a new function, provides dynamic links for functions within the same server and allows functions belonging to the same server to share computational resources.

Chapter 3

VEREFOO

This chapter introduces a framework example that implements the improved SFC approach with the addition of SDN and NFV technologies. The modules that compose it are briefly described, the characteristics and functionalities for which this thesis has contributed to the development, which will be better described in the following chapters. An overview of the logical topologies that the framework can accept as input is provided, including the Service Graph, Allocation Graph, and Network Security Requirements.

The second part of the chapter provides a detailed description of the two graphs, Service Graph and Allocation Graph, with their differences and describes the graph model through an example of implementation in the XML schema.

The last part of the chapter focuses on the description of the Network Security Requirements, on the model and on the different types of requirements used in this thesis work: Reachability Properties, Isolation Properties and Protection Properties. Finally, after presenting the usefulness of the three requirements, an example of implementation of the three requirements is provided, taken from an XML file of the input framework.

3.1 Introduction to VEREFOO

The technologies described in the previous paragraphs make it possible to make the configuration of network services more dynamic so that manual activities can be replaced by automatic approaches so as to drastically reduce the number of errors. An activity that is benefiting from automation is the positioning and configuration of network security functions (NSF) that are introduced to meet network security requirements (NSR). In this case, an automated solution is preferred over a manual one since manual activity can lead to errors and suboptimal results. Formal correctness and the achievement of an optimal solution leads to a decrease in the computational resources used and to a maximization of performance.

A similar approach is used by the VEREFOO framework whose main objectives are the allocation and automatic configuration of network security functions on a Service Graph given the appropriate NSRs. To do this, it uses the z3Opt engine to

solve a Maximum Satisfiability Theories Module (MaxSMT) problem in which it works in contact with an NFV orchestrator to receive the correct implementation of the selected NSFs.

3.2 Model description

VEREFOO (VErified REfinement and Optimized Orchestration) is a framework that manages the allocation and optimal configuration of network security functions (NSF) on a Service Graph received in input to meet the security requirements (NSR) expressed by the administrator using a high-level language and finally also manages the positioning on physical servers [2]. To do this VEREFOO uses the z3Opt engine which solves the (MaxSMT) problem in order to have the correct implementation of the Network Security Functions.

The framework guarantees high flexibility by allowing users to define a repository for the NSRs and to create the list of functions present in the system. Among the graphs passed as input to the system we must distinguish:

- **Service Ghaph:** logical topology formed by network functions that together constitute a complete end-to-end service. Unlike a simple SFC, in a Service Graph there can be loops, in fact there are multiple paths to reach the different endpoints. In the creation of a Service Graph no security requirements are required, in fact a Service Graph does not include any firewall, vpn gateway and antispam filters;
- **Allocation Graph:** logical topology constituted by the same set of functions of the Service Graph, in addition it adds between each pair of placeholder functions called place allocation (AP), in which it is possible to allocate the network security functions. The framework allocates the network security functions in the APs located in optimal positions, in the remaining APs it allocates the forwarders so that they can propagate packets in all possible paths;

In Figure 3.1, a complete overview of the framework is presented, highlighting the main components in order to describe the complete flow of VEREFOO. The following phases can be distinguished:

1. Input phase, where the service designer introduces the following parameters:
   - A set of network security requirements which are the constraints that must be met by the network administrator. Introduced through a GUI Policy, which facilitates its creation, they can be defined using a high or medium level language.
   - A Service Graph or Allocation Graph where you can allocate and configure network security functions. Through a Service GUI, the user can choose, from a catalog of network functions or network security functions, which functions to allocate directly on the graph.

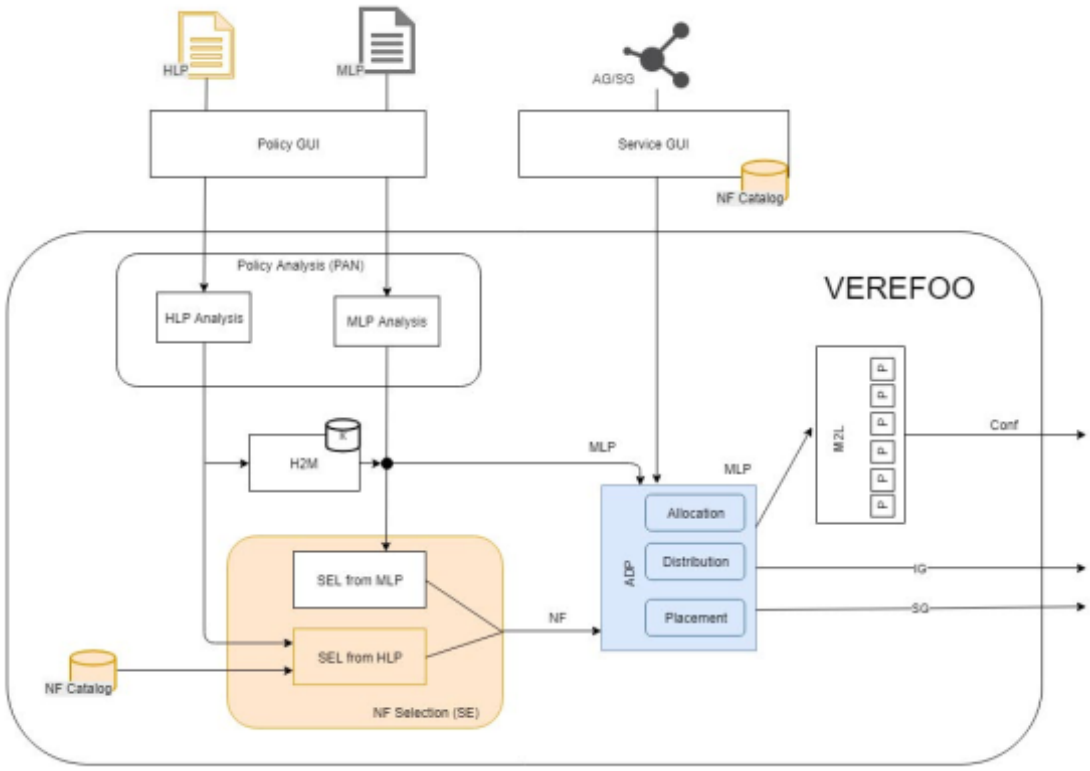

Figure 3.1. VEREFOO model

2. Preliminary phase conducted by the Policy ANalysis (PAN) module, in which the conflict analysis of the Network Security Requirements passed in input is performed, in order to have the minimum number of constraints that the network administrator must enforce on the network. In case of an error it can also provide a report notifying the error occurred.

3. If the network security requirements introduced in input are in a high-level language, the High-to-Medium (H2M) module implements a requirements improvement process so as to transform them into medium-level requirements that contain all the information necessary for create NSFs and for low-level configuration of VNFs located on the physical network.

4. In the next phase, an important task is carried out by the NF Selection (SE) module which selects the appropriate Network Security Functions needed to meet the high and medium level Network Security Requirements. These functions are chosen from a catalog that is the same as the one accessed by the network designer via the Service GUI. In this step it is possible to have an optimization process in which the optimal set of NSFs is selected, although a limitation is given by the lack of knowledge of the Service Graph topology.

5. The most important phase is carried out by the Allocation, Distribution and Placement (ADP) module, whose main task is to calculate a Services Graph

with the addition of the new automatically allocated Network Security Functions or a Physical Graph implementing the functions on physical devices. Thanks to the Medium-to-Level (M2L) module, ADP can produce the translation of the configuration from medium to low level, so that it can be set in the correct functions regardless of the type of supplier. More details on the implementation are provided in the following chapters, as ADP is the module for which this thesis has provided developments.

3.3 Service Graph

A Service Graph is a generalization of the Service Function Chain and represents the logical topology of an end-to-end service with no network function that contribute to system security.

For the definition of a Service Graph, the system designer has at his disposal a set of network functions that contribute to the designation of the service through their structuring, but can also influence the configuration of the Network Security Functions given the security constraints to be respected. The functionality classes that perform these functions are as follows:

- **load balancing:** it forwards the traffic via the decisions made by an internal logic, which makes decisions based on the load of the various servers. Since the policies of the Network Security Functions concern only the end hosts, it is not possible to define where the traffic can be forwarded.
- **traffic monitoring:** carry out a monitoring action on the network packets received and in the event of a possible anomaly they modify the value of the internal counter so as to notify the network administrator to resolve it. However, they do not modify the content of the received packets, so their presence does not affect the configuration of the Network Security Functions.
- **web caching:** makes forwarding decisions based on the information present in the URL of the resource that a client has requested from a web server and checks if is already present in its internal cache. However, since it is based on higher level checks than the checks made at the IP address level, it is not possible to determine in advance when a packet will be discarded or not, so it does not affect the behaviour of the configuration of the Network Security Functions.
- **NAT:** is able to modify some fields of the packet header, since the main task of this feature is to replace private addresses with public addresses. This implies that the rules configured on the Network Security Functions may be different depending on their allocation in the Service Graph, if they precede or follow a NAT function.

The type of a Service Graph passed in input to the framework and where no Network Security Function has been allocated is the following:

$$Gs = (Ns, Ls)$$

where:

- Ns is the set of all network nodes of the Service Graph and terminal nodes including client nodes, server nodes or subnets in which there are multiple endpoints. Each node Ns is identified by a non-negative integer k;
- Ls it is the set of links that unite the various Ns nodes, in order to make the Service Graph a direct graph.

A Service Graph by itself does not allow the allocation of Network Security Functions, since each node of the graph is already characterized by a specific network function. Therefore, an automatic transformation of the Service Graph into an Allocation Graph will be required to allow the allocation and configuration of the NSFs.

3.3.1 XML schema of the Service Graph

A Service graph is represented through an XML schema and is identified through the *graph* element. Each scheme is identified by a unique number used to represent several graphs at the same time, and also consists of a Boolean attribute that takes the value false if you want to represent an Allocation Graph and the value true if you want to represent a Service Graph. An example of a Service Graph in XML schema is provided through Listing 3.1 which structures the network shown in Figure 3.2.

Listing 3.1. Service Graph XML schema example

```
<graph id= "1" serviceGraph= "true">

<node functional_type= "WEBCLIENT" name= "10.0.0.2">
  <neighbour name= "20.0.0.1"/>
  <configuration description= "WebClient␣description" name= "WC2">
    <webclient nameWebServer= "30.0.0.1"/>
  </configuration>
</node>

<node functional_type= "WEBCLIENT" name= "10.0.0.1">
  <neighbour name= "20.0.0.1"/>
  <configuration description= "WebClient␣description" name= "WC1">
    <webclient nameWebServer= "30.0.0.1"/>
  </configuration>
</node>

<node functional_type= "NAT" name= "20.0.0.1">
  <neighbour name= "10.0.0.1"/>
  <neighbour name= "10.0.0.2"/>
  <neighbour name= "30.0.0.1"/>
```

```
    <configuration description= "NAT␣description" name= "NAT2"> >
      <nat>
        <source>10.0.0.1</source>
        <source>10.0.0.2</source>
      </nat>
    </configuration>
  </node>

  <node functional_type= "WEBSERVER" name= "30.0.0.1">
    <neighbour name= "20.0.0.1"/>
    <configuration description= "WebServer␣description" name= "WS2">
      <webserver>
        <name>30.0.0.1</name>
      </webserver>
    </configuration>
  </node>

  </graph>
```

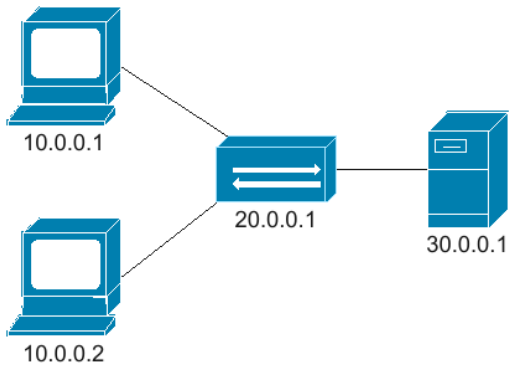

Figure 3.2. Example of a Service Graph

Each node element represents the basic unit of the logical topology on which a network function or an NSF can be specified.

Each node element is characterized by:

- **a name item:** a unique string in which the IP of the node is represented;
- **a functional type item:** rappresenta quale funzionalità deve svolgere quell'elemento del grafico;
- **a set of neighbor item:** where the neighbour nodes to which traffic can be forwarded are represented;
- **a configuration item:** represents the configuration of the network function, therefore the behavior that function must have to satisfy with the network security requirements.

3.4 Allocation Graph

As seen above, it is not possible to allocate any network security functions in a Service Graph since each node corresponds to a specific network function. For this reason, the internal representation of the Service Graph provided in input is modeled in a structure called Allocation Graph, in which Allocation Places are added, one for each pair of nodes, so as to allow the allocation of NSFs.

If the service designer has good knowledge about configuring security devices, he can force an NSF to be placed in a specific AP, preventing the solver from removing it, or he can prohibit insertion between two specific nodes of an AP so as to improve the capacity of the framework and bring benefits on execution time. Furthermore, if the designer has a clear idea of how the final Service Graph would be calculated and which are the best places on which to allocate an NSF, he can decide to manually calculate an Allocation Graph and introduce it as input to the framework. These improvements that are provided by a manual contribution of the configuration, if used by people who are not security experts, in addition to bringing benefits can lead to an unsatisfactory optimization of the problem.

When running the framework, each Allocation Place of the Allocation Graph can be treated in two different ways:

1. if the Allocation Place is between the source and destination of the specified NSRs, an NSF may be allocated in this AP, so hard and soft constraints must be added in constructing the MaxSMT problem;

2. if the Allocation Place is not between the source and the destination of the specified NSRs, no NSF will be allocated in this AP and therefore no hard and soft constraint must be added for the construction of the MaxSMP problem, then the AP can be removed from the network topology.

After the calculation of the MaxSMT problem, two possible situations can occur for each AP:

1. an NSF is allocated and configured so as to satisfy all network security requirements passed in input;

2. NSF is not allocated, in this case a forwarder function is allocated to allow the transmission of traffic, this function will be deleted later from the framework with a post-processing.

After each node of the graph has been assigned a specific role, the virtual implementation can be transformed into the configuration of the substrate servers of a Physical Graph. This configuration can take place in two different ways, in the first a physical middlebox is associated with each logical node of the Allocation Graph, while in the second case it is possible to distribute multiple VNFs simultaneously on the same server in order to reduce the amount of traffic and the actual number of servers used.

The model of an Allocation Graph automatically generated from a Service Graph received as input from the framework is the following:

$$Ga = (Na, La)$$

where:

- Na is the set of all network nodes of the Allocation Chart and consists of terminal nodes such as client nodes, servers or subnets in which there are multiple endpoints, a set of nodes composed of service functions that are necessary to provide a complete end-to-end service and from a set of Allocation Place (PA) where an NSF can be allocated. Each Na node is identified by a non-negative integer k.
- it is the set of links that unite the various Na nodes, in order to make the Allocation Graph a direct graph. contenuto...

3.4.1 XML schema of the Allocation Graph

The implementation of the XML schema of the Allocation Graph was made starting from the XML schema of the Service Graph shown in Listing 3.1, where node elements were added without any type of functional attribute and without any internal configuration, since they will be added later by framework. Listing 3.2, shows an XML example of an Allocation Place that can be added manually by the network designer or automatically by the framework, in the configuration of the Allocation Graph XML file.

Listing 3.2. XML of an Allocation Place

```
<node name= "40.0.0.1" >
  <neighbour name= "10.0.0.1"/>
  <neighbour name= "10.0.0.2"/>
  <neighbour name= "20.0.0.1"/>
</node>
```

Listing 3.3. Allocation Graph XML schema example

```
<graph id= "1">

<node functional_type= "WEBCLIENT" name= "10.0.0.2">
  <neighbour name= "40.0.0.1"/>
  <configuration description= "WebClient␣description" name= "WC2">
    <webclient nameWebServer= "30.0.0.1"/>
  </configuration>
</node>

<node functional_type= "WEBCLIENT" name= "10.0.0.1">
  <neighbour name= "40.0.0.1"/>
```

```
    <configuration description= "WebClient description" name= "WC1">
      <webclient nameWebServer= "30.0.0.1"/>
    </configuration>
  </node>

  <node name= "40.0.0.1" >
    <neighbour name= "10.0.0.1"/>
    <neighbour name= "10.0.0.2"/>
    <neighbour name= "20.0.0.1"/>
  </node>

  <node functional_type= "NAT" name= "20.0.0.1">
    <neighbour name= "40.0.0.1"/>
    <neighbour name= "40.0.0.2"/>
    <configuration description= "NAT description" name= "NAT2"> >
      <nat>
        <source>10.0.0.1</source>
        <source>10.0.0.2</source>
      </nat>
    </configuration>
  </node>

  <node name= "40.0.0.2" >
    <neighbour name= "20.0.0.1"/>
    <neighbour name= "30.0.0.1"/>
  </node>

  <node functional_type= "WEBSERVER" name= "30.0.0.1">
    <neighbour name= "40.0.0.2"/>
    <configuration description= "WebServer description" name= "WS2">
      <webserver>
        <name>30.0.0.1</name>
      </webserver>
    </configuration>
  </node>

  </graph>
```

A complete example of an XML file representing an Allocation Graph that has been modeled in the MaxSMT problem from a Service Graph is provided via Listing 3.3 which represents the network structure depicted in Figure 3.3.

3.5 Network Security Requirements

3.5.1 Description

The second input of VEREFOO is represented by the Network Security Requirements, which according to the experience of the system designer, can be formulated in two different representations:

- if the designer has a good understanding of the configuration of the security functions, he can choose to use a medium level representation containing

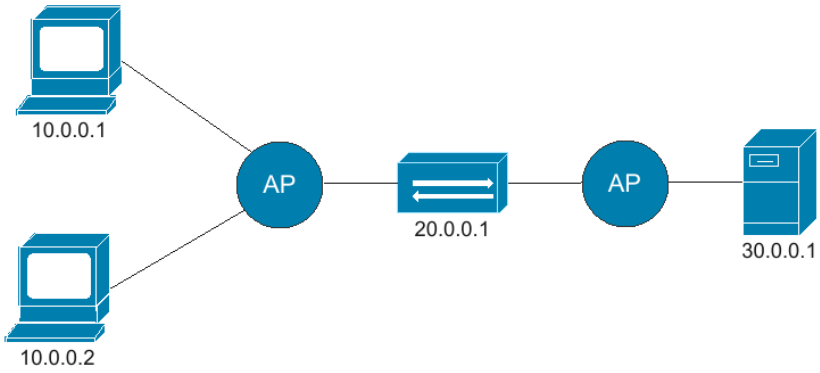

Figure 3.3. Example of an Allocation Graph

specific details such as IP addresses and ports of the nodes involved. In this way the translation operation is avoided since the middle level representation of the NSRs can be used for the creation of the MaxSMT problem;

- if the designer does not have a good understanding of the configuration of the safety functions, he can use a high-level representation, flexible to manage and easy to understand. It is a user-friendly approach where each end point of the service is identified by a unique number which can later be mapped to an IP address. contenuto...

The VEREFOO ADP module receives input only constraints in a medium level language, in this case, in case of high level constraints, a fundamental role is played by the H2M module which translates the high level constraints received into medium level constraints to be passed in input to the ADP module.

The Network Security Requirements used in this thesis are requirements that connect or protect the communication between two endHosts and can be classified into three different types:

1. **reachability property:** a specified end point must be allowed to reach another end point through a path;
2. **isolation property:** a specified end point must not be allowed to reach another end point through all available paths;
3. **protection property:** a specified end point must be allowed to communicate securely with another end point.

Each Network Security Requirement is formulated with the following components:

$$[ruleType, IPSrc, IPDst, portSrc, portDst, transportProto]$$

where:

- **the ruleType:** indicates which type of Security Requirement must be satisfied, can take the value of IsolationProperty, ReachabilityProperty and ProtectionProperty;
- **the IPSrc:** represents the source IP of the communication for the specific security requirement;
- **the IPDst:** represents the destination IP of the communication for the specific security requirement;
- **the portSrc:** represents the source port of the transport-layer of the traffic flow for the specific security requirement;
- **the portDst:** represents the destination port of the transport-layer of the traffic flow for the specific security requirement;;
- **the transportProto:** represents the traffic flow transport-layer protocol for the specific security requirement.

Each IP address (IPSrc, IPDst), is defined with the traditional point-decimal notation, that is:

$$ip1.ip2.ip3.ip4$$

where ipi , $\forall i \in \{1,2,3,4\}$, it can be an integer belonging to the range [0-255]or a wildcard element identified with the symbol $*$. With this symbol it is possible to represent network addresses, for example the representation 30.0.2.$*$ represents the end points present in the network 30.2.2/24, or corresponding network masks, for example the representation 30.2.$*$.$*$ represents the network 30.2.0.0/16.

The $portSrc$ and $portDst$ values are formulated as an integer or a range of integers belonging to the range [0-65535]. The satisfiability of the property is guaranteed if the package in question has a value of $portSrc$ and $portDst$ within the range specified in the requirement.

Finally, the $transportProto$ value represents the layer 4 protocol used in the requirement. This element can take on the TCP and UDP values or both values simultaneously through the use of the wildcard character $*$.

3.5.2 Reachability Requirements

A Reachability Requirement is used by VEREFOO to ensure that a source end-host can communicate with a destination end-host, i.e. that packets of this communication are not discarded by any packet filter present in the path. This can be done through a configuration of the packet filter in whitelist or blacklist mode, in the first case the transit should be blocked for all communications for which no specific requirement has been formulated, in the second case the transit should be blocked at all communications for which at least one specific requirement has been formulated.

For a correct satisfaction of the Reachability Requirement, the following constraints must be met:

1. There must be a node in the network that has the source address as its IP address and one that has the destination IP address of the requirement, otherwise the packet will never be sent from a source node nor received from a destination node;
2. The source node of the security requirement must be able to send the traffic that satisfies it to at least one first hop, since the existence of a path between source and destination is sufficient to satisfy the requirement;
3. The requirement destination node must receive satisfying traffic from at least one of its first-hops, as this means that at least one path has been found between the source and destination.

Listing 3.4. Example of a Reachability Property

```
-Reachability Property from 10.0.0.1 to 20.0.0.1.
```

A complete example is presented below, where the allocation graph shown in Figure 3.4 and the requirement 3.4 are passed as input to the framework.

