Politecnico di Torino

Master of Science in Computer Engineering

Master Degree Thesis

Analysis and characterization of the VPN configuration problem

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

Candidate
Carlo Formica

Academic Year 2022-2023

Summary

The automated and optimized configuration of secure communications inside networks is an indispensable necessity. The transmitted data must be protected to guarantee confidentiality, integrity, and availability. From this point of view, VPNs allow the configuration of secure channels between hosts, even over untrusted networks such as the Internet. However, the introduction of virtual systems based on virtualization significantly changes networks, which have become more dynamic and flexible, but also more complex to configure. Therefore, operations require specialized skills and knowledge of the network and the functions to be implemented, which cannot be managed entirely by humans. One possible solution is to make such operations automatic through a policy-based model.

This thesis contributes to the development of one of these automated approaches, called VEREFOO (VErified REFinement and Optimized Orchestration). From user-specified security requirements, this framework has the ability to automatically generate an optimal network configuration. Full automation, formal correctness, and optimization are the main principles that need to be fulfilled. Full automation is achieved because the final network configuration, except for input specification, can be obtained without human intervention. Formal correctness is ensured by solving the MaxSMT problem, which generates a correct solution through hard constraints without requiring a-posteriori verification. Optimization can be reached by solving soft constraints. This thesis is based on this approach to evaluate the performance of the framework for configuring secure communications using VPNs.

In particular, a generator is proposed that can create a complex and branched network structure, in which different types of nodes can be inserted. This generator allows the evaluation of the framework's performance in the optimal allocation and configuration of a minimum number of VPN gateways to enable secure communications and traffic protection. Tests are conducted by increasing the size and considering different scenarios to assess the impact of different parameters. The results demonstrate the framework's ability to correctly configure and allocate security functions, even in the presence of a large number of input security requirements. Performance is evaluated according to the execution time taken to find the optimal and correct solution. To achieve the results, two different models, Atomic Flows and Maximal Flows, are used to evaluate which one offered greater advantages when subject to the same conditions.

Acknowledgements

I would like to thank my supervisors for their support throughout the entire process of my thesis. Furthermore, I would like to dedicate special thanks to my family, who represent the most important part of my life. Without them, I would not be here today and would not have become the person I am now. I would like to express my deepest gratitude to my mother, who supported me unconditionally, always encouraging me to believe in myself and never give up. Thanks also to my father and sister, who have always done everything possible to ensure my happiness and have been a constant source of inspiration.

Contents

List of Figures 7

List of Tables 9

Listings 10

1 Introduction 12

1.1 Thesis introduction . . . 12

1.2 Thesis description . . . 13

2 Virtualized Networks 15

2.1 Virtualization Paradigms . . . 15

2.1.1 Network Functions Virtualization (NFV) . . . 15

2.1.2 Software-Defined Networking (SDN) . . . 17

2.1.3 Service Function Chaining (SFC) . . . 19

2.2 Virtual Network Functions (VNF) . . . 21

2.2.1 Virtual Network Security Functions (VNSF) . . . 23

2.2.2 Limitations of Manual Configuration . . . 25

3 The VEREFOO Approach 28

3.1 Inputs and Outputs of the framework . . . 29

3.1.1 Input: Service Graph and Allocation Graph . . . 29

3.1.2 Input: Network Security Requirements . . . 30

3.1.3 Expected outcome . . . 30

3.2 MaxSMT problem formulation . . . 30

3.3 Traffic Flow Models . . . 31

3.3.1 Atomic Flows . . . 32

3.3.2 Maximal Flows . . . 33

3.4 VPN Implementation in VEREFOO . . . 33

3.4.1 Communication Protection Model . . . 35

4 Thesis Objective 39

5 Proposed Network Generator 41

5.1 Network and security requirements model . . . 41

5.1.1 Network Node Types . . . 41

5.1.2 Security Requirements Model . . . 49

5.2 Network Generator . . . 53

5.2.1 Input parameters . . . 53

5.2.2 Network structure . . . 53

5.2.3 Network structure with inspector node . . . 56

5.2.4 Generation of Security Requirements . . . 57

5.2.5 Illustrative Examples . . . 58

5.3 Integration with VEREFOO . . . 64

6 Implementation and Testing 65

6.1 Test Description . . . 65

6.2 Test 1: Basic Structure . . . 67

6.2.1 Atomic Flow Results . . . 67

6.2.2 Maximal Flow Results . . . 68

6.2.3 Interpretation of Test Results . . . 69

6.3 Test 2: Structure with Inspector Node . . . 72

6.3.1 Atomic Flow Results . . . 73

6.3.2 Maximal Flow Results . . . 74

6.3.3 Interpretation of Test Results . . . 75

6.4 Test 3: APs vs Policies Impact . . . 77

6.4.1 Fixed APs Results . . . 77

6.4.2 Fixed Policies Results . . . 79

6.4.3 Interpretation of Test Results . . . 80

6.5 Improving Framework Performance: Analysis Work . . . 81

6.5.1 Reducing Solver Constraints . . . 82

6.5.2 Redundancy Elimination for Improved Performance . . . 84

7 Conclusions 86

Bibliography 88

A Setting Up the Generator for Testing 92

List of Figures

2.1 Differences between traditional network appliances and the NFV-based approach . . . 16
2.2 Simplified view of an SDN architecture . . . 18
2.3 SFC example . . . 20
2.4 VNF internal achitecture . . . 22
2.5 Taxonomy of Security VNFs. [1] . . . 25

3.1 The VEREFOO Approach . . . 29
3.2 Network topology with untrusted node . . . 37
3.3 Solution . . . 37

5.1 Node object . . . 42
5.2 Allocation of VPN Gateways at endpoints . . . 43
5.3 Network topology . . . 44
5.4 Solution 1 . . . 45
5.5 Other possible configurations . . . 45
5.6 Configuration with inspector node IN . . . 46
5.7 Complex configuration with two inspector nodes (IN1, IN2) . . . 47
5.8 Solution 1 . . . 48
5.9 Solution 2 . . . 49
5.10 "Basic structure - Client" complete and not complete . . . 54
5.11 "Basic structure - Server" complete and not complete . . . 55
5.12 Basic structures connected to "Central AP" . . . 55
5.13 Generated structure with inspector node IN . . . 56
5.14 Use case 1 . . . 59
5.15 Use case 2 . . . 61
5.16 Network topology generated . . . 62

6.1 Test 1: Table of Atomic Flows Results . . . 68

6.2 Test 1: Graph of Atomic Flows Results 68
6.3 Test 1: Table of Maximal Flows Results 69
6.4 Test 1: Graph of Maximal Flows Results 69
6.5 Time trends in the Atomic Flows model 70
6.6 Test 1: Comparison of the Atomic Flows and Maximal Flows model 71
6.7 Test 1: Generation of constraints in the Atomic Flows and Maximal Flows model 72
6.8 Test 2: Table of Atomic Flows Results 73
6.9 Test 2: Graph of Atomic Flows Results 73
6.10 Test 2: Table of Maximal Flows Results 74
6.11 Test 2: Graph of Maximal Flows Results 74
6.12 Impact of the inspector node in the Atomic Flows model 75
6.13 Impact of the inspector node in the Maximal Flows model 76
6.14 Test 2: Generation of constraints in the Atomic Flows model 76
6.15 Test 2: Generation of constraints in the Maximal Flows model . . . 77
6.16 Test 3: Table of Fixed APs Results 78
6.17 Test 3: Graph of Fixed APs Results 78
6.18 Test 3: Table of Fixed APs Results 79
6.19 Test 3: Graph of Fixed APs Results 79
6.20 Constraints Generated with Fixed APs 80
6.21 Constraints Generated with Fixed Policies 81
6.22 Table with constraint reduction 83
6.23 Graph with constraint reduction 83
6.24 Table with redundancy elimination 84
6.25 Graph with redundancy elimination 85

List of Tables

Listings

3.1 CPP XML representation 37
3.2 VPN GATEWAY 1 XML configuration 38
3.3 VPN GATEWAY 2 XML configuration 38
5.1 Client node creation in VEREFOO 43
5.2 Server node creation in VEREFOO 43
5.3 Added VPN capability in VEREFOO 44
5.4 Untrusted node creation in VEREFOO 46
5.5 Inspector node creation in VEREFOO 47
5.6 Allocation Place creation in VEREFOO 49
5.7 Network Protection Policies node definition in VEREFOO 52
6.1 Reducing Solver Constraints 82
A.1 Configuration of parameters for running tests 92

Chapter 1

Introduction

1.1 Thesis introduction

One of the major challenges in networks is to implement systems capable of configuring secure communications between hosts, even on unreliable networks such as the Internet. This means that any data transmitted needs to preserve the properties of confidentiality, availability, and integrity. To create secure channels within untrusted networks, one available technology is the Virtual Private Network (VPN), which applies security mechanisms such as encryption, MAC computation, and digital signature to make data unreadable to third parties and verify the authenticity and integrity of received traffic.

Nowadays, for secure communications, the evolution of the modern network due to virtualization should be considered. New paradigms such as Network Functions Virtualization (NFV), Software-Defined Networking (SDN), and Service Function Chaining (SFC) improve network implementation and management, moving away from traditional system approaches. In particular, NFV transforms network functions into software applications that can now be allocated and executed on generic hardware. SDN separates the network control logic from the underlying devices, enabling centralized and programmable network management of the network itself. With SFC, virtual functions can be combined into a service chain to provide a customizable end-to-end system.

However, while networks are more dynamic and flexible thanks to virtualization, their complexity and size increase. This makes their configuration and security functions a non-trivial problem, especially when performed by human beings. For the protection of communication, specific virtual security functions need to be configured. This configuration requires specific skills and is not a trivial task, especially if done manually. The administrator in charge must have a high level of knowledge about the network and the functions to be implemented, in order to avoid anomalies or malfunctions. According to the Verizon report, in just the first months of 2023, 74% of breaches are due to the human element, including social engineering attacks, errors, or improper use.

In light of these considerations, it becomes essential to implement tools capable of automating operations within a network. Several studies have been conducted

with the aim of implementing automated models to reduce human intervention in the configuration of security functions within virtualized networks. These models introduce benefits in the implementation of security systems and are used in various sectors, including SDN switches [2], and Smart Homes [3].

One of the frameworks that address these needs is VEREFOO (VErified REFinement and Optimized Orchestration), which automatically allocates and configures network functions with policy-based management. In this case, the user has only the task to define security requirements to be applied to the network. VEREFOO, with an input that includes the logical topology of the network and the security requirements defined by the user, generates an optimized and correct final solution using an approach based on a *partial weighted Maximum Satisfiability Modulo Theories* (MaxSMT) resolution.

VEREFOO has already achieved excellent success in the literature. The works [4][5][6] provide a general overview of the framework for the proper implementation of policy-based security within virtual networks. Other studies analyze the model with specific security functions, including packet filter [7][8][9], demonstrating its effectiveness and paving the way for potential improvements in intermediate stages [10] and policy conflicts [11][12].

Based on these studies, the aim of this thesis work is to evaluate VEREFOO to enable secure communication between two nodes within the network, focusing specifically on the use of Virtual Private Networks (VPNs). Some work has already been developed in this direction [13], but in this case, the behavior and effectiveness of the framework are evaluated concerning the variation of specific parameters present in the structure.

1.2 Thesis description

The rest of the thesis is divided into the following chapters:

- **Chapter 2**: it provides an overview of the context in which the work is situated. It describes the fundamental paradigms that emerge in virtualized networks, such as NFV, SDN, and SFC, focusing in particular on virtual network functions and their implementation in security. It also examines the benefits of automating network functions over their manual configuration.
- **Chapter 3**: it focuses on the general functioning of the VEREFOO framework. It analyzes the framework's inputs and outputs, MaxSMT resolution, and the process of VPN allocation and configuration. In addition, the two models used for testing, Atomic Flows and Maximal Flows, are discussed.
- **Chapter 4**: it explains the objective of the thesis.
- **Chapter 5**: it provides a detailed description of the proposed network generator in the thesis, which has been implemented to create a complex and realistic network. This chapter describes the different types of nodes that can compose a network and how they are integrated within VEREFOO. The logic

used to connect these nodes is also outlined, along with a description of how the generator is integrated with the framework.

- **Chapter 6**: it presents the tests conducted on the framework and the corresponding results achieved. For each test, the results obtained with the two models, Atomic Flows and Maximal Flows, are shown in order to compare which one is more advantageous given the same conditions.
- **Chapter 7**: conclusions.
- **Appendix A**: it shows how the generator is configured to conduct the tests.

Chapter 2

Virtualized Networks

This chapter provides a general overview of the concept of virtualization, which is an important solution to the problems of complexity and flexibility of traditional networks. This technology uses software implementations to create virtual networks that simulate the functionality of a physical network, simplifying management and configuration and improving overall flexibility and efficiency. This means that each network function is implemented as a software application instead of using dedicated hardware devices. This concept has given origin to new networking paradigms such as NFV (Network Functions Virtualisation), SDN (Software-Defined Networking), and SFC (Service Function Chaining), which will be presented in the following section. For each paradigm, the general concepts with their advantages and limitations will be explained.

The second part of the chapter will analyze the structure of these software applications, called Virtual Network Functions (VNF). Special emphasis will be on the virtual functions responsible for the security of systems, the Virtual Network Security Functions (VNSF). Finally, the importance of configuring these network security functions automatically, rather than manually, will be explained. This helps to ensure system stability and avoid possible human errors due to the complexity of networks.

2.1 Virtualization Paradigms

2.1.1 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV)[14] [15] introduces an important change in network implementation by separating network functions from dedicated hardware and turning them into software applications, called Virtual Network Functions (VNFs). In this way, a particular service can be broken down into a collection of VNFs, which can be implemented using software running on one or multiple industry-standard physical servers. This allows to perform multiple network functions on a single physical device, without requiring the purchase and the installation of dedicated hardware.

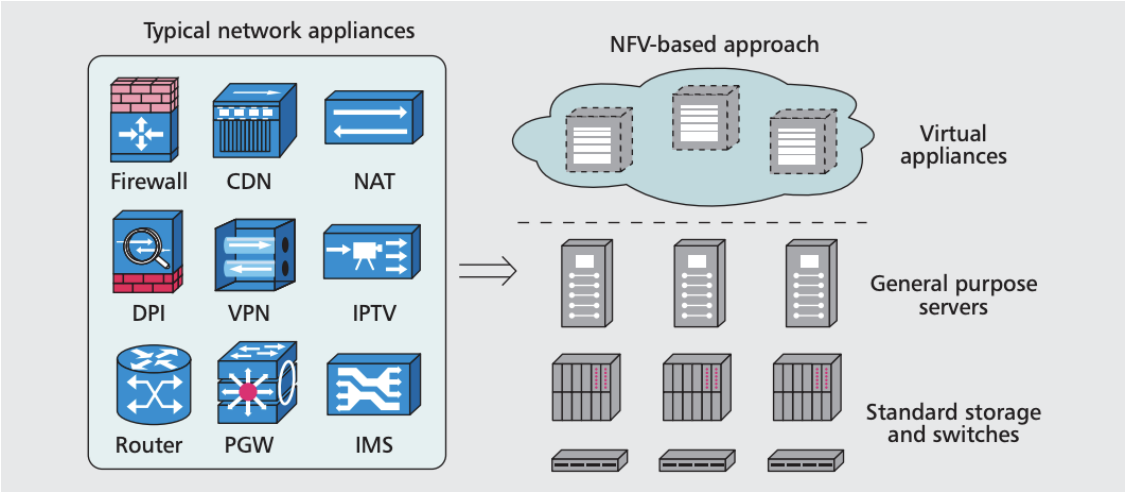

Figure 2.1. Differences between traditional network appliances and the NFV-based approach

NFV uses a software abstraction layer, called a hypervisor, to create multiple virtual machines (VMs) on a single physical device[16]. These VMs host the different NFVs, which perform the same network functions as traditional hardware. For example, routing, load balancing and security functions can be implemented and executed on a single virtual machine without the need of additional hardware devices.

The ETSI standards have defined a reference architecture for NFV composed of three main components. The first is the Virtualized Network Function (VNF), which represents the actual virtual network function. The second is the Network Function Virtualization Infrastructure (NFVI)[17], where the specific VNFs are allocated, executed and managed. This architecture includes all the elements required to support the virtual environment. Finally, there is the NFV Management and Orchestration (NFV-MANO)[18], responsible for the administration of infrastructure resources and virtualized functions.

The changes described above introduces many benefits, including:

1. Cost-effectiveness: virtual network functions run on a standard server controlled by a hypervisor, which is more cost-effective than purchasing dedicated and proprietary devices. In addition, using a single device for multiple network functions further reduces hardware costs and lowers operating and maintenance expenses.

2. Scalability: as needed, the number of VNFs can dynamically adapt to meet requirements for CPU, memory, storage, and network. This maximizes available capacity and reduces energy consumption.

3. Reliability: in case of a hardware failure or network topology changes, VNFs can be easily migrated from one server to another, maintaining constant service availability. This ability to easily move a network function to another

server increases flexibility as the provider can adapt the VNF to changing demands, accelerating the deployment of services and applications.

4. On-demand service creation: thanks to virtualization, virtual devices can be created only when they are needed, saving resources and reducing costs in comparison to a physical device that would remain on, even if not in use.

5. Remote device update operations: if an update or replacement of a physical system is needed, operations can be performed remotely in virtual environments without reaching the device itself where it is installed.

On one side, virtualization introduces several advantages that improve network principles, while on the other side the potential risks of this technology have to be evaluated. The main considerations concern security within virtual infrastructures. Indeed, virtualization increases the complexity of the network and this can make easier for attackers to discover vulnerabilities and new types of attacks. Moreover, within a virtual function, it is more difficult to contain malware between virtual components running on a virtual machine than between hardware devices which may be isolated or physically separated. The hypervisor installed in the virtual network is another point of weakness since an attacker could take control and have full access to the system, running malicious programs without being detected by the virtual machines. This type of attack is known as *hyperjacking* [19].

2.1.2 Software-Defined Networking (SDN)

Software-Defined Networking (SDN)[20] is a paradigm introduced to solve problems related to network configuration and management in traditional environments and provides a software-based network architecture, which is fully programmable. SDN breaks vertical integration by moving the network device control plane, which handles traffic forwarding, from hardware to software, while the data plane, which forwards traffic, remains in the dedicated hardware. With this separation of control and data planes, network switches assume the role of simple forwarding entities, while the centralized controller takes charge of executing the control logic, allowing network administrators to manage and schedule the entire network from a centralized user interface without requiring additional hardware. This means that decisions on data routing, network configuration, and security management are taken by a centralized software control component that has a global view of the structure.

SDN consists of three main components that may be located in different physical locations or proximity (Figure 2.2).

- SDN controller (control layer): it is a central component that manages and controls the entire network by receiving information from the switches and sending them instructions on how to forward traffic.

- Networking devices (infrastructure layer): these are the physical devices that forward traffic. They communicate with the SDN controller to get information regarding how to route the packets.

- Applications (application layer): they are programs that use the SDN controller to manage the network. They can be used to implement features such as security management, load balancing, and dynamic routing.

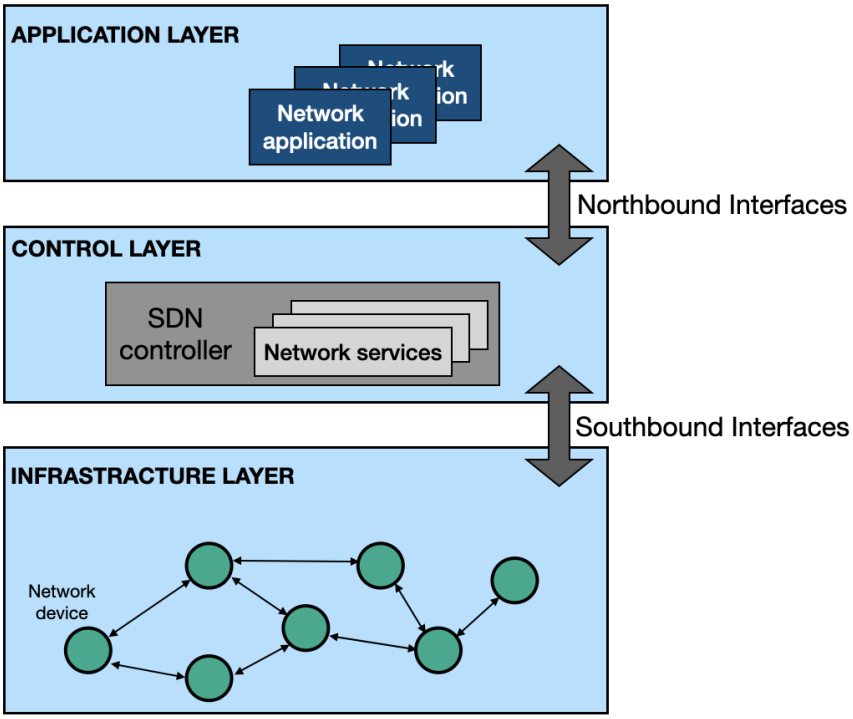

Figure 2.2. Simplified view of an SDN architecture

By separating the control logic from the physical infrastructure, administrators can program and control the entire structure from one centralized point, the SDN controller. This makes network configuration and management more flexible, and scalable and allows an easier creation and introduction of new network abstractions. Furthermore, as illustrated in Figure 2.2, well-defined interfaces are implemented between the different layers to facilitate communication and control.