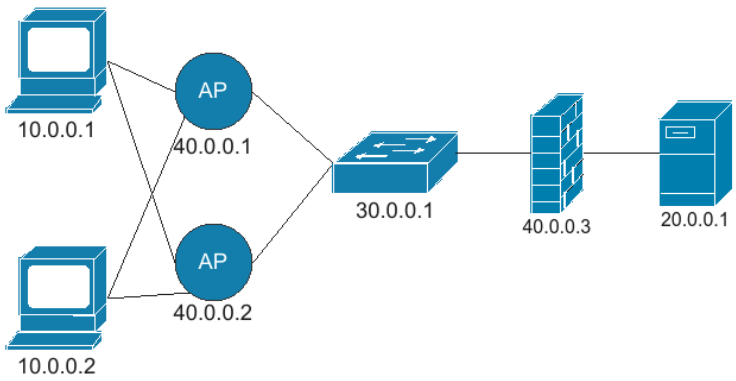

Figure 3.4. Allocation Graph example

considering the behavior of packet filters in blacklist mode, a possible solution is shown in Figure 3.5.

It can be seen that in the graph of Figure 3.5 no packet filter has been allocated, with the exception of the packet filter 40.0.0.3 already present in the input Allocation Graph. To allow the reachability of traffic from the source 10.0.0.1 to the destination 20.0.0.1, simple Forwarders have been allocated in the APs of the input Allocation Graph which have the task of sending all incoming traffic to output. Finally, no configuration rule has been configured in the packet filter since the default configuration, blacklisting, allows the transit of all traffic for which no specific requirement has been formulated.

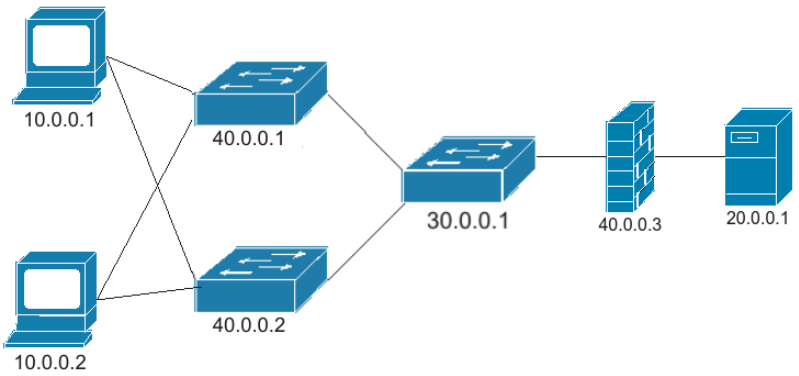

Figure 3.5. Example satisfiability solution Reachability Property

It is important to underline that in this solution the fundamental constraints discussed above are respected, for the first it can be noted that in the network there are nodes 10.0.0.1 and 20.0.0.1 which are respectively the source node and the destination node of the reachability constraint , for the other two it can be seen that the node 10.0.0.1 has at least one first-hop to send the traffic to in the path that goes to the destination of the requirement and the destination 20.0.0.1 has at least one end-hop from which to receive the traffic coming from the source of the requirement.

3.5.3 Isolation Requirements

An Isolation Requirement is used by VEREFOO to ensure that a source end-host cannot communicate with a destination end-host, i.e. that the packets of this communication are discarded by all packet filters present in the path. This can be done through a configuration of the packet filter in whitelist or blacklist mode, in the first case the transit should be blocked for all communications for which no specific requirement has been formulated, in the second case the transit should be blocked at all communications for which at least one specific requirement has been formulated.

For a correct satisfaction of the Isolation Requirement, the following constraints must be met:

1. There must be a node in the network that has the source address as its IP address and one that has the destination IP address of the requirement, otherwise the packet will never be sent from a source node nor received from a destination node;

2. The source node of the security requirement must be able to send the traffic that satisfies it to at least one first hop, since the existence of a path between source and destination is sufficient to satisfy the requirement;

3. The destination node of the requirement must not receive traffic that satisfies it from all of its first hops, as this means that at least one path has been found between the source and the destination.

A complete example is presented below, where the allocation graph shown in Figure 3.4 and the requirement 3.5 are passed as input to the framework.

Listing 3.5. Example of a Isolation Property

```
-Isolation Property from 10.0.0.1 to 20.0.0.1.
```

considering the behavior of packet filters in blacklist mode, a possible solution is shown in Figure 3.6.

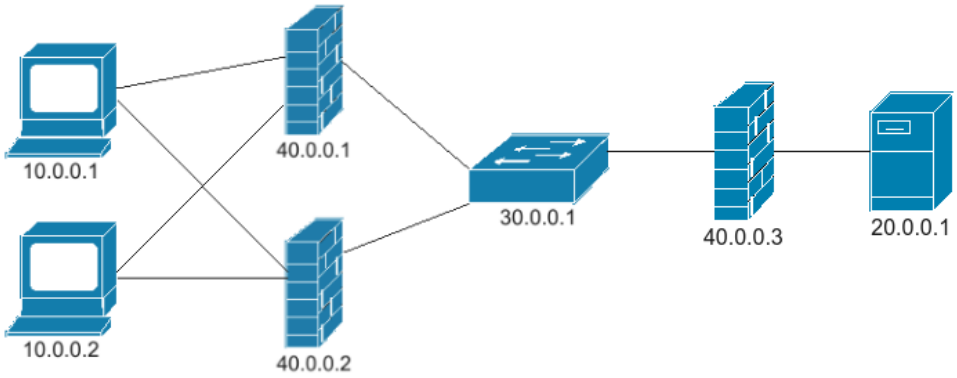

Figure 3.6. Example satisfiability solution Isolation Property

It can be seen that in the graph of Figure 3.6, to ensure the isolation of traffic from the source 10.0.0.1 to the destination 20.0.0.1, two packet filters have been allocated in addition to the packet filter 40.0.0.3 already present in the input graph. These packet filters have been configured to discard all packets with source 10.0.0.1 and destination 20.0.0.1, since the default configuration of packet filters would allow the transit of all traffic for which no requirement has been formulated specific.

Finally, it is important to underline that in this solution the fundamental constraints discussed above are respected, for the first it can be noted that in the network there are nodes 10.0.0.1 and 20.0.0.1 which are respectively the source node and the destination node of the isolation constraint , for the other two it is seen that the node 10.0.0.1 can send the traffic in the paths that go to the destination of the requirement to all its first-hops and the destination 20.0.0.1 has no end-hops from which to receive the traffic coming from the source of the requirement because packets are blocked by packet filters.

3.5.4 Communication Protection Requirements

The security requirement of communication protection is the security requirement that can establish secure communication between two endpoints, creating one or more secure channels depending on how untrusted nodes and inspector nodes are located on the network. The characteristics of the security channel must comply with the set of properties specified in the protectionInfo present in the network security requirements. The properties that a secure communication must respect are:

- **securityTechnology:** technology to be used for the creation of the secure channel;
- **authenticationAlgorithm:** algorithm to use for package integrity and authentication;
- **encryptionAlgorithm:** algorithm to use for packet encryption.

In defining the communication protection requirements, the topologyInfo must also be defined, characterized by a set of nodes and connections including:

- **untrustedNodes:** are those nodes where the application of security requirements is mandatory. By default, secure communication often considers untrusted nodes, all nodes that are in the path among the source and the destination. In this way, even if a valid secure communication is produced since no node in the path has access to the data in the clear text, this is an ordinary and inflexible solution. For this reason, if the security manager has a good knowledge of the network and the nodes that compose it, he can decide to differentiate between trusted nodes and untrusted nodes so as to create a larger space of the solution and save in terms of allocated resources;
- **inspectorNodes:** are those nodes in which the traffic must flow without any protection since it must be analyzed. Such nodes (for example IDS), which performs the function of analyzing network traffic and seeing if there is anything harmful in them, cannot perform their task if they find themselves analyzing encrypted traffic. In fact, this category of nodes was designed to facilitate the inspection of traffic but could lead to a slowdown in communication due to the various encryption and decryption actions;
- **untrustedLinks:** are those links where the application of security requirements is mandatory. By default, secure communication often considers untrusted links, all links that are in the path among the source and the destination. In this way, even if a valid secure communication is produced since no link in the path has access to the data in clear text, this is an ordinary and inflexible solution. For this reason, if the security manager has a good knowledge of the network, of the nodes and links that compose it, he can decide to differentiate between trusted links and untrusted links so as to create a larger space of the solution and save in terms of allocated resources.

For a correct satisfaction of the Channel Protection Requirement, the constraints that have been remodeled in the work of this thesis must be satisfied and are the following:

1. for each traffic flow that belongs to the requirement, in the presence of an untrusted node, the number of predecessor nodes that insert protection must be greater than the number of nodes that remove protection. This constraint is critical to ensuring that traffic is secure as it traverses the untrusted node;

2. for each traffic flow that belongs to the requirement, in the presence of an inspector node or destination node, the number of predecessor nodes that execute an add security action must be equal to the number of nodes that perform a non-protection action. This constraint is critical to ensure that traffic is unsecured when it passes through the inspection node or flow destination.

3. for each traffic flow that belongs to the requirement, if you have an untrusted link, the number of predecessor nodes performing a security action must be greater than the number of nodes performing a non-security action. This constraint is critical to ensuring that traffic is secure when traversing the untrusted link.

Listing 3.6. Example of a Protection Property

```
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1 and inspectorNode 30.0.0.2
```

Below is a complete example of satisfying a communications security requirement, where the Allocation Graph shown in Figure 3.7 and the requirement 3.6 are passed as inputs to the framework.

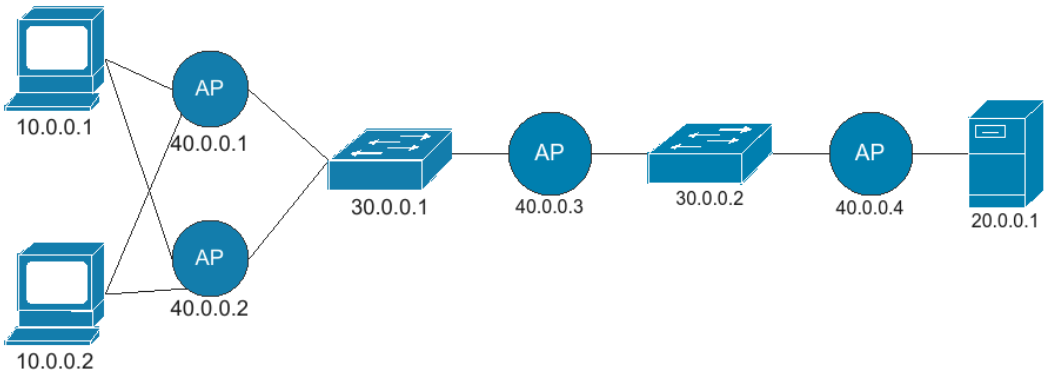

Figure 3.7. Example of an Allocation Graph

a possible solution of the framework output is shown in Figure 3.8.

It can be seen that in the graph of Figure 3.8, three VPN gateways have been allocated to ensure the protection of traffic from the source 10.0.0.1 to the destination 20.0.0.1. The VPN gateways 40.0.0.1 and 40.0.0.2 have been configured in ACCESS mode in order to protect all packets coming from the source 10.0.0.1 and destined for 20.0.0.1, instead the VPN gateway 40.0.0.3 has been configured in EXIT mode in so as to remove the protection of all packets protected by previous VPN gateways.

Finally, it is important to underline that in this solution the fundamental constraints discussed above are respected. For the first it can be noted that given the

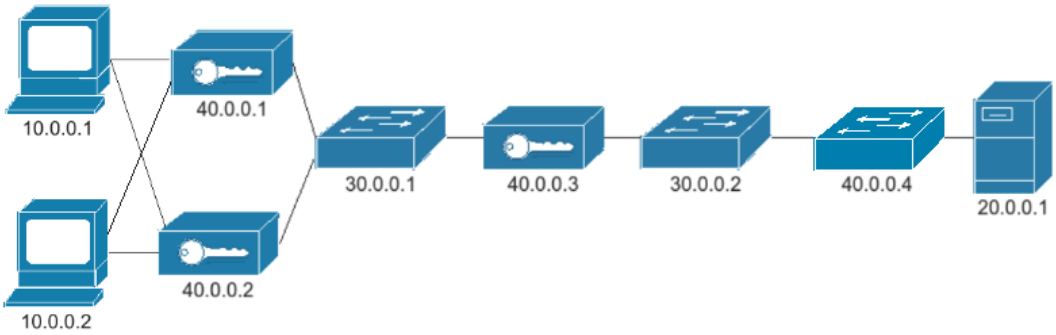

Figure 3.8. Example satisfiability solution Protection Property

untrusted node 30.0.0.1, the sum of the predecessor VPN gateways that provide protection is greater than the sum of the VPN gateways that take away protection. For the second constraint it can be noted that given the inspector node 30.0.0.2 or the destination 20.0.0.1, the sum of the predecessor VPN gateways that provide protection is equal to the sum of the predecessor VPN gateways that remove protection. Finally, the third constraint is useless given the absence of untrusted links in the requirement.

3.5.5 XML schema of the NSRs

The implementation of the XML schema of the Security Requirements is characterized by Property elements, each of which consists of the following attributes: *name*, *src*, *dst*, *src_port*, *dst_port* and *l4_proto*. In addition, in the Protection Properties, in addition to the attributes described above, the protectionInfo element is defined in which the encryption and authentication algorithms, all possible inspector nodes and all possible untrusted nodes / links are specified. Here are a couple of examples taken from an XML framework input file.

Listing 3.7. XML example of a Reachability Requirement

```
<Property graph="0" name="ReachabilityProperty" src="10.0.0.1" dst="20.0.0.1"
    dst_port="80" lv4proto="TCP"/>
```

Listing 3.7, represents an XML example of a Reachability Requirement where node 10.0.0.1 must be able to reach destination node 20.0.0.1 at port 80 using the TCP protocol.

Listing 3.8. XML example of a Isolation Requirement

```
<Property graph="0" name="IsolationProperty" src="10.0.0.2" dst="20.0.0.1"
    dst_port="80" lv4proto="TCP"/>
```

Listing 3.8, represents an XML example of an Isolation Requirement where node 10.0.0.2 must not be allowed to reach destination node 20.0.0.1 on port 80 using the TCP protocol.

Listing 3.9. XML example of a Protection Requirement

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.-1" dst="20.0.0.1"
    dst_port="80" lv4proto="TCP">
<protectionInfo encryptionAlgorithm="AES_256_CBC"
    authenticationAlgorithm="SHA2_512">
<inspectorNode node="30.0.0.2"/>
<untrustedNode node="30.0.0.1"/>
</protectionInfo>
</Property>
```

Listing 3.9 represents an XML example of a channel security requirement in which each node belonging to subnet 10.0.0.0/24 must open secure TCP communication when it wants to contact destination 20.0.0.1 on port 80. Furthermore, the property specifies the authentication algorithm: $SHA2\_512$, the encryption algorithm: AES_256_CBC and the information on the nodes belonging to the network, in fact the node 30.0.0.1 is classified as an untreated node instead the node 30.0.0.2 is classified as an inspector node. Finally it can be noted that the value -1 is used to express the wildcard character $*$ in the framework.

Chapter 4

Firewall and Channel Protection Configuration

This chapter presents all the problems that arise in the configuration of a security device. The first part of the chapter lists the problems of manually configuring firewalls, describing the possible solutions provided by the automation of the configuration. Subsequently, the fundamental fields of a Packet Filter, Firewall used in this thesis are listed and an example is presented in the XML code.

The second part describes the configuration of the Channel Protection Systems, with an overview of the possible anomalies introduced by the network administrators during the manual configuration of the safety devices, which underlines the importance of the problem of the wrong configuration, solved by the automation of the devices. Next, the basic fields of a Channel Protection System are listed and an example is presented in the XML code.

Finally, conflicts (Intrapolicy and Interpolicy) that can occur between different policies on the same security device or on different devices are analyzed, so that network security administrators can manage them in order to eliminate any network vulnerabilities.

4.1 Firewall Configuration and Packet Filter

A firewall is a network security device that acts as a protective shield for the network, so that it can protect it from unwanted attacks from outside. This is achieved by configuring policies that are based on the type of network, such as public or private, and by setting security rules that block or allow access to prevent potential hacker or malware attacks.

The correct configuration of the firewalls, through the Filtering Policy, is an essential aspect, since the predefined features could not provide maximum protection against a cyber attack. This configuration performed manually, in addition to requiring time and a not negligible responsibility, is subject to human errors that can lead to conflicts and anomalies. To overcome the limitations due to the manual configuration of the Filtering Policy, the concept of Security Automation is used, a

paradigm that through automatic control processes establishes the policies of the security features, so as to completely reduce possible human errors.

4.1.1 Manual and Automatic Configuration of Firewalls

A firewall is a network device based on correspondence-action rules and is configured so that if a packet meets the requirements provided by a rule, the related actions are performed. For example, in Figure 4.1, where an example of a Filtering Policy configuration is shown, line 1 denies all tcp traffic with source address 140.192.37.10 and destination port 80 and line 12 allows you to configure the default behaviour of the firewall to discard all traffic that doesn't match any rules.

However, a manual configuration of the Filtering Policy presents some problems that can lead to configuration errors or conflicts, such as:

- The probability of causing an error during manual configuration is high because there is no type of automatic check that verifies the correctness of the rules and any redundancies with the rules already installed on the devices;
- Multiple Firewalls on a network cause complexity and probability of failure as multiple filtering policies need to be managed rather than a single rule repository;
- An incorrect configuration could affect the internal security of the network.

		Source		Destination		
	Protocol	Address	Port	Address	Port	Action
1:	tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2:	tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3:	tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4:	tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5:	tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6:	tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7:	tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8:	tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9:	udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10:	udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11:	udp,	140.192.38.*,	any,	161.120.35.*,	any,	accept
12:	udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

Figure 4.1. Filtering Policy example

One solution [3] to solve these configuration problems is the Firmato toolkit, which allows you to automatically configure traditional firewalls using network security policies defined with a Model Definition Language (MDL). The Model Compiler then performs the translation from the mid-level language into the specific firewall configuration file, distributing the rules to multiple instances of this function. Firmato aspects have influenced the design of the VEREFOO ADP module, even if the new NFV and SDN network technologies are not used in its application.

4.1.2 Packet Filter Firewall

Packet filtering is a firewall technology used to control access to the network by monitoring outgoing and incoming packets and allowing them to pass or stop based on preliminary checks on Internet Protocol (IP) addresses, protocols and ports of origin and destination. Packet filtering checks the source and destination IP addresses of the packet in question, the source and destination protocols UDP (User Datagram Protocol) and TCP (Transmission Control Protocol), and the ports. If all parameters find at least one match in the device configuration and the packet filter is configured in whitelist mode, the packet is considered safe and verified.

Despite the easy implementation, the configuration of a packet filter is not trivial, since the rules concerning the application layer, the TCP and UDP protocols, should be translated into rules of the IP layer. The level of security provided by the device is also low as decisions are based on a small number of information, although overall the performance is quite good since the screening action performed can easily drop most of the packets belonging to prohibited communications, avoiding possible Denial-Of-Service attacks that can overload the most complex network monitoring equipment.

4.1.3 Packet Filter model

A Firewall is constituted by a Filtering Policy which represents the configuration according to which the device decides whether a received packet must be forwarded or must be discarded. The filtering policy can be set automatically by the ADP module or manually by the service designer, if they have sufficient knowledge in the field of security. This policy is characterized by two components:

- a set of policies Ψ, which determines the rules that must be satisfied by the package in question;
- a default action Ω;

The default action Ω can assume two values:

1. **DENY:** if the firewall configuration is in whitelist mode;
2. **ALLOW:** if the firewall configuration is in blacklist mode.

The pattern of each rule Ψ is shown below:

$$[actionType - IPSrc - IPDst - portSrc - portDst - Protocol]$$

where:

- ***the actionType:*** through the values DENY or ALLOW it specifies the action to be performed on the packet;
- ***the IPSrc:*** identifies the source address of the traffic processed;
- ***the IPDst:*** identifies the destination address of the traffic processed;
- ***the portSrc:*** it is the source port at the transport layer of the packet to be processed;
- ***the portDst:*** it is the destination port at the transport layer of the packet to be processed;
- ***the Protocol:*** represents the transport layer protocol of the packet to be processed.

When a packet filter receives a packet, it must first analyze whether the rules applied to the packet field belong to the policy rule set Ψ, then if the match is positive, the corresponding action is performed which can be ALLOW or DENY, otherwise if there is no match with the package fields, the default action Ω, action with lower priority, is performed.

Packet filters are one of the two Network Security Functions used in this thesis and are automatically allocated and configured on the input Allocation Graph from the ADP module, which for optimal allocation and configuration must be subject to two optimality constraints:

1. the allocation on the Allocation Graph of the minimum number of Firewalls in order to minimize the resource consumption of the VNFs that will implement them and not to increase the complexity of the network;
2. the minimization of the number of rules configured on each Filtering Policy of a Firewall, so as to facilitate the reading by the service designer so that he can manage and modify it without major problems. Furthermore, a small number of rules would require a smaller amount of memory to store them on the corresponding VNFs.

In fact, after receiving a set of Network Security Requirements as input, the ADP module must establish the minimum set of rules to be configured on the firewalls in order to optimally satisfy all the security requirements. Initially, it is necessary to identify the requirements that for their satisfiability require a specific configuration on the packet filter, for example given a packet filter in blacklisting, only the isolation requirements are considered, since those of reachability are satisfied by the default action .

After the requirements simplifications, soft and hard constraints are inserted into the MaxSMT problem, used to minimize the number of Firewalls allocated and the number of rules configured in a Firewall's Filtering Policy. The reduction of the number of configuration rules has a lower priority than the reduction of the number of Firewalls allocated, this is due to the fact that the allocation of a Network Security Function costs more than the configuration of a Filtering Policy, for this reason it was decided to differentiate the value of the two priorities.

4.1.4 Packet Filter Implementation in the XML schema

In XML schemas, a packet filter is represented by a node element that has the value FIREWALL as a functional attribute. This node, in addition to the traditional elements such as IP addresses, ports, level 4 protocol and list of neighbours, is also characterized by a specific configuration that contains a firewall element. Each firewall element contains an attribute of type defaultAction that can take the value DENY or ALLOW depending on whether the configuration is in whitelisting or blacklisting.

The XML model of a filtering policy rule is shown in the xsd code block depicted in Figure 4.2.

```
<xsd:element name= "elements">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name= "action" type= "TypeAction" minOccurs="0" default=
     "ALLOW"/>
      <xsd:element name= "source" type= "xsd:string"/>
      <xsd:element name= "destination" type= "xsd:string"/>
      <xsd:element name= "protocol" type= "ProtocoloL4" minOccurs= "0"
          default= "ANY"/>
      <xsd:element name= "src_port" type= "xsd:string" minOccurs= "0"/>
      <xsd:element name= "dst_port" type= "xsd:string" minOccurs= "0"/>
      <xsd:element name= "directional" type= "xsd:boolean" minOccurs=
          "0"default= "true"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.2. xsd schema for the Filtering Policy

This Figure shows a model of a single Firewall filtering policy rule, with the various parameters used with the appropriate occurrences. As you can see in the model we have as actionsType the default value ALLOW, so as to set the packet filter in blacklist mode.

Also, an example of a Filtering Policy that the framework could automatically generate is shown in Listing 4.1.

Listing 4.1. XML schema of the Packet Filter

```
<node functional_type= "FIREWALL" name= "30.0.0.1" >
  <neighbour name= "10.0.0.1"/>
  <neighbour name= "20.0.0.1"/>
  <configuration description= "Firewall_configuration" name= "confFw1" >
    <firewall defaultAction= "ALLOW" >
      <elements>
        <action>DENY</action>
        <source>10.0.0.1</source>
        <destination>20.0.0.1</destination>
        <protocol>ANY</protocol>
        <src_port>*</src_port>
        <dst_port>*</dst_port>
      </elements>
    </firewall>
  </configuration>
</node>
```

As can be seen from the example shown in Listing 4.1, the firewall element is structured by the following attributes:

- ***action element:*** indicates the action of the configured rule, which can be ALLOW or DENY;
- ***source element:*** indicates the source IP address of the rule which can be a single address or a subnet that includes multiple hosts;
- ***destination element:*** indicates the destination IP address of the rule which can be a single address or a subnet that includes multiple hosts;
- ***protocol element:*** indicates the transport layer protocol of the rule, it can assume the value of TCP, UDP or ANY;
- ***src_port element:*** indicates the source port at the transport level, it can take a value between 0 and 65535 or the value * to indicate any port number;
- ***dst_port element:*** indicates the destination port at the transport level, it can take a value between 0 and 65535 or the value * to indicate any port number;
- ***directional element:*** indicates whether the rule is bidirectional or unidirectional.

4.2 Virtual Private Network Configuration

A secure communication is a communication that fully satisfies all security requirements, including header integrity, data integrity and data confidentiality. A channel is a directional exchange of data that can take place in a protected manner with some security properties (protected channel) or with none (non-secure channel). These security properties are applied on the source host and are removed on the destination host and are defined by specifying:

1. Source and destination;
2. Security protocol to use;
3. Security properties;
4. Passages crossed and traffic to be protected.

However, applying these network security properties is not a trivial task, in fact it requires specific skills and a lot of competence on the part of security administrators, who often find themselves configuring network devices with basic tools and without the aid of any tools to debug and check if the applied policy is correct and compliant with high-level security requirements.

4.2.1 Manual Configuration

The manual configuration of NSFs is a complex process that can lead to errors and anomalies that compromise the correct functioning of the network and breach of security, an incident that causes access to information without any authorization. The increase in the complexity of networks and the presence of various network security functions has drastically increased the possibility of human error. Often the security manager does not have the appropriate skills to prevent these errors and to create secure communication.

In article [4] 19 anomalies that can arise during the configuration of the security devices were listed, which are divided into 5 macro categories according to a classification based on the effects. A description is provided below:

1. **Insecure Communication:** unsecured communication occurs when the communication security level is lower than the minimum level. For example, a channel that does not meet the minimum level of security defined in the policy generates an inadequacy anomaly.
2. **Unfeasible Communication:** it occurs if communication cannot be established due to a configuration error. These anomalies are very serious because they completely prevent the exchange of data.

3. **Potential Errors:** anomalies created randomly by network administrators, so their resolution is complicated. For example, when an administrator creates policies with different levels of security for the integrity and confidentiality priorities, the one with the highest priority will be chosen to protect the channel.

4. **Suboptimal Implementations:** it occurs when one or more policies reduce the throughput of the network generating an overload on the nodes. They are not very problematic but their resolution can be useful as it improves the performance of the network making it less vulnerable to DoS attacks.

5. **Suboptimal Walks:** it occurs when there are multiple alternative paths in a communication between two endpoints, so that the path followed by the data is unnecessarily long. This does not lead to a configuration error but reduces the performance and security of the network.

These anomalies, as seen above, can appear in the real world scenario when an administrator performs a manual configuration, so it is essential to create a model that is able to detect and resolve them so as to completely eliminate the configuration errors of the security functions. of network.

4.2.2 Automatic Configuration

The automation of the configuration and allocation of NSFs, thanks to the innovations brought about by the new SDN and NFV technologies, has been the path taken to solve the limits due to manual configuration. However, due to the considerable configuration complexity, the automation process of some security functions proceeds very slowly. For example, the firewalls, seen above, are one of the most used network security functions and also one of the simplest to configure, instead the Channel Protection Systems are very difficult to configure since they have to manage secure communications guaranteeing the integrity, packet authentication and encryption.

The first approaches for generating security policies for a series of requirements passed in input are proposed in [5] and are the following:

1. **The bundle approach:** it is the first approach proposed for the resolution of Ipsec conflicts. The bundle algorithm is divided into two basic parts. In the first, the entire traffic is separated into different sets of disjoint traffic, called bundles, each of which is tied to a set of security requirements. In the second phase, each bundle is assigned a set of actions to ensure that the corresponding security policies are generated.

2. **The direct approach:** in which tunnels are created so that they do not overlap with existing ones. If overlapping tunnels are generated, the tool generates two other non-overlapping tunnels. This approach, compared to the previous one, improves the efficiency of the generation and updating of policies.

3. **The combined approach:** composed of taking charge of the bundle and direct approaches. First use the direct approach and in case of solution used also the bundle approach.

In [6] the ordered split algorithm is presented which, in addition to calculating a correct solution, selects the optimal one with the minimum number of tunnels allocated, managing the encryption and authentication requirements separately. First, starting from the original constraints, it creates an unconstrained set of requirements and then generates canonical solutions of minimum size for the new requirements. A canonical solution is a solution where no pair of tunnels start or end at the same point. An unconstrained set of requirements is a set of requirements without two requirements that share the same from value and the same to value. This algorithm generates fewer tunnels than the approaches described above.

In [7] a heuristic algorithm is presented to generate the tunnels iteratively starting from the longest. Thanks to this approach, conflict-free policies are automatically generated and all security requirements are met, saving on the number of tunnels and on computing power.

Finally in [8] an automatic configuration and allocation of the Channel Protection System without conflicts is carried out, which will be the starting point of this thesis work.

As you can see in these approaches just listed, you can only perform an automatic configuration of the network functions, but not an allocation. Apart from the work done in [8], in all other articles the input of the algorithms used is a network topology where the network security functions are already located. This is due to the non-use of virtualization of network functions and therefore each function is represented in a different physical box lacking in flexibility. Another important aspect is the lack of optimality research, the only one to build an optimal approach is [yang] with the use of a heuristic algorithm. Finally, the lack of scalability can be noted as most of the algorithms have been tested with a number of security requirements that are inconsistent with the size of current networks.

4.2.3 Virtual Private Network model

A VPN gateway is able to operate in three different modes: in ACCESS mode, if it inserts the protection of unsecured traffic; in EXIT mode, if it removes the protection of the protected traffic; in FORWARDER mode, if it receives traffic that must be forwarded to the next node without adding or removing security.

Furthermore, the VPN gateway, unlike other security functions such as the Firewall, is a network function that modifies the packets that pass through it by adding or removing external headers. The two behaviors occur if:

- **adding the external header:** the VPN gateway is configured in ACCESS mode, so for each received packet, if it meets the requirements, it inserts an external header in the packet with the source address its own address and the termination address as the destination address of the tunnel;

- **remove the outer header:** the VPN gateway is configured in EXIT mode, so for each received packet, if it meets the requirements, it removes the outermost header of the packet.

A GatewayVPN is characterized by a Security Policy that represents the configuration according to which the device decides whether protection must be added or removed to a received packet or must be forwarded without any modification. The security policy can be set automatically by the ADP module or manually by the service designer, if they have sufficient knowledge in the field of security. This policy is characterized by two components:

- the forwarder action Ω.
- the set of policy rules Ψ, where the actions to be performed on all packet that satisfy the expressed conditions are established;

The model of the rules present in the Ψ set is shown below:

$$[behavior - start - end - conditions - authenticationAlg - encryptAlg]$$

where:

- ***behavior:*** specifies the behavior must be performed on each packet that satisfies the conditions of the rule and can assume the values ACCESS and EXIT;
- ***start:*** is the initiator node of the tunnel;
- ***end:*** is the terminator node of the tunnel;
- ***conditions:*** $[IPSource - IPDestination - portSrc - portDst - Protocol]$:
  - IPSource: IP address of the traffic source for which the action is applied;
  - IPDestination: IP address of the traffic destination for which the action is applied;
  - portSrc: source port at the transport level of the channel for which the action is applied;
  - portDst: destination port at the transport level of the channel for which the action is applied;
  - Protocol: protocol at the transport level of the channel for which the action is applied;
- ***authenticationAlg:*** authentication algorithm used to authenticate all packets that meet the conditions of the rule;
- ***encryptAlg:*** Encryption algorithm used to encrypt all packets that meet the conditions of the rule.

When a packet is received by a VPN gateway, the conditions of each rule of the set are applied Ψ, if the correspondence between a rule and the packet is positive, the action expressed in the behavior value is performed which can be an action of added of the protection, ACCESS value, or a unprotection action, EXIT value. Finally, if there is no match, the packet is forwarded without any changes.

Furthermore, in addition to the VPN gateways, the end-Hosts are also able to create secure channels without the aid of additional VPN gateways. In fact, thanks to an improvement in the hardware of these devices, the end-Hosts are able to add protection if the end-Host is a source node or remove the protection if the end-Host is a destination node. To specify this behavior, the hasVPNCapability predicate was introduced in the end-Host, which specifies whether or not the VPN functionality is enabled on that node. This addition has led to significant improvements in the consumption of computational resources, in fact a solution with end-Hosts enabled in the construction of tunnels is preferable to a solution with VPN gateway allocation.

4.2.4 VPN Implementation in the XML schema

n the XML schema, a VPN gateway is represented by a node element that has the value VPNGateway as a functional attribute. This node, in addition to the traditional elements such as IP addresses, ports, layer 4 protocol and list of neighbors, is also characterized by a specific configuration that contains a vpnGateway element. A vpnGateway element consists of a set of SecurityAssociations that represent the rules of the Security Policy expressed through a medium level representation.

In Figure 4.3, it is represented by a block of xsd code, a model of a single rule of the security policy in which the parameters used and the appropriate occurrences are provided.

```
<xsd:complexType name= "SecurityAssociationType" >
  <xsd:sequence>
    <xsd:element name= "behavior" type= "BehaviorTypes" minOccurs= "0"/>
    <xsd:element name= "startChannel" type= "xsd:string" minOccurs= "0"/>
    <xsd:element name= "endChannel" type= "xsd:string" minOccurs= "0"/>
    <xsd:element name= "source" type= "xsd:string"/>
    <xsd:element name= "destination" type= "xsd:string"/>
    <xsd:element name= "protocol" type= "L4ProtocolTypes" minOccurs= "0"
        default= "ANY"/>
    <xsd:element name= "src_port" type= "xsd:string" minOccurs= "0"/>
    <xsd:element name= "dst_port" type= "xsd:string" minOccurs= "0"/>
    <xsd:element name= "authenticationAlgorithm" type=
        "AuthenticationAlgorithmType" minOccurs= "0"/>
    <xsd:element name= "encryptionAlgorithm" type="EncryptionAlgorithmType"
        minOccurs= "0"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4.3. xsd schema for the Security Association

An example of a Security Policy that the framework could automatically generate is shown in Listing 4.2.

Listing 4.2. XML schema of the VPNGateway

```
<node functional_type= "VPNGateway" name= "30.0.0.1" >
  <neighbour name= "10.0.0.1"/>
  <neighbour name= "40.0.0.1"/>
  <configuration description= "gatewayVPN" name= "confVPN1" >
    <vpnGateway>
      <securityAssociation>
        <behavior>ACCESS</behavior>
        <startChannel>30.0.0.1</startChannel>
        <endChannel>30.0.0.2</endChannel>
        <source>10.0.0.1</source>
        <destination>20.0.0.1</destination>
        <protocol>ANY</protocol>
        <src_port>*</src_port>
        <dst_port>*</dst_port>
        <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
        <encryptionAlgorithm>AES_128_CBC</encryptionAlgorithm>
      </securityAssociation>
    </vpnGateway>
  </configuration>
</node>
```

As can be seen from the example shown in Listing 4.2, the vpnGateway element is constituted by the following attributes:

- ***behavior element:*** indicates the behavior that the VPN gateway must assume and can have the value ACCESS or EXIT;
- ***startChannel element:*** indicates the IP address of the tunnel initiator node;
- ***endChannel element:*** indicates the IP address of the tunnel terminator node;
- ***source element:*** indicates the source IP address of the rule which can be a single address or a subnet that includes multiple hosts;
- ***destination element:*** indicates the destination IP address of the rule which can be a single address or a subnet that includes multiple hosts;
- ***protocol element:*** indicates the transport layer protocol of the rule, it can assume the value of TCP, UDP or ANY;
- ***src_port element:*** indicates the source port at the transport layer, it can take a value between 0 and 65535 or the value $*$ to indicate any port number.
- ***dst_port element:*** indicates the destination port at the transport layer, it can take a value between 0 and 65535 or the value $*$ to indicate any port number.

- ***authenticationAlgorithm element:*** indicates the authentication algorithm used to create the tunnel;

- ***encryptionAlgorithm element:*** indicates the encryption algorithm used by the for the creation of the tunnel.

4.3 Firewall and Channel Protection Conflict

Due to increasing network attacks, network security devices such as firewalls and IPSec gateways have become critical elements for designing not only corporate networks but also small home networks. Deploying these technologies on your network offers tremendous flexibility, so you can customize the right security for different networks and applications. However, configuring network security policies remains a complex and error-prone task due to the rules and policy interaction in the network. In fact, unawareness of the types of policy conflicts and a lack of automated support can lead network administrators to fail device security certification. These errors can cause serious security and network breaches such as blocking authorized traffic, authorizing unwanted traffic, and unsecured data transmission. Just think that an ordering of the rules can make some rules obsolete so as to create a security flaw vulnerable to network attacks or that the criticisms of the IPSec tunnels in can allow the transmission in the clear of reserved traffic, thus violating the security policies.

As we will see below, these conflicts occur due to misconfiguration of rules within a single policy (intrapolicy conflicts) or between policies on different devices (interpolicy conflicts) as described in [9].

4.3.1 Intrapolicy Access List Conflicts

The relationship or dependence between the rules is fundamental to determine any conflict in the security policy, in fact if two rules are disjoint (there is no relationship between the two), any ordering of the rules in the security policy is valid. Therefore, classifying all possible relationships between rules is the first step in understanding the source of conflicts due to incorrect policy configuration. The relationships of the filtering rules can be defined as follows:

- **Exactly matching rules** ($Rx = Ry$)**:** The Rx and Ry rules match exactly if each field in Rx is equal to the corresponding field in Ry.

- **Inclusively matching rules** ($Rx \subset Ry$)**:** The Rx rule inclusively matches Ry if the rules do not match exactly and if each field in Rx is a subset of or equal to the corresponding field in Ry. Rx is called subset match while Ry is called superset match.

- **Correlated rules** ($Rx \bowtie Ry$)**:** Rules Rx and Ry are correlated if at least one field in Rx is a subset or partially intersects with the corresponding field in Ry, and at least one field in Ry is a superset or partially intersects with the corresponding field in Rx, and the rest of the fields are equal. This means that there is an intersection between the address space of the correlated rules, although neither rule is a subset of the other.

- **Disjoint rules:** Rules Rx and Ry are completely disjoint if every field in Rx is not a subset and not a superset and not equal to the corresponding field in Ry.

After identifying the possible relationships between the rules, the first type of conflict is illustrated, which is created between rules present in the same secutiry device, where it is very common to find rules interconnected with each other. In this case, different rule orders can imply different behaviors and even execution errors. Therefore, the rules not ordered with care can be hidden by others so as to obtain an incorrect configuration. The possible conflicts that can arise between the rules of a single security device are classified below:

- **Intrapolicy Shadowing:** a rule is obscured when it is preceded by a rule with a different action. As a result, this rule will never apply.

- **Intrapolicy Correlation:** There is a correlation conflict when two rules are related to each other and have different type of action. This dependency is not intuitive and leads to ambiguity in the definition of the security policy.

- **Intrapolicy Exception:** a rule is an exception to the next rule if it has different actions and the following rule is a superset match of the rule itself.

- **Intrapolicy Redundancy:** a rule is redundant when there is another rule that obscures it and therefore packages will never use it.

4.3.2 Interpolicy Access List Conflicts

This type of conflict concerns conflicts that can occur between different device policies. For example, an upstream device could block traffic allowed by another downstream device or it could protect it, which is not allowed by the downstream device. In both cases the traffic will not reach the destination as it will be discarded by the upstream devices. Based on the flow direction, a previous device is called an upstream device while a subsequent one is called a downstream device. Conflicts arise if a further downstream device allows traffic blocked by one of the upstream devices and if a upstream device allows traffic blocked by one of the downstream devices. On the other hand, all intermediate devices should allow / exclude any traffic allowed / protected by the most upstream and downstream devices so that the flow can reach its destination. Among the interpolicy conflicts we distinguish the following:

- **Interpolicy Shadowing:** Unlike intrapolicy shadowing, this conflict occurs between two rules of different devices, in fact it occurs if an upstream policy blocks part of the traffic allowed by a downstream policy.
- **Interpolicy Spuriousness:** Traffic is spurious if the upstream device allows some of the traffic that is blocked by the downstream device. This has serious consequences for network security because it allows unwanted traffic to pass through the network.

4.3.3 Considerations

As seen above, configuring security device policies is a critical task, particularly in large networks such as enterprise networks. Given the large number of policy rules, a manual configuration can cause numerous conflicts causing serious vulnerabilities and network threats such as flooding attacks and insecure transmission. These conflicts can include improper traffic control, such as shadowing and spuriousness conflicts, or incorrect traffic protection, such as nested / overlapped security session conflicts. Network security, like any other technology, requires adequate management support, including automatic conflict analysis and verification in order to provide the required security service. In fact, it is the task of the network administrator to take charge of all possible conflicts internal or external to the device and try to resolve them so as to eliminate all possible errors, making the network less vulnerable to external attacks.

Chapter 5

Thesis Goals

This chapter introduces the main objective of this thesis work, describing the techniques and approaches used to meet the overall purpose of the thesis. The first part explains how the main objective was divided into smaller objectives, the achievement of which is essential for achieving the total objective of the thesis.

Finally, the approaches used in this work to achieve the individual objectives are introduced in detail, motivating the choices made and showing the advantages over other types of approaches.

5.1 Objectives

In the previous chapters it was shown how the application of network security in computer systems is a task of fundamental importance even if it is very complex and delicate to manage. It was highlighted that manually managing the configuration of network security functions such as Firewall and Channel Protection System is a very difficult task for network administrators, as it requires very specific skills and a high level of competence. Often they find themselves managing problems due to their manual configuration or to possible conflicts, generated between the policies of the same device or between policies of different devices. As seen, these problems can be overcome thanks to the automation of computer systems by exploiting innovations such as Software Defined Network and Network Function Virtualization, in order to give flexibility, agility and network resilience to the solutions in which this approach is applied.

The automation of the configuration and allocation of network security functions was the main objective of this thesis work, which took place thanks to the creation of a tool capable of simultaneously allocating and configuring the Firewall and Channel Protection System (CPS), managing any conflicts and respecting some optimality criteria in order to satisfy a set of communication requirements. In particular, the main purpose was the expansion and improvement of VEREFOO, a framework already able to meet the network security requirements by separately managing the allocation and configuration of the two security functions. This purpose is broken down into other small goals, whose achievement of the individual goals is fundamental to the main purpose. These goals are:

- The first objective was the reformulation of some constraints of the Communication Protection Network Security in order to improve the performance and increase the scalability of the framework. This was possible thanks to the remodeling of the hard constraints present in the ADP module of the framework, the purpose of which is to ensure the correct number and correct positioning of the VPN gateways within the network to be protected. These constraints, initially created with integer variables, have been rewritten with Boolean variables, in order to simplify the MaxSMT problem to be passed to solver z3.
- The second objective, which was also the main objective of this thesis work, was to expand the ADP module of VEREFOO making it able to perform an automatic allocation and / or configuration of Firewalls and / or Channel Protection System at the same time, after having received in input a set of requirements to be respected, a Service Graph or an Allocation Graph, following optimality criteria through the resolution of a MaxSMT problem. This feature has also been enabled in the presence of network functions which modify the addresses of the traffic that passes through them, an example is given by the NAT function.
- Finally, the third objective was to translate the framework's output into the input accessible to the StrongSwan platform, so that the results obtained from the automatic configuration can be used by the platform for the configuration of Channel Ipsec to be allocated on the network. This was possible thanks to a translator that translates the configuration language of the CSPs present in the VEREFOO output into an input language supported by the StrongSwan platform, so that it can configure correctly on the Ipsec Channel network.

5.2 Thesis Approach

5.2.1 Hard Constraints of Channel Protection System

In the first part of the thesis, the problem of remodeling some constraints built for the allocation / configuration of the Channel Protection System was addressed. These constraints, together with others, constitute the Maximum Satisfiability Module Theories problem (MaxSMT), an extension of the SMT problem, in which the final goal is the maximum satisfaction of the predicative clauses present in the problem.

The MaxSTM problem like the SMT problem, at the worst-case computational complexity level is an NP-complete problem. The main difference between the two problems is given by the maximization of the satisfied clauses, this is done by inserting a weight for each clause to be satisfied, in the case of the SMT problem the weight is unitary.

In this thesis work the partial weighted MaxSTM problem is used, weighted because each clause can be attributed a different weight and consequently the best

solution is given by the clauses of greater value; weighted because in addition to the non-relaxable constraints, constraints in which their satisfiability is necessary in order to obtain a satisfactory solution, relaxable constraints are introduced, where their fulfillment is not fundamental for the achievement of a satisfactory solution. The main constraints present in the weighted partial MaxSMT problem are:

- **Hard Constraints:** non-relaxable constraints, the fulfillment of which is fundamental for the satisfiability of the final solution. Mainly used to satisfy all those actions that must always be guaranteed, such as the actions of the components of the VEREFOO ADP module.
- **Soft Constraints:** relaxable constraints, the fulfillment of which is not fundamental for the satisfiability of the final solution. For each soft constraint a weight is specified which represents the priority of satisfaction of the constraint, in fact in the final solution the highest value of the sum of all the soft constraints satisfied is considered in order to reach an optimal solution.

The hard constraints to be remodeled are represented in formulas 5.1, 5.2, 5.3 and constitute the fundamental constraints to be satisfied for a correct allocation of the Channel Protection System.

$$\sum_{n_i \in \pi(f) | n_i < n} PROTECT(n_i, f) \quad > \sum_{n_i \in \pi(f) | n_i < n} UNPROTECT(n_i, f) \tag{5.1}$$

$$\forall r \in R, \forall f \in \phi(r), \forall n \in \pi(f) \wedge \in v(r)$$

$$\sum_{n_i \in \pi(f) | n_i < n} PROTECT(n_i, f) \quad = \sum_{n_i \in \pi(f) | n_i < n} UNPROTECT(n_i, f) \tag{5.2}$$

$$\forall r \in R, \forall f \in \phi(r), \forall n \in \pi(f) \wedge (n \in \iota(r) \vee n.dstAddr = r.dstAddr)$$

$$\sum_{n_i \in \pi(f) | n_i < l.dstNode} PROTECT(n_i, f) \quad > \sum_{n_i \in \pi(f) | n_i < l.dstNode} UNPROTECT(n_i, f)$$

$$\forall r \in R, \forall f \in \phi(r), \forall l \in \pi(f) \wedge \in \lambda(r) \tag{5.3}$$

where:

- $v(r)$ is the set of untrusted nodes relative to r;
- $\iota(r)$ is the set of inspector nodes relative to r;
- $\lambda(r)$ is the set of untrusted links relative to r;

– $\pi(f)$ is the set of nodes crossed by the flow f;

– $\phi(r)$ is the set of flows relative to r;

– N constitutes the set of network nodes.

– F constitutes the set of network flow.

and the functions take on the following values:

- $PROTECT(n, f) \Longrightarrow$ Boolean with $n \in N, f \in F$
- $UNPROTECT(n, f) \Longrightarrow$ Boolean with $n \in N, f \in F$

we have that:

- Formula 1: given an untrusted node, considering the predecessor nodes, the sum of those that remove the traffic protection must be less than the sum of those that put it. This is an essential constraint to ensure that traffic passes through an untrusted node in a secure manner so as to comply with Network Security Requirements.

- Formula 2: given a final node or an inspector node, considering the predecessor nodes, the sum of those that remove the traffic protection must be equal to the sum of those that put it. This is an essential constraint to ensure that the traffic passes through the inspector node or arrives at the final node in an unprotected way so as to comply with the Network Security Requirements.

- Formula 3: given an untrusted link, considering all the predecessor nodes of the destination node of the untrusted link, the sum of those that remove the traffic protection must be less than the sum of those that put it. This is an essential constraint to ensure that traffic crosses an untrusted link in a secure manner so as to comply with Network Security Requirements.

The constraints defined above in formulas 5.1, 5.2, 5.3, modeled with free integer variables, constitute the hard constraints defined in formulas 1,2,3, in which the sum of the protection gateways and the sum of the unprotection gateways are calculated before a given node.

These constraints form part of the constraints present in the MaxSMT problem to be passed to the z3 solver. Given a non-optimal use of the integers by the z3 solver, this solution is not very scalable from the computational point of view compared to a solution modeled with Boolean variables. In fact, the first objective of this thesis work was the rewriting of these constraints, replacing the integer variables with Boolean variables and the summations with logical operators.

5.2.2 Firewall with Channel Protection System

The main part of this thesis work concerned the expansion of the VEREFOO framework, making it capable of simultaneously allocating and / or configuring network security systems such as Firewalls and Channel Protection Systems. The objectives were to redesign and modify a coherent part of the original framework to comply with the new formulas designed, the analysis of possible conflicts resulting from the presence of two different network security functions and the achievement of excellent computational performance. Thanks to this expansion, the network security administrator can simultaneously specify all three network security requirements enabled in the framework:

1. Isolation requirement;
2. Reachability requirement.
3. Communication protection requirement.

To satisfy the first two network security requirements, automatic allocation and / or automatic configuration of the Firewall is required, for the third network security requirement, automatic allocation and / or automatic configuration of the Channel Protection System is required .

To achieve this goal, a sequential approach was preferred, which considers the final solution as the sum of two optimal partial solutions, compared to an approach given by the merge of the two existing Firewall and CPS configuration / allocation solutions. This choice is motivated by the complexity of the latter solution, since a merge of the two solutions would have required new constraints and a rewriting of the MaxSMT problem to be passed to the solver. In addition to the complexity of the solution, another point against this solution concerns the significant increase in the number of constraints and the complexity of the MaxSMT problem so as to make it not very scalable as the number of NSRs and APs increases. A point against the sequential solution is given in certain situations by the non-reachability of the optimal solution.

To ensure that some solutions are not cut by suboptimal allocation of security functions, we have chosen to run the CPS allocation / configuration tool first. In this way, an incorrect allocation of a Firewall would compromise the satisfiability of the final solution.

In fact, as shown in Figure 5.1, the incorrect allocation of the Firewall compromises the allocation of the VPN Gateways for the satisfaction of the Channel Protection constraints, since it is not possible to allocate any VPN Gateway before the untrusted node given by the ForwarderA.

Another approach, to consider a greater number of satisfiability solutions, so that the sum of the two optimal solutions is also an optimal solution, was to add for each Allocation Place where a VPN gateway is allocated two additional APs, one before and after the considered VPN gateway. This addition of Allocation Place is fundamental to overcome the limit of the sequential solution, in fact, thanks to this

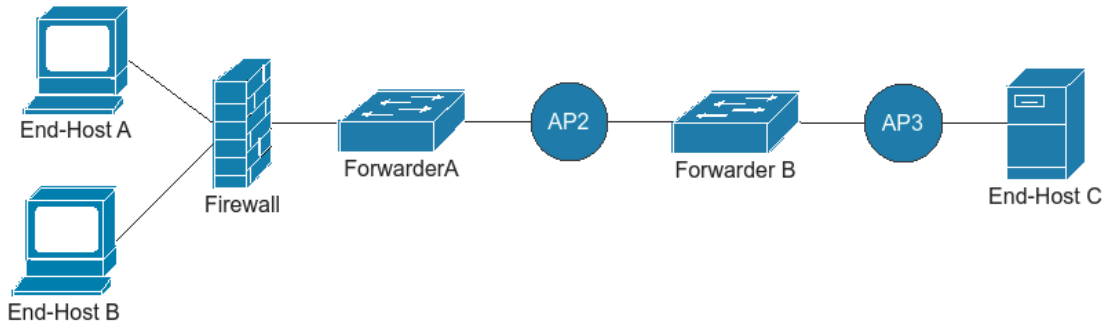

Figure 5.1. Allocation Graph with additional APs

approach, the set of possible final solutions is expanded, increasing the percentage of obtaining an optimal solution.

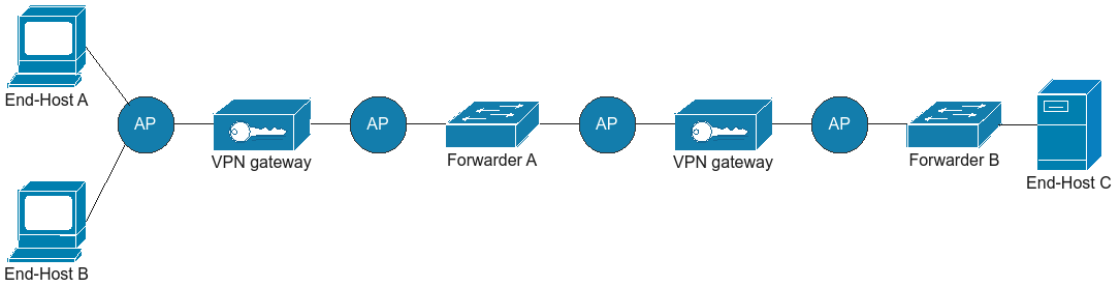

Figure 5.2. Example with the execution of the Firewall allocation / configuration tool

An example is presented in Figure 5.2, which shows an Allocation Graph in which two Allocation Places have been added for each VPN gateway allocated by the framework, in order to consider a greater number of possible solutions in the allocation of any Firewall.

Finally, to fully achieve the set goal, some aspects of the already existing VPN Gateway allocation and configuration tool have been improved, due to an incomplete translation of the z3 output in a medium level language or the configuration of Firewall and VPN Gateway when there are functions that are changing the IP address in progress, such as NAT functions or the VPN functions themselves.

As you can see in the example in Figure 5.3, the presence of the NAT function 70.0.0.1 will result in the modification of the packet header, changing the source IP address with the IP address of the same function. In fact, the Firewall 60.0.0.1 and VPN Gateway 50.0.0.1 functions allocated after the NAT function must not consider the address of the actual traffic source as the source address, but the address of the NAT function.

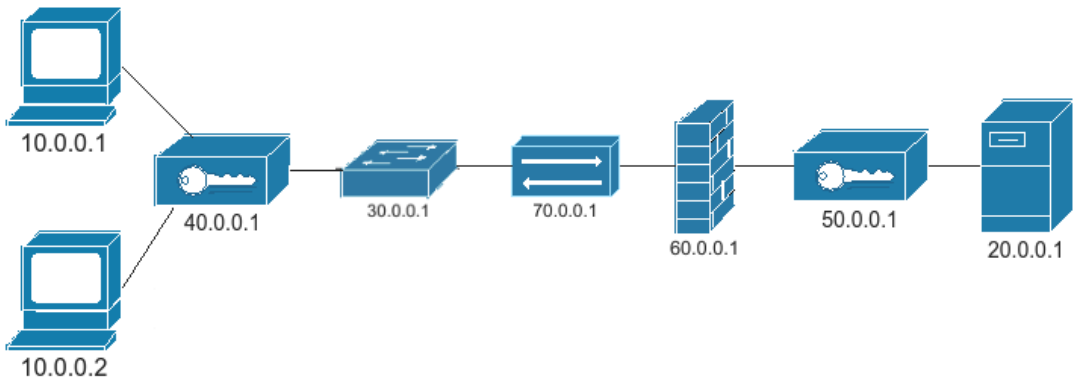

Figure 5.3. Example of a Network with NAT

5.2.3 StrongSwan configuration

In the last goal of this thesis, it was set to create, starting from the configuration of the output VPN Gateways, a configuration for a platform that creates protected Ipsec channels, so as to actually implement the configuration of the security functions. To do this, it was decided to use the StrongSwan platform, an Open Source implementation based on the IKEv2 key exchange protocol and X.509 certificates, capable of creating Ipsec channels for secure communication between two or more End Point groups.

The main objective is, starting from the VEREFOO output configuration, to create an ipsec.conf file so that we can automatically configure the Ipsec channels of our network. This was possible thanks to a translator that translates the configuration language of the VPN Gateways present in the VEREFOO output into an input language supported by the StrongSwan platform, so that it can be configured correctly on the Ipsec Channel network.

Listing 5.1. Example of output VPN Gateway configuration