The adoption of SDN has significant advantages over traditional networking. The SDN network provides higher control, increased speed, and greater flexibility. Using a standard software-based controller, SDN allows administrators to control the network, modify the configuration, allocate virtual resources, and increase capacity, thanks to a centralized user interface, without requiring additional hardware. In terms of security[21], the greater global visibility and the ability to define secure paths offered by SDN allows a more complete view of threats and intrusions. However, centralizing network management in a single software-based controller also presents risks to security and stability. If not adequately protected, the controller can be a point of vulnerability. Furthermore, since SDN allows a fully programmable network configuration, the number of bugs or vulnerabilities may increase, compromising the stability of the system. For this reason, it is important to implement robust security measures to maintain the system secure and reliable.

The implementation of SDN is increasingly popular in various sectors due to its automation, programmability, and flexibility capabilities, which also lead to efficiency and cost benefits. A concrete example of a successful large-scale implementation of SDN is the B4 architecture, the Global Software Defined WAN (SD-WAN) developed by Google[22]. By exploiting the advantages offered by SDN, B4 has achieved levels of scalability, fault tolerance, cost efficiency, and control that would not have been possible with traditional network architectures.

In conclusion, SDN shares some features with NFV, but they are not dependent on each other. Both rely on virtualization and network abstraction, but they use different ways to separate functions and abstract resources. Despite this, NFV and SDN can be combined in a single solution to achieve the best performance. SDN provides NFV with the benefits of a programmable connection between virtualized network functions, while NFV gives SDN the ability to implement network functions through the software on standard generic servers. It is this coexistence between the two paradigms that allow exploiting SDN technology to determine the sequence of functions to direct traffic, creating what are known as Service Function Chains (SFCs).

2.1.3 Service Function Chaining (SFC)

Service Function Chaining (SFC)[23] is a technology for creating an end-to-end network service where traffic is processed through a series of Service Functions (SF). These functions are concatenated in a predefined order specified by the network administrator that meets the designer's needs or user requirements.

The concept of Service Function Chain (SFC) was formally introduced in RFC 7665[24], providing a comprehensive definition. In the context of network architecture, an SFC is an ordered collection of abstract service functions, with specific ordering constraints. Its purpose is to specify the sequential application of these service functions to packets, frames, or flows that have been selected based on classification. For instance, consider an abstract service function such as a firewall. It's important to note that the ordering of service functions may not necessarily follow a linear progression. The SFC architecture allows for branching SFCs, where multiple paths are supported, and it also allows flexibility in the order of applying service functions. In addition, the article defines a Service Function (SF) as a function that carries out specific actions on received network packets. It operates at various layers of the protocol stack, such as the network layer or other OSI layers. Service Functions can be implemented either virtually or integrated into physical network elements. Moreover, a single network element has the capability to accommodate multiple service functions, and it is possible for multiple instances of the same service function to coexist within the same administrative domain.

The SFC works by routing network packets to the first element of interconnected service functions. Once the packet has been processed by this function, it is sent to the next one. This process continues until the packet has been processed by all the service functions in the chain.

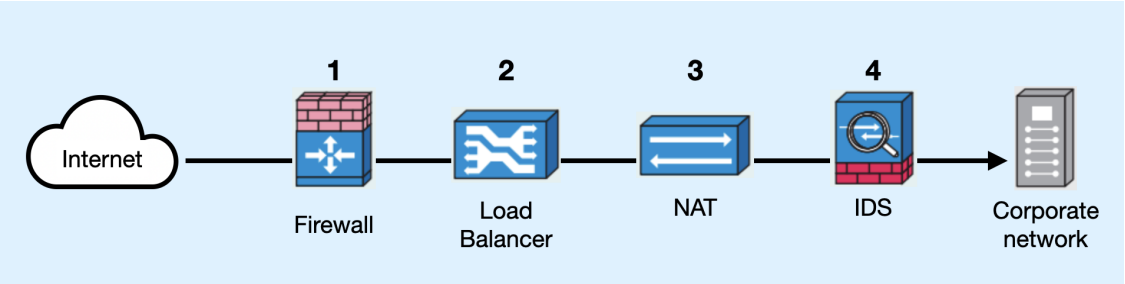

Figure 2.3. SFC example

Figure 2.3 is an example of how a SFC could be implemented in a corporate network that receives traffic from the Internet:

1. Traffic is initially directed to a firewall that manages access to the network's resources and services.
2. The traffic allowed by the firewall is forwarded to a load balancing function, which distributes the traffic to prevent overloading any server and ensure that users can access all services.
3. After the traffic has been processed by the load balancing function, it is routed to a Network Address Translation (NAT) function, which translates public IP addresses into private ones.
4. Then, the traffic passes through an Intrusion Detection System (IDS), which monitors the traffic for any potential security risks such as malware and cyber-attacks.
5. Finally, the traffic is forwarded to the corporate servers where it is processed to provide the services and resources requested by users.

This represents one of the many possible scenarios for implementing an SFC architecture. An important aspect is that the company can decide the order in which network functions are applied, depending on its needs. For example, it could be decided to place an IDS immediately after the firewall to increase security. This allows greater flexibility, but at the same time can create complexity problems in finding the correct design to avoid inconsistencies or conflicts when there are several subnets. To avoid possible anomalies that could potentially lead to errors or unexpected network behavior, some solutions allow the requirements to be verified before their implementation [25]. This preliminary verification process helps to ensure that the network configuration is correct and in line with the established specifications, reducing the risk of possible malfunctions.

In addition to complexity, other problems need to be considered [26]. Among them, the topological dependency of service functions introduces several difficulties when implemented at the hardware level, such as the need to follow a precise order in the sequence of service functions. Moreover, it increases the complexity when

functions are added or removed, requiring in some cases the change of physical network topology. This can lead to inefficient use of network resources and limited and slow flexibility in service delivery.

However, one possible solution is to combine SFC with the new NFV and SDN networking paradigms. Integrating SFC with NFV enables the provision of customized and dynamic network services by implementing network functions through software instead of dedicated hardware. In this way, network functions can be quickly configured and updated to satisfy the specific needs of network services without having to configure or replace specialized hardware. On the other hand, SFCs can be integrated with the SDN to provide centralized management of traffic passing through a set of service functions. This is achieved by combining an SDN controller, responsible for managing the traffic flow, with middleboxes that perform specific functions. Integration with SDN allows SFCs to apply complex network policies such as security, performance optimization, and quality of service (QoS) for data packets.

In conclusion, SFCs are an essential technology for optimizing the management of network traffic, providing greater accuracy, control, and security. With SFCs, organizations can offer reliable, secure, and scalable services at an affordable cost, as well as simplify the monitoring and maintenance of service functions by organizing them in a sequential chain. This is why it is a widely adopted solution by organizations.

2.2 Virtual Network Functions (VNF)

Virtual Network Functions (VNFs) are software components designed to provide network functions in a virtualized environment. This means that traditional network functions, performed by dedicated and proprietary hardware, can now be managed more efficiently and flexibly by this software running on standard devices. Different types of VNFs can exist, e.g. those for security, such as firewalls, intrusion detection and prevention systems (IDS/IPS), and VPNs, which offer protection against threats and intrusions, or those that manage traffic, such as load balancers and virtual routers. Given the wide availability of functions, it becomes crucial to evaluate the needs of a network and service to select the correct VNF. However, one of the most critical aspects is the proper placement and configuration of VNFs within the virtual network in order to have a reliable system. This is achieved through the use of existing models and tools, which allow to perform this task in a fully automatic and optimized way.

The internal architecture of a VNF is presented in Figure 2.4, in accordance with the ETSI Industry Specification Group (ISG) standard[27]. The figure shows that a VNF consists of software modules, running inside containers provided by the virtualization layer, and well-defined interfaces with the orchestration, the NFVI, its EM (Element Management), and other VNFs. The VNF runs on an NFVI and is managed by a Virtualized Network Orchestrator (NFVO) and a Virtualized Network Functions Manager (VNF Manager).

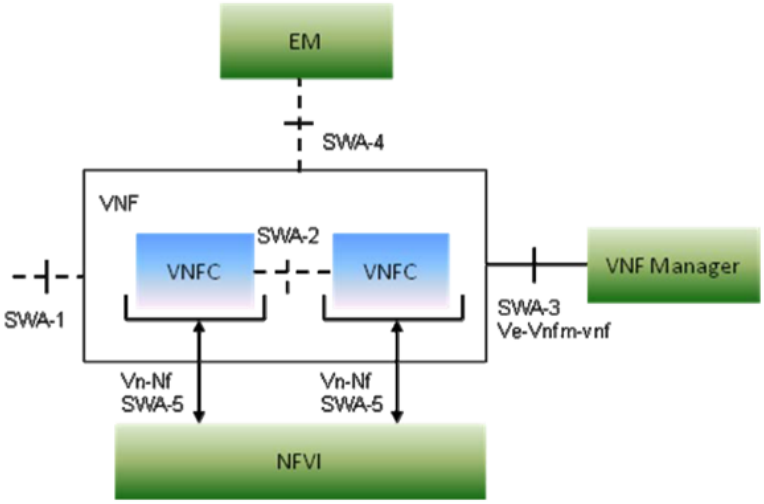

Figure 2.4. VNF internal achitecture

As for software modules, VNF vendors can choose to design their software modules monolithically or by breaking them down into simpler components known as VNF Components (VNFC), each mapped 1:1 to an interface to the NFVI infrastructure. The choice of the structure depends on performance, scalability, service reliability, and security requirements.

As for VNF interfaces, there are several types with different functionalities:

- SWA-1 is a well-defined interface that enables communication between different VNFs within the same or different network services, either through the transmission of data or network function control signals.
- SWA-2 refers to interfaces internal to the VNF, i.e., for communication between VNFC components. These interfaces are defined by the network function providers and are usually not visible to VNF users. SWA-2 interfaces exploit the underlying communication mechanisms implemented on the NFV infrastructure; however, other technologies can be exploited to improve latency performance, such as channel-based communications or shared-memory mechanisms. In the latter case, there may be security concerns and it should be used only in controlled contexts.
- SWA-3 represents the management and control interface for virtualized network functions, which is used for communication between the VNF Manager, the orchestrator, and the VNF.
- SWA-4 is an interface used by the management system (EM) to communicate with a VNF and is dedicated to the runtime management of the VNF.
- SWA-5 represents an abstraction of the set of interfaces for communication between specific VNFs and the underlying infrastructure (NFVI). This

interface may change depending on the type of VNF and the services required at the computation, memory, and network levels.

After defining VNFs, it is important to understand how this technology can be used in modern networks. These virtual functions are designed to be integrated with new network paradigms such as SDN, NFV and SFC to provide more efficient, flexible and scalable solutions. Integrating VNFs with NFV, the benefits of virtualization make the network more flexible and dynamic, where functions are implemented as software applications. Inside the SDN architecture, on the other hand, a VNF can be managed in a centralized manner, enabling a more efficient and customized configuration of individual virtualized functions. This optimizes the utilization of resources and reduces downtime, improving the overall quality of service. Finally, using SFC technology, individual VNFs can be concatenated to create an end-to-end system with a dynamic and customizable service flow that can be adapted to the needs of modern networks.

2.2.1 Virtual Network Security Functions (VNSF)

To ensure security in a virtual environment, there is a specific type of VNF called Virtual Network Security Functions (VNSFs). VNSFs consist of a set of functions used to implement security measures and protect virtual networks from internal and external threats such as network attacks, intrusions, malware, and data theft. Since they operate in a virtual environment, VNSFs, like all VNFs, enjoy the benefits of virtualization and the network paradigms (SDN, SFC, and NFV) with which they can be integrated. The ability to manage and configure network functions in a centralized manner simplifies the security management of virtual networks. This makes security management more efficient and convenient, as configurations can be changed and updated in real-time from a single location.

According to [28], security functions can be divided into four categories: attack detection, prevention, deception, and mitigation. The first type includes technologies that monitor network traffic to identify anomalies. The second type, in addition to detecting anomalies, implements preventive measures to block attacks. The third category, deception attack, as proposed in the article, differs from prevention as its purpose is to deceive or disorient the attacker, rather than simply blocking the attack. This allows defenders to gather valuable information about the attacker's behavior and waste his resources. On the contrary, the category of attack mitigation tries to limit the impact of attacks through a combination of various functions (including those from other categories) when complete prevention is not possible.

The following presents and explains some of the possible types of VNSFs that can be used within a virtual network, depending on the requirements, and they fall into one of the categories described above. Traditionally, most of these functions would have been implemented on dedicated hardware to process network traffic along the data path. Today, these functions are deployed as VNFs, which offer the same functionality as dedicated physical devices but are implemented as software.

Virtual Intrusion Detection System (IDS)

Virtual IDS is a security function for attack detection designed to monitor the network for anomalous or potentially malicious behavior. If suspicious traffic is detected, it issues an alert that notifies security managers to take further action.

Virtual Firewall

The virtual firewall is a prevention solution for the protection of a virtual network from external threats, preventing intrusions and attacks. It performs controls of incoming and outgoing connections and blocks unauthorized requests. In this way, only traffic that meets predefined rules or user-configured settings is allowed. Virtual firewalls can be cheaper and easier to manage than traditional physical firewalls, especially when there are multiple virtual machines and large network infrastructures. In addition, virtual firewalls can be more easily updated to meet network requirements.

Virtual Intrusion Prevention System (IPS)

The Intrusion Prevention System (IPS) is a security system designed to prevent intrusions or attacks within a network or system. Its operation is similar to that of the IDS but includes the ability to immediately activate security measures to block any malicious traffic before it can reach the protected network or system.

Virtual VPN Gateway

The Virtual VPN Gateway is an attack prevention software solution used to implement a Virtual Private Network (VPN) that creates a private and secure connection between two private networks over a public network, such as the Internet. By using security protocols like IPsec or SSL/TLS, the VPN Gateway ensures data encryption and protection against potential threats. The VPN Gateway is a critical solution for secure communications, and therefore it needs to be properly allocated and configured. This thesis proposes a solution for this purpose, using the VEREFOO framework.

Honeypot

A virtual Honeypot is a simulated environment designed to attract and detect potential attackers and malicious activities. It acts as a decoy by intentionally placing vulnerabilities to make attackers believe they have found a real system to attack. In reality, it allows monitoring their activities, analyzing their methods, and gathering information about their tactics and tools. This helps organizations better understand and respond to cyber threats and improve their overall security measures.

Figure 2.5 shows some available VNSFs. For each topology, a brief description and the use of this technology in networking is given.

VNF	Description	Use in security
Antispam	Email filtering	Malware detection, Phishing prevention
Antivirus	Email/Web scanning/Endpoint security	Virus/Trojan/Malware detection
DLP	Data Loss/Leakage Prevention	Data exfiltration detection
DPI	Payload analysis	Spam Filtering, Intrusion detection, DDoS detection, Malware detection, Security Analytics
Honeypot	Traffic redirection and inspection	Spam filtering, Malware detection, SQL database protection, Security Analytics
IDS	Traffic inspection (header and payload)	Intrusion detection, Malware detection, DDoS detection, Security Analytics
IPS	Traffic filtering based on header and payload	Intrusion prevention, DDoS prevention
NAT[1]	IP address mapping	Intrusion prevention
Packet Filter Firewall	Header-based packet filtering	Intrusion prevention
Parental Control	Media content filtering	Blocking access to inappropriate content
VPN Gateway	Site-to-site VPN connection over unsecured networks	Data Tunneling/Encryption
WAF	HTTP traffic monitoring, filtering, logging	Prevention of SQL injection, cross-site scripting

[1] NAT is not a security function but inherently provides packet filtering similar to a firewall.

Figure 2.5. Taxonomy of Security VNFs. [1]

Choosing VNSFs to protect a company's virtual networks is an important process that requires careful evaluation of needs, market options, and technical aspects of different solutions. Firstly, the security objectives of the company and the specific requirements of the virtual networks that need protection must be defined. This could involve securing networks and systems against external intrusions, managing remote access to business services, or monitoring network traffic for suspicious activities.

Then, various VNSFs security solutions available in the market can be explored, considering their strengths and weaknesses. Once a suitable VNSF security solution has been chosen, it becomes essential to configure it correctly to ensure optimal efficiency and protection. This involves defining security policies and configuration rules while regularly testing the solution to identify any potential issues or vulnerabilities.

2.2.2 Limitations of Manual Configuration

The softwarization of networks, made possible by the SDN and NFV paradigms, simplifies network management operations and replaces physical middleboxes with virtual functions. This enables the creation of flexible and scalable networks that can adapt to the needs of users and applications.

In the past, the configuration of networks and VNSF was manually performed by administrators in order to meet the necessary security requirements for network users. For example, if an administrator wanted to block all traffic to a certain

website, they would add the corresponding filtering rule to a firewall. If an attack occurred, the behavior would be changed to prevent a possible repetition. This approach worked only with small or static networks, where network accesses could be easily known.

However, the increase in the varieties of virtual technologies available, which are more complex than the previous physical middleboxes, and the growing number of possible solutions have an impact on the complexity and size of networks. This makes it more difficult to find the best solution in terms of correctness and optimization. In addition, the dynamism introduced by virtualization requires continuous reconfiguration of the security system due to the frequent changing of IP addresses and ports of virtual services by NFV controllers [29].

These difficulties introduced by virtualization come on top of the common problems of configuring a network. One of these concerns communication between the network administrator and the security manager, who often operate separately. A lack of communication between them can lead to making trivial mistakes that can compromise network security. In addition, the security manager may not always be up-to-date on the latest cyber threats or the best protection solutions, increasing the network's vulnerability to external attacks. Another problem is the trial-and-error approach to configuration, which often leads to complexity in configuration files. This can make network management more difficult and increase troubleshooting time [30].

Not only configuration but also security orchestration is afflicted by problems related to manual operations. For example, when there is a transition from one network security state to another, the changes must be applied to the network as quickly as possible and must be orchestrated to minimize the number of intermediate states in which security can be compromised.

To solve the problems described, automated tools are used to allocate and configure virtual functions correctly and efficiently. Automation simplifies network management, improves security, and reduces troubleshooting time. The use of these tools reduces human errors and improves the overall quality of the network. An further benefit is the reduction in the level of expertise or experience required in network security, as management operations are automated and users only have to deal with monitoring the tools that perform the operations.

In addition, automation allows the large size and heterogeneity of modern computer networks to be managed more efficiently because of the comprehensive view of the entire structure that automated tools provide. The heterogeneity of different implementations of the same security function can be abstracted so that the automated system treats them as the same functionality, adapting the result produced to the correct vendor-dependent commands only later. Moreover, it not only allows a correct solution to be found and configured but also the optimal one in terms of resource consumption.

In conclusion, the proper implementation and configuration of network functions is a key step in ensuring data security and network business continuity. Automatic configuration of network functions, with its many benefits, is the ideal solution to achieve these goals. This is the path taken by many companies so as to minimize the number of operations performed manually by human users. With automatic

configuration, systems require only a few specific inputs from humans or other systems to operate autonomously, reducing downtime and increasing the security of corporate data.

Chapter 3

The VEREFOO Approach

The previous chapter shows how the constant evolution of computer systems made virtualization an essential element for organizations due to its numerous advantages. Consequently, the search for tools capable of providing and configuring reliable services in an efficient and secure manner, with limited use of resources, has increased. To address this challenge, it has been decided to develop policy-based configuration tools that support operators in creating and configuring trusted security services (Policy-Based Management [31]). These tools can verify and fulfill a set of input-specified security requirements to achieve the intended goals. Automation of this process is essential to minimize human error, as already highlighted in the previous chapter. In addition to automation, two other crucial aspects are considered in the development of these tools. The first, concerns the possibility of obtaining a correct solution by construction through formal verification, guaranteeing maximum reliability and security of the systems. The second aspect, on the other hand, is the need to search for an optimal solution that can allocate and configure all security functions without conflict, minimizing the use of resources. In this way, reliable, efficient, and sustainable security services are achieved.

This chapter illustrates the framework called VEREFOO (VErified REfinement and Optimized Orchestration), which represents an approach to automate the creation of reliable security services, guaranteeing their formal correctness and optimization. The VEREFOO approach is designed to be a general method that can be applied to any type of NSFs. However, in this thesis, it is used for the automatic configuration of VPNs for secure communications.

VEREFOO represents the first innovative methodology combining automation, formal verification, and optimization for the allocation and configuration of Network Security Functions in virtual networks. This approach is based on the policy-based management paradigm, which requires the specification of network security policies to describe the required security behavior in the virtual network. In this way, VEREFOO ensures that user-specified security policies are satisfied during the allocation and configuration of NSFs in the virtual network.

The general functioning of VEREFOO is schematized in Figure 3.1. The user provides as input the Service Graph (SG) and a set of Network Security Requirements (NSR). VEREFOO processes the input data and uses a solver to generate an optimal solution. If at least one solution is found, the engine generates a new

SG enriched by the allocated NSFs and their configuration rules, which satisfy the security policies specified by the user. Otherwise, a not-enforceability report is generated to help the user fix the input and obtain a correct solution in the next execution.