```
<node name="40.0.0.1" functional_type="VPNGateway">
  <configuration name="AutoConf">
    <vpnGateway>
      <securityAssociation>
        <behavior>ACCESS</behavior>
        <startChannel>40.0.0.1</startChannel>
        <endChannel>50.0.0.1</endChannel>
        <source>10.0.0.1</source>
        <destination>20.0.0.1</destination>
        <protocol>ANY</protocol>
        <src_port>*</src_port>
        <dst_port>*</dst_port>
      </securityAssociation>
    </vpnGateway>
  </configuration>
</node>

<node name="50.0.0.1" functional_type="VPNGateway">
  <configuration name="AutoConf">
```

```
      <vpnGateway>
        <securityAssociation>
          <behavior>EXIT</behavior>
          <startChannel>40.0.0.1</startChannel>
          <endChannel>50.0.0.1</endChannel>
          <source>10.0.0.1</source>
          <destination>20.0.0.1</destination>
          <protocol>ANY</protocol>
          <src_port>*</src_port>
          <dst_port>*</dst_port>
        </securityAssociation>
      </vpnGateway>
    </configuration>
  </node>
```

To better understand the goal, in Listing 5.1 an example of a final configuration of two VPN Gateways is presented, one in ACCESS mode which puts protection and the other in EXIT mode which removes protection and in Figure 5.4 an ipse file is shown with with a possible configuration of a VPN Gateway to be passed to the StrongSwan platform.

```
# /etc/ipsec.conf - strongSwan IPsec configuration file

config setup
        charondebug="all"
        uniqueids=yes
        keyingtries=%forever
        ikelifetime=28800s
        lifetime=3600s
        dpddelay=30s
        dpdtimeout=120s
        dpdaction=restart