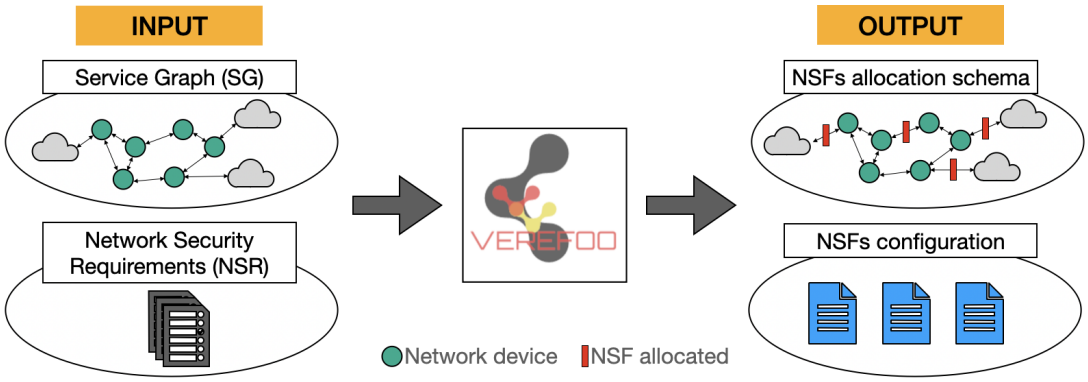

Figure 3.1. The VEREFOO Approach

3.1 Inputs and Outputs of the framework

3.1.1 Input: Service Graph and Allocation Graph

A Service Graph (SG) represents the logical topology of a virtual network, i.e. the interconnection of service functions and network nodes that provide a complete end-to-end network service. The service designer can leverage Network Functions (NF) to create an SG that provides a reliable and high-performance network service to users, whose access points are represented by the SG endpoints (e.g. clients, servers, subnets).

Once the SG is created, VEREFOO processes it and generates the Allocation Graph (AG), which represents the optimal virtual network topology for the security requirements specified by the user. The AG is composed of placeholder elements called Allocation Place (AP), placed between any pair of nodes or functions in the network. In this location, VEREFOO may decide to insert an NSF to achieve the optimal allocation scheme. However, a security service designer can force the allocation of an NSF in a specific AP or forbid the use of specific APs, increasing the flexibility of the proposed methodology.

This manual customization of the AG also reduces computation time, as it decreases the solution space that the algorithm must search for solving the problem. On the other hand, the user's interaction with the AG configuration may lead to a non-optimized solution or the impossibility to find one, as some acceptable - and potentially optimal - solutions may be discarded based on the user's input.

3.1.2 Input: Network Security Requirements

VEREFOO also requires a set of Network Security Requirements (NSR), which define the security criteria to be implemented in the network. These requirements are specified by users using an intermediate-level language, which abstracts from the specific characteristics of Network Security Functions (NSF) implementations provided by different vendors. This language allows users to define security requirements without requiring in-depth knowledge of network security, making the VEREFOO approach accessible even to a network administrator with limited skills.

There are different types of security requirements that a user can define for a network. This thesis focuses on the Communication Protection Policies (CPP) that enable the creation of VPNs to ensure confidentiality, integrity and availability of an information flow from a source to a destination. With the automated VEREFOO framework, it is possible to verify the correctness of user-defined protection requirements in an effective and reliable manner.

3.1.3 Expected outcome

After receiving the SG and the CPP, the VEREFOO solver generates a Security Service Graph (SSG), enriching the original SG with the appropriate allocated NSFs and configuration rules for each security function, to satisfy the security policies specified by the user. For example, if the user has defined a security policy for specific traffic, the allocated security function will contain a rule to guarantee the security of that specific communication. The allocation process aims to minimize the number of NSFs required to enforce all input policies, thus reducing resource consumption. Furthermore, the configuration of each allocated NSF is optimized to use the minimum number of configured rules, reducing memory usage and optimizing NSF performance.

3.2 MaxSMT problem formulation

The formally correct and optimal solution to the automatic NSF allocation and configuration problem is achieved through the formulation and resolution of a weighted partial MaxSMT problem, which is a generalization of the traditional SMT problem. This optimized formulation of the SMT problem distinguishes between two sets of input constraints: “hard” and “soft” clauses. The “hard clauses” represent the constraints that must always be satisfied to obtain a solution to the problem, while the “soft clauses” have an assigned weight and their satisfaction is not strictly necessary, but is subject to the optimization objective, which is to maximize the sum of the weights of the satisfied clauses.

This formulation allows for an optimized correctness-by-construction approach, where no further verification of the correctness or optimality of the solution is required. The partial and weighted formulation of MaxSMT is fundamental to achieve the three main objectives of the VEREFOO approach: full automation, optimization, and formal correctness. Full automation is possible because a MaxSMT

problem can be solved without human intervention, except for input specifications. Optimization is achieved by expressing optimization goals as "soft clauses", while formal correctness is ensured by representing correctness requirements as "hard clauses". The formal correctness-by-construction approach is advantageous because it not only increases assurance and confidence in the correctness of the computed solution but also avoids the need for a-posteriori formal verification. In fact, the solution can already be considered formally correct as far as all problem components are correctly modeled. Such models must capture all information that may influence the correctness of the solution, including the security requirements and forwarding behavior of the network in which the NSFs are to be applied. At the same time, the number and complexity of constraints in the MaxSMT problem must be limited to ensure the scalability of the approach. However, the adoption of this approach represents an important step forward in solving the problem of automatic allocation and configuration of NSFs, offering an optimal, correct, and fully automated solution.[8]

For solving SMT and MaxSMT problems, there are highly efficient automatic solvers, such as *z3*[32], developed at Microsoft Research. This solver can automatically check if a set of logical formulae called a set of constraints, has a satisfiable solution. This means that z3 can determine if there is a combination of values that satisfies all given constraints, with very good performance.

3.3 Traffic Flow Models

After illustrating the general aspects of the VEREFOO framework, it is important to go deeper into the different network modelling approaches used to perform the tests described in the Chapter 6.

These approaches are based on the concept of Traffic Flow, which represents a flow of packets through the network. The traffic flow model describes how a certain class of packets is forwarded and transformed by the various network functions between a source node and a destination node. Analysing a flow of packets instead of individual packets offers several advantages, as it allows for a simpler modeling of network security functions and reduces the number of packets to consider. In particular, these approaches aim to solve the problem of automated configuration and correctness verification of security mechanisms defined within a virtual network.

In this thesis, two different approaches are used for the identification and computation of flows. The first approach, called Atomic Flows, simplifies a class of packets as much as possible by managing their atomic flows. The second model, known as Maximal Flows, reduces the overall number of flows by aggregating multiple flows together.

Before describing the functioning of these two models, it is necessary to define two fundamental concepts: the *Predicate* and the *Traffic Flow*, whose definitions can be found in Article [9].

Definition 3.3.1 ***Predicate****: A class of packets, also called traffic, is modeled as a predicate defined over variables that represent some packet fields. In this way, packets that do not differ in these fields are represented by the same class.*

There are various ways of representing a predicate, but in this study the approach introduced in [12] is adopted. According to this approach, a predicate is composed of sub-predicates, each of which corresponds to a specific field of the package. The conjunction of these sub-predicates forms the complete predicate, which is represented by a tuple. In the context of this study, predicates are modelled by the IP quintuple, which includes the following fields: IP source, IP destination, source port, destination port and protocol type. In addition, each sub-predicate can represent a single value, a range of values or the entire range, indicated by the wildcard "*". For example, a predicate may be represented by the following tuple (10.0.0.5, 20.0.0.*, 8080, *, tcp). In this case, the IP address 10.0.0.1 indicates the source, the IP addresses of the subnet 20.0.0.0/24 are the destination, the source port is 8080, the destination port can take any value, and the protocol type is TCP.

Related to the predicate concept there is the Traffic Flow. As reported in [9] and [12], a Traffic Flow represents a flow of packets that can cross the network. This term describes how a class of packets can be forwarded or transformed by the various nodes in the network that are between the source and destination of this traffic. The definition of traffic flow is as follows:

Definition 3.3.2 ***Traffic flows****: A Traffic Flow $f \in F$ is formally modeled as a list of alternating nodes and predicates, [n_s, t_{sa}, n_a, t_{ab}, n_b,...,n_k, t_{kd}, n_d].*

In this representation, n indicates a node along the path, while t_{xy} is the traffic generated by source node n_x with destination node n_y.

3.3.1 Atomic Flows

The Atomic Flows model is based on the idea of simplifying a class of packets as much as possible by handling only atomic flows. This Atomic Predicate concept, first presented in [33], allows a set of network predicates that are disjoint and minimal (atomic) to be handled instead of all predicates being created. Each predicate of the main set can be expressed as a disjunction of predicates of the atomic set. In this way, each complex predicate is decomposed into a set of simpler atomic predicates.

The operation of the model involves the initial creation of atomic predicates relating to the user-defined policies as input. Subsequently, the selected atomic predicates are used to compute the relevant Atomic Flows. An Atomic Flow represents a flow of packets in which every traffic within it is an atomic predicate.

Definition 3.3.3 ***Atomic Flows****: A flow f=[n_s, t_{sa}, n_a, t_{ab}, n_b,...,n_k, t_{kd}, n_d] is defined as atomic if each traffic t_{ij} is an atomic predicate.*

The important feature of atomic predicates is that they are disjoint and unique. This offers an important advantage in that each atomic predicate is assigned an integer value to identify it, replacing a more complex representation. The use of an integer identifier makes it much easier for the solver to handle the atomic predicate to solve the MaxSMT problem.

3.3.2 Maximal Flows

A completely opposite approach to the previous one is the Maximal Flows model. In this case, instead of dividing the flows into several atomic predicates, the number of representative flows is reduced by considering a smaller but equally significant subset. This subset is called Maximal Flows. In the Maximal Flows model, all flows represented by the same Maximal Flow behave similarly when crossing the network, so it is sufficient to consider only the Maximal Flow instead of each individual flow.

Definition 3.3.4 *Called F_r the set of possible flows of the network, the corresponding set of Maximal Flows ${F_r}^M$ matches the following definition: ${F_r}^M = \{{f_r}^M \in F_r \mid \nexists\, f \in F_r.(f \neq {f_r}^M \wedge {f_r}^M \subseteq f)\}$*

The set ${F_r}^M$ is defined as a subset of F_r (set of all possible flows in the network) that contains only those flows that are not subsets of any other flow in F_r. In other words, as many different flows as possible are aggregated into representative maximal flows that exhibit similar behaviour when passing through the network nodes. This allows only the maximal flows to be considered instead of each individual represented flow. Even in the case of Maximal Flows, flows are still modelled as a list of alternating nodes and predicates, but the predicates within a Maximal Flow are not necessarily atomic. They express the disjunction of multiple quintuples. Therefore, they can no longer be identified by a simple integer value, but require the use of more complex data structures for their representation.

Atomic Flows vs Maximal Flows

Comparing the two models, the most significant differences arise in the following aspects.

In the Atomic Flows model, predicates are atomic and completely disjoint from each other. This allows predicates to be identified by an integer value, offering an advantage when solving the MaxSMT. Problem formulation is simpler when only integer values are used instead of complex representations. However, in the case of Maximal Flows, this feature is not present.

Another difference concerns the processing of atomic predicates and flows. In the case of Atomic Flows, it is necessary to initially compute the atomic predicates and flows before solving the overall problem. This can lead to an initial delay in processing. In contrast, in Maximal Flows, this initial delay does not occur. A partial solution to mitigate the initial delay in Atomic Flows model could be the introduction of parallelization in the computation of Atomic Flows from Atomic predicates. However, this is not necessary in the context of Maximal Flows.

3.4 VPN Implementation in VEREFOO

Virtual Private Networks (VPNs) are a technology for establishing a secure connection through a public network, such as the Internet. To protect these communications, VPNs create a secure tunnel through which data is encrypted and

transmitted. This means that even if an insecure network is used, the data sent through the VPN will be protected and inaccessible to third parties.

VPNs can be implemented using various technologies, including IPSec and TLS, which use different protocols. VPNs with IPSec provide network-level protection via the IPSec protocol, while VPNs with TLS offer transport-level protection using the TLS (Transport Layer Security) protocol. Both technologies are widely used and considered secure. However, this thesis focuses on analyzing the performance of VPNs with IPSec within the VEREFOO framework.

The VEREFOO framework was developed to also support virtual security functions that enable secure communications by combining three fundamental characteristics: full automation, formal verification and optimization.

The operation of VEREFOO with VPNs follows a similar approach to other virtual security functions. The user provides as input the Service Graph (SG) and a set of Channel Protection Policies (CPP), which represent the security requirements for channel protection. These inputs are processed and managed by the solver which, through the formulation and resolution of the MaxSMT problem, finds an optimal and correct solution that satisfies the requirements. If at least one solution has been found, the solver generates a Security Service Graph (SSG), which is the original SG with the allocated Channel Protection System (CPS), responsible for channel protection, and the CPS Configuration (CC), which includes all device configuration rules. In case no solution is found, a not-enforceability report is generated. The operation of the solver is based on a constraint-based approach since the search for a correct and optimal solution relies on compliance with some hard and soft constraints.

The enforcement of the CPPs is expressed with hard constraints that must be satisfied to achieve a correct solution. These hard constraints, as described in [13], include the following:

- In the SG provided as user input, at least two CPSs must be allocated for each flow that satisfies the CPP conditions. The first CPS must protect traffic passing through a set of untrusted nodes, while the second, placed after these untrusted nodes, removes the protection.
- When flows cross the untrusted middleboxes, they must respect the security properties expressed by the CPP.
- When flows pass through inspector middleboxes, they must be unprotected to allow traffic analysis.
- When traffic reaches the destination node, it must be unprotected.
- When traffic reaches one of the two nodes corresponding to the VPN endpoints, the traffic should not be tunneled. However, if a node does not correspond to a VPN endpoint but is located between the two endpoints, the traffic must be tunneled.

Other hard constraints are necessary to express the configuration and allocation decisions of a CPS for each network node. Regarding CPS allocation, each AP node

in the SG can be considered as a potential candidate position for the allocation of a CPS. This is represented by the *allocate* predicate, which is a free variable whose value will be determined by the solver. As for configuring a rule within a CPS, the responsible predicate is *configure*, which takes the value of true if that rule is actually configured, otherwise false. Additionally, if a node has at least one CPP configured, it is necessary to allocate a CPS in that node. This behavior is ensured by an additional hard constraint.

With the presence of these constraints, the solution space is reduced, allowing a faster search for a correct solution, since the solver needs to analyze a smaller set of valid solutions to identify the best one. However, the absence of a correct solution to the problem may occur.

The hard constraints described so far are sufficient to guarantee a correct, but not an optimal solution. Therefore, it is necessary to define soft constraints to achieve an optimal solution. Each soft constraint is defined using the notation $Soft(c,w)$ where c represents the constraint and w the assigned weight.

The first optimization objective is to minimize the number of CPSs allocated in the SG in order to reduce resource consumption. This is achieved by using soft clauses that require, when possible and without violating hard constraints, not to install a CPS. The solver will try to satisfy as many of these soft clauses as possible, with positive assigned weights.

The second optimization objective is to minimize the number of rules configured within CPSs to enforce all the CPPs. The purpose is to improve the efficiency of security operations by avoiding redundant rules. A soft clause is defined for each possible rule, and the best situation is to have no rules configured. The solver will try to satisfy as many of these soft clauses as possible, taking into account the assigned weights.

3.4.1 Communication Protection Model

The behaviour of each CPS allocated in the network depends on a set of rules that specify the actions to be performed and the conditions that identify the traffic subject to those actions. When a VPN is configured, the systems responsible for creating secure communication are the VPN Gateways, which are network devices capable of adding and removing traffic protection.

A VPN Gateway can operate in two different ways:

- ACCESS: indicates the start of secure communication, where incoming traffic is protected.
- EXIT: indicates the end of secure communication, where protection is removed from the traffic.

When a VPN Gateway is created, a Security Association is added defining the configuration of that VPN Gateway and the actions to be taken when a packet is received. When a packet reaches a VPN Gateway and its fields match the gateway's

Security Association, the actions configured for that VPN Gateway are applied. This means, if the conditions of the packet match, protection is added or removed based on the VPN Gateway's behaviour (ACCESS or EXIT). If no rules match, the packet is simply forwarded.

Each security association is represented by the following model:

[*behavior - startChannel - endChannel - Conditions - authAlg - encAlg*]

- **Behaviour**: describes the VPN Gateway's behaviour with packets that meet the rule conditions. It can be ACCESS or EXIT.
- **startChannel**: indicates the starting point of the secure channel.
- **endChannel**: indicates the ending point of the secure channel.
- **Conditions**: are the conditions of the packet for which protection needs to be added or removed. Conditions are specified in the format [*IPSrc - IPDst - portSrc - portDst - transportProto*]:
  - *IPSrc*: is the source IP address of the packet;
  - *IPDst*: is the destination IP address of the packet;
  - *portSrc*: is the source port of the transport layer;
  - *portDst*: is the destination port of the transport layer;
  - *transportProto*: is the transport layer protocol;
- **authAlg**: Indicates the authentication algorithm used to authenticate packets that meet the specified conditions.
- **encAlg**: Indicates the encryption algorithm used to encrypt packets that meet the specified conditions.

Example of VPN Implementation

Considering a simple network (Fig.3.2), a security requirement is defined to protect the traffic crossing the network from client (IP = 10.0.0.1) to server (IP = 20.0.0.1), knowing that UN (IP = 30.0.0.1) is an untrusted node. In addition, the two endpoints, the client and the server, are configured in such a way that they cannot support a VPN.

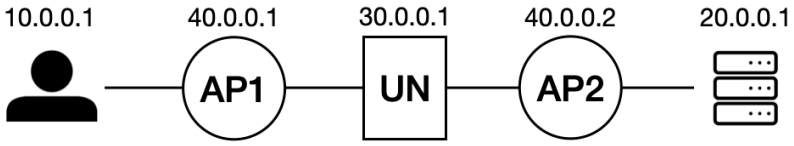

Figure 3.2. Network topology with untrusted node

The requirement, represented in XML format, is as follows:

Listing 3.1. CPP XML representation

```
<PropertyDefinition>
    <Property name="ProtectionProperty" graph="0" src="10.0.0.1"
    dst="20.0.0.1" lv4proto="ANY" src_port="null" dst_port="null"
    isSat="true">
        <protectionInfo>
            <untrustedNode node="30.0.0.1"/>
        </protectionInfo>
    </Property>
</PropertyDefinition>
```

The solution is shown in Fig.3.3, where the solver has allocated two VPN gateways, one in node AP1 to protect the traffic and one in node AP2 to remove the protection, allowing node B to receive the traffic in clear.

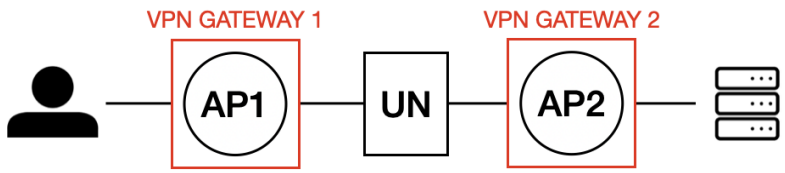

Figure 3.3. Solution

Regarding the configuration, the two VPN gateways are set up in the following manner:

Listing 3.2. VPN GATEWAY 1 XML configuration

```
<node name="40.0.0.1" functional_type="VPNGateway">
    <neighbour name="10.0.0.1"/>
    <neighbour name="30.0.0.1"/>
    <configuration name="AutoConf">
        <vpnGateway>
            <securityAssociation>
                <behavior>ACCESS</behavior>
                <startChannel>40.0.0.1</startChannel>
                <endChannel>40.0.0.2</endChannel>
                <source>10.0.0.1</source>
                <destination>20.0.0.1</destination>
                <protocol>ANY</protocol>
                <src_port>*</src_port>
                <dst_port>*</dst_port>
            </securityAssociation>
        </vpnGateway>
    </configuration>
</node>
```

Listing 3.3. VPN GATEWAY 2 XML configuration

```
<node name="40.0.0.2" functional_type="VPNGateway">
    <neighbour name="30.0.0.1"/>
    <neighbour name="20.0.0.1"/>
    <configuration name="AutoConf">
        <vpnGateway>
            <securityAssociation>
                <behavior>EXIT</behavior>
                <startChannel>40.0.0.1</startChannel>
                <endChannel>40.0.0.2</endChannel>
                <source>10.0.0.1</source>
                <destination>20.0.0.1</destination>
                <protocol>ANY</protocol>
                <src_port>*</src_port>
                <dst_port>*</dst_port>
            </securityAssociation>
        </vpnGateway>
    </configuration>
</node>
```

Chapter 4

Thesis Objective

As mentioned in the introduction, the advent of virtualization has changed the implementation of traditional networks. Virtualization paradigms such as NFV, SDN, and SFC have transformed hardware devices into software applications, offering more dynamic and flexible management of networks. However, despite the numerous advantages offered by these solutions, there are some limitations to consider. As illustrated in Section 2.2.2, one of the main difficulties is the increased complexity of configuring such systems, especially when done manually. This complexity can have a negative impact, especially on security, since an incorrect configuration of the network can make it vulnerable to attacks. Therefore, the main challenge is to find models that can simplify configuration and perform these operations in an automatic manner.

One tool that addresses these needs is VEREFOO, which has been discussed in Chapter 3. This framework automatically allocates and configures specific security functions to ensure adequate system protection. In its current implementation, VEREFOO can correctly configure network functions such as packet filters to provide security functionality. However, its usage has been extended to configure secure communications through VPNs, implemented using the IPSec protocol. The reason for this new extension is due to the increasing priority of organizations to configure secure communications between users within networks. To ensure secure communication, traffic must pass through channels capable of maintaining confidentiality, integrity, and availability properties. Users who wish to set up secure communication between two points in the network only need to define security policies, called Communication Protection Policies (CPP). For each security policy, the source and destination for each communication to be protected are indicated. Based on these policies, VEREFOO configures and allocates VPN gateways to introduce traffic protection when necessary. For example, within a network, especially in a public network like the Internet, some nodes or systems may be insecure, and in such cases, traffic should be protected using VPNs. In addition, each VPN gateway must be able to remove such protection allowing the destination node to receive clear traffic.