conn 40.0.0.1-to-50.0.0.1
        type=tunnel
        auto=start
        keyexchange=ikev2
        authby=secret
        left=40.0.0.1
        leftsubnet=10.0.0.1[%any,%any]
        right=50.0.0.1
        rightsubnet=20.0.0.1[%any,%any]
        ike=aes256-sha1-modp1024!
        esp=AES_256_CBC-SHA2_512!
        aggressive=no
```

Figure 5.4. Example of ipsec.conf file

As you can see the configuration file referred to in the figure above refers to the StrongSwan configuration of the VPN Gateway 40.0.0.1, in fact it can be seen that in its configuration there is only one connection with the other VPN Gateway 50.0.0.1 .

Chapter 6

Reformulation of the Constraints of the Channel Protection System

This chapter deals with the implementation of the first part of this thesis work, that is the reformulation of three constraints in the tool that performs the allocation and configuration of the VPN Gateway. In the first part of the chapter, a brief description is made of the goal to be achieved and the constraints to be remodeled are described. Subsequently, the reformulated constraints are presented with the respective formulas and finally for each constraint the operation is described by solving some examples.

6.1 Introduction to the Problem

The first part of the implementation of this thesis concerned the reformulation of some constraints of the Communication Protection Network Security in order to improve performance and increase the scalability of the framework. This was done by reshaping some of the hard constraints present in the ADP module of the framework, the purpose of which is to ensure the correct number and correct positioning of the VPN gateways within the network to be protected. These constraints, initially created with integer variables, have been rewritten with Boolean variables, in order to simplify the MaxSMT problem to be passed to solver z3.

The formulas in question, presented in the five chapter, have been remodeled with the use of Boolean variables instead of integer variables and state that:

1. given an untrusted node, considering the predecessor nodes, the sum of those that remove the traffic protection must be less than the sum of those that put it. This is an essential constraint to ensure that traffic passes through an untrusted node in a secure manner so as to comply with Network Security Requirements.

2. given a final node or an inspector node, considering the predecessor nodes, the sum of those that remove the traffic protection must be equal to the sum of those that put it. This is an essential constraint to ensure that the

traffic passes through the inspector node or arrives at the final node in an unprotected way so as to comply with the Network Security Requirements.

3. given an untrusted link, considering all the predecessor nodes of the destination node of the untrusted link, the sum of those that remove the traffic protection must be less than the sum of those that put it. This is an essential constraint to ensure that traffic crosses an untrusted link in a secure manner so as to comply with Network Security Requirements.

Given the set Network Security Requirements R, security requirements introduced in input by the network system engineer, for each requirement r belonging to the R set, there can be zero or more inspector nodes, untrusted links or untrusted nodes belonging to a traffic flow or multiple traffic flows that connect the requirement source host with the destination host. Having said that, we have that:

- $\iota(r)$ is the set of inspector nodes relative to r;
- $v(r)$ is the set of untrusted nodes relative to r;
- $\lambda(r)$ is the set of untrusted links relative to r;
- $\pi(f)$ is the set of nodes crossed by the flow f;
- $\phi(r)$ is the set of flows relative to r.
- $\gamma(f)$ the set of links crossed by the flow f.

with:

- $PROTECT(n, f) \Longrightarrow$ Boolean with $n \in N, f \in F$
- $UNPROTECT(n, f) \Longrightarrow$ Boolean with $n \in N, f \in F$

where:

- N constitutes the set of network nodes;
- F constitutes the set of network flow;
- $PROTECT(n, f)$ returns *TRUE* if the node belonging to the flow puts protection, otherwise it returns FALSE;
- $UNPROTECT(n, f)$ returns *TRUE* if the node belonging to the flow removes the protection, otherwise it returns *FALSE*.

we have the following formulas which are the reformulation of the formulas 5.1, 5,2 and 5,3 described in the five chapter:

$$\begin{aligned}&\forall r \in R, \forall f \in \phi(r), \forall n \in \pi(f) \wedge \in v(r), \exists n' \in \pi(f) | n' < n, \\ &(PROTECT(n', f) = true \wedge (\forall \hat{n} \in \pi(f) | n' < \hat{n} < n, \\ &UNPROTECT(\hat{n}, f) = false))\end{aligned} \tag{6.1}$$

$$\begin{aligned}&\forall r \in R, \forall f \in \phi(r), \forall n \in \pi(f) \wedge (n \in \iota(r) \vee n.dstAddr = \\ &r.dstAddr), \forall n' \in \pi(f) | n' < n, PROTECT(n', f) = true \\ &\Rightarrow \exists \hat{n} \in \pi(f) | \hat{n} > n', UNPROTECT(\hat{n}, f) = true)\end{aligned} \tag{6.2}$$

$$\begin{aligned}&\forall r \in R, \forall f \in \phi(r), \forall l \in \gamma(f) \wedge \in \lambda(r), \exists n' \in \pi(f) | \\ &n' < l.dstNode, (PROTECT(n', f) = true \wedge (\forall \hat{n} \in \pi(f) | \\ &n' < \hat{n} < l.dstNode, UNPROTECT(\hat{n}, f) = false))\end{aligned} \tag{6.3}$$

6.2 Constraint1

Formula 6.1: demonstrates that taken an NSR, for each flow of the security requirement, for each node present in the path, if it is an untrusted node, then there must be a preceding node that performs a protection action where among all the successor nodes of the node there is no node performing unprotection action. This is an essential constraint to ensure that traffic passes through it in a secure manner.

The constraint of Formula 7.1 is modeled in z3 with the following model:

$$(or(and(PROTECT(n', f))(and(= UNPROTECT(\hat{n}, f) false))))$$

$$\text{with } n0 < n' < n \text{ and } n' < \hat{n} < n \ .$$

Some examples of the application of this constraint are presented below:

Considering the network in Figure 6.1, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with untrusted node 30.0.0.1, three VPN gateways two configured in ACCESS mode (40.0.0.1, 40.0.0.2), one in EXIT mode (40.0.0.3), the MaxSMT problem created by the solver is as follows:

(((PROTECT (10.0.0.1) and ((UNPROTECT (40.0.0.1) = false))) or (PROTECT (40.0.0.1)))

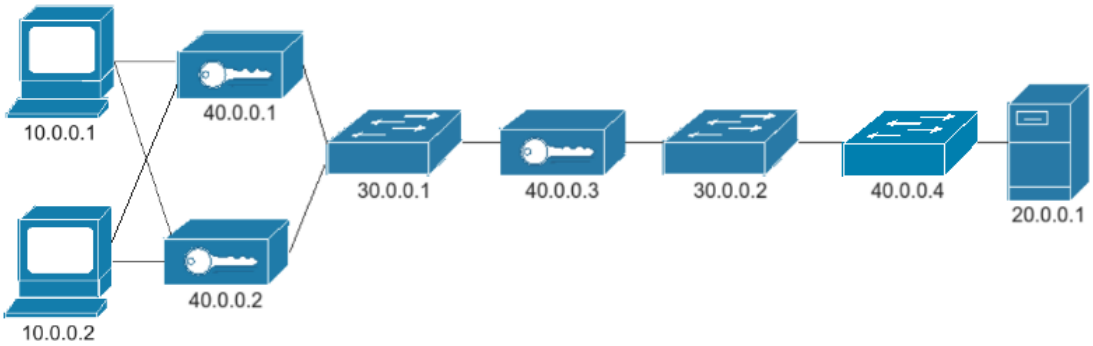

Figure 6.1. Network example1 for channel protection system constraints

the expression with Boolean values is as follows:

((false) and (true)) or (true)

where the solution of the constraint is TRUE then a satisfactory solution is reached. In fact, it can be seen that before the untrusted node 30.0.0.1 there is at least one node (40.0.0.1) that performs a traffic protection action.

Considering the network in Figure 6.1, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with untrusted node node 30.0.0.2, three VPN gateways two configured in ACCESS mode (40.0.0.1, 40.0.0.2), one in EXIT mode (40.0.0.3), the MaxSMT problem created by the solver is as follows:

(((PROTECT(10.0.0.1) and ((UNPROTECT(40.0.0.1) = false) and (UNPROTECT(30.0.0.1) = false) and (UNPROTECT(40.0.0.3)=false))) or ((PROTECT(40.0.0.1) and((UNPROTECT(30.0.0.1) = false) and ((UNPROTECT(40.0.0.3)=false))) or ((PROTECT(30.0.0.1) and ((UNPROTECT(40.0.0.3) = false))) or ((PROTECT(40.0.0.3)))

the expression with Boolean values is as follows:

(((false) and ((true) and (true) and (false))) or ((true) and ((true) and (false))) or ((false) and ((false))) or ((false)))

In this case the solution of the constraint is FALSE therefore a satisfactory solution is not reached. In fact, it can be noted that before the untrusted node 30.0.0.2 there is no node that performs a traffic protection action without its successors carrying out an unprotection action, in fact the node 40.0.0.1 performs a traffic protection action and node 40.0.0.3 performs a unprotection action.

Considering the network in Figure 7.2, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with untrusted node node 30.0.0.1, the MaxSMT problem created by the solver is the following:

(((PROTECT (10.0.0.1) and ((UNPROTECT (40.0.0.1) = false))) or (PROTECT (40.0.0.1)))

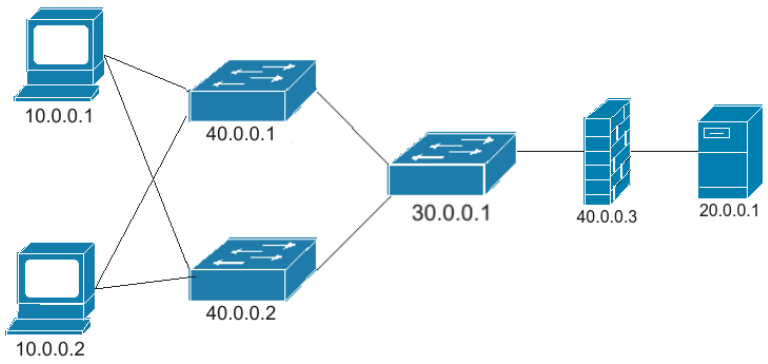

Figure 6.2. Network example2 for channel protection system constraints

the expression with Boolean values is as follows:

((false) and (true)) or (false)

Also in this case the solution of the constraint is FALSE therefore a satisfactory solution is not reached. In fact, it can be seen that before the untrusted node 30.0.0.1 there is no node that performs a traffic protection action since no VPN gateway is allocated in the network.

6.3 Constraint2

Formula 6.2: demonstrates that taken an NSR, for each flow of the security requirement, for each node present in the path, if there is an inspector node or an end node, for any predecessor node of the node in question if there is a node that performs a protection action , there must be at least one successor node of the node that performs an unprotection action. This is an essential constraint to ensure that traffic passes through it in an unprotected way.

The constraint of Formula 6.2 is modeled in z3 with the following model:

$$(and(ite(PROTECT(n', f))(or(UNPROTECT(\hat{n}, f)))(true))$$

$$\text{with } n0 < n' < n \text{ and } n' < \hat{n} <= n \ .$$

Some examples of the application of this constraint are presented below:

Considering the network in Figure 6.1, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with inspector node 30.0.0.2, three VPN gateways two configured in ACCESS mode (40.0.0.1, 40.0.0.2), one in EXIT mode (40.0.0.3), the MaxSMT problem created by the solver is as follows:

((if (PROTECT(10.0.0.1) (UNPROTECT(40.0.0.1) or UNPROTECT(30.0.0.1) or UNPROTECT(40.0.0.3) or UNPROTECT(30.0.0.2)) else (true)) and (if (PROTECT(40.0.0.1) (UNPROTECT(30.0.0.1) or UNPROTECT(40.0.0.3) or UNPROTECT(30.0.0.2)) else (true)) and (if (PROTECT(30.0.0.1) (UNPROTECT(40.0.0.3) or UNPROTECT(30.0.0.2)) else (true)) and (if (PROTECT(40.0.0.3) (UNPROTECT(30.0.0.2)) else (true)))

the expression with Boolean values is as follows:

((true) and (false or true or false) and (true) and (true))

where the solution of the constraint is TRUE then a satisfactory solution is reached. In fact, it can be noted that before the node inspector 30.0.0.2, if there is a node that performs a protection action (40.0.0.1) then there must be at least one node that performs a unprotection action (40.0.0.3).

Considering the network in Figure 6.3, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1, a VPN gateway configured in ACCESS mode (40.0.0.1), with no ability to create tunnels by the end host, the MaxSMT problem created by the solver to verify the constraint on the end node is the following:

((if (PROTECT(10.0.0.1) (UNPROTECT(40.0.0.1) or UNPROTECT(30.0.0.1) or UNPROTECT(20.0.0.1)) else (true)) and (if (PROTECT(40.0.0.1) (UNPROTECT(30.0.0.1) or UNPROTECT(20.0.0.1)) else (true)) and (if (PROTECT(30.0.0.1) (UNPROTECT(20.0.0.1)) else (true)))

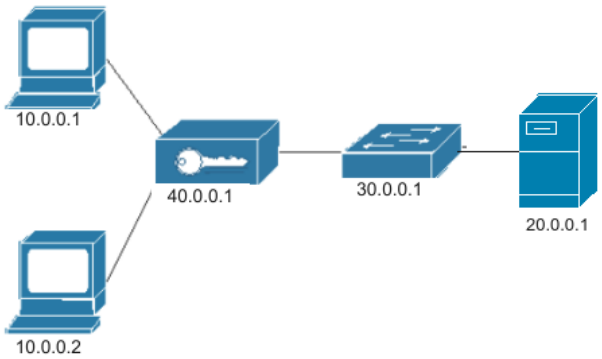

Figure 6.3. Network example3 for channel protection system constraints

the expression with Boolean values is as follows:

((true) and (false or false) and (true))

in this case the solution of the constraint is FALSE therefore an unsatisfactory solution is reached. In fact it can be seen that before the end node 20.0.0.1 there is a node that performs a protection action (40.0.0.1) without any subsequent node that performs a unprotection action, so the destination 20.0.0.1 is reached by of protected traffic.

6.4 Constraint3

Formula 6.3: demonstrates that taken an NSR, for each flow of the security requirement, for each link present in the path, if an untrusted link is found, then there must be a preceding node that performs a protection action where among all the successor nodes of the node there is no node performing an unprotection action. This is an essential constraint to ensure that traffic passes through it in a secure manner.

The constraint of Formula 6.3 is modeled in z3 with the following model:

$$(or(and(PROTECT(n', f))(and(= UNPROTECT(\hat{n}, f) false))))$$

$$\text{with } n0 < n' < l.dstNode \text{ and } n' < \hat{n} < l.dstNode \ .$$

Some examples of the application of this constraint are presented below:

Considering the network in Figure 6.1, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with untrusted link the link going from node 40.0.0.1 to node 30.0.0.1, three VPN gateways, two configured in mode ACCESS (40.0.0.1, 40.0.0.2), one in EXIT mode (40.0.0.3), the MaxSMT problem created by the solver is as follows:

(((PROTECT (10.0.0.1) and ((UNPROTECT (40.0.0.1) = false))) or (PROTECT (40.0.0.1)))

the expression with Boolean values is as follows:

((false) and (true)) or (true)

where the solution of the constraint is TRUE then a satisfactory solution is reached. In fact, it can be seen that before the untrusted link there is at least one node (40.0.0.1) that performs a traffic protection action.

Considering the network in Figure 6.1, a channel protection requirement between source 10.0.0.1 and destination 20.0.0.1 with untrusted link the link going from node 40.0.0.3 to node 30.0.0.2, three VPN gateways two configured in ACCESS mode (40.0.0.1, 40.0.0.2), one in EXIT mode (40.0.0.3), the MaxSMT problem created by the solver is as follows:

(((PROTECT(10.0.0.1) and ((UNPROTECT(40.0.0.1) = false) and (UNPROTECT(30.0.0.1) = false) and (UNPROTECT(40.0.0.3)=false))) or ((PROTECT(40.0.0.1) and((UNPROTECT(30.0.0.1) = false) and ((UNPROTECT(40.0.0.3)=false))) or ((PROTECT(30.0.0.1) and ((UNPROTECT(40.0.0.3) = false))) or ((PROTECT(40.0.0.3)))

the expression with Boolean values is as follows:

(((false) and ((true) and (true) and (false))) or ((true) and ((true) and (false))) or ((false) and ((false))) or ((false)))

in this case the solution of the constraint is FALSE therefore a satisfactory solution is not reached. In fact, it can be noted that before the untrusted link there is no node that performs a traffic protection action without its successors performing an action of unprotection of the traffic, in fact the node 40.0.0.1 performs a protection action and node 40.0.0.3 performs a unprotection action.

Chapter 7

Packet Filter and Channel Protection System implementation

This chapter describes the way in which the second objective of this thesis work has been achieved, that is the expansion of the VEREFOO ADP module, making it able, at the same time, to perform an automatic allocation and / or configuration of Firewall and Channel Protection System, after having received in input a series of requirements to be respected, a Service Graph or an Allocation Graph, following optimality criteria through the resolution of a MaxSMT problem. In the first part of the chapter, the implementation choices made to achieve the main objective are described and motivated. A total working example of the framework is given next.

The second part of the chapter presents how the configuration conflicts of the Network Security Functions due to the allocation of Firewalls and VPNs on the same network have been overcome.

Finally, in the final part of the chapter, the operation and possible configuration of the two security functions in the presence of the NAT function is described.

7.1 Packet Filter and Channel Protection System Sequential Implementation

The second part of the implementation of this thesis concerned the expansion of the VEREFOO framework, making it capable of simultaneously allocating and / or configuring network security systems such as Firewalls and Channel Protection Systems. This was developed by implementing a sequential solution, which builds the final solution by adding the partial optimal solutions returned by the execution of the framework. This choice, after a preliminary study, turned out to be the best in terms of simplicity of constraint writing and computational performance, since a non-sequential solution would have increased the number of constraints of the MaxSMT problem so as to drastically increase execution times.

In order for some solutions not to be cut off from a non-optimal allocation of the safety functions, we have chosen to run the CPSs allocation / configuration tool first, so that the output of this solution is the input of the allocation / configuration tool Firewalls, so that an incorrect allocation of a Firewall would not compromise the satisfiability of the final solution.

To understand this choice, in the examples presented below we will distinguish the two characteristics of VEREFOO in the allocation and configuration of NSFs in:

- **tool1:** VEREFOO tool for the configuration and allocation of Packet Filter.
- **tool2:** VEREFOO tool for the configuration and allocation of CPSs.

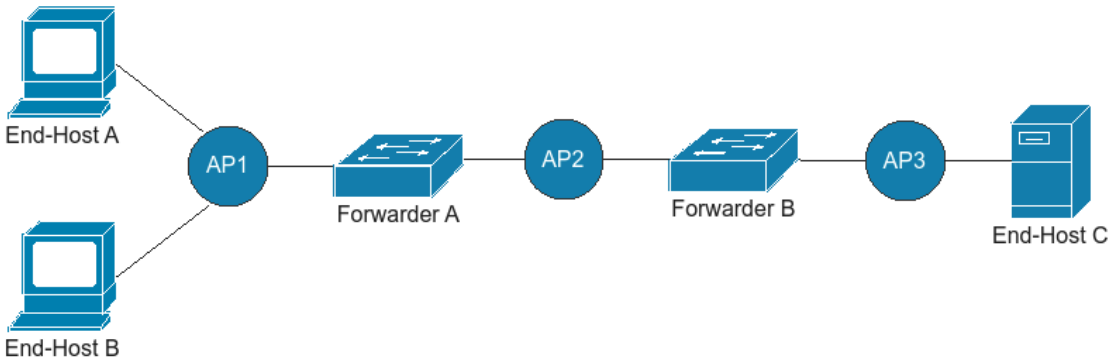

Figure 7.1. Allocation Graph example

An example explains the reason for the implementation choice and the advantages that this choice brings to the final solution. Considering the network shown in Figure 7.1 in which we want to guarantee the protection of the channel for all the traffic coming from the End-HostA source and destined for the End-HostC, given that the ForwarderA is present in the path which is an untrusted node and at the at the same time we want to ensure that the traffic coming from the End-HostB source does not arrive at the End-HostC destination, a solution that first implements the Firewall allocation / configuration tool could lead to a total configuration that does not meet the two security requirements since the Firewall could be allocated in the only place available for the VPN Gateway, an example of this situation is shown in Figure 7.2.

The execution of tool1 returned a SAT solution by allocating a Firewall in AP1 and satisfying the requirement of isolation of all the traffic coming from the End-HostB source and destined for the End-HostC. We can see that this solution meets the isolation requirement but does not meet the communication security requirement, as it is not possible to allocate any VPN gateway before the ForwarderA to protect the traffic before it passes through the untrusted node. In fact, the solution of the tool2, a tool in which the CPS are allocated / configured, would be UNSAT.

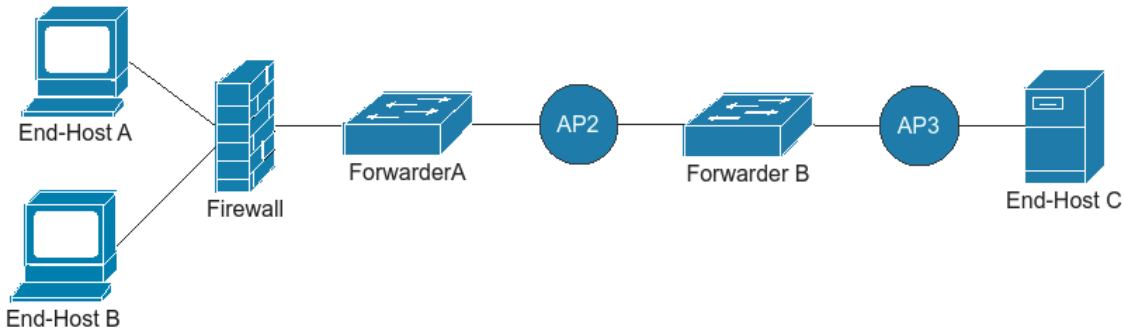

Figure 7.2. Allocation Graph with Firewall

On the other hand, considering first the execution of the Channel Protection System allocation and configuration tool 2, a possible output solution of the framework is presented in Figure 7.3.

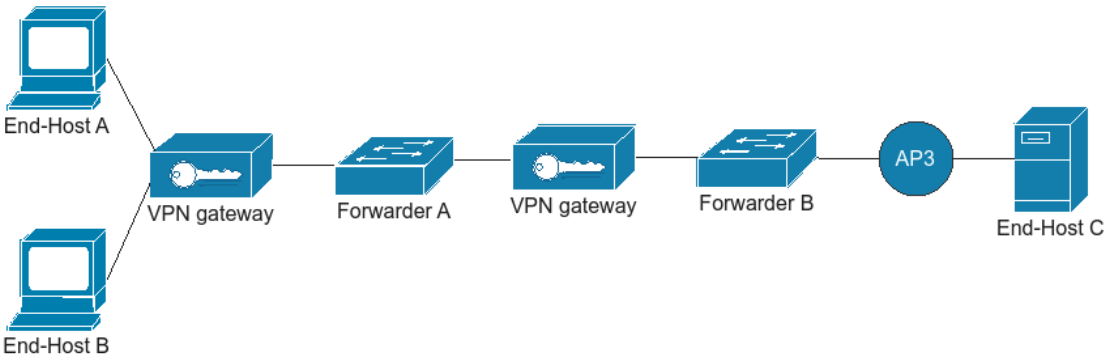

Figure 7.3. Allocation Graph with VPN Gateway

The execution of tool2 returned a SAT solution by allocating two VPN Gateways, one in AP1 and the other in AP2 and satisfying the requirement of Protection of all traffic coming from the End-HostA source and destined for the End-HostC since the traffic is secured before the untrusted node. Then running tool1 on the network shown in Figure 7.3 would produce the output result shown in Figure 7.4.

The execution of tool1 returned a SAT solution by allocating a Firewall in AP3 and satisfying the requirement of isolation of all the traffic coming from the End-HostB source and destined for the End-HostC. In this case it can be seen that the execution sequence of the two tools led us to a satisfactory solution by correctly allocating the three safety functions in the 3 Allocation Places. For this reason, in this thesis it was chosen to run first the tool2 that performs the allocation and configuration of the channel protection system and then in sequence the tool1 in order to avoid incorrect allocations such as the one shown in the example in Figure 7.2.

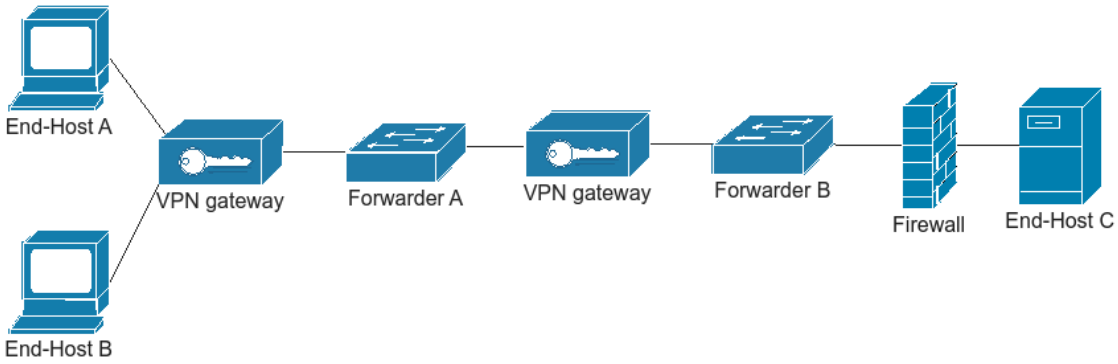

Figure 7.4. Example of output network

Another fundamental part in achieving this goal was the addition of Allocation Place, two for each VPN Gateway allocated in the network, so as to discard the least number of solutions so as to be able to approach the excellent total. This addition is done in the network output by the VPN Gateway's allocation and configuration tool, so that an Allocation Place can host two Network Security Functions at the same time. Each Place of Allocation added to the network has a unique IP address in the network which is assigned to the security function in case of allocation. An example of this addition is shown in image 7.5, where you can see the presence of two APs for each VPN Gateway allocated in the network, one that precedes it and the other that follows it.

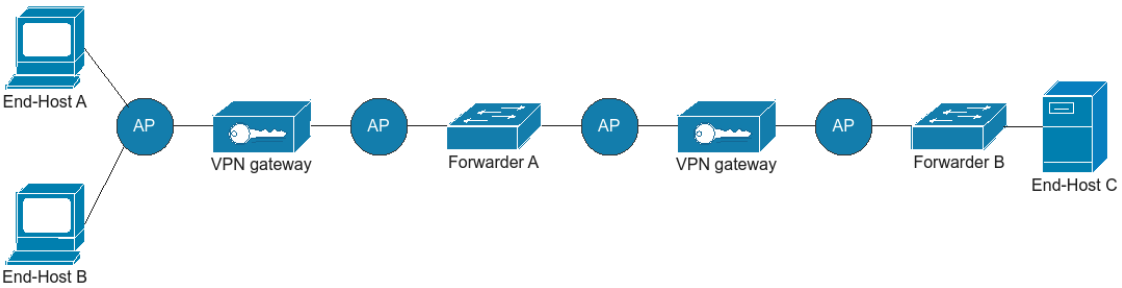

Figure 7.5. Allocation Graph with additional APs

The execution of tool1 on the network in Figure 7.5 would produce as output the network shown in Figure 7.6, where we can see that the framework has allocated and configured a Firewall in the first available AP to discard the traffic coming from the End-HostB source and destined for the End-HostC as soon. Subsequently, the framework eliminated all superfluous APs in order to simplify the final solution.

This solution is better than the one proposed in Figure 7.4, since thanks to the advance allocation of the Firewall it is possible to discard unwanted traffic as soon as possible, avoiding that it transits unnecessarily on the network so as to increase the Throughput of the various links.

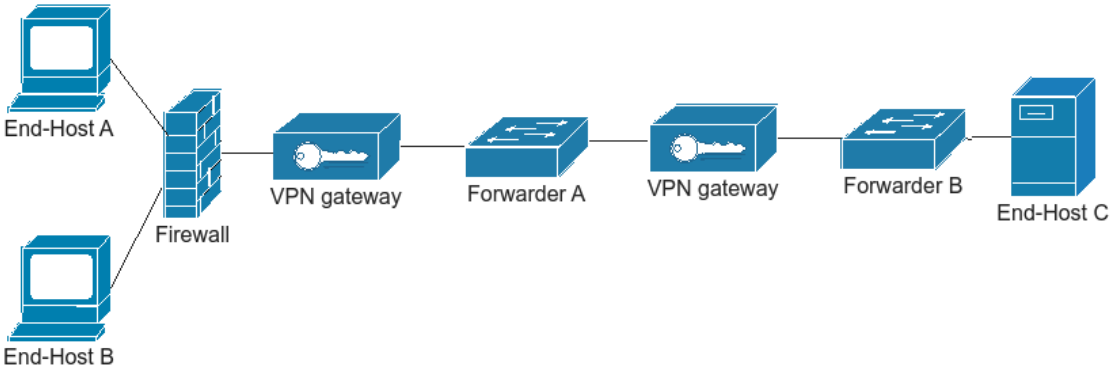

Figure 7.6. Example optimal solution

These Allocation Places have a lower priority than the APs initially allocated in the input Allocation Graph, in fact if in the first case a Forwarder is allocated in case of non-allocation of a network security function, in the second in case of non-use they are directly removed from framework.

Below, to better understand the changes made, is an example of the execution of the framework, which shows how the Allocation Graph shown in Figure 7.7 is modeled in sequence by the execution of the tool.

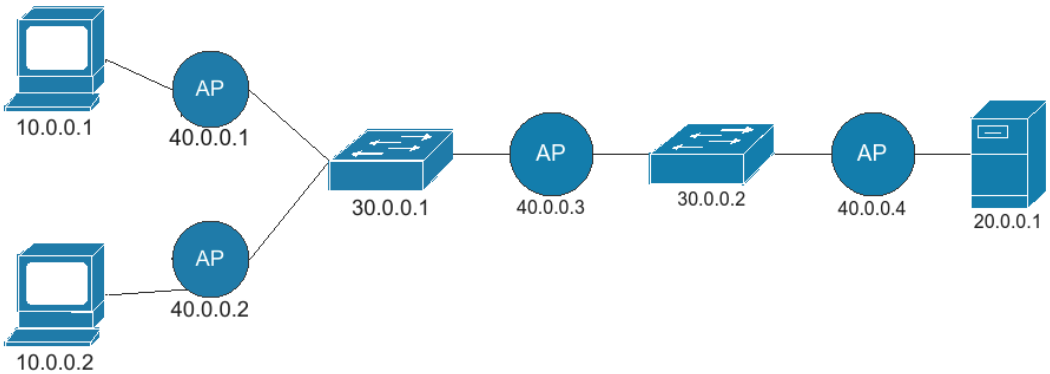

Figure 7.7. Allocation Graph example

Figure 7.7 shows a network made up of two WebHosts, a WebServer, two Forwarders and four Allocation Places where, if possible, the network security functions covered in this thesis will be allocated. Having passed the network of Figure 7.7 and the Security Requirements of the Listing 7.1 as input to the framework, a possible

intermediate solution given by the execution of the first part of the tool is shown in Figure 7.8.

Listing 7.1. Network Security Requirements

```
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
-Protection Property from 10.0.0.2 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
-Isolation Property from 10.0.0.1 to 20.0.0.1;
```

As you can see in Figure 7.8, the execution of the first part of the tool caused the allocation and configuration of three VPN gateways, two in ACCESS mode in nodes 40.0.0.1 and 40.0.0.2 and one in EXIT mode in node 40.0.0.3.

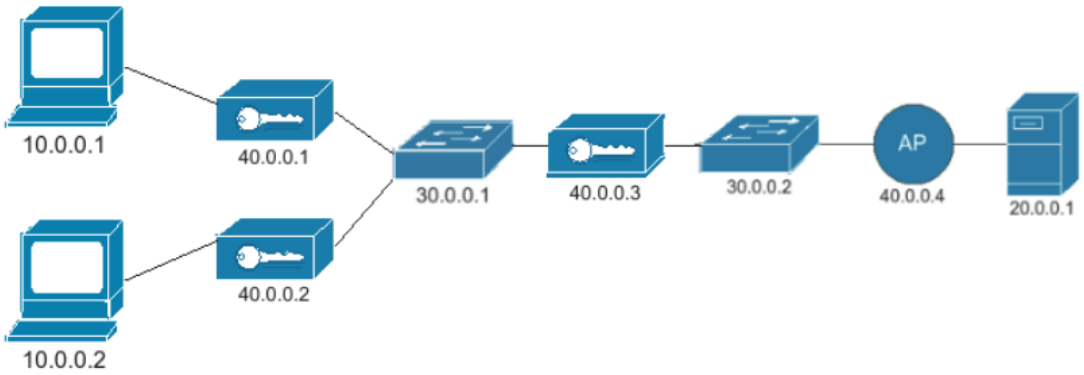

Figure 7.8. Allocation Graph with VPN gateway

With this allocation, the framework has satisfied the two protection constraints passed in input but still has to satisfy the isolation constraint between node 10.0.0.1 and node 20.0.0.1. This constraint will be satisfied later by the second part of the tool.

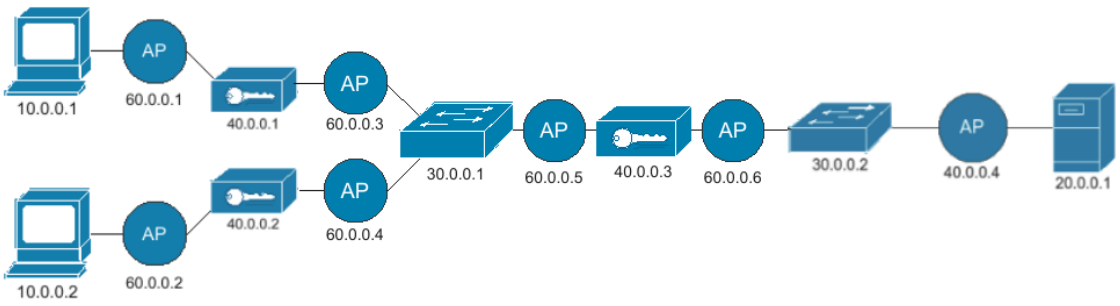

Figure 7.9. Allocation Graph with the addition of APs

Before running the second part of the tool, given the Allocation Graph in Figure 7.8, the framework for each allocated VPN gateway adds two Allocation Places, one preceding it and one following it. This modification shown in Figure 7.9, is useful to avoid discarding optimal final solutions.

Figure 7.10. Final Allocation and Configuration

Finally, Figure 7.10 shows the final network topology, in which the framework has allocated a Firewall to satisfy the isolation requirement from node 10.0.0.1 to node 20.0.0.1. This requirement does not conflict with the reachability requirement given by the protection requirement between node 10.0.0.1 and node 20.0.0.1, since in this case the reachability requirement is created as a soft constraint and therefore its priority is lower with respect to the isolation constraint. For this reason the framework has allocated the Firewall before the VPN gateway, so that the traffic is discarded as soon as possible.

7.2 Conflicts between Packet Filters and Channel Protection Systems

A Packet Filter placed on the path between the source and destination of a secure channel could render the work done by a VPN gateway or a host with VPN capabilities useless as it would prevent traffic from reaching its destination. In fact, a Firewall in white-list mode, given a non-correspondence of the protected traffic with the rules configured in the Filtering Policy, would block all the protected traffic preventing the source from being able to communicate with the destination. This happens because in the Filtering Policy of the Packet Filter in whit-list mode, appropriate rules are configured to satisfy the Reachability Properties passed as input, but no reachability rule is configured for the traffic flows belonging to the Protection Properties. This aspect is of fundamental importance because in the presence of a Firewall no protected communication would reach its destination.

In this thesis this aspect was also addressed, trying to find solutions to best configure the Channel Protection System in the presence of Firewalls in the network. Examples are presented below to better explain how the problem of configuring the two devices was solved.

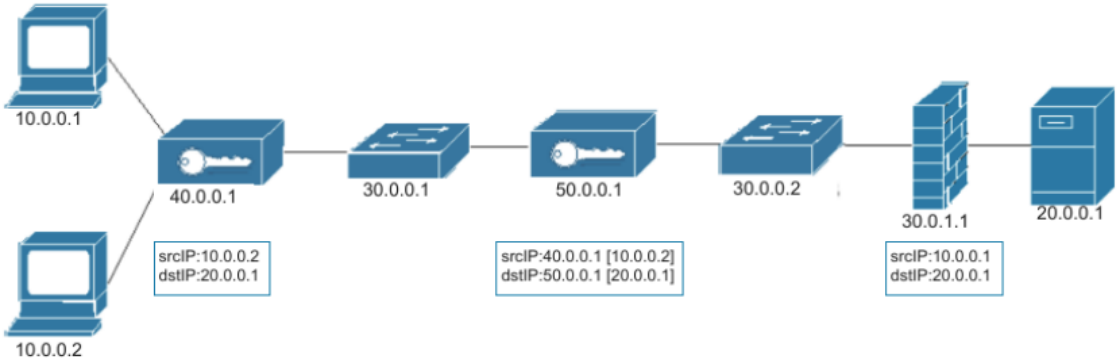

Figure 7.11. Example1 of a network with Packet Filter and VPN

Figure 7.11 shows a first example of the framework output, given the constraints shown in the Listing 7.2 as inputs. For the resolution of the first reachability constraint between node 10.0.0.1 and node 20.0.0.1, given the presence of a firewall 30.0.1.1 configured in white-list mode, a rule is inserted in the firewall that allows all to be forwarded to the exit packages with srcIP: 10.0.0.1 and dstIP: 20.0.0.1.

Listing 7.2. Network Security Requirements1

```
-Reachability Property from 10.0.0.1 to 20.0.0.1;
-Protection Property from 10.0.0.2 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
```

For the resolution of the second traffic protection constraint between node 10.0.0.2 and node 20.0.0.1, two VPN gateways are allocated in the graph, one previous and the other successor to the untrusted node 30.0.0.1. As you can see from the figure, even if all packets with srcIP: 10.0.0.2 and dstIP: 20.0.0.1 are protected correctly before the untrusted node, but the traffic will never reach the destination 20.0.0.1 due to the presence of the Firewall.

Therefore, another capability that has been added to the solver, through the addition of the ReachabilityDerived property, is the ability to find a solution that, in addition to guaranteeing the protection of packets flowing from source to destination according to the security requirements, also that protected packets reach their destination in the presence of a Packet Filter placed somewhere in the path between the source and destination of the stream. In fact, for each traffic flow belonging to a Protection Property, a Reachability Derived Property is created with the same values:

$$[IPSrc, IPDst, portSrc, portDst, Protocol]$$

of the Protection Property in question.

– A **ReachabilityDerived Property** is a reachability property similar to the already existing Reachability Property but with a lower satisfaction priority. The substantial difference between the two properties lies in the creation of the constraints that make up the MaxSMT problem, in fact the Reachability Property is modeled with hard constraints in which their satisfiability is fundamental for the achievement of a satisfactory final solution, instead the *ReachabilityDerived* is modeled with soft constraints, where their satisfiability is not essential for the achievement of a satisfactory final solution.

This is done to give the isolation properties a higher priority than the protection properties. In fact, if you have as input the properties shown in the Listing 7.2, where there are two properties, one for the isolation of the traffic from the source 10.0.0.1 to the destination 20.0.0.1 and one for the protection of the traffic from the source 10.0 .0.1 to the destination 20.0.0.1, the Reachability Derived Porperty built starting from the Protection Property, will not be satisfied as its satisfiability would prevent the satisfaction of the Isolation Property. This is possible thanks to the soft constraint formulation of the ReachabilityDerived Property.

Figure 7.12 illustrates a possible framework output solution, given the constraints shown in the Listing 7.2 as input, with the addition of ReachabilityDerived Property. In this case a new Reachability Property will be created by the framework with srcIP: 10.0.0.2 and dstIP: 20.0.0.1 in order to allow the traffic not to be blocked by the Firewall. A solution similar to the one shown in Figure 6.1 is obtained, but with a small difference in adding a new rule in Firewall Filter Policy 30.0.1.1 to allow the flow from the Protection Property to reach its destination. In this case the rules are merged into a single rule since the two source addresses can be coupled in the 10.0.0.0/24 network.

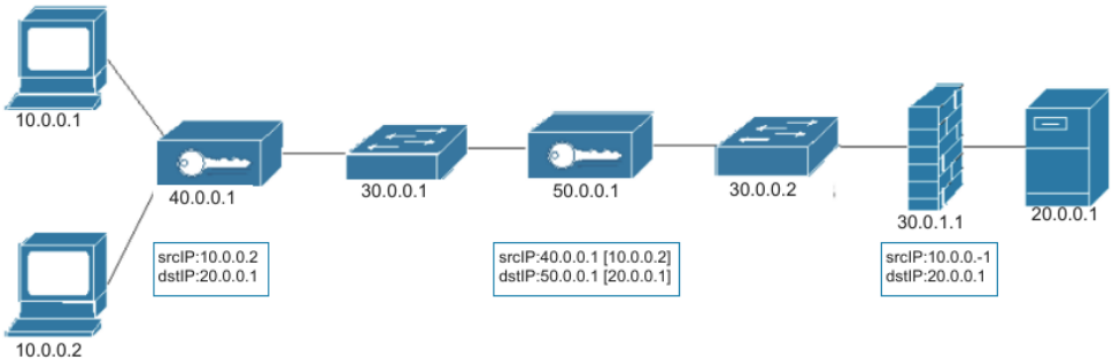

Figure 7.12. Example2 of a network with Packet Filter and VPN

In addition, another possible configuration of output data input the properties of the Listing 7.2, is shown in the example of Figure 7.13, where the Firewall in white-list mode is allocated between the two VPN gateways. In this case, two rules are configured in the Firewall Filtering Policy, one that allows the transit of traffic with srcIP: 10.0.0.1 and dstIP: 20.0.0.1 to satisfy the first reachability requirement, the other rule that allows the transit of the traffic with srcIP: 40.0.0.1 and dstIP: 50.0.0.1, is added to satisfy the *ReachabilityDerived* Property created to allow protected packets to reach their destination.

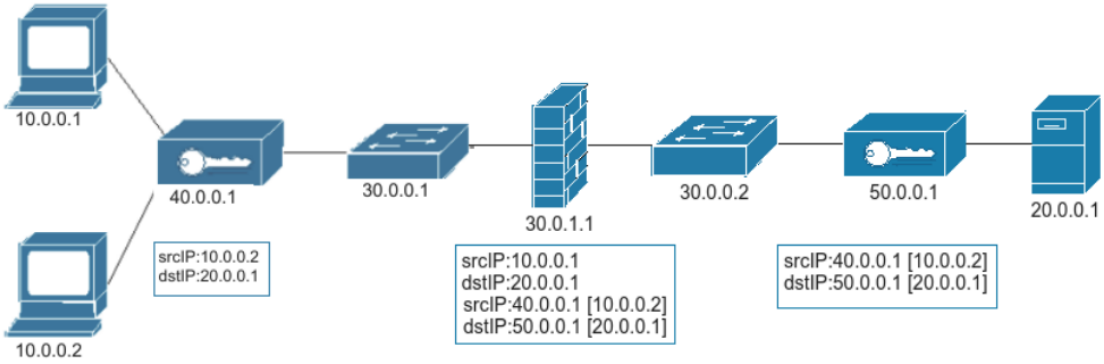

Figure 7.13. Example3 of a network with Packet Filter and VPN

Finally, in Figure 7.14 a last case study is presented, given in input the security properties shown in the Listing 7.3 and a Firewall configured in black-list.

Listing 7.3. Network Security Requirements2

```
-Isolation Property from 10.0.0.1 to 20.0.0.1;
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
```

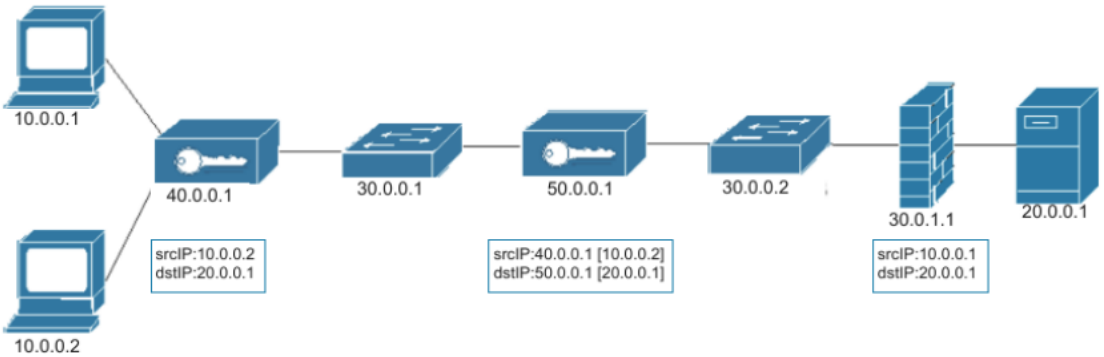

Figure 7.14. Example4 of a network with Packet Filter and VPN

It can be seen that in this case the configuration of the Firewall 30.0.0.1 blocks all traffic with srcIP: 10.0.0.1 and dstIp: 20.0.0.1, so as to make the encryption operation of the VPN gateways allocated on the network useless, since the 'Isolation Property has higher priority to be satisfied than the ReachabilityDerived Property created to satisfy the reachability of protected packets.

7.3 Firewalls and CPSs with NAT functions

Another important point of the work of this thesis was the configuration of the Network Security Functions in the presence of the NAT function. This function performs the task of isolating a network by obscuring the original IP address of the traffic. This is done by changing the packet header, in fact for each packet that passes through the NAT function, the source address is replaced with the address of the same NAT function, so as to hide the IP address of the real traffic source. For this reason, this situation has aroused considerable interest since without a particular implementation of the framework, in the presence of a NAT function, an incorrect configuration of the network security functions would occur.

This functionality was added in the phase of creating the Allocation Graph relating to the Service Graph passed in input. In fact, for each node crossed by a traffic flow belonging to a property, it is checked whether that node is preceded by a NAT function, if yes, the IP address of the NAT function will be inserted as the traffic source to the new node created, otherwise the original address of the traffic source. Here are some examples to better understand how network security functions are configured in the presence of the NAT function.

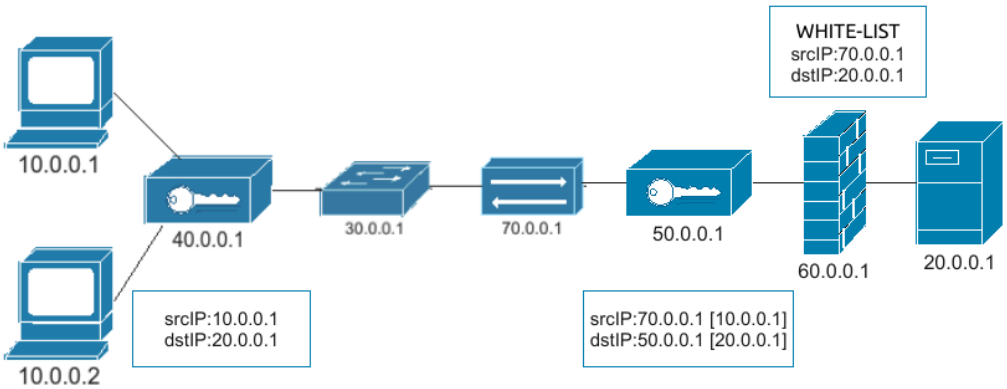

Figure 7.15. Example1 of a network with NAT function

Figure 7.15 shows a final network configuration that meets the requirements given in Listing 7.4. As you can see, two VPN Gateways and a Firewall have

been allocated and configured in this network. The two VPN Gateways have been allocated and configured to meet the traffic protection requirement from source 10.0.0.1 to destination 20.0.0.1 given the untrusted node 30.0.0.1. In fact from the figure it can be seen that the VPN Gateway 40.0.0.1 has been allocated before the untrusted node 30.0.0.1 and has been configured in ACCESS mode so that it can protect all packets that have as srcIP: 10.0.0.1 and as dstIP: 20.0.0.1.

Listing 7.4. Network Security Requirements1 for NAT function

```
-Reachability Property from 10.0.0.2 to 20.0.0.1;
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
```

The second VPN Gateway 60.0.0.1 has been allocated after the untrusted node and has been configured in EXIT mode, so that it can remove the protection from all packets that have in the outermost header, as source address, the address of the first VPN Gateway and as destination address the address of the same VPN Gateway. However, in this case the presence of the NAT function has obscured the IP addresses of the devices that precede it,in fact, in the VPN Gateway 50.0.0.1 as the source IP address, the VPN Gateway address 40.0.0.1 does not appear but the address of the NAT function.

Finally, a Firewall configured in white-list mode had already been allocated in the input network, so to satisfy the Reachability property between the source 10.0.0.2 and the destination 20.0.0.1 and to allow the protected traffic to reach the final destination, the Firewall has been configured so that all packets with source 70.0.0.1 are allowed, since both source 10.0.0.1 and source 10.0.0.2 precede the NAT function and are therefore obscured.

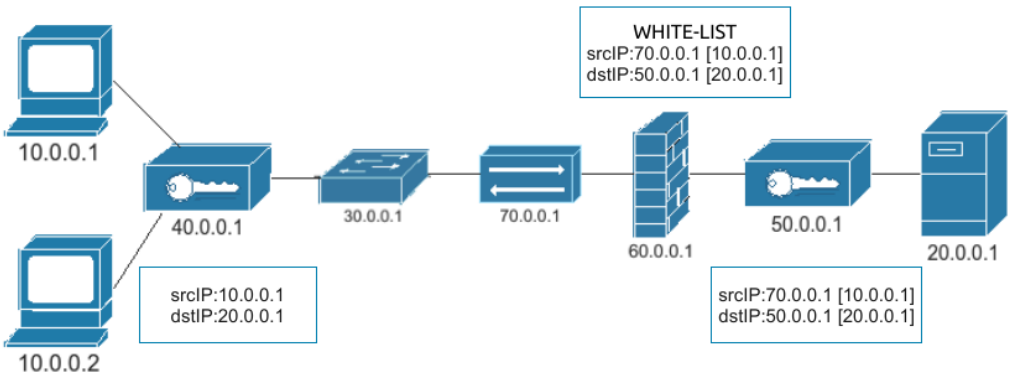

Figure 7.16. Example2 of a network with NAT function

Figure 7.16 shows another example of configuring the security functions in the presence of the NAT function. In this network as in the previous one, to satisfy the requirements shown in the Listing 7.5, two VPN Gateways and a Firewall have been allocated. The two VPN Gateways have been allocated and configured to meet the traffic protection requirement from source 10.0.0.1 to destination 20.0.0.1 given the untrusted node 30.0.0.1. In fact from the figure it can be seen that the VPN Gateway 40.0.0.1 has been allocated before the untrusted node 30.0.0.1 and has been configured in ACCESS mode so that it can protect all packets that have as srcIP: 10.0.0.1 and as dstIP: 20.0.0.1. The VPN Gateway 50.0.0.1, as in the previous example, has been allocated after the untrusted node 30.0.0.1 and has been configured in EXIT mode, so that it can remove the protection from all packets that have in the outermost header the following addresses: srcIP: 70.0.0.1 and dstIP: 50.0.0.1. Also in this case, as in the previous one, the source address is the address of the NAT function.

Listing 7.5. Network Security Requirements2 for NAT function

```
-Isolation Property from 10.0.0.2 to 20.0.0.1;
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1,with encryption algotithm AES_128_CBC
   and authentication algotithm SHA2_256;
```

Finally, the framework to meet the Traffic Isolation Requirement from source 10.0.0.2 to destination 20.0.0.1 has allocated the Firewall 60.0.0.1 and configured it in white-list mode. This Firewall was placed between the two VPN Gateways and after the NAT function. The default behavior of the Firewall in white-list mode satisfies the traffic protection requirement but also blocks all traffic flow produced by the Protection Property. For this reason, an entry has been configured in the Firewall to allow all secure traffic, this entry has the NAT IP as its source IP address and the 50.0.0.1 VPN Gateway address as its destination address.

Chapter 8

StrongSwan Configuration

This chapter describes how the third objective of this thesis work has been achieved, namely the writing of a translator who performs the translation from the VEREFOO output language to the language accessible from the StrongSwan platform. The first part of the chapter gives an overview of the most important parameters present in the ipsec.conf file, a fundamental file for configuring Channel Ipsec using StrongSwan.

Finally, in the final part of the chapter, through examples, it is described how the translator, starting from the configuration of a VPN Gateway in the VEREFOO output, creates the ipsec.conf configuration file.

8.1 Implementation

The third part of the implementation of this thesis work involved the writing of a translator who carries out the translation from a medium-level language to a low-level language accessible from the StrongSwan platform, so that the results obtained from the automatic configuration can be used by the platform for the configuration of Channel Ipsec to be allocated on the network. This translator, consisting of a java class, translates the configuration language of the CSPs present in the VEREFOO output into an input language supported by the StrongSwan platform, so that they can be configured correctly on the Ipsec Channel network.

The translation result is saved in an *ipsec_conf_ \IP_of_Node \.txt* file which can be directly used in the StrongSwan configuration. This file consists of the following parameters:

- **config setup:** Allows you to specify general configuration information for IPsec connections.
  - *charondebug:* defines the amount of debug debug output to log.
  - *uniqueids:* Specifies whether an ID is unique.

  - *keyingtries:* indicates how many times a connection must be attempted to negotiate.
  - *ikelifetime:* indicates the duration time of a connection before being renegotiated.
  - *lifetime:* defines the time of a link instance, from negotiation to expiration date.
  - *dpddelay:* specifies the time interval in which the peer can be notified via messages.
  - *dpdtimeout:* time to elapse before all peer connections are deleted in case of inactivity
  - *dpdaction:* defines the use of the Dead Peer Detection (DPD) protocol for connection management.

- **conn:** defines the name of the connection.
  - *type:* defines the type of connection.
  - *auto:* how the connection should be handled when IPSec starts.
  - *keyexchange:* defines what version of the IKE protocol are using.
  - *authby:* defines how peers must authenticate with each other.
  - *left:* defines the left IP address of the network interface that creates the channel.
  - *leftsubnet:* left node IP network address, ports and protocol.
  - *right:* defines the right IP address of the network interface that closes the channel.
  - *rightsubnet:* right node IP network address, ports and protocol.
  - *ike:* list of encryption and authentication algorithms for building IKE SA.
  - *esp:* list of encryption and authentication algorithms for correct communication.
  - *aggressive:* if you are using the aggressive main mode.

8.2 Implementation Examples

The network depicted in Figure 8.1 represents the output of the framework where two VPN gateways have been allocated and configured so that the traffic with the source node 10.0.0.1 and destination node 20.0.0.1 is protected given the presence

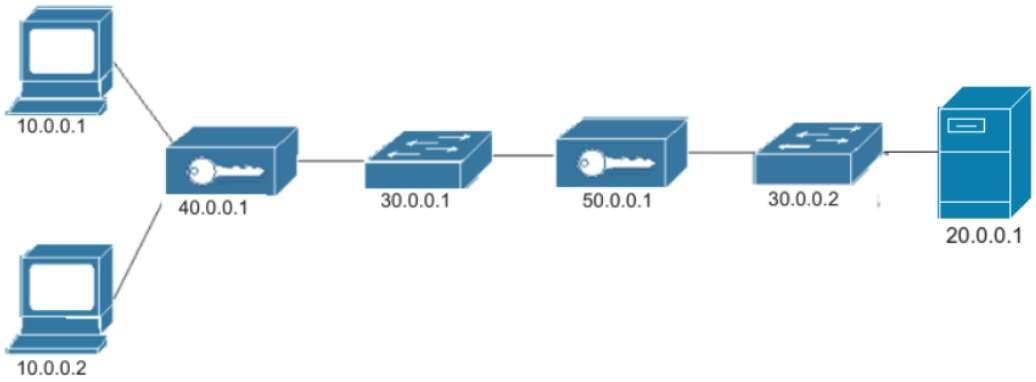

Figure 8.1. Network example1 for StrongSwan configuration

in the path of the untrusted node 30.0.0.1. With this network configuration, the translator present in the framework will output the following ipsec_conf.txt files:

Listing 8.1. ipsec_conf_40.0.0.1.txt