In light of these considerations, the extension of VEREFOO in configuring secure communications requires further evaluation. In addition to verifying its proper

functioning, a main objective of this thesis is to evaluate the framework's performance in scenarios where networks are complex and the number of defined protection policies is high. To achieve this, a network generator will be proposed to create realistic network scenarios characterized by complexity and the presence of multiple traffic flows requiring protection. Its performance will be evaluated by analyzing the execution time required by the framework to find the correct and optimal solution based on input requirements. A series of specific tests will be conducted to determine the impact of different parameters on the configuration and overall performance of the system. The first two tests aim to evaluate the framework's performance as the network size and the number of security policies defined increase. In particular, the second test will include an inspector node within the generated network, whose task is to monitor traffic without protection. The final test aims to evaluate which parameter has a greater impact on performance by comparing the number of Allocation Places (APs) and the number of defined protection policies under certain conditions. For each test, two different models previously implemented in VEREFOO, Atomic Flows, and Maximal Flows, will be used. Both models are based on different implementations but share a common idea of analyzing flows instead of individual packets. The execution of these tests, along with the implementation of the generator, will provide data describing the framework's performance and behavior, as well as evaluate which model offers greater advantages. This will help identify possible areas that require future improvements.

In conclusion, this thesis aims to achieve the following objectives:

- Verify the correctness of the VEREFOO framework in supporting the configuration of secure communications, with particular emphasis on VPNs with IPSec.
- Develop a complex network generator to evaluate the effectiveness of VEREFOO in configuring secure communications in realistic scenarios.
- Evaluate the performance of the VEREFOO framework in correctly configuring and allocating IPSec VPNs, identifying the parameters that most significantly influence performance.
- Evaluate which model, Atomic Flows or Maximal Flows, offers greater advantages under the same conditions.
- Provide a critical evaluation of the performance and capabilities of the extended framework in the context of secure communications and complex networks.

By achieving these objectives, the aim is to contribute to the development of automated and reliable solutions for configuring secure communications within virtualized networks, enabling more efficient and secure network management.

Chapter 5

Proposed Network Generator

This chapter provides an analysis of the specific characteristics and functionalities of each type of node within a network topology used in the tests described in the next chapter. For each node, its placement within the network, the need to protect the traffic it handles, and its implementation within the VEREFOO framework are described. In addition, it explores the fundamental security requirements needed to guarantee secure communication between nodes.

Then, the implementation of the proposed network generator is discussed in more detail. This includes the process of generating the network topology, the criteria used to establish connections between nodes, and the distribution of security requirements. In the last section, it is examined how this generator interacts with VEREFOO, providing an efficient and optimized model for the automatic configuration of VPNs.

5.1 Network and security requirements model

When a secure communication is required, the first step is to analyze the nodes present between source A and destination B, to identify those where traffic must be protected. There are nodes that are considered trustworthy where traffic can pass unencrypted, while others have possible vulnerabilities and the traffic needs some protection. Each node is classified and managed differently, depending on the function installed or its reliability.

5.1.1 Network Node Types

Before discussing the classification of nodes, it is important to understand how a network node is defined within the VEREFOO framework.

In VEREFOO, the *Node.java* class is responsible for defining nodes with a set of attributes and methods. Each node has a *name* attribute, which is its IP address. This IP address is composed of 4 bytes separated by a dot but is represented as a single string. When a node is created, each byte of its IP address is randomly

generated, with a check to make sure that it is unique in the network. A Node object also has a *FunctionalType* attribute, which specifies the role of the node within the network. Possible values are *WEBCLIENT*, *WEBSERVER*, *FORWARDER*, *VPN_GATEWAY*, and others. An important feature that a node may have is the ability to configure a VPN gateway, indicated by the Boolean attribute *vpnCapabilities*. If this value is set to TRUE, it means that that node can configure a VPN gateway, otherwise, it is set to FALSE.

In addition to these attributes, each node has a list of the neighbours to which it is connected. This is implemented in Java via a list of Neighbour objects, which contain attributes such as the name (IP address) and ID of the Node object.

For example, considering the client in the Figure 5.1, its attributes will include the following values:

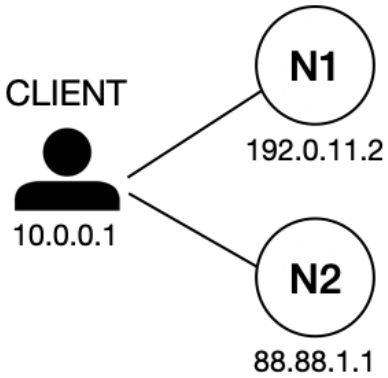

Figure 5.1. Node object

- Name = "10.0.0.1"
- VpnCapabilities = FALSE
- FunctionalType = WEBCLIENT
- Neighbours = {N1, N2}

Endpoint Nodes

The endpoint represents the edge points of a network connection. In the proposed network generator, an endpoint can be a client or a server and certain security requirements must be defined to protect the data transmitted between them, which can be the source and destination of the communication.

Assuming a network with two clients and one server, as in the Figure 5.2, if the clients want to communicate with the server but the security of the network cannot be guaranteed, it is necessary to protect the traffic from the clients to the server. For this purpose, it is possible to configure the clients and the server with a VPN gateway. One possible solution is:

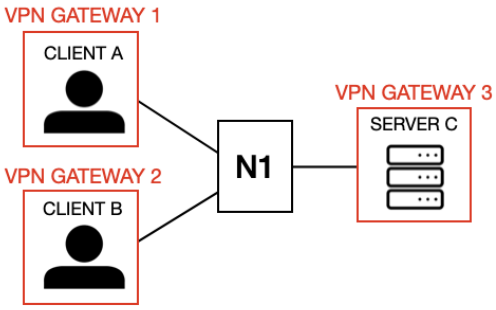

Figure 5.2. Allocation of VPN Gateways at endpoints

Endpoint Nodes in VEREFOO

Within VEREFOO, clients and servers are implemented similarly without significant distinction. The main difference is the *FunctionalType* defined when the node is created, where a value *WEBCLIENT* is set for the client and a *WEBSERVER* is set for the server. In addition, each client and server has a *vpnCapabilities* property which, if set to true, indicates that the node is capable of setting up a VPN gateway. This node then can be the starting or end point of the VPN, enabling secure traffic protection.

Listing 5.1. Client node creation in VEREFOO

```
for (int i = 1; i <= numberWebClients; i++) {
    String IPClient = createRandomIP();
    Node client = new Node();
    client.setFunctionalType(FunctionalTypes.WEBCLIENT);
    client.setName(IPClient);
    Configuration confC = new Configuration();
    confC.setName("confA");
    Webclient wc = new Webclient();
    confC.setWebclient(wc);
    client.setConfiguration(confC);
    allClients.add(client);
}
```

Listing 5.2. Server node creation in VEREFOO

```
for (int i = 1; i <= numberWebServers; i++) {
    String IPServer = createRandomIP();
    Node server = new Node();
    server.setFunctionalType(FunctionalTypes.WEBSERVER);
    server.setName(IPServer);
    Configuration confS = new Configuration();
    confS.setName("confB");
    Webserver ws = new Webserver();
    ws.setName(server.getName());
    confS.setWebserver(ws);
    server.setConfiguration(confS);
    allServers.add(server);
}
```

If the node supports VPN, these lines of code must be added during the definition. With the first method, the value of the variable *vpnCapabilities* is set to TRUE, while with the second one, it is defined which VPN technology is supported. In this case, only IPSec is supported, but TLS can also be added.

Listing 5.3. Added VPN capability in VEREFOO

```
client.setVpnCapabilities(true);
client.getVpnTechnology().add(SecurityTechnologyType.IPSEC);
```

Untrusted Nodes

Untrusted nodes are devices or systems that cannot be considered trusted or secure for the transmission of data. These nodes are a threat to network security as they can intercept or manipulate data in transit. To protect data transmitted through these nodes, specific security requirements must be set for each node that is considered untrusted. An approach to ensure secure communication is to consider all nodes between the source and destination as untrusted. This strategy, although produces valid secure communication, is inflexible and can only be used only if one does not know the network and its components. Instead, considering a set of specific untrusted nodes increases the space of possible solutions and, in most cases, optimizes the allocation of resources. However, if there are doubts about the reliability of a node, it is safer to consider that node untrusted to guarantee the security of communication.

Considering the network shown in Figure 5.3, the network administrator wants to protect traffic from clients A and B to server C, knowing that along the path there is N4, an untrusted node. The other nodes in the network, including endpoints, can set up a VPN gateway.

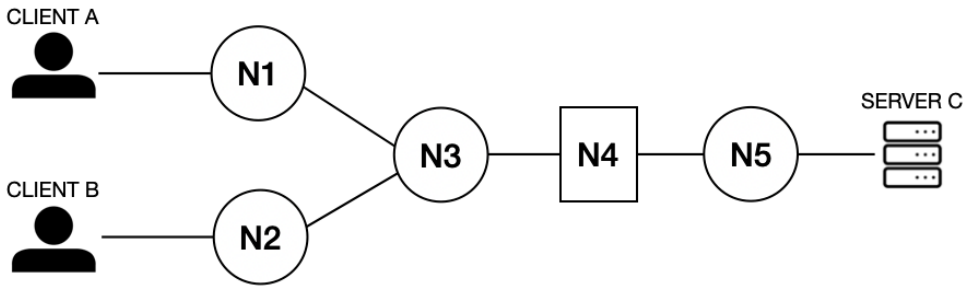

Figure 5.3. Network topology

Without the possibility of specifying a set of untrusted nodes in the requirements, all intermediate nodes must be considered untrusted. Therefore, the only configuration available to satisfy the requirements is the installation of three VPN gateways: the first two in clients A and B to protect the traffic and the third in server C to remove the protection and allow the reception of unencrypted traffic.

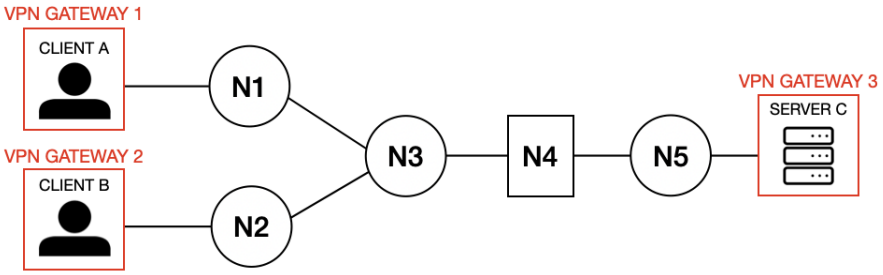

Figure 5.4. Solution 1

However, thanks to VEREFOO's functionality, which allows untrusted node sets to be specified within the security requirements, it is possible to optimize the network configuration and find alternative solutions. In the example of the network described above 5.3, two other possible configurations can be considered.

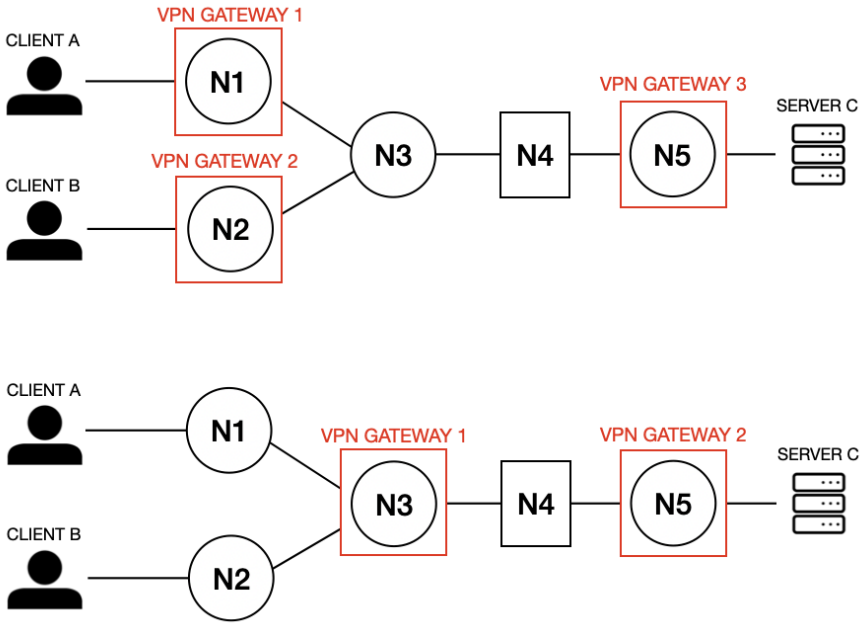

Figure 5.5. Other possible configurations

In configuration 3, a VPN gateway is placed in N3, which protects the traffic, and another VPN gateway in N5, which removes the protection. This provides a correct and optimised solution with less use of allocated resources than the previous configuration.

Untrusted Nodes in VEREFOO

In the VEREFOO framework, the untrusted node is configured as a normal network node, with a *FunctionalType* set to *FORWARDER*. The main difference from other nodes is that, during the generation of security requirements, the set of untrusted nodes in the network is stored. This information is used by the framework to search for solutions with network security functions allocated to protect data transmitted through untrusted nodes.

Listing 5.4. Untrusted node creation in VEREFOO

```
for (int i = 0; i < numberUntrustedNodes; i++) {
    String ipUntrustedNode = createRandomIP();
    Node untrustedNode = new Node();
    untrustedNode.setName(ipUntrustedNode);
    untrustedNode.setFunctionalType(FunctionalTypes.FORWARDER);
    untrustedNodes.add(untrustedNode);
}
```

Inspector Nodes

Inspector nodes, within a network, perform the function of monitoring network traffic for anomalies or suspicious activity. The presence of inspector nodes increases network security, as they quickly identify and detect potential threats. However, to enable analysis, the traffic in these nodes must be unprotected. This means that, if the traffic is encrypted, it must be decrypted and this process may result in some delay in communication.

Considering the network previously illustrated (Figure 5.3) with the addition of inspector node N6, it is possible to identify a configuration that requires a minimum number of VPN gateways configured to protect traffic from clients A and B to server C. A possible solution could be the following:

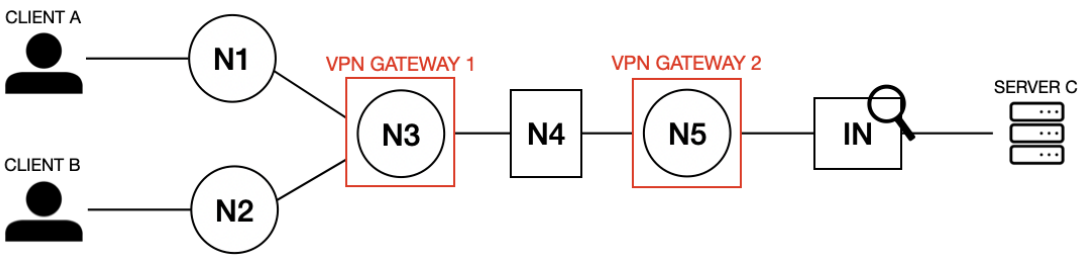

Figure 5.6. Configuration with inspector node IN

In this case, there are no significant delays in communication since the solution found is very similar to the one obtained previously with only the addition of an inspector node before the server.

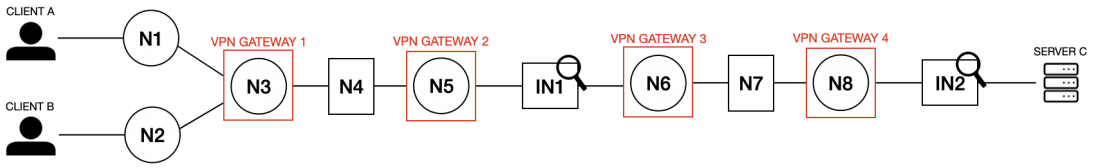

Figure 5.7. Complex configuration with two inspector nodes (IN1, IN2)

However, in the case of a more complex network like the one in Figure 5.7, the presence of an inspector node IN1 between untrusted nodes can cause significant delays. In this case, traffic must be encrypted and decrypted several times during communication. In the configuration shown, the traffic is initially encrypted by VPN Gateway 1 to protect the traffic passing through the untrusted node N4, then it is decrypted by VPN Gateway 2 to be analyzed by inspector node IN1. Next, the traffic is encrypted again by VPN Gateway 3 due to the presence of the untrusted node N7 and finally decrypted by VPN Gateway 4 before being sent to destination C unprotected. This solution leads to a significant increase in allocated resources and a considerable delay in communication.

Inspector Nodes in VEREFOO

In VEREFOO, an inspector node is configured in the same way as an untrusted node, with a *FunctionalType* set to *FORWARDER*. However, the main difference is in the security requirements. In this case, each requirement stores a set of untrusted nodes where traffic must be protected and a set of inspector nodes where protection must be removed. This means that, when configuring a security requirement in VEREFOO, it is possible to specify both sets of untrusted nodes and sets of inspector nodes in that network, allowing optimization of the allocation of security resources. In this way, traffic passing through an inspector node can be analyzed without significant delays in communication, as protection is removed only at the necessary points and only for the time strictly necessary to analyze the traffic.

Listing 5.5. Inspector node creation in VEREFOO

```
String ipInspectorNode = createRandomIP();
inspectorNode.setName(ipInspectorNode);
inspectorNode.setFunctionalType(FunctionalTypes.FORWARDER);
inspectorNodes.add(inspectorNode);
```

Trusted Nodes

In a network, it is possible to identify reliable and secure nodes where traffic protection is at the discretion of the solver. In VEREFOO, these nodes are configured like any other node, with the *FunctionalType* set to *FORWARDER*. Furthermore, when defining security requirements, it is not necessary to specify these nodes,

since each requirement only stores untrusted and inspector nodes. Consequently, all middleboxes that are not considered untrusted or inspectors will be considered trusted by the solver.

Allocation Places

An Allocation Place (AP) is a network node where virtual network security functions can be allocated. It is defined according to various factors, such as the availability of hardware resources, network latency, or required processing capacity. Furthermore, the AP is closely related to network security requirements, as it can be used to prevent a VNF from being allocated on an unprotected or compromised node. This means that the AP ensures that VNFs are only allocated on secure nodes and that traffic protection can be provided effectively, improving the reliability and security of the network.

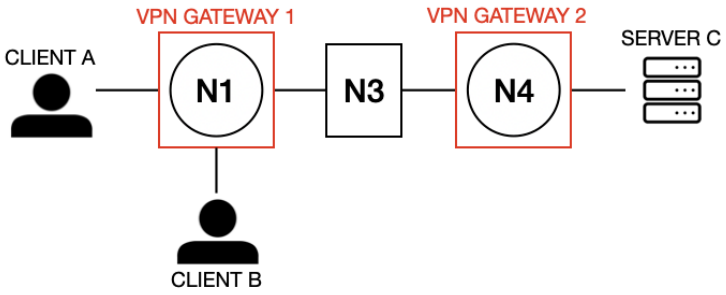

Figure 5.8. Solution 1

Considering a simple network topology such as the one in the figure 5.8, it can be seen that there are three endpoints (A, B, C) and one untrusted node (N3), while the remaining nodes (N1, N4) are APs because they are considered trusted and able to support security functions. In this topology, the administrator defines a security requirement for traffic between client A and destination server C. The only possible solution to fulfill the security requirement is to install a VPN gateway in N1 to protect the traffic and another VPN gateway in N4 to remove the protection.

If a new AP is added between N1 and the untrusted node N3 and client B is connected to it, there is no longer only one possible solution, but two. The first solution is similar to the previous one, with a VPN gateway in N1, while the second is to install the same VPN gateway in the new node N2. In both cases, the solution satisfies the security requirements imposed by the administrator. This example demonstrates that the addition of a single AP node has an impact on the number of potential solutions.

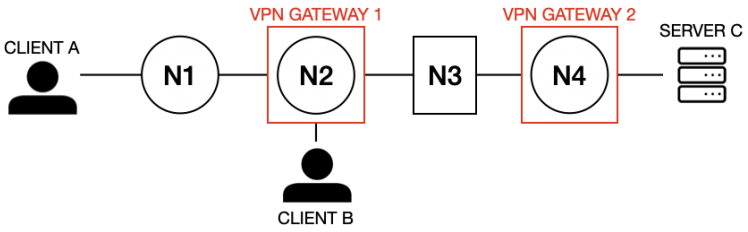

Figure 5.9. Solution 2

Allocation Places in VEREFOO

Within the VEREFOO framework, an AP is configured like all other nodes and is considered by the solver as a trusted node. Thus, during solution processing, the solver will try to include a security function in each AP to obtain an optimal and correct solution according to the security requirements specified by the user. If the number of APs within a framework is increased, the space of possible solutions will also increase.

Listing 5.6. Allocation Place creation in VEREFOO

```
for (int i = 1; i <= numberAllocationPlaces2; i++) {
    String ipAllocationPlace = createRandomIP();
    Node allocationPlace = new Node();
    allocationPlace.setName(ipAllocationPlace);
    allAPs.add(allocationPlace);
}
```

The VEREFOO framework also allows the definition of an endpoint with the ability to configure a VPN Gateway. In this case, the client or the server is managed by the framework like a normal AP. To do this, the user must set the endpoint to support VPN and add the type of security technology, IPSec or TLS.