```
config setup
   charondebug="all"
   uniqueids=yes
   keyingtries=%forever
   ikelifetime=28800s
   lifetime=3600s
   dpddelay=30s
   dpdtimeout=120s
   dpdaction=restart

conn 40.0.0.1-to-50.0.0.1
   type=tunnel
   auto=start
   keyexchange=ikev2
   authby=secret
   left=40.0.0.1
   leftsubnet=10.0.0.1[%any,%any]
   right=50.0.0.1
   rightsubnet=20.0.0.1[%any,%any]
   ike=aes256-sha1-modp1024!
   esp=AES_256_CBC-SHA2_512!
   aggressive=no
```

Listing 8.2. ipsec_conf_50.0.0.1.txt

```
config setup
```

```
    charondebug="all"
    uniqueids=yes
    keyingtries=%forever
    ikelifetime=28800s
    lifetime=3600s
    dpddelay=30s
    dpdtimeout=120s
    dpdaction=restart

conn 50.0.0.1-to-40.0.0.1
    type=tunnel
    auto=start
    keyexchange=ikev2
    authby=secret
    left=50.0.0.1
    leftsubnet=10.0.0.1[%any,%any]
    right=40.0.0.1
    rightsubnet=20.0.0.1[%any,%any]
    ike=aes256-sha1-modp1024!
    esp=AES_256_CBC-SHA2_512!
    aggressive=no
```

Listings 8.1 and 8.1 show the contents of the two files ipsec_conf_40.0.0.1.txt and ipsec_conf_50.0.0.1.txt, which contain the StrongSwan language configuration of the network nodes 40.0.0.1 and 50.0.0.1 respectively. It can be seen that the configurations of the two nodes share the same setup configuration but differ in the connection configuration, as node 40.0.0.1 is the initiator of the tunnel and node 50.0.0.1 is its terminator.

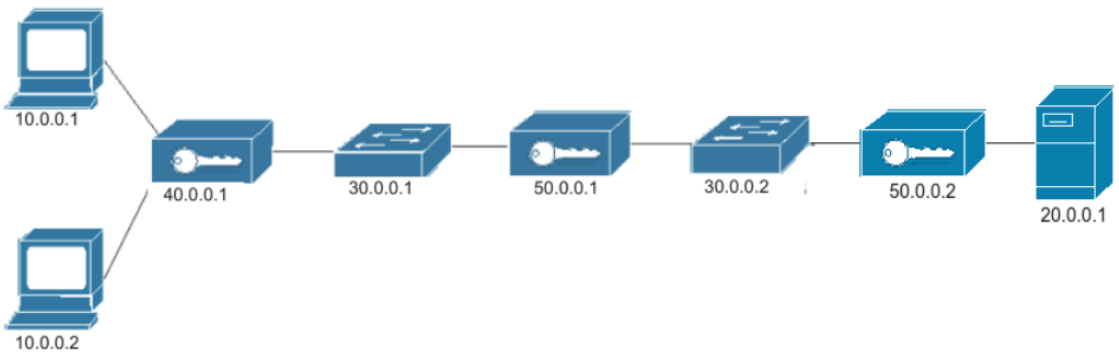

Figure 8.2. Network example2 for StrongSwan configuration

The network depicted in image 8.2 represents the output of the framework where

three VPN gateways have been allocated and configured so that the traffic with the source node 10.0.0.1 and destination node 20.0.0.1 is protected given the presence in the path of the untrusted node 30.0.0.1 and of the inspector node 30.0.0.2 and the traffic with source 10.0.0.2 and destination 20.0.0.1 is protected given the presence in the path of the untrusted node 30.0.0.2. With this network configuration, the translator present in the framework will output the following ipsec_conf.txt files:

Listing 8.3. ipsec_conf_40.0.0.1.txt

```
config setup
   charondebug="all"
   uniqueids=yes
   keyingtries=%forever
   ikelifetime=28800s
   lifetime=3600s
   dpddelay=30s
   dpdtimeout=120s
   dpdaction=restart

conn 40.0.0.1-to-50.0.0.1
   type=tunnel
   auto=start
   keyexchange=ikev2
   authby=secret
   left=40.0.0.1
   leftsubnet=10.0.0.1[%any,%any]
   right=50.0.0.1
   rightsubnet=20.0.0.1[%any,%any]
   ike=aes256-sha1-modp1024!
   esp=AES_256_CBC-SHA2_512!
   aggressive=no

conn 40.0.0.1-to-50.0.0.2
   type=tunnel
   auto=start
   keyexchange=ikev2
   authby=secret
   left=40.0.0.1
   leftsubnet=10.0.0.2[%any,%any]
   right=50.0.0.2
   rightsubnet=20.0.0.1[%any,%any]
   ike=aes256-sha1-modp1024!
   esp=AES_256_CBC-SHA2_512!
   aggressive=no
```

Listing 8.4. ipsec_conf_50.0.0.1.txt

```
config setup
   charondebug="all"
   uniqueids=yes
   keyingtries=%forever
   ikelifetime=28800s
   lifetime=3600s
   dpddelay=30s
   dpdtimeout=120s
   dpdaction=restart

conn 50.0.0.1-to-40.0.0.1
   type=tunnel
   auto=start
   keyexchange=ikev2
   authby=secret
   left=50.0.0.1
   leftsubnet=10.0.0.1[%any,%any]
   right=40.0.0.1
   rightsubnet=20.0.0.1[%any,%any]
   ike=aes256-sha1-modp1024!
   esp=AES_256_CBC-SHA2_512!
   aggressive=no
```

Listing 8.5. ipsec_conf_50.0.0.2.txt

```
config setup
   charondebug="all"
   uniqueids=yes
   keyingtries=%forever
   ikelifetime=28800s
   lifetime=3600s
   dpddelay=30s
   dpdtimeout=120s
   dpdaction=restart

conn 50.0.0.2-to-40.0.0.1
   type=tunnel
   auto=start
   keyexchange=ikev2
   authby=secret
   left=50.0.0.2
   leftsubnet=10.0.0.2[%any,%any]
   right=40.0.0.1
   rightsubnet=20.0.0.1[%any,%any]
```

```
    ike=aes256-sha1-modp1024!
    esp=AES_256_CBC-SHA2_512!
    aggressive=no
```

Listings 8.3, 8.4 and 8.5 show the contents of the three files, ipsec_conf_40.0.0.1.txt, ipsec_conf_50.0.0.1.txt and ipsec_conf_50.0.0.2.txt, which contain the StrongSwan language configurations of network nodes 40.0.0.1, 50.0.0.1 and 50.0.0.2. respectively. It can be seen that in the configuration file of node 40.0.0.1 there are two connections one towards node 50.0.0.1 and the other towards node 50.0.0.2, since it is the initiator node of the tunnels. In the other two nodes there is only one connection to node 40.0.0.1, since they are the terminating nodes of the tunnel.

Chapter 9

Validation and Testing

This chapter tests the real functionality of the framework and the computational performance as the number of Allocation Places and the number of Network Security Requirements vary. In the first part, the correctness of the implementation is tested by means of specific tests where given the inputs the outputs obtained are verified. These tests are described through three Use Case examples.

Finally, in the second part of the chapter, the computational performance of the framework is tested after the changes made in the implementation of this thesis work and the parameters that impact the most on the scalability of the system are identified. These tests are described and represented by graphs in which the total execution time of the framework is considered.

9.1 Validation

This section presents four use cases to show all the features explained and see how the ADP module actually works in the automatic and manual configuration of Firewall and VPN gateway.

9.1.1 Use Case1

Considering the Allocation Graph in Figure 9.1 and the Network Security Requirements in the Listin 9.1 as input to the framework, a possible solution provided by VEREFOO could be the one shown in Figure 9.2.

The solution in Figure 9.2 shows the allocation and configuration on the Graph of two VPN Gateways and a Firewall, the VPN Gateways for meeting the protection requirement, enforcing a site-to-site tunnel from 40.0.0.1 which is the ACCESS gateway or the CPS in charge of protecting the traffic and the 50.0.0.1 which is the EXIT Gateway or the CPS in charge of removing the protection on the packet and forwarding it to the destination node. The Firewall to block all traffic coming from 10.0.0.2 and destined for node 20.0.0.1.

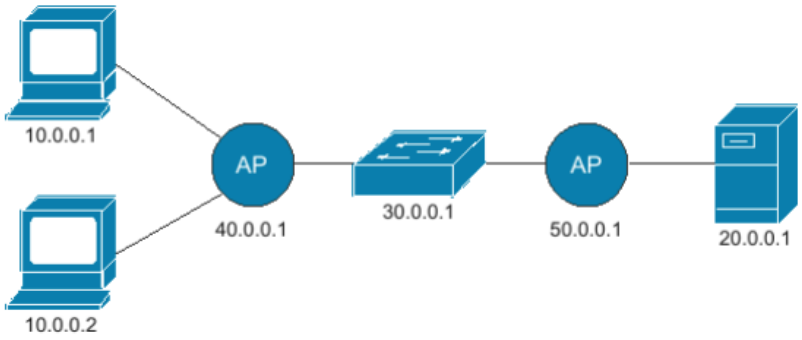

Figure 9.1. Input UseCase1

Listing 9.1. Input SecurityRequirements1

```
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1
-Isolation Property from 10.0.0.2 to 20.0.0.1
```

In the protection requirement no encryption and authentication algorithm has been specified, consequently the default ones have been adopted in the configurations. In the configuration of the Firewall, given the presence of a single Isolation constraint, the behavior in whitelist was chosen as the default, so as to discard all traffic and only allow communication between the node 10.0.0.1 and the node 20.0.0.1.

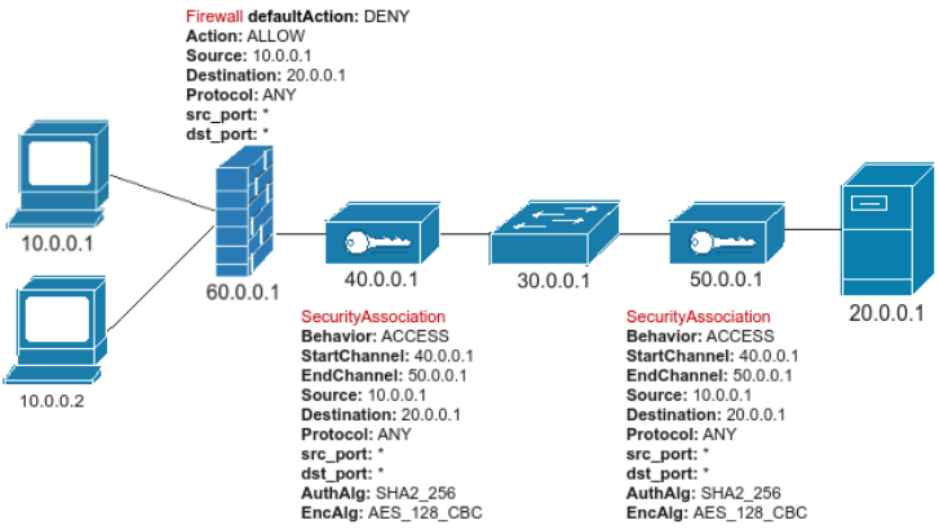

Figure 9.2. Output UseCase1

9.1.2 Use Case2

Considering as the framework input the Service Graph shown in Figure 9.3, in which two VPN Gateways are allocated and configured, an ACCESS Gateway in node 40.0.0.1 and an EXIT Gateway in node 50.0.0.1 and the Network Security Requirements in the Listing 9.2, one possible solution provided by VEREFOO could be the one shown in Figure 9.4.

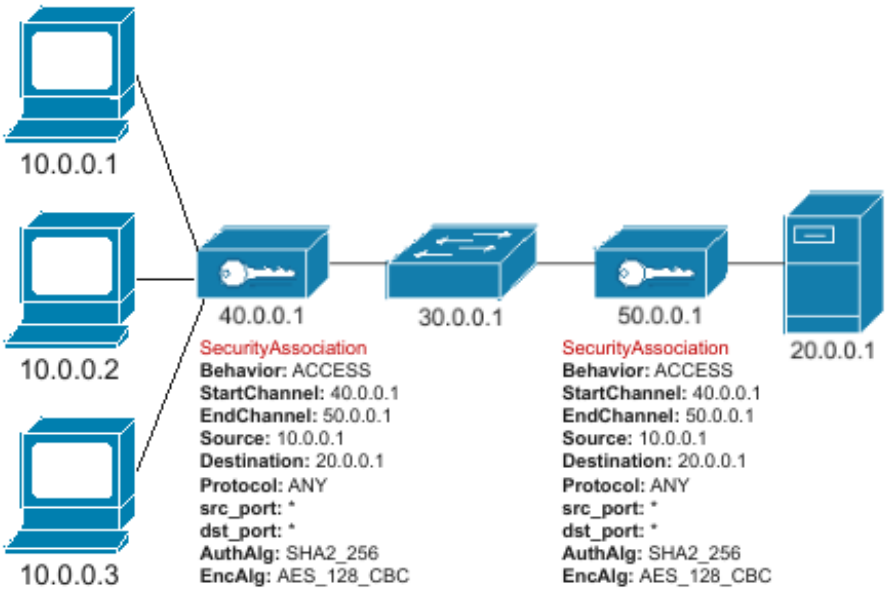

Figure 9.3. Input UseCase2

In the solution shown in Figure 9.4 the framework has verified the correctness of the allocation and manual configuration of the two VPN Gateways passed in input so as to satisfy the channel protection requirement from node 10.0.0.1 to node 20.0.0.1 and automatically allocated and configured a Firewall to block all traffic coming from node 10.0.0.2 and destined for node 20.0.0.1 and to allow all traffic coming from node 10.0.0.3 and destined for node 20.0.0.1.

Listing 9.2. Input SecurityRequirements2