5.1.2 Security Requirements Model

Network security requirements are one of the two inputs of VEREFOO. These requirements are necessary because the framework relies on a PBM approach to automate network security, in which security managers do not have to manage each network function manually, but can specify the desired behavior through policies that are automatically refined to configure the functions correctly. Although this makes configuring the system easier, it can also make it more vulnerable to errors in policy definition, especially when made by inexperienced users. To solve this problem, it is important to provide an easy-to-use policy definition language and modules for translating rules and detecting conflicts or anomalies in the specified policies.

To improve the usability and effectiveness of policies, two levels of abstraction have been introduced, as explained in paper [34]. The difference between these

levels is the end users to whom they are designated, based on their skills and experience in programming security controls.

The first level is HSLP, which allows security requirements to be expressed in a way similar to natural language, making it very easy to understand and use even without in-depth knowledge of network configuration and security functions. HSLP is designed to be simple, flexible enough to create any type of security policy, and extensible to support different extensions. For example, policies such as 'Block network traffic to and from known malicious IP addresses' or 'Allow network access only to devices that satisfy specified security requirements' can be defined.

MSPL represents the second layer, designed to abstract configuration languages, offering a vendor-independent format and security control. To do this, MSPL requires more detail about the nodes and security functions in the network. MSPL must fulfill the requirements of abstraction, diversity, flexibility, extensibility, and continuity. Abstraction requires that the language contain abstract security configurations that are independent of vendor- or product-specific representation and storage. Diversity implies the capability to include configurations for a wide range of safety functions, without being limited to a single product or vendor. Flexibility and extensibility allow the introduction of new security controls, without the need to change the entire security policy. Finally, continuity is essential to ensure the consistency of the policy chain, from the initial definition in HSPL to the actual implementation of security controls, allowing the policy actually applied and the user associated with it to be tracked.

In this thesis, the focus is on protection policies oriented to define security functions that protect network traffic between two points. The structure of these policies will be presented in the next section.

Communication Protection Network Security Requirements

Each protection policy is characterized by the following elements:

- **Condition Set**: this is the information used to identify the traffic to be protected. Each condition is defined by means of a tuple with the following fields:
  - *IPSrc* is the source of the traffic flow to be protected;
  - *IPDst* is the destination of the traffic flow to be protected;
  - *portSrc* is the source port at the transport layer of the traffic flow to be protected;
  - *portDst* is the transport layer destination port of the traffic flow that needs to be protected;
  - *transportProto* is the transport layer protocol of the traffic flow that is to be protected.

To represent the source and destination IP addresses (*IPSrc* and *IPDst*), the traditional dot-decimal notation is used:

IP1.IP2.IP3.IP4

where IP_i, with i {1,2,3,4}, may be an integer between 0 and 255 (including extremes), or the wildcard character *, representing the entire range [0,255]. If the wildcard is used for IP_i, the other fields to its right must also represent a range instead of a single number. This symbol simplifies the process by combining the declaration of a network address and its corresponding netmask into a single element, eliminating the need for separate declarations. For example, the representation 192.6.10.* may be used to express the end nodes in the 192.6.10.0/24 network, while the representation 30.6.*.* identifies the network 30.6.0/16. When using this notation, VEREFOO does not implement a single security requirement but splits it into several security policies. For example, if the designer wants to protect traffic between sources 10.0.0.1 and 10.0.0.2, he may define a security requirement using the wildcard. From this requirement, VEREFOO creates two separate security policies, one with source 10.0.0.1 and the other with source 10.0.0.2.

The source and destination transport ports (*portSrc* and *portDst*) can be expressed with a single number or a range of numbers, considering a range from 0 to 65535. These fields provide greater flexibility. For example, if you want to protect traffic from a source to a destination, but only through a specific port, you can specify that port. Alternatively, you can use the wildcard to protect traffic passing through any port.

Finally, the *transportProto* field represents the layer 4 protocol used over the IP layer and can have TCP or UDP as possible values, or the wildcard character *.

- **Security properties**: these are the security properties that must be applied to the traffic to guarantee the security of communications. Each property is specified with a security technology to fulfill the security requirement, an authentication algorithm to guarantee integrity and authentication, and an encryption algorithm for information confidentiality. In case these values are not specified in the security requirement, VEREFOO uses the default values AES_128_CBC for encryption and SHA_2_256 as the authentication algorithm.

- **Set of middleboxes**: represents the set of nodes in the network that are considered untrusted or are inspector nodes. As described in the previous section, in the case of untrusted nodes, protection must be applied to the traffic, while for inspector nodes, the traffic must be unencrypted so that it can be analyzed.

- **Rule types**: indicates the type of policy required to ensure secure communication. For secure communication, protection policies are required to guarantee the security of the information exchanged.

Protection Requirements in VEREFOO

In VEREFOO, each security requirement is generated by the following function:

createPolicy(PName type, NFV nfv, Graph graph, String IPClient, String IPServer)

- **PName type**: indicates the type of security requirement that will be defined. In this case, to create a protection requirement, the value of the variable must be *PName.PROTECTION _PROPERTY*.
- **NFV nfv**: is an object that contains all information, including the generated structure, connections between nodes, defined requirements, and so on.
- **Graph graph**: is the network structure generated with an ID and a list of the nodes present.
- **String IPClient**: indicates the IP address of the source node.
- **String IPServer**: indicates the IP address of the destination node.

Listing 5.7. Network Protection Policies node definition in VEREFOO

```
private void createPolicy(PName type, NFV nfv, Graph graph, String IPClient,
    String IPServer) {

    Property property = new Property();
    property.setName(type);
    property.setGraph((long) 0);
    property.setSrc(IPClient);
    property.setDst(IPServer);
    ProtectionInfoType protectionInfoType = new ProtectionInfoType();

    for (Node un : untrustedNodes) {
        NodeRefType nrt = new NodeRefType();
        nrt.setNode(un.getName());
        protectionInfoType.getUntrustedNode().add(nrt);
    }

    for (Node in : inspectorNodes) {
        NodeRefType nrt = new NodeRefType();
        nrt.setNode(in.getName());
        protectionInfoType.getInspectorNode().add(nrt);
    }
    property.setProtectionInfo(protectionInfoType);
    nfv.getPropertyDefinition().getProperty().add(property);
}
```

5.2 Network Generator

Part of this thesis focused on the implementation of the network generator used to evaluate the performance of VEREFOO. The reason behind this implementation is to create a network as realistic and complex as possible, able to represent a common situation. In particular, the presence of several branches is considered between nodes, where traffic flows overlap and have different lengths and directions. The complexity of the generated network represents an important challenge for performance evaluation and for testing VEREFOO's ability to allocate and configure the security functions in a fast and efficient way, which is necessary to satisfy the security requirements of the input. Furthermore, the simulation of complex scenarios, in which traffic flows overlap and extend over several ramifications, allows the effectiveness and efficiency of VEREFOO to be tested under realistic conditions.

In the course of the chapter, it will be examined how the generator creates a realistic and complex network, both with the presence of an inspector node and without it. It will also be explained how security requirements are defined to guarantee secure communication between the various points in the network. Finally, concrete examples will be provided to clarify the concepts presented above and make it easier to understand the network generator's behavior.

5.2.1 Input parameters

The network generator is designed to create a network with various types of nodes, including clients, servers, untrusted nodes that need traffic protection, allocation places (APs) that are trusted nodes where security functions can be allocated, and inspector nodes that monitor network traffic. In this generator, it is possible to manually set the number of clients and the number of servers that must be present within the network. The other parameters, such as the number of trusted and untrusted nodes and the node disposition, are computed automatically according to the design rules. In this way, the configuration process is simpler and better adapted to the user's needs. As far as inspector nodes are concerned, there are two versions of network topology available. One version includes a single inspector node, which is only placed in the topology if the number of servers defined as input is greater than two.

After manually defining the number of clients and servers in the network topology, three security policies are created for each client and server. These security policies define the rules to protect the network traffic from each node. Each policy has the client or server itself as the source and a different destination. This configuration makes it possible to handle traffic flows of various lengths and directions, making the network topology as realistic as possible.

5.2.2 Network structure

The basic structure consists of three parts: the "Basic structure - Client", the "Basic structure - Server" and a third element consisting of two clients connected to a single AP, that is the central AP to which the two previous parts are connected.

Basic structure - Client

This basic structure can have from 1 to 6 clients connected to APs, followed by a single untrusted node (UN). Each client can be connected to a single AP and each AP can be connected to a maximum of 2 clients. In addition, the untrusted node can be connected to a maximum of 3 APs to where clients are connected. This layout makes it possible to create a structure that can be described as complete. However, if the specified parameters require more nodes than the complete structure, new structures will be created until all input clients have been allocated. A complete structure is shown on the left and an incomplete structure on the right.

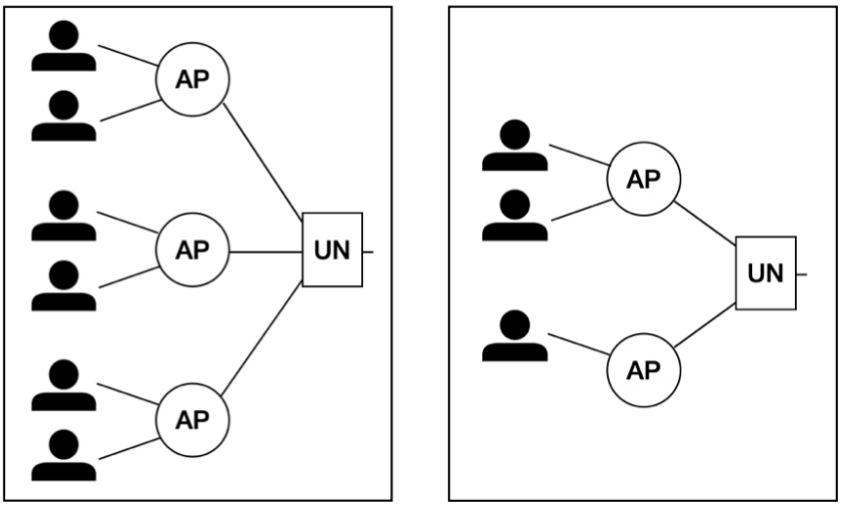

Figure 5.10. "Basic structure - Client" complete and not complete

Basic structure - Server

The "Basic structure - Server" is similar to the previous one. In this case, there are both client and server nodes. The structure consists of a single server connected to an AP followed by an untrusted node. In this structure, the untrusted node is connected to a maximum of 2 APs, while each AP can be connected to a server, a client and an untrusted node. Once the servers are allocated, a client is connected to one of the two APs in the structure. Again, when a complete structure is reached, new structures will be created. In Figure 5.11, the complete structure is on the left and the incomplete structure on the right.

Central AP

The third part is an AP to which only two clients are always connected. Once the structures of the 3 parts have been defined, the single untrusted nodes in the "Basic structure - Client" and the "Basic structure - Server" are connected to this central AP. At the end of this process, the basic structure shown in the figure below is generated. The parameters in this case are 7 clients and 2 servers set by the user.

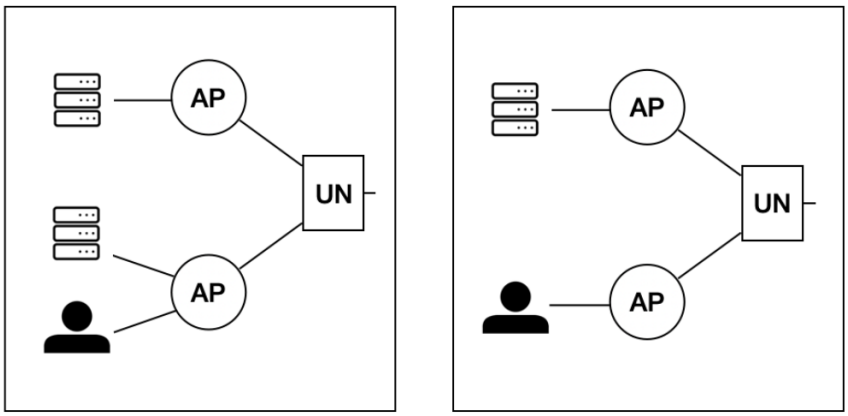

Figure 5.11. "Basic structure - Server" complete and not complete

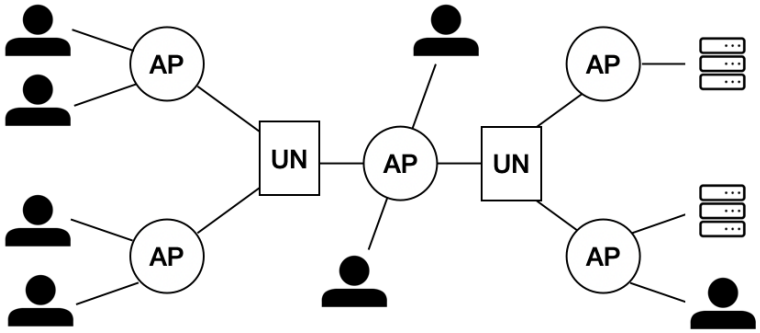

Figure 5.12. Basic structures connected to "Central AP"

In Figure 5.12, only a "Basic structure - Client" and a "Basic structure - Server" are represented. However, by increasing the input parameters, the number of basic structures will increase. In this case, not all structures can be connected to the central AP, since with a large number of clients and servers, the final network would consist of N structures all connected to a single AP. This could be critical for two reasons: VEREFOO could allocate a single VPN gateway in that central AP with all the rules configured and a single point of failure would be created, a solution that is rarely adopted in real networks.

For this reason, when the network topology grows, the implemented generator connects the various base structures using a different logic. If there are several "Basic structure - Client", the first structure is connected to the central AP as seen in Figure 5.12. Then, the second "Basic structure - Client" is connected to the untrusted node in the first server structure, the third client base structure is connected to the untrusted node in the second server structure, and so on. If there are more "Basic structure - Client" than the number of untrusted nodes in the "Basic structure - Server", the "Basic structure - Client" begin to connect to the APs of the "Basic structure - Server". In this case, connecting all structures to a single node is avoided and a very branched network is generated.

Once all "Basic structure - Client" have been allocated, the same procedure is followed for the "Basic structure - Server", but using the nodes of the "Basic structure - Client". The first two server base structures are connected to the central AP, while the others are first connected to the untrusted nodes and then to the APs of the client structure.

Some examples will be presented at the end of this section.

5.2.3 Network structure with inspector node

If the user decides to add an inspector node to the network, the network generation process follows a similar process to that described previously. However, there are some differences when connecting the second "Basic structure - Server". Instead of connecting the first two "Basic structure - Server" to the central AP, the second structure is connected to an AP, followed by an inspector node. After that, the inspector node is connected to the central AP. The process then continues normally, connecting the server base structures, if present, first to the untrusted nodes and then to the APs of the "Basic structure - Client". In this way, compared to the previous version, the resulting topology will have an additional inspector node and an additional AP node.

An example of this configuration is shown in the figure below:

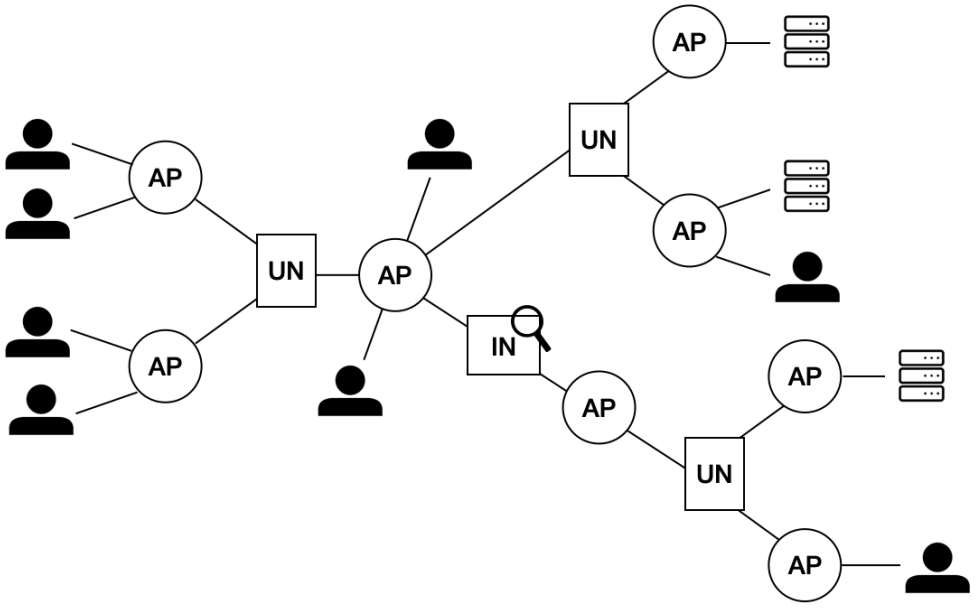

Figure 5.13. Generated structure with inspector node IN

5.2.4 Generation of Security Requirements

After defining the network topology and connecting the nodes, the security policies are defined. As mentioned earlier, three security policies are created for each client and server. Each policy has the client or server itself as its source and a different destination. The policies are generated according to the type of source node. For clients in the "Basic structure - Client", the destination of the policy will be one of the servers in the network topology. It is important to note that the destination changes between different policies created for the same client. For the nodes in the "Basic structure - Server", each node will have as its destination one of the clients present in the "Basic structure - Client" or one of the clients connected to the central AP. This configuration allows traffic from the servers to be protected by redirecting it to different clients. Finally, for clients connected to the central AP, the policy will have as its destination one of the nodes of the base structures. This configuration ensures that the policies defined are of various types, adapting to the complexity of the network generated and allowing traffic flows of various lengths and directions to be correctly protected.

5.2.5 Illustrative Examples

A complete examples are now presented. The first two examples show how the network topology grows as the input parameters increase, while the third one explains how the security requirements are defined.

Use case 1: Network Structure

Input parameters specified in the generator:

- Number of Clients: 24
- Number of Servers: 4
- Number of Inspector Nodes: 0

Parameters automatically computed based on input:

- Number of APs: 15
- Number of Untrusted Nodes: 6

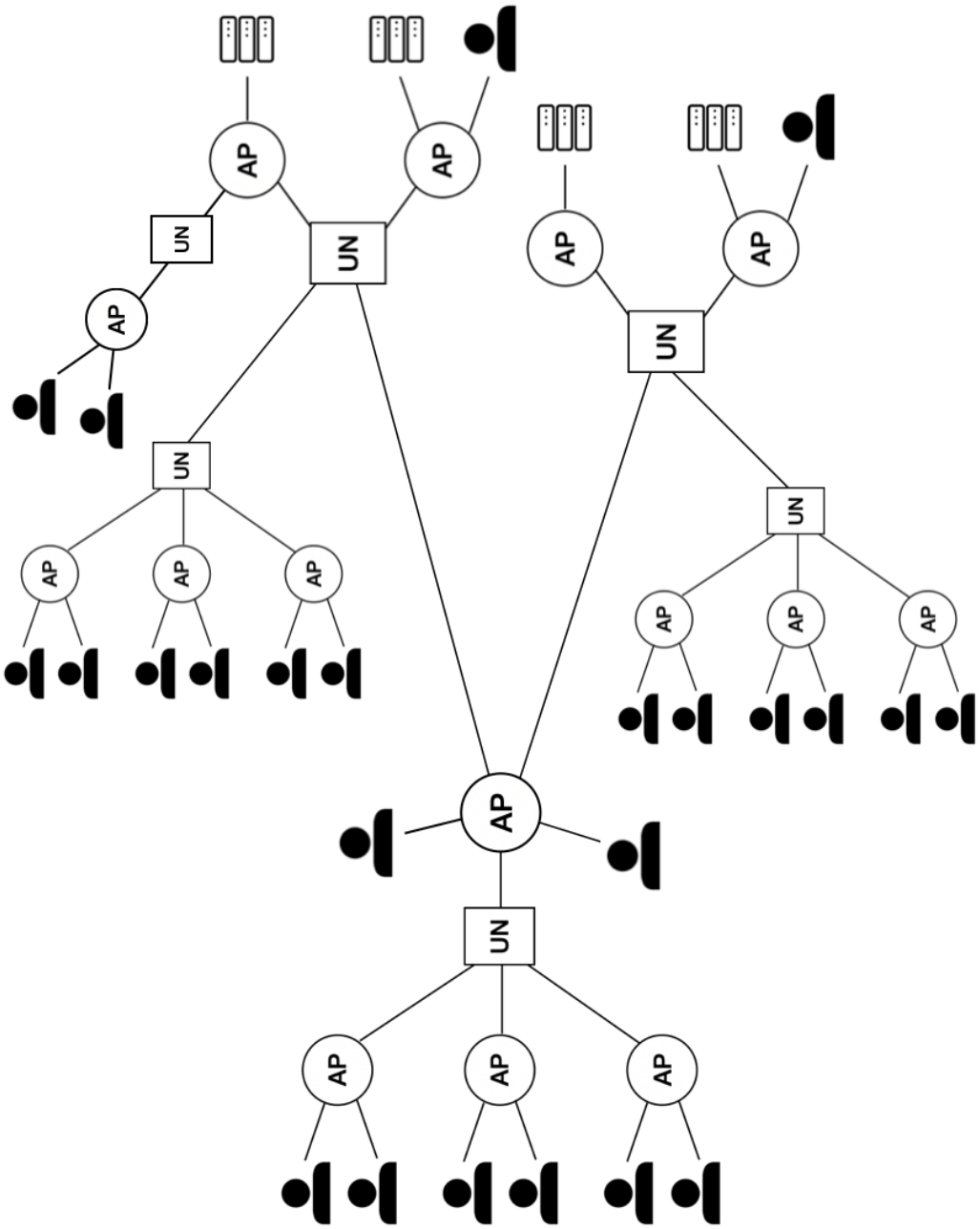

Figure 5.14. Use case 1

Use Case 2: Network Structure with Inspector node

Input parameters specified in the generator:

- Number of Clients: 36
- Number of Servers: 5
- Number of Inspector Nodes: 1

Parameters automatically computed based on input:

- Number of APs: 24
- Number of Untrusted Nodes: 9

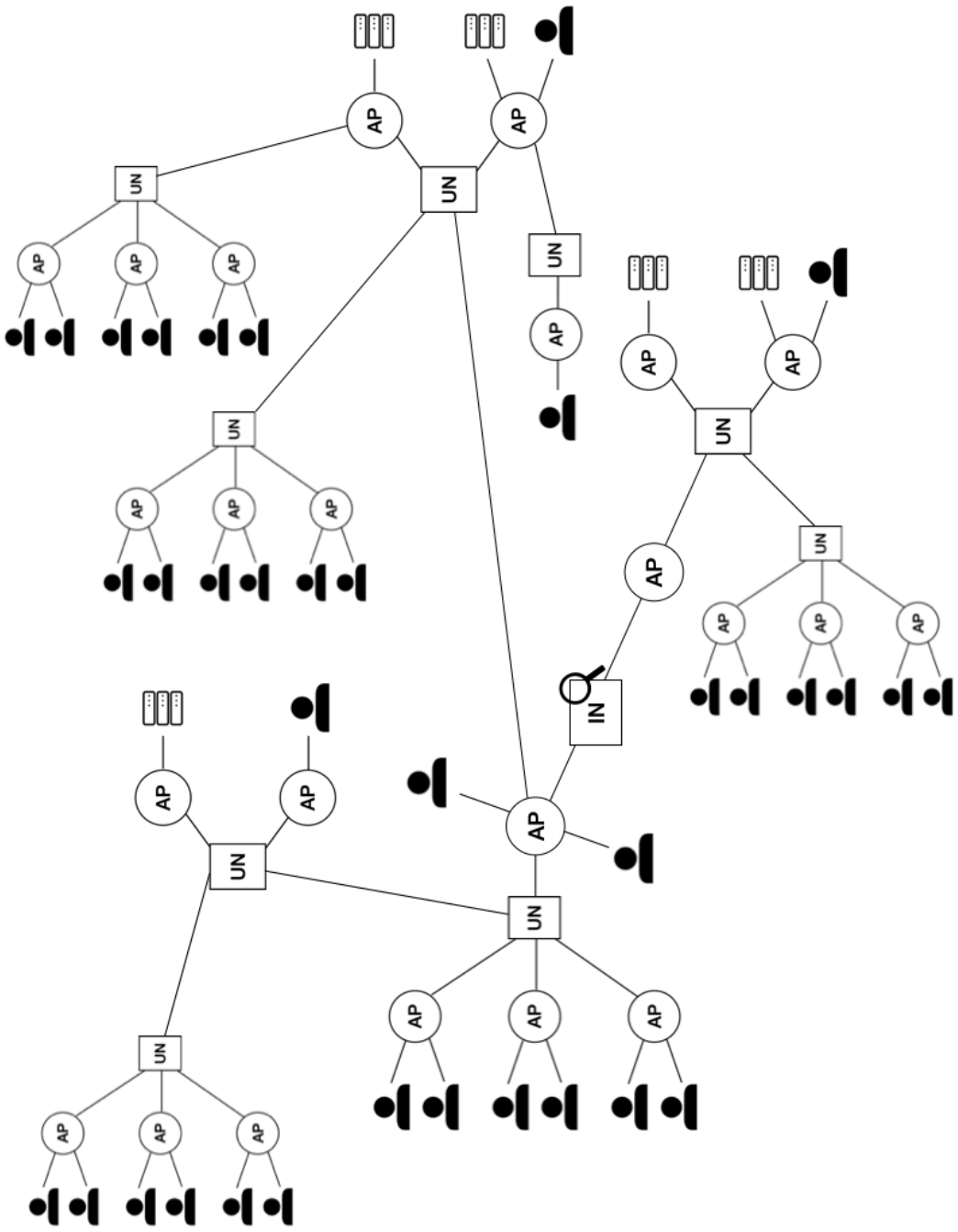

Figure 5.15. Use case 2

Use Case 3: Policies Definition

Input parameters specified in the generator:

- Number of Clients: 6
- Number of Server: 3
- Number of Inspector Nodes: 0

Parameters automatically computed based on input:

- Number of APs: 6
- Number of Untrusted Nodes: 3
- Number of Policies: 27

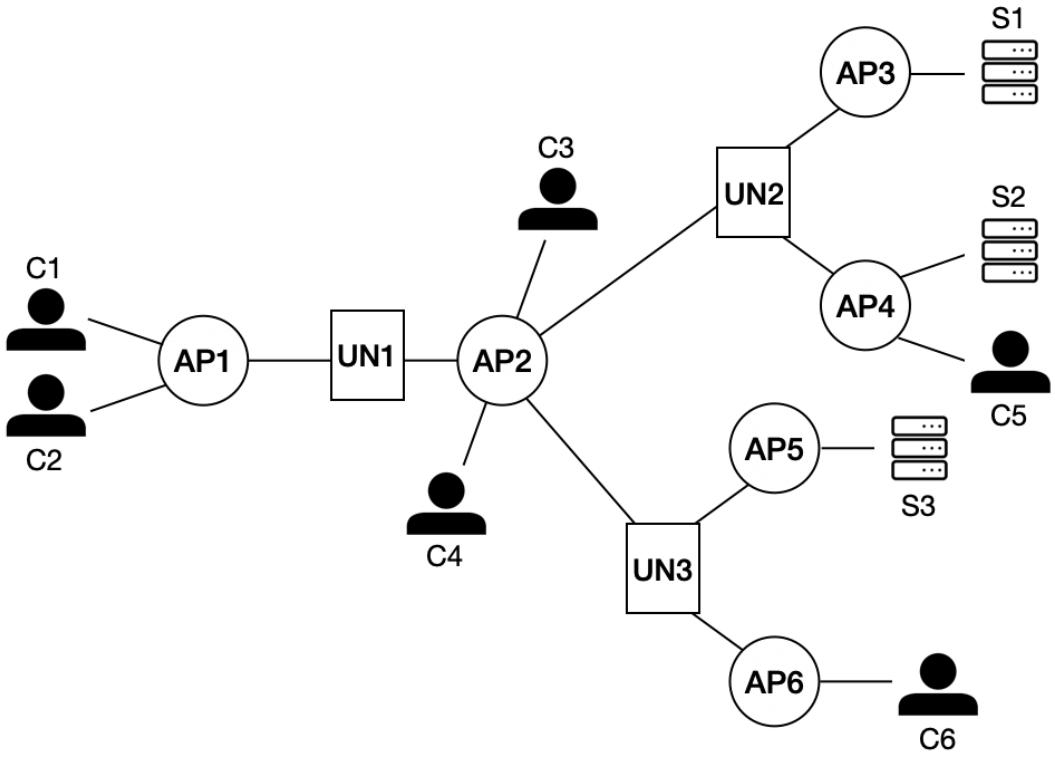

Figure 5.16. Network topology generated

Figure 5.16 illustrates the generated network topology composed of 9 endpoints. Since each endpoint requires 3 different security requirements, the total number of requirements that need to be defined is 27. The table below lists the source and destination nodes that will be used as IP source and IP destination, respectively, for each security requirement.

SOURCE NODE	**DESTINATION NODE**
C1	S1
C1	S2
C1	S3
C2	S1
C2	S2
C2	S3
C3	C1
C3	S1
C3	C2
C4	S2
C4	C1
C4	S3
C5	C1
C5	C2
C5	C3
C6	C4
C6	C1
C6	C2
S1	C1
S1	C2
S1	C3
S2	C4
S2	C1
S2	C2
S3	C3
S3	C4
S3	C1

5.3 Integration with VEREFOO

The network generator described previously has been designed to integrate with the VEREFOO framework in order to provide a realistic and complex network for performance evaluation tests. The integration of the generator with VEREFOO is done as follows:

1. Definition of input parameters. Before using the network generator, input parameters such as the number of clients and servers must be specified. These parameters allow the user to personalize the network configuration. It is also possible to manually set the version with or without an inspector node.

2. Calling the network generator. Once the input parameters have been defined, the network generator is called within the VEREFOO framework. It performs its function of generating the corresponding network topology with the various types and branches between nodes.

3. Definition of protection requirements. After the network topology has been generated, the generator defines the corresponding security policies for each client and server.

4. Choice of security functions. Once the security policies have been defined, the VEREFOO framework allocates and configures the security functions required to comply with these policies. This operation is performed by the solver, which computes the optimal solution by solving the MaxSMT problem.

5. Performing performance evaluation tests. With the network generated and the security functions allocated, performance evaluation tests can be started. These tests allow the effectiveness and efficiency of VEREFOO in managing complex networks and providing secure communication to be evaluated.

Through the integration of the network generator in the framework, it is possible to simulate realistic and complex scenarios, define suitable security policies, and evaluate the overall performance of the system. This integration is a key step in ensuring the effective implementation of VEREFOO and providing a reliable test environment to evaluate its functionality.

Chapter 6

Implementation and Testing

After analyzing the network generator, this chapter presents the results of tests performed to evaluate the performance of the VEREFOO framework under specific conditions. In these cases, the framework was used to configure VPNs with IPSec to satisfy the input requirements. For each test, the considered network structure is described and the results obtained with the two different models presented in the background are reported: Atomic Flows (section 3.3.1) and Maximal Flows (section 3.3.2). In this way, based on the results, performance can be evaluated to determine which model is most advantageous to use under the same conditions.

The aim is to provide an evaluation of VEREFOO's ability to automatically establish secure communications between nodes in a network, to find potential future improvements to be applied to the framework.

6.1 Test Description

The experimental validation of the framework was conducted to analyze its scalability and identify which parameters have the greatest impact on performance. All tests were performed with a 4-core Intel i7-6700 3.40 GHz system with 32 GB of RAM, and for each input, the test was repeated 50 times, to obtain an average performance evaluation when specific parameters changed. Performance was evaluated according to the computation time it takes VEREFOO to find the optimal solution that satisfies the security requirements defined by the user input. This means the final execution time includes the time it takes to generate the network topology and security requirements according to the user input values, properly allocate and configure the virtual network security functions, and finally, search for the optimal solution.

The metrics of most interest for these tests are the following:

- the number of input security requirements, as the solution computed by the framework must satisfy each defined requirement;
- the number of AP in the network, which has an impact on the possible solution space, since each AP is a possible candidate for the allocation of the security function.

In addition to these metrics of major interest, the values of each parameter used within the generator were also shown for each test. The parameters reported are:

- Number of Endpoints: indicates the total number of clients and servers in the network, where each node has a random IP address and the attribute *VPNcapabilities* set to FALSE. This means that the node cannot configure a VPN gateway and is not considered an allocation place.
- VPNcap Endpoint: represents the value that the *VPNcapabilities* attribute assumes in each client and server during definition, which refers to all endpoints present within the network. If it is set to FALSE, both client and server nodes in the network are unable to support VPN and are not considered as APs.
- Number of Network Security Policies: indicates the number of security requirements defined within the network. For each endpoint, three different security requirements are defined.
- Number of APs: represents the number of nodes in the network where a security function can be allocated. In the value, clients and servers supporting VPN are also counted when the *VPNcapabilities* attribute is set to TRUE.
- Number of Inspector Nodes: indicates the number of inspector nodes presents in the network where traffic must be unencrypted, without protection. The set of inspector nodes is stored within each security requirement that is defined, which is why the value is marked as "Number of Inspector Nodes per Policy".
- Number of Untrusted Nodes: is the number of untrusted nodes in the network where traffic requires protection. The set of untrusted nodes is stored within each security requirement that is defined, which is why the value is marked as "Number of Untrusted Nodes per Policy".
- Computation Time: this is the final average execution time computed as the mean of 50 iterations with the same parameters. It includes the time needed to generate the network topology and security requirements based on user input values, allocate and properly configure the virtual network security functions, and finally determine the optimal solution that satisfies the input requirements. The value is expressed in seconds (sec).

The tests have all these characteristics in common. However, it should be noted that specific modifications have been applied to each model, concerning certain parameters and constraints. The differences are as follows.

Atomic Flows tests

Using the Atomic Flows model, not only the average computation time was recorded, but also the times relating to:

- Atomic Predicates Time: is the total time required to compute all atomic predicates related to the defined security requirements;

- Atomic Flows Time: is the total time needed to compute the Atomic Flows related to the predicates;
- MaxSMT Time: is the total time taken by the solver to find the correct and optimal solution;

This made it possible to analyze in detail which part has the greatest impact on the final computation time, which is the parameter of major interest.

Maximal Flows tests

The tests with Maximal Flows were performed without the soft constraints relating to the aggregation of the rule condition fields. Initially, there were four soft constraints, each of which verified that the byte of the address was equal to the wildcard. However, with these constraints, the performance of the framework significantly decreased, making it impossible to obtain good results. Furthermore, the aggregation of rules is also not supported by the algorithm based on Atomic Flows, so the conditions under which the models were tested are also comparable in this aspect, making it possible to remove these constraints.

6.2 Test 1: Basic Structure

The first test is based on analyzing the performance of VEREFOO when clients, servers, untrusted nodes, and APs are present within the network topology. In this configuration, therefore, no inspector nodes were considered. This means, that each security requirement was defined by storing only the set of untrusted nodes present in the network. In this way, the only requirement the framework has to fulfill is to protect the traffic when it crosses within these untrusted nodes. For the tests of both the Atomic Flows and Maximal Flows models, initial low values were used for the parameters, and each time the number of clients and servers was increased, so as all other related values. In this case, no parameters were left fixed, except for the number of inspector nodes in the network, which was kept at zero. The parameters were increased to a reasonable value for which it was interesting to keep the record.

The structure considered is the one explained in section 5.2.2.

6.2.1 Atomic Flow Results

With the Atomic Flows model, the obtained results are presented in Figure 6.1, which displays the values of all parameters for each execution. The last columns show the average computation times for Atomic Predicates, Atomic Flows, solving the MaxSMT problem and the overall average execution time for those inputs. After gathering the data in a table, the trend of the average computation time was represented in Graph 6.2. The Y-axis shows the computation time in seconds, while the X-axis represents the number of defined security policies.

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Atomic Predicates Time (sec)	Atomic Flows Time (sec)	MaxSMT Time (sec)	Computation Time (sec)
10	False	30	6	0	3	**0,104**	**0,018**	**0,424**	**0,553**
15	False	45	9	0	4	**0,327**	**0,030**	**1,580**	**1,943**
20	False	60	11	0	4	**0,606**	**0,051**	**3,690**	**4,355**
28	False	84	15	0	6	**1,298**	**0,117**	**15,127**	**16,550**
33	False	99	18	0	7	**1,940**	**0,188**	**51,895**	**54,033**
38	False	114	20	0	7	**2,596**	**0,239**	**113,293**	**116,139**
44	False	132	23	0	8	**3,578**	**0,328**	**641,218**	**645,141**
48	False	144	25	0	9	**4,318**	**0,415**	**1544,869**	**1591,164**

Figure 6.1. Test 1: Table of Atomic Flows Results

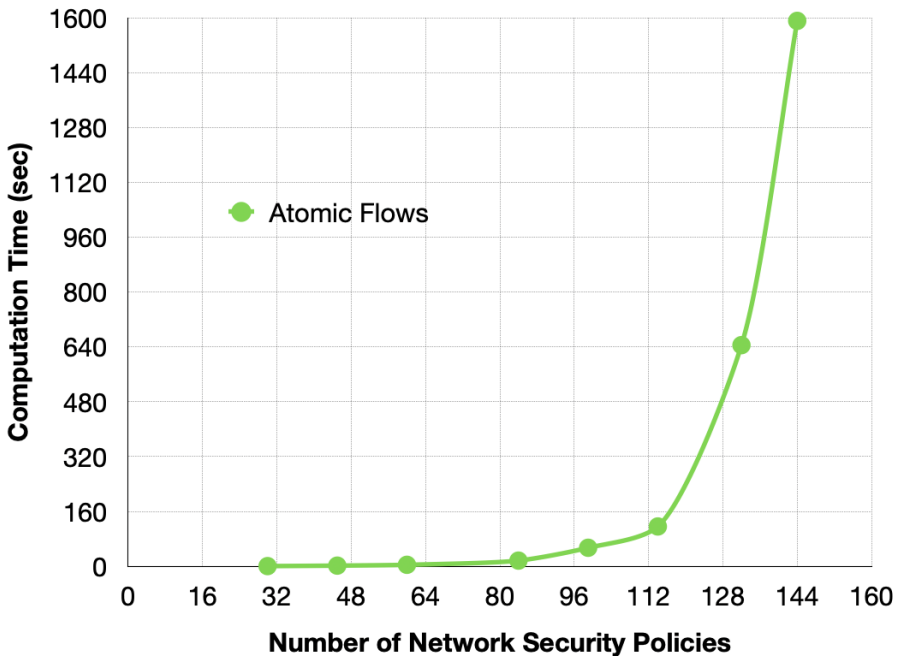

Figure 6.2. Test 1: Graph of Atomic Flows Results

6.2.2 Maximal Flow Results

The Figure 6.3 shows the results with the Maximal Flows model. The trend in execution time in relation to the number of defined security policies is depicted in Graph 6.4.

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Computation Time (sec)
10	False	30	6	0	3	**0,645**
15	False	45	9	0	4	**1,445**
20	False	60	11	0	4	**2,554**
28	False	84	15	0	6	**4,450**
33	False	99	18	0	7	**6,767**
38	False	114	20	0	7	**9,262**
48	False	144	25	0	9	**16,450**
58	False	174	32	0	11	**28,226**
78	False	234	40	0	14	**49,064**
88	False	264	45	0	16	**73,325**
118	False	354	60	0	21	**148,516**
142	False	426	72	0	25	**256,143**
248	False	744	125	0	42	**1229,586**

Figure 6.3. Test 1: Table of Maximal Flows Results

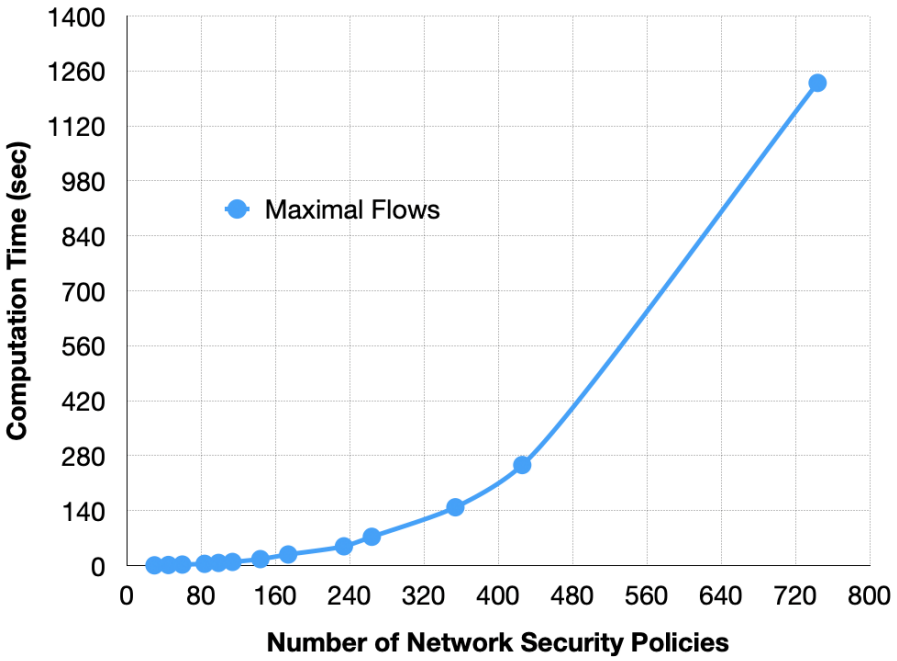

Figure 6.4. Test 1: Graph of Maximal Flows Results

6.2.3 Interpretation of Test Results

Based on the analysis of graphs 6.2 and 6.4, a similar behavior can be observed between the two models, Atomic Flows and Maximal Flows, in terms of execution

time in relation to the number of defined security policies.

Initially, both models show an almost quadratic increase in execution time, but once a certain threshold is reached, performance begins to deteriorate exponentially. For example, in the Atomic Flows model, the execution time doubles when the number of security policies increases from 99 to 114, while it quintuples from 114 to 132. A similar trend is also observed in the Maximal Flows model.

The factor that has the greatest impact on performance is the time taken by the solver for the MaxSMT problem. Whenever a security policy is added, additional soft constraints are generated with a deterioration in overall performance. This can be observed from Figure 6.1 of the Atomic Flows model, where it can be seen that the final execution time is mainly influenced by the time spent by the MaxSMT, which shows exponential growth, while the time needed to calculate the flows and atomic predicates increases, but less markedly.

This impact of the time taken by the solver for the MaxSMT problem on the execution time can also be seen from Graph 6.5 in which all the time trends in the Atomic Flows model are shown. For better understanding, the Y-axis uses a logarithmic scale.

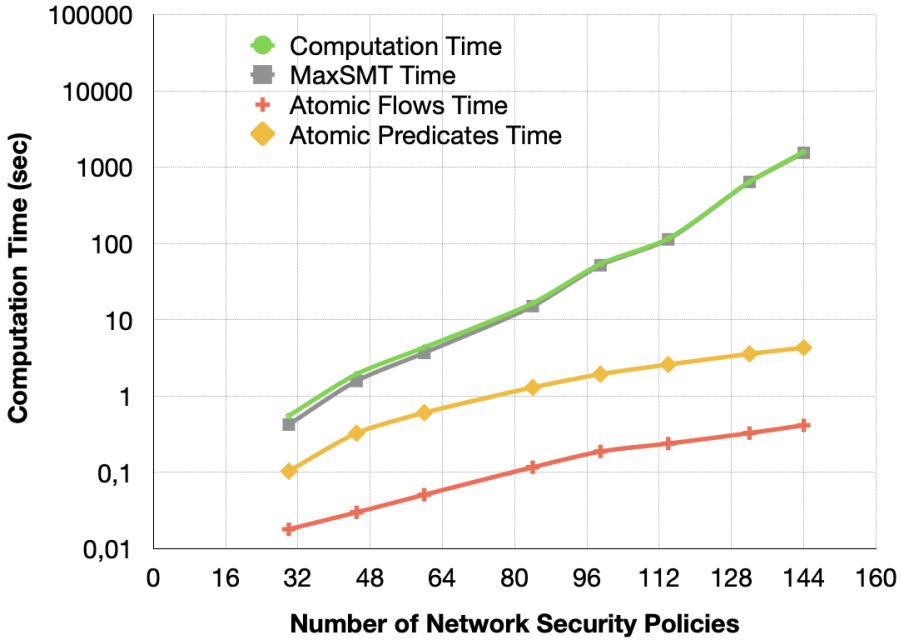

Figure 6.5. Time trends in the Atomic Flows model

For comparing the performance of the two models, Atomic Flows and Maximal Flows, Graph 6.6 was created to identify any differences. To provide a better and more complete view, results obtained with less number of policies than in the previous tests, not shown in Figure 6.1 and 6.3, were also included. Even in this case, the Y-axis uses a logarithmic scale.

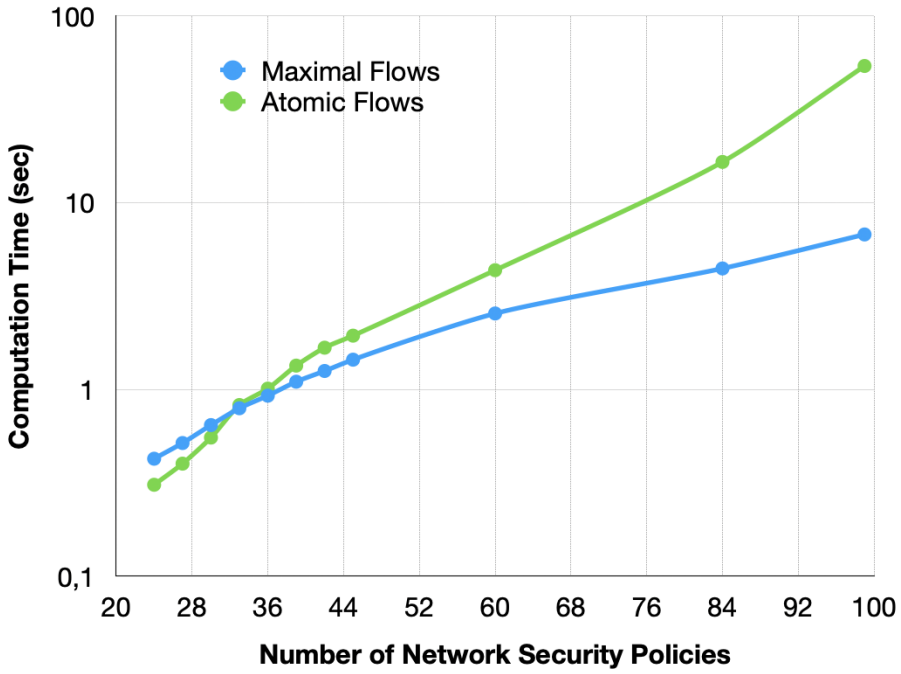

Figure 6.6. Test 1: Comparison of the Atomic Flows and Maximal Flows model

From the analysis of Graph 6.6, it is possible to determine which model offers better performance. With a limited number of policies defined in the network, the Atomic Flows model is more advantageous. However, as the number of policies increases, performance deteriorates and in the long term, the Maximal Flows model becomes the more time-effective choice. These results can be explained by two main factors: the number of constraints generated in the two models and their resource management capability.

Concerning resource management, the Atomic Flows model has an advantage when the number of resources is limited. This initial gain can be attributed to the small number of initial Atomic Flows and the fact that these flows are identified with integer values, which are easier to handle by the solver in solving the MaxSMT problem. With a limited number of resources, the time for computing predicates and atomic flows is almost negligible. On the contrary, the Maximal Flows model takes more time with limited resources because each maximal flow is no longer identified by integer values, but rather by more complex representations that reduce performance. However, when the number of security policies increases, the comparison between the two models is no longer so favorable for the Atomic Flows model. The Maximal Flows model shows better performance when more resources are used, which is not the case for the Atomic Flows model. This deterioration in performance can be attributed to the large number of Atomic Flows that are generated by increasing the defined security policies. Although each flow is identified by an integer value, which is easier for the solver to handle, a large number of flows it has to manage cancels out the initial advantage, thus favoring the Maximal Flows model in terms of performance.

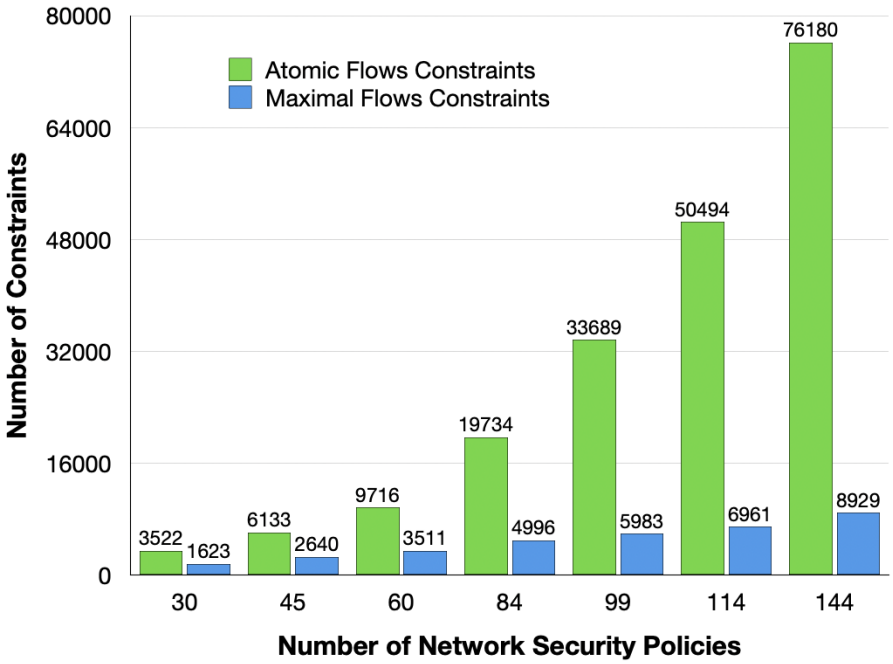

Figure 6.7. Test 1: Generation of constraints in the Atomic Flows and Maximal Flows model

The other main factor is the number of constraints generated and passed to the solver in the two models. An increase in the number of constraints affects the resolution of the MaxSMT problem, which, as pointed out, has a strong impact on the final execution time (approximately 90% in each run). For this reason, it is important to consider the number of constraints generated in the two models. This difference is evident in Graph 6.7, where it can be observed that the Atomic Flows model generates a much larger number of constraints than the Maximal Flows model. This inevitably leads to a deterioration in performance for the Atomic Flows model when the resources increase, and consequently also the constraints that the solver has to handle.

6.3 Test 2: Structure with Inspector Node

The second test is different from the first due to the presence of an inspector node within the network topology. In this case, the test was performed to evaluate how performance decreases when traffic must be encrypted and decrypted several times during communication. The inspector node is placed between two untrusted nodes, so protection must be removed so that the inspector node can analyze the traffic and then re-established to protect this traffic in the untrusted node that is next. As with the first test, parameter values were increased to evaluate the impact on computation time. The structure considered is the one explained in section 5.2.3.

6.3.1 Atomic Flow Results

Similar to the previous test, the last columns of Figure 6.8 show the average computation times, while Graph 6.9 illustrates the trend of the execution time in relation to the number of defined policies.

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Atomic Predicates Time (sec)	Atomic Flows Time (sec)	MaxSMT Time (sec)	Computation Time (sec)
10	False	30	7	1	3	**0,156**	**0,007**	**0,406**	**0,575**
15	False	45	10	1	4	**0,551**	**0,021**	**0,888**	**1,441**
20	False	60	12	1	4	**0,933**	**0,037**	**4,344**	**5,319**
23	False	69	14	1	5	**1,326**	**0,052**	**6,155**	**7,539**
28	False	84	16	1	6	**2,071**	**0,085**	**64,870**	**67,035**
29	False	87	17	1	6	**2,341**	**0,125**	**89,339**	**91,823**
31	False	93	18	1	6	**2,792**	**0,133**	**316,041**	**318,983**
33	False	99	19	1	7	**2,941**	**0,138**	**1289,633**	**1312,217**

Figure 6.8. Test 2: Table of Atomic Flows Results

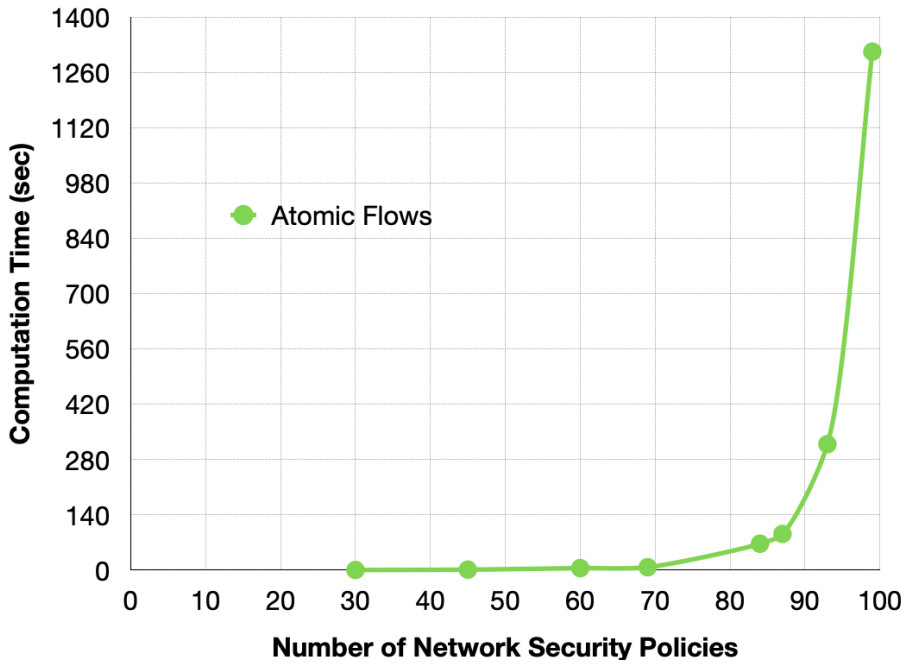

Figure 6.9. Test 2: Graph of Atomic Flows Results

6.3.2 Maximal Flow Results

Figure 6.10 shows the results, while the trend of computation time in relation to the number of defined security policies is depicted in Graph 6.11.

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Computation Time (sec)
10	False	30	7	1	3	**0,811**
15	False	45	10	1	4	**1,866**
20	False	60	12	1	4	**3,082**
28	False	84	16	1	6	**5,853**
33	False	99	19	1	7	**8,893**
58	False	174	31	1	11	**34,811**
78	False	234	41	1	14	**69,081**
108	False	324	56	1	19	**155,865**
208	False	624	106	1	36	**787,545**

Figure 6.10. Test 2: Table of Maximal Flows Results

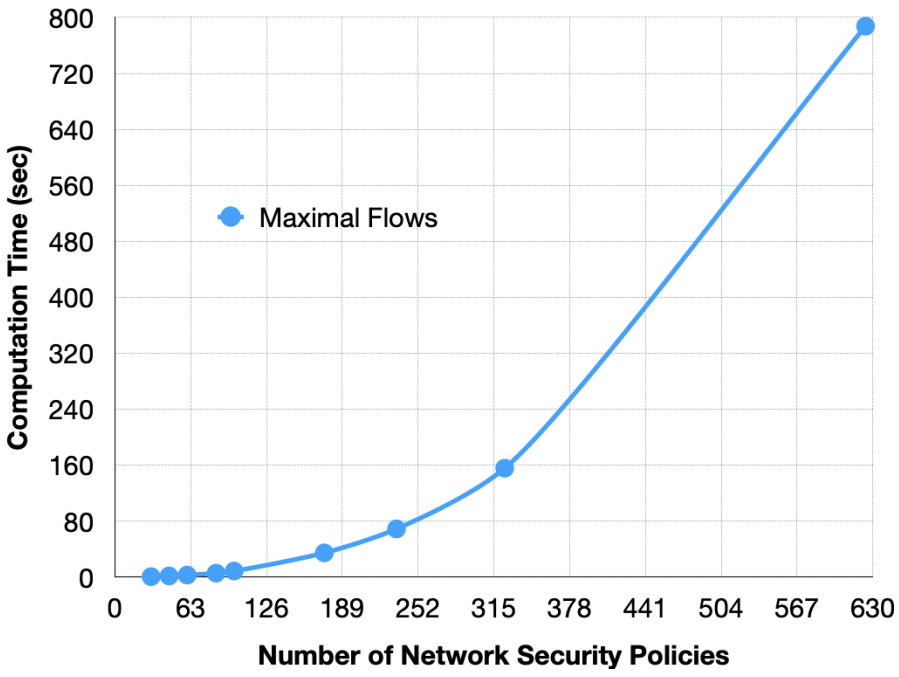

Figure 6.11. Test 2: Graph of Maximal Flows Results

6.3.3 Interpretation of Test Results

The results of this test showed a similar trend to that observed in Test 1 for both Atomic Flows and Maximal Flows models. The Atomic Flows model maintains an initial advantage with a limited number of resources, while the Maximal Flows model proves to be more advantageous in the long run. The reasons for this behavior, therefore, remain the same.

However, in this test, an inspector node was introduced into the network structure, which influenced the overall performance of both models. The performance deterioration is attributed to the inspector node, which is responsible for analyzing the traffic passing through it without protection. This led to a deterioration caused by two possible factors. Firstly, the number of constraints generated and passed to the solver increases, since those relating to the inspector node must be added. Secondly, the number of allocated and configured VPN gateways within the network also increases. This is due to the fact that the inspector node is located between two untrusted nodes and therefore traffic protection must be added and removed several times.

In Figure 6.12 and Figure 6.13, two graphs are presented. The first one shows the trend of computation time with the Atomic Flows model when the network structure does or does not have an inspector node. On the other hand, in the second graph the same trend is displayed but with the Maximal Flows model. To highlight the difference, the Y-axis uses a logarithmic scale.

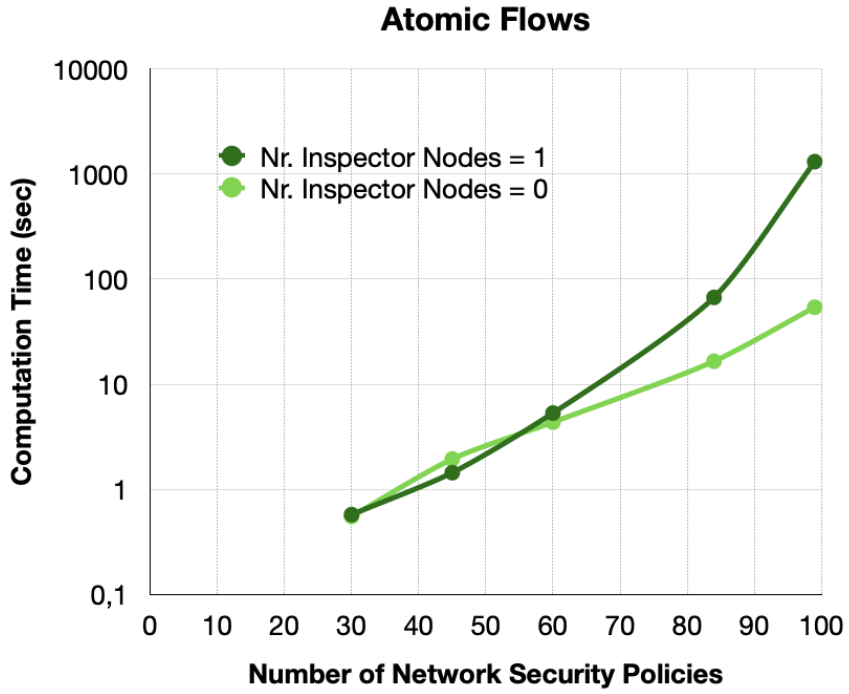

Figure 6.12. Impact of the inspector node in the Atomic Flows model

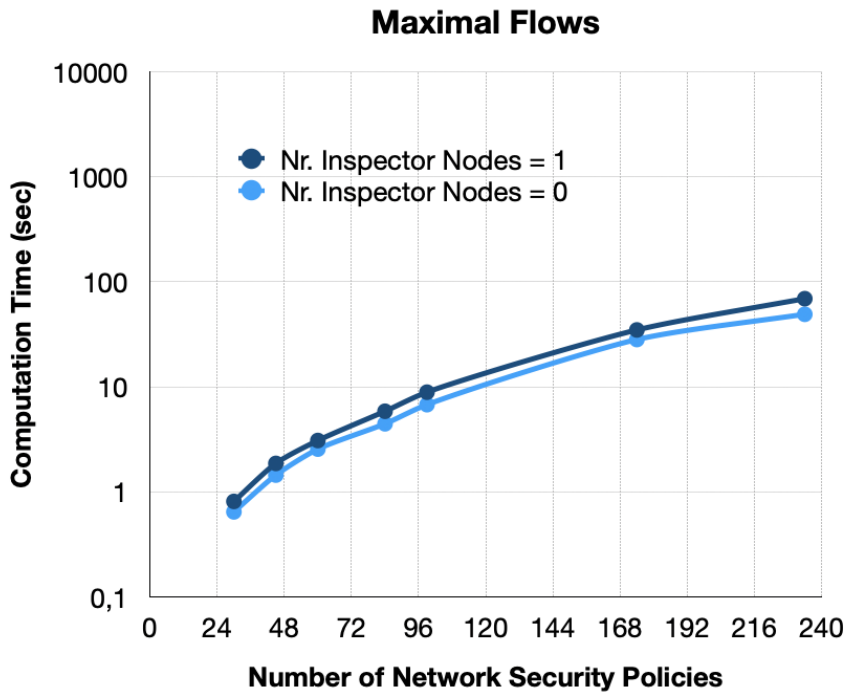

Figure 6.13. Impact of the inspector node in the Maximal Flows model

The following graphs show the total number of constraints generated in the two models.

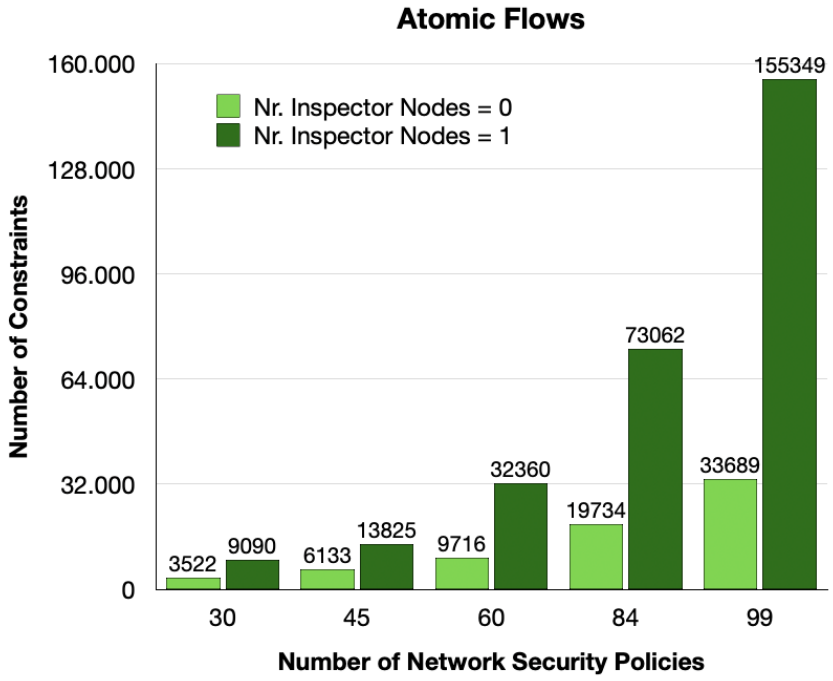

Figure 6.14. Test 2: Generation of constraints in the Atomic Flows model

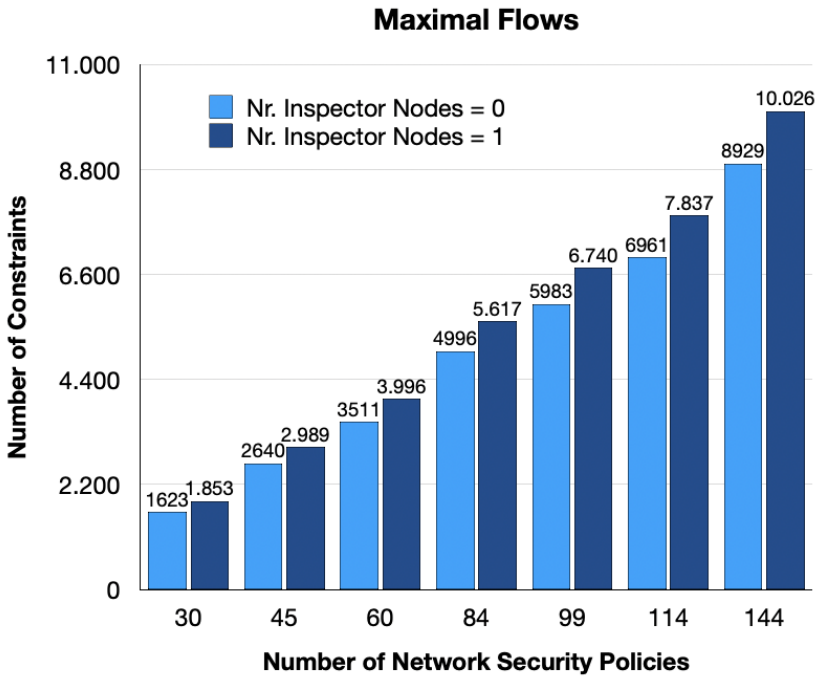

Figure 6.15. Test 2: Generation of constraints in the Maximal Flows model

In conclusion, the introduction of the inspector node in this test confirmed the overall trend observed in Test 1 for both models but resulted in a performance degradation. This deterioration can be attributed to the increased number of constraints generated and passed to the solver to define the behavior of the inspector node, and to the higher number of VPN gateways in the network due to the need for traffic to pass in clear.