```
-Protection Property from 10.0.0.1 to 20.0.0.1 with
   untrustedNode 30.0.0.1
-Isolation Property from 10.0.0.2 to 20.0.0.1
-Reachability Property from 10.0.0.3 to 20.0.0.1
```

In the configuration of the Firewall, given the presence of two Reachability constraints, one hard and the other soft derived from the traffic protection requirement

from node 10.0.0.1 to node 20.0.0.1, the behavior in blacklist was chosen as default, so to allow all traffic and discard only communication between node 10.0.0.2 and node 20.0.0.1.

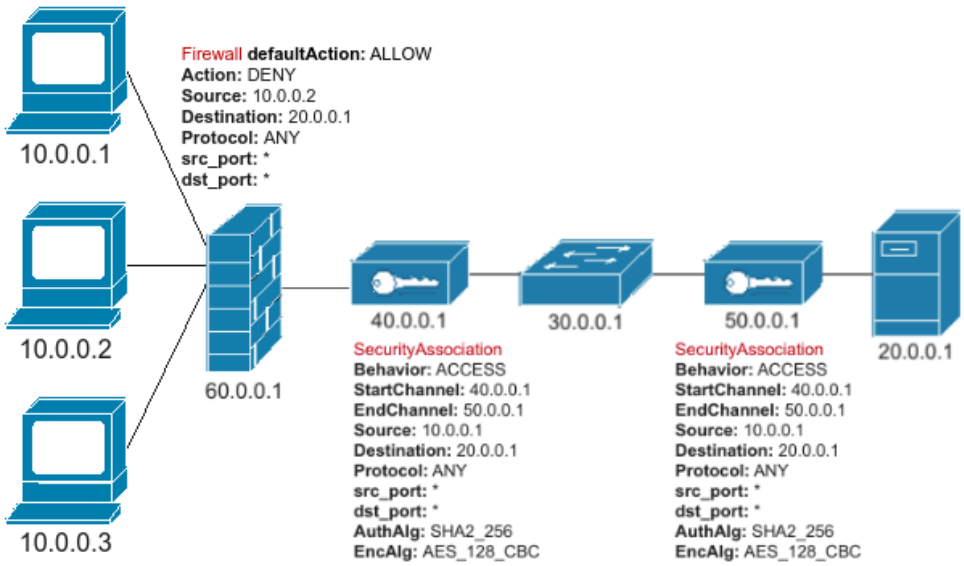

Figure 9.4. Output UseCase2

9.1.3 Use Case3

Considering as the framework input the Allocation Graph shown in Figure 9.5, in which there is a NAT that obscures the addresses of the source nodes 10.0.0.1 and 10.0.0.2 and a Firewall manually allocated in whitelist mode, and the Network Security Requirements in the Listing 9.1, a possible solution provided by VEREFOO could be the one shown in 9.6.

The solution in Figure 9.6 shows the allocation and configuration on the Graph of two VPN Gateways and a Firewall, the VPN Gateways for meeting the protection requirement, enforcing a site-to-site tunnel from 40.0.0.1 which is the ACCESS gateway or the CPS in charge of protecting the traffic and the 50.0.0.1 which is the EXIT Gateway or the CPS in charge of removing the protection on the packet and forwarding it to the destination node. In Gateway 50.0.0.1 it can be seen that the startChannel parameter is no longer the Gateway 40.0.0.1 but the NAT 70.0.0.1, the same goes for the value of the traffic source.

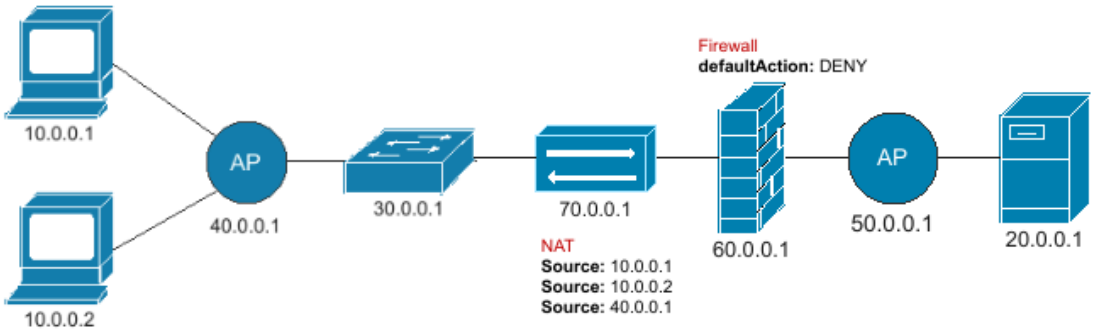

Figure 9.5. Input UseCase3

The Firewall initially allocated manually, was configured by the framework to allow only protected traffic to pass through. Since the Firewall has been allocated between the two VPN Gateways and follows the NAT, the source value is not the actual value of the traffic source but the NAT 70.0.0.1 and in turn the destination value is not the value of the real destination but the VPN Gateway 50.0.0.1.

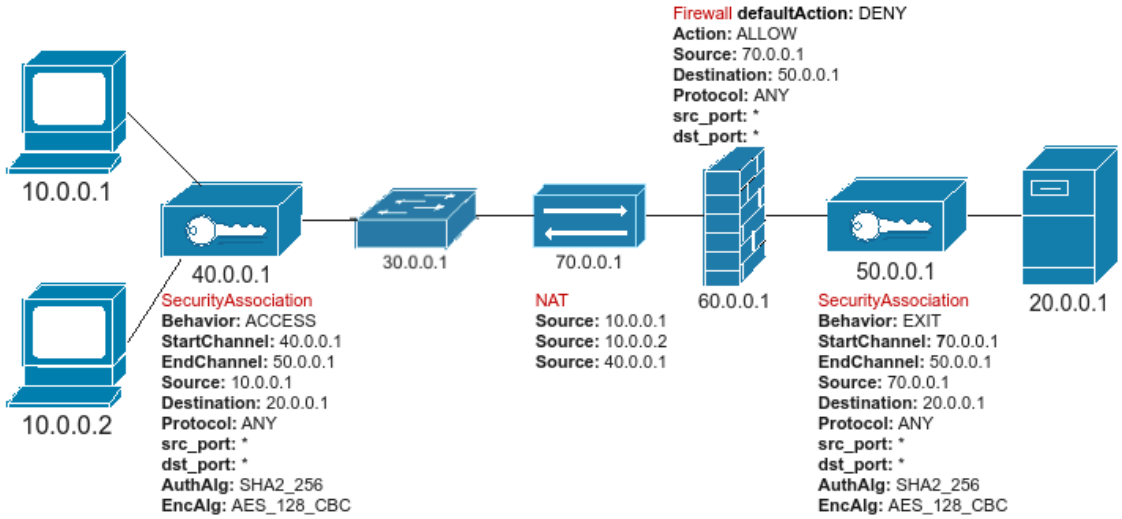

Figure 9.6. Output UseCase3

9.2 Testing

To test the first part of the work of this thesis and the scalability of the framework, performance tests were created and run. These tests were performed to verify the performance and scalability of the framework after the insertion of the new constraints for the allocation of VPN Gateways, in order to verify the improvements and highlight the factors that most impact the scalability of the system. The tests were performed on a 4-core Intel i7-6700 3.40 GHz workstation with 32 GB of RAM and for each input value, each test was repeated 50 times. The two input values on which the tests were carried out are:

- **APs:** number of Allocation Places present in the Allocation Graph;
- **NSRs:** number of Network Security Requirements passed in input to the framework.

Starting from the two input values, the approach followed for the tests was to test the behavior of the system as one of the two metrics increases, setting the other to a fixed value and finally as both metrics vary.

To display the results graphically, two types of graphs were created, one that shows the trend of the data function in input the two metrics and returning in output the average of the execution time of the 50 repetitions, instead in the second type of graph, whisker plot, given the 50 output values, the following values are represented:

- *the first and third quartiles:* values represented by each box;
- *the median:* value represented by the segment inside the box;
- *the average:* value represented by the segment inside the box;
- *maximum value and minimum value:* representatives of the outermost segments and are called whiskers;
- *exceptional value:* represented by the external point;

The results of the first two types of tests are shown in the following four graphs, where the graphs in Figure 9.7 and Figure 9.8, represent the trend of the solution as the number of Allocation Places increases and keeping the number of Network Security Requirements fixed and the graphs in Figure 9.9 and Figure 9.10, represent the trend of the solution as the number of Network Security Requirements increases and keeping the number of Allocation Places fixed.

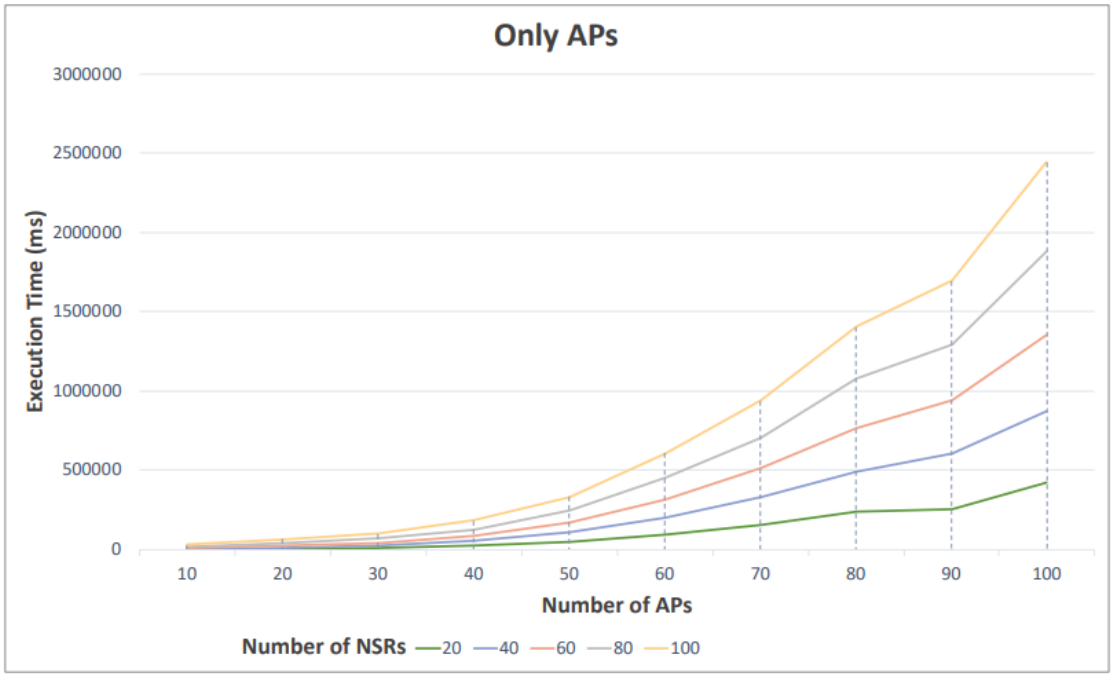

Figure 9.7. APs impact

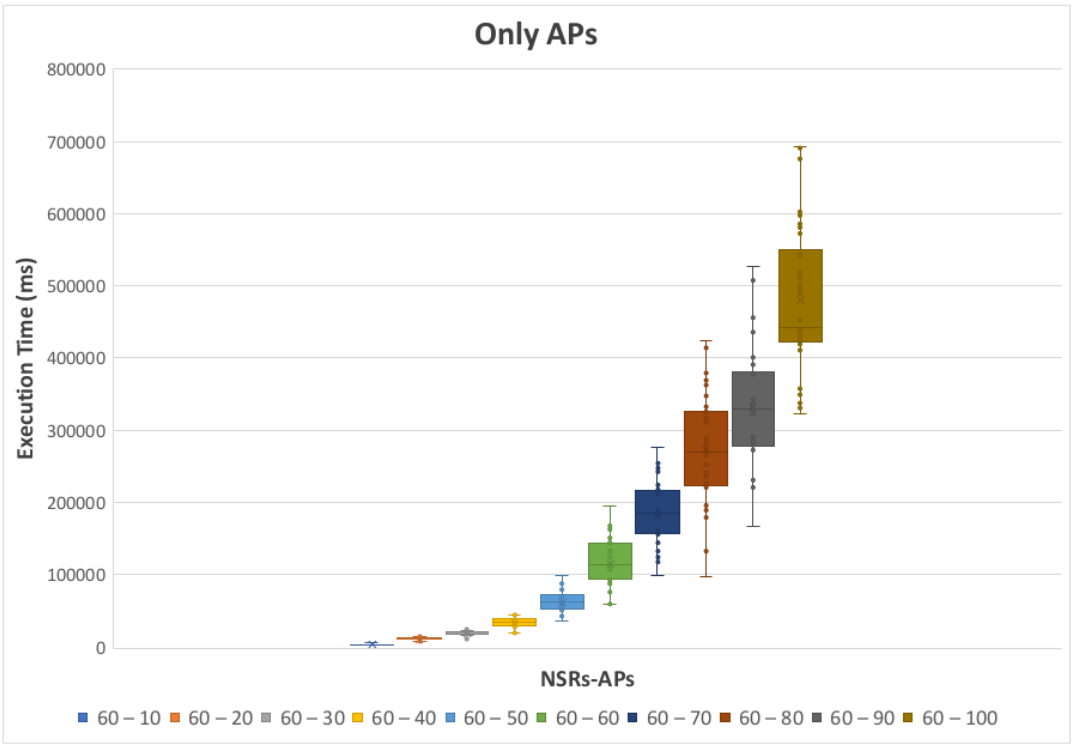

Figure 9.8. APs impact whisker plots

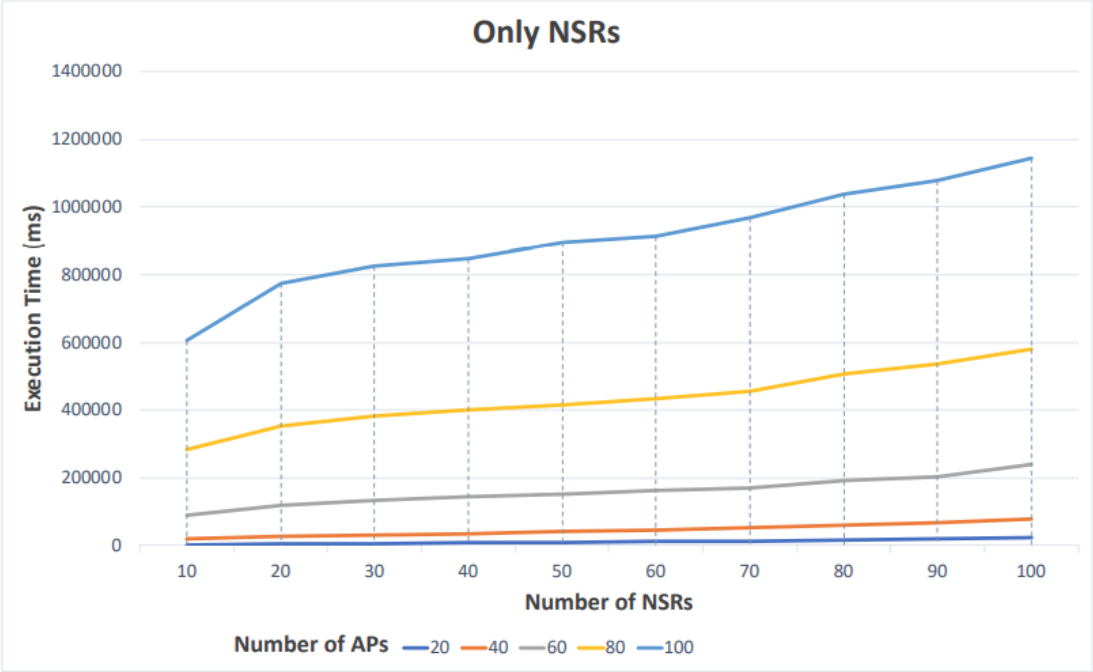

Figure 9.9. NSRs impact

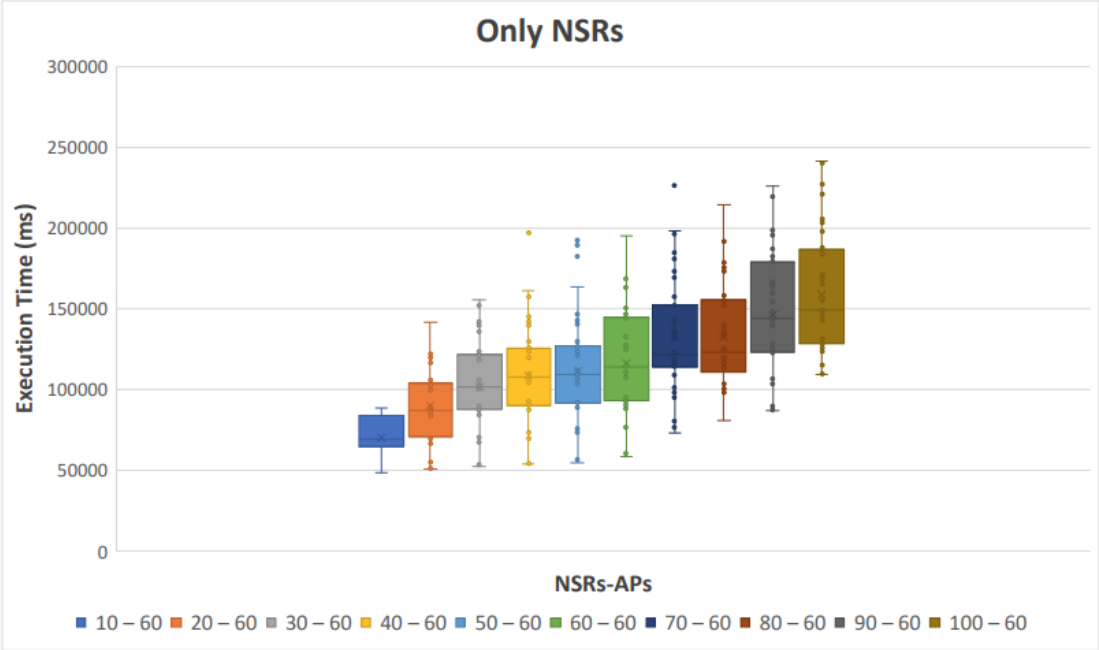

Figure 9.10. NSRs impact whisker plots

From the four graphs it can be immediately seen that the increase in execution time does not grow exponentially, this is already a fundamental result considering the computational cost of the MaxSMT problem. Another fundamental result that can be deduced from the graphs is that at the level of computational scalability, the number of APs impacts much more than the number of NSRs. This is due to the excessive number of IP addresses and the particular handling of integers by z3. This can also be seen from the heterogeneity of the boxes in Figure 9.8 compared to a more accentuated homogeneity between the boxes in Figure 9.10, where the greater number of IP addresses of the first solution generates a greater indeterminacy on the final execution time.

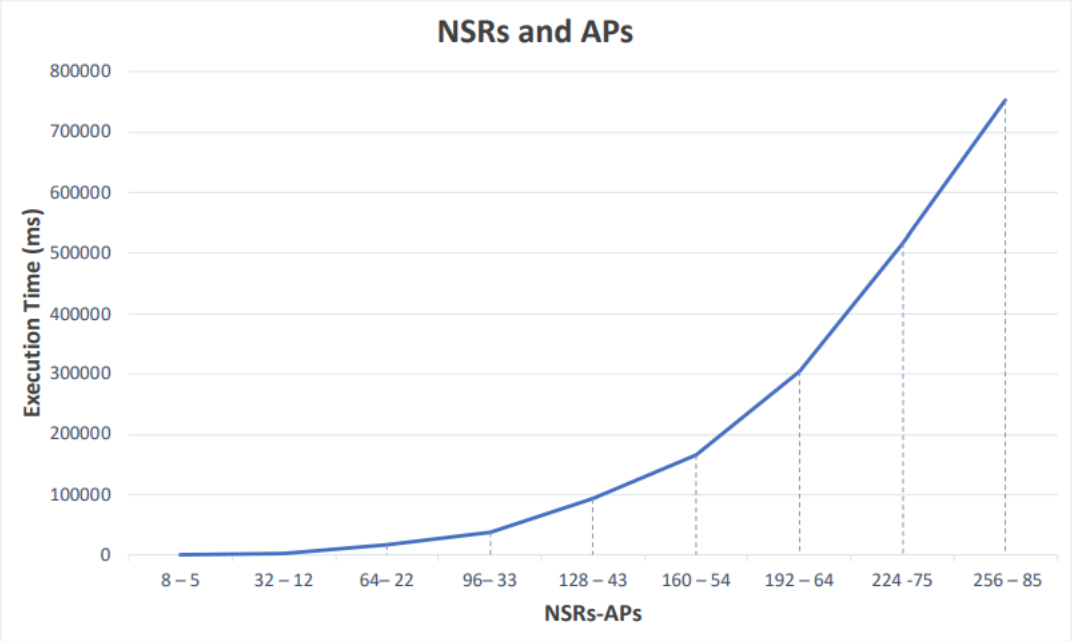

Figure 9.11. Scalability test graphic

In the last part of the tests, the computational performance was tested as both the number of Allocation Places and the number of Network Security Requirements increased, so as to test a possible real configuration of the framework. Given the greater impact on the scalability of the number of APs compared to the number of NSRs, it was chosen to take a progressive number of metrics (APs and NSRs) for each test, so that it can be verified that the flow of each NSR passes through at least 4 APs.

As can be seen, in the graphs depicted in Figure 9.11 and Figure9.12, also in this case the execution time does not grow exponentially so that even with high enough input values it is possible to have reasonable execution times. For example the average execution time to satisfy 192 NSRs given an Allocation Graph consisting

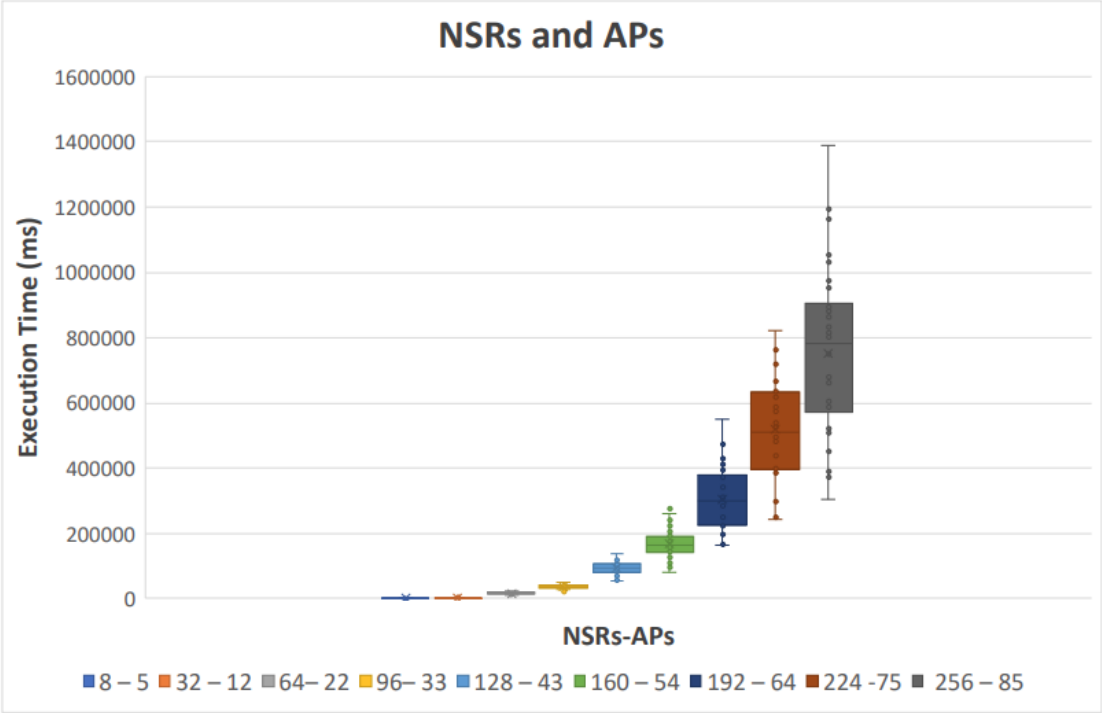

Figure 9.12. Scalability test graphic whisker plots

of 64 APs is 300000 ms (5 minutes) and the maximum time value does not exceed 600000 ms (10 minutes).

Chapter 10

Conclusions

In this thesis work a new version of the VEREFOO ADP module was created, more efficient than the previous one and able to allocate and configure two network security functions at the same time such as Firewall and Channels Protection Systems. Thanks to this improvement, the network designer, in the same iteration of the framework, can automatically allocate and configure, in the input Service Graph, two different conflict-free network security functions.

First of all, some constraints of the Communication Protection Network Security have been reformulated in order to improve the performance and scalability of the framework. In fact these constraints, formulated in the previous version of the framework with integer variables, have been rewritten with Boolean values in order to considerably simplify the problem passed to solver z3 and significantly lower the total execution time.

Subsequently, after various changes and improvements on the previous version of the framework, we worked to make it able to simultaneously perform an allocation and an automatic configuration of Firewalls and Channels Protection System, after having received a series of requirements to be respected, a Service Graph or an Allocation Graph, following optimality criteria through the resolution of a MaxSMT problem. This was done by running the Channels Protection System allocation / configuration tool first, so that the output of this solution is the input of the Firewalls allocation / configuration tool, so that an incorrect allocation of a Firewall does not compromises the satisfiability of the final solution.
By adding the ReachabilityDerived Property, the reachability of packets belonging to a secure communication has been guaranteed, when a Packet Filter configured in white-list mode is allocated in the path between the source and the destination, so as to allow these packets to reach the final destination.
All possible conflicts between the two network security functions have also been resolved, especially when a Firewall is allocated between two VPN Gateways and therefore the source and destination address of the packet is no longer the original one but becomes that of the two VPN Gateways creators of the tunnel.
The allocation and configuration functionality of the two network security functions was then enabled even in the presence of network functions that modify the addresses of the traffic that passes through them, an example is given by the NAT function.

Finally, through the writing of a translator, the ability to create, through the configuration of the VPN Gateways present in the VEREFOO output, a configuration file supported by the StrongSwan platform has been added to the framework, so that they can be created and configured automatically Channels Ipsec on the network.

At the end, performance tests were performed to verify the correctness and scalability of the framework, after the insertion of the new constraints for the assignment of VPN Gateways, in order to verify the improvements and highlight the factors that have the greatest impact on scalability of the system. These tests showed a better scalability of the total execution time compared to the previous version of the framework and that the number of Allocation Places has a greater impact on the scalability of the system than the number of Network Security Requirements. It has also been verified that the framework is able to find an optimal solution in a fairly reasonable time even in the presence of a large number of Allocation Places and a large set of Network Security Requirements.

Possible future innovations on the VEREFOO ADP module could be, on the one hand, the expansion of the set of network security functions such as the anti-spam filter, the Intrusion Detection System (IDS) and the Web Application Firewall (WAF), from the another is the definition of other types of Network Security Requirements useful for adding more functionality to the framework.

Bibliography

[1] J. M. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," *RFC*, vol. 7665, pp. 1–32, 2015. [Online]. Available: https://doi.org/10.17487/RFC7665

[2] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Towards a fully automated and optimized network security functions orchestration," in *2019 4th International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, October 10-12, 2019*. IEEE, 2019, pp. 1–7. [Online]. Available: https://doi.org/10.1109/CCCS.2019.8888130

[3] ——, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/NOMS47738.2020.9110402

[4] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2601–2614, 2017. [Online]. Available: https://doi.org/10.1109/TNET.2017.2708096

[5] Z. Fu and S. F. Wu, "Automatic generation of ipsec/vpn security policies in an intra-domain environment," in *Operations & Management, 12th International Workshop on Distributed Systems, DSOM 2001, Nancy, France, October 15-17, 2001. Proceedings*, O. Festor and A. Pras, Eds. INRIA, Rocquencourt, France, 2001, pp. 279–290. [Online]. Available: http://www.simpleweb.org/ifip/Conferences/DSOM/2001/DSOM2001/proceedings/S8-3.pdf

[6] Y. Yang, C. U. Martel, and S. F. Wu, "On building the minimum number of tunnels: an ordered-split approach to manage ipsec/vpn policies," in *Managing Next Generation Convergence Networks and Services, IEEE/IFIP Network Operations and Management Symposium, NOMS 2004, Seoul, Korea, 19-23 April 2004, Proceedings*. IEEE, 2004, pp. 277–290. [Online]. Available: https://doi.org/10.1109/NOMS.2004.1317665

[7] C. Chang, Y. Chiu, and C. Lei, "Automatic generation of conflict-free ipsec policies," in *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*, ser. Lecture Notes in Computer Science, F. Wang, Ed., vol. 3731. Springer, 2005, pp. 233–246. [Online]. Available: https://doi.org/10.1007/11562436_18

[8] M. Rossberg, G. Schaefer, and T. Strufe, "Distributed automatic configuration of complex ipsec-infrastructures," *J. Netw. Syst. Manag.*, vol. 18, no. 3, pp. 300–326, 2010. [Online]. Available: https://doi.org/10.

1007/s10922-010-9168-7

[9] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *IEEE Commun. Mag.*, vol. 44, no. 3, pp. 134–141, 2006. [Online]. Available: https://doi.org/10.1109/MCOM.2006.1607877

[10] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," 2020/11/13. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3411505.3418439

[11] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *Risks and Security of Internet and Systems - 9th International Conference, CRiSIS 2014, Trento, Italy, August 27-29, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 8924. Springer, 2014, pp. 148–163. [Online]. Available: https://doi.org/10.1007/978-3-319-17127-2_10

[12] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, "Improving the formal verification of reachability policies in virtualized networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, pp. 713–728, 2021. [Online]. Available: https://doi.org/10.1109/TNSM.2020.3045781

[13] F. Valenza, S. Spinoso, and R. Sisto, "Formally specifying and checking policies and anomalies in service function chaining," *J. Netw. Comput. Appl.*, vol. 146, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.102419

[14] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2929325

[15] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Introducing programmability and automation in the synthesis of virtual firewall rules," in *6th IEEE Conference on Network Softwarization, NetSoft 2020, Ghent, Belgium, June 29 - July 3, 2020*, F. D. Turck, P. Chemouil, T. Wauters, M. F. Zhani, W. Cerroni, R. Pasquini, and Z. Zhu, Eds. IEEE, 2020, pp. 473–478. [Online]. Available: https://doi.org/10.1109/NetSoft48620.2020.9165434

[16] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "A novel approach for security function graph configuration and deployment," in *7th IEEE International Conference on Network Softwarization, NetSoft 2021, Tokyo, Japan, June 28 - July 2, 2021*, K. Shiomoto, Y. Kim, C. E. Rothenberg, B. Martini, E. Oki, B. Choi, N. Kamiyama, and S. Secci, Eds. IEEE, 2021, pp. 457–463. [Online]. Available: https://doi.org/10.1109/NetSoft51509.2021.9492654