6.4 Test 3: APs vs Policies Impact

This third test evaluates the impact of the number of APs and security policies on the performance of the framework. In the first case, the number of APs within the network remains constant while the number of endpoints and security policies increase. In the second case, instead, the network topology increases while the number of security policies is kept fixed.

The aim was to determine which parameter, between the number of APs and security policies, has the greatest impact on overall performance. For this purpose, only the Maximal Flows model was used, which proved to have better performance than the Atomic Flows model. Furthermore, to avoid delays and factors that could further affect overall performance, an inspector node was not included in the network.

6.4.1 Fixed APs Results

The structure considered consists of 20 APs. For each test run, the number of endpoints and security policies is increased, while the number of APs (20) and the

number of untrusted nodes (7) remain fixed. In this case, the structure topology does not grow as described in section A, but the added endpoints are connected to the APs of the basic structure (client and server), without creating new APs or untrusted nodes. This test was designed to evaluate the impact of security policies independently of the number of APs.

The results are shown in Table 6.16 and illustrated in Graph 6.17 .

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Computation Time (sec)
53	False	159	20	0	7	**25,291**
61	False	183	20	0	7	**31,799**
82	False	246	20	0	7	**71,378**
98	False	294	20	0	7	**100,752**
118	False	354	20	0	7	**179,640**
154	False	462	20	0	7	**391,957**
206	False	618	20	0	7	**1002,403**
238	False	714	20	0	7	**1918,530**

Figure 6.16. Test 3: Table of Fixed APs Results

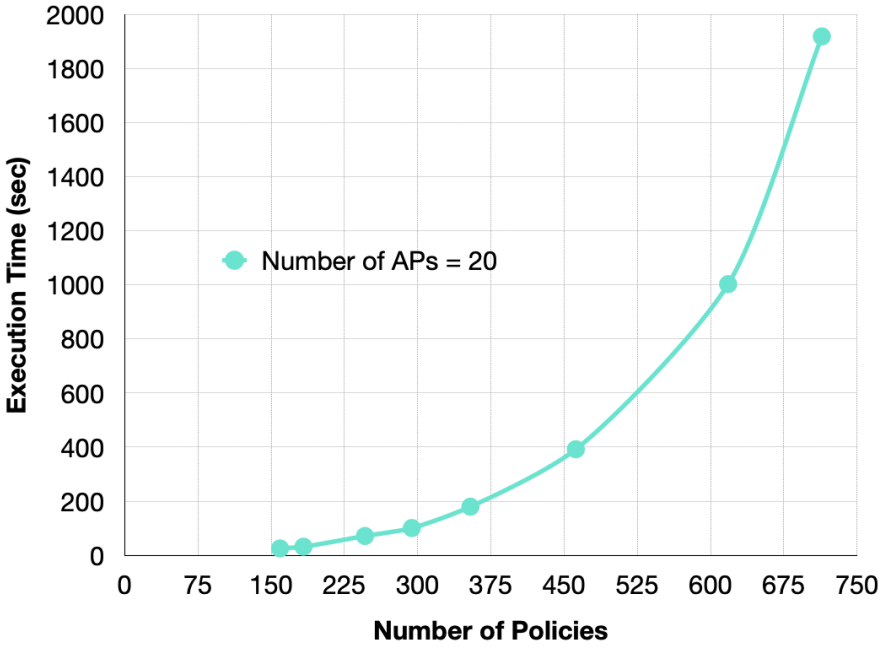

Figure 6.17. Test 3: Graph of Fixed APs Results

6.4.2 Fixed Policies Results

In this second case, the same network structure as described in Section A is used. The logic for expanding the topology when the parameters change remains the same. However, the difference is that the number of security policies is fixed at 100. These policies are defined in such a way as to require the protection of traffic flows of different lengths and directions. This approach allows the performance of the framework to be evaluated under conditions that are as realistic as possible.

Number of Endpoints	VPNcap Endpoint	Number of Network Security Policies	Number of APs	Number of Inspector Nodes per Policy	Number of Untrusted Nodes per Policy	Computation Time (sec)
38	False	100	20	0	7	**7,566**
48	False	100	25	0	9	**7,886**
58	False	100	30	0	11	**9,447**
98	False	100	50	0	17	**15,905**
118	False	100	60	0	21	**20,303**
198	False	100	100	0	34	**66,774**
298	False	100	150	0	51	**197,190**

Figure 6.18. Test 3: Table of Fixed APs Results

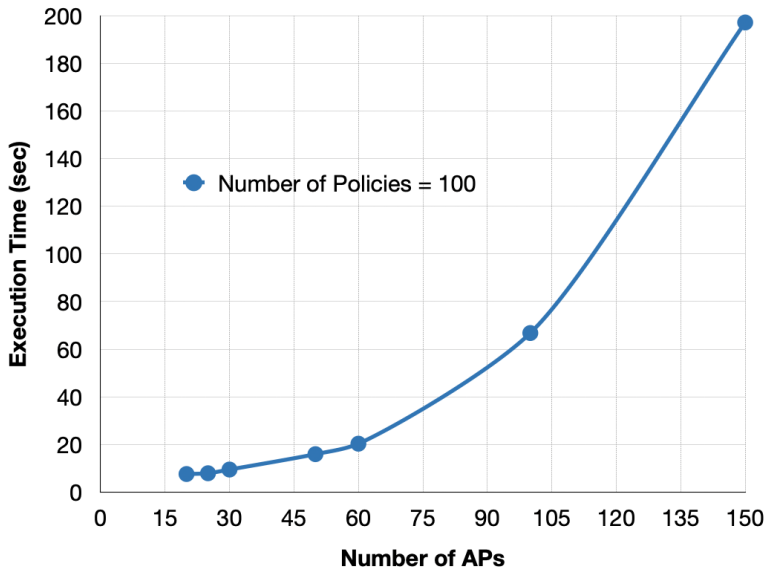

Figure 6.19. Test 3: Graph of Fixed APs Results

6.4.3 Interpretation of Test Results

From the analysis of the results obtained and the computation time in relation to the variable parameter in the specific case, it is possible to determine that the number of network security policies is the factor that has the greatest impact on the framework's performance.

In case 2, where the policies were kept fixed, it can be observed that the time does not change significantly with the increase in the size of the network structure and consequently the number of APs. It was possible to create a complex network topology, composed of 150 APs and 298 endpoints, with an execution time of approximately 200 seconds. On the contrary, in the case with the fixed number of APs, considering the same computation time, the network is much simpler and smaller, with about 120 endpoints. From this, it can be deduced that the parameter with the greatest impact on execution time is the number of defined security policies.

However, this observation can also be explained by analyzing the different number of constraints generated and passed to the solver. As shown by Figure 6.20 and Figure 6.21, this difference is significant. By increasing the number of APs while keeping the security policies constant, the number of generated constraints does not increase significantly. On the other hand, if the number of APs is kept fixed but the number of policies is increased, the opposite result is obtained. The number of defined constraints is much higher than in the previous case. This can be a reason why the number of policies has a greater impact on the overall performance.

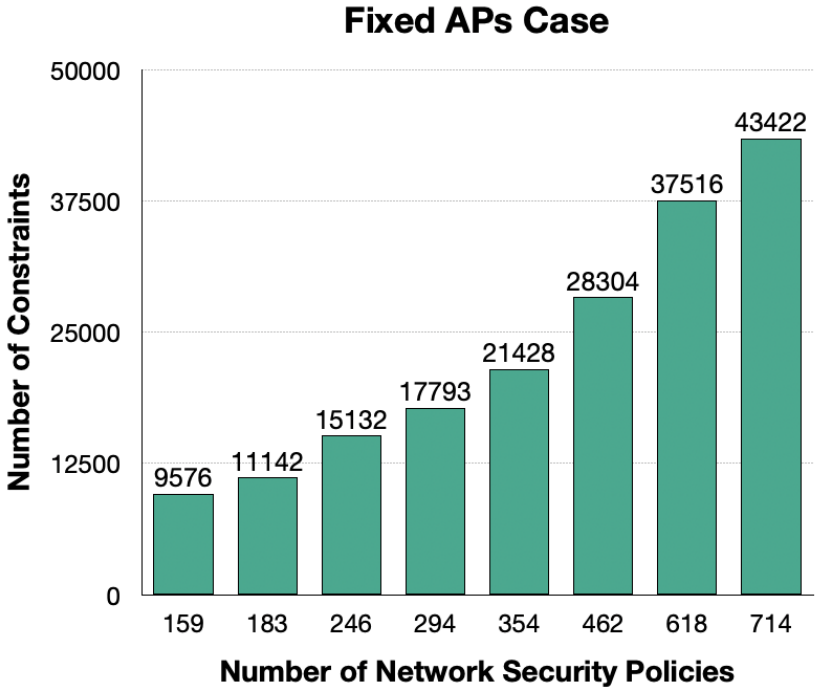

Figure 6.20. Constraints Generated with Fixed APs

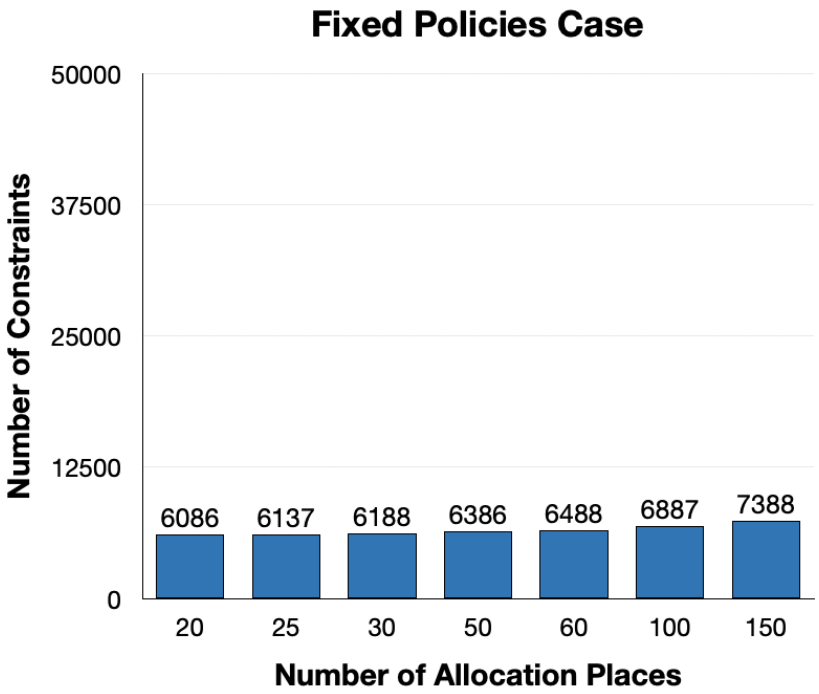

Figure 6.21. Constraints Generated with Fixed Policies

In conclusion, increasing the number of security policies has a significant impact on the framework's execution time compared to the number of APs in the network. Therefore, to optimize performance, it is important to carefully manage the number and complexity of security policies, aiming to reduce the number of constraints generated and passed to the solver.

6.5 Improving Framework Performance: Analysis Work

This section describes the analysis work performed on the framework to identify possible modifications for performance optimization. From the test results described in the previous section, it emerged that, without aggregation constraints, the Maximal Flows model is more advantageous than the Atomic Flows model, especially with a larger number of resources. Therefore, the analysis was conducted on the Maximal Flows model.

The analysis process followed a well-defined approach, starting with an analysis of the code to identify possible areas of intervention and ending with an evaluation of the performance with the new modifications implemented. The principal aim was to optimize the part that has the greatest influence on execution time, i.e. the MaxSMT. The key factor in solving the MaxSMT problem is the number of soft and hard constraints generated and passed to the solver. Therefore, an attempt was made to optimize by reducing the number of constraints or their impact, avoiding the generation of unnecessary constraints in certain situations. The changes made were to improve performance without compromising the correctness and optimality

of the solution provided by the solver. Then, tests were conducted to assess the actual performance improvement achieved with the implemented changes.

This work has led to several potential solutions, which are now described along with the results and improvements obtained.

6.5.1 Reducing Solver Constraints

The first intervention concerns the reduction of the constraints related to VPN capabilities for each node. The modification was made within the Checker.java file, where the protection constraints to be passed to z3 for each user-defined security requirement are created.

During this process, an inconsistency was identified. For each node in the requested flow to be protected by a security requirement, constraints related to the node's ability to support a VPN Gateway were generated. However, the possibility that some types of nodes, such as endpoints, untrusted nodes, or inspector nodes, cannot support a VPN Gateway had not been considered. A simple solution to this problem was to add a check to ensure that constraints are only defined for the appropriate nodes, before actually generating them. Thus, constraints are only generated for endpoints that support VPN and for AP where a security function can be allocated. The following box shows the construct introduced to perform this check.

Listing 6.1. Reducing Solver Constraints

```
private void createProtectionConstraints(SecurityRequirement sr) {
    ...
    if (node.getNode().isVpnCapabilities() || node.getTypeNF() ==
        FunctionalTypes.VPN_GATEWAY) {
    ...
    }
    ...
}
```

The application of this change has led to significant improvements in the number of constraints created and passed to the solver. Figure 6.22 and Figure 6.23 show a reduction of approximately 6.7% in the number of constraints, a significant percentage of reduction.

In terms of performance, however, no significant improvement was observed. Although a considerable reduction in the constraints created, the resulting performance shows only a slight increase, which is not significant. Therefore, it can be deduced that the constraints related to the VPN capabilities supported by the nodes do not have a substantial impact on performance. Thus, the absence of these constraints does not provide any advantage.

Number Endpoints	Number of Network Security Policies	Number of Constraints without modification	Number of Constraints with modification	% of Reduction.
10	30	**1623**	**1518**	**6,47%**
20	60	**3511**	**3269**	**6,89%**
33	99	**5983**	**5567**	**6,95%**
48	144	**8929**	**8324**	**6,78%**
78	234	**14553**	**13563**	**6,80%**
118	354	**22264**	**20756**	**6,77%**
142	426	**26912**	**25102**	**6,73%**

Figure 6.22. Table with constraint reduction

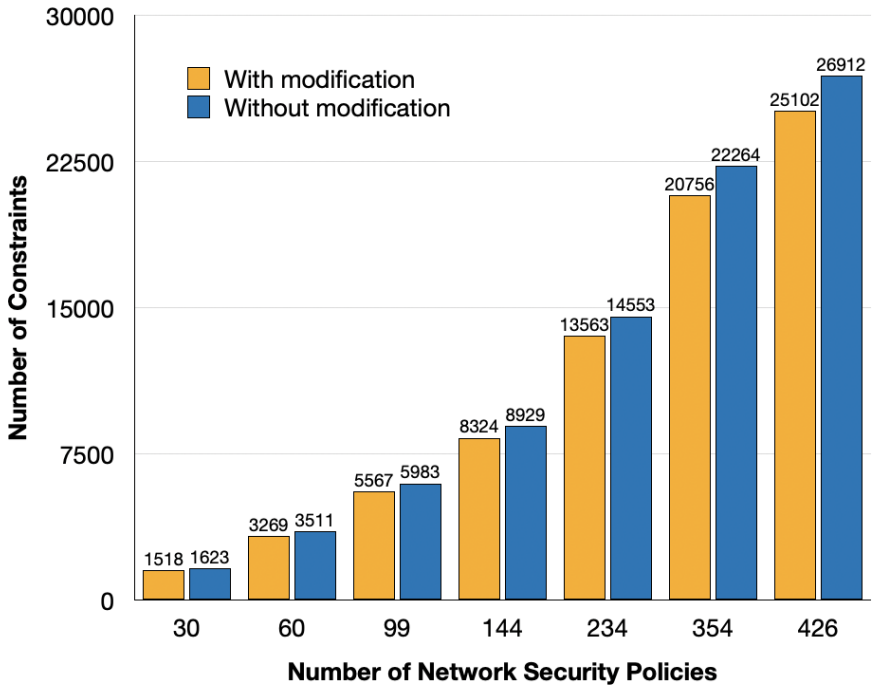

Figure 6.23. Graph with constraint reduction

Other modifications are made with the aim of reducing the number of constraints, such as the elimination of possible duplicated constraints before they are passed to the solver. However, these changes do not produce interesting results either in terms of performance or in the total number of constraints.

6.5.2 Redundancy Elimination for Improved Performance

Another change concerns the removal of 'low-level' constraints implemented to improve the readability of solver results and simplify translation into XML format. It turns out that these constraints are redundant and could be removed. The final solution remains the correct and optimal one.

Compared to the previous modification, significant performance improvements are achieved in this case. As shown in Figure 6.24 and Figure 6.25, the computation time improves considerably. Initially, with 744 network security policies, the search for the solution requires around 1200 seconds, whereas with the modification made, the solution is achieved in only 900 seconds, resulting in a saving of 300 seconds.

Number Endpoints	Number of Network Security Policies	Computation Time with redundancy (no modification) (sec)	Computation Time without redundancy (sec)
9	27	**0,511**	**0,399**
23	69	**3,129**	**2,126**
33	99	**6,767**	**4,491**
52	156	**20,438**	**12,936**
88	264	**73,325**	**53,206**
142	426	**256,143**	**168,567**
248	744	**1229,586**	**966,23**

Figure 6.24. Table with redundancy elimination

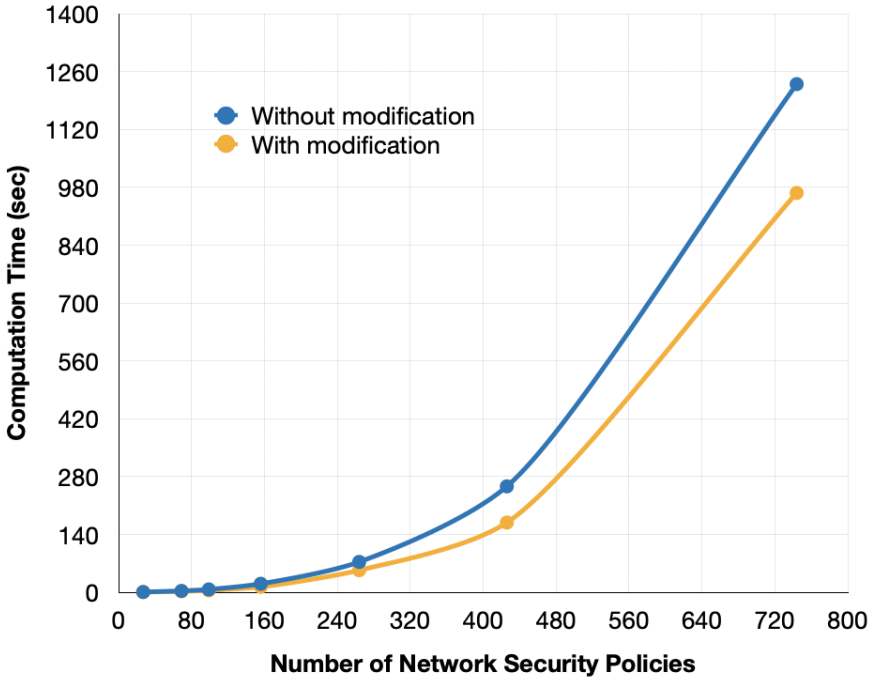

Figure 6.25. Graph with redundancy elimination

Chapter 7

Conclusions

The main objective of this thesis was to analyze a model for the configuration of secure communication within the network, extending the functionalities of VEREFOO. The model is based on the principles of full automation, optimization, and formal verification to reduce errors deriving from human intervention and compute a correct and optimal solution compared to the others available. This was achieved by solving the MaxSMT problem, which allows obtaining a correct VPN gateway configuration within the network that satisfies the user-defined input requirements.

A network generator capable of creating a complex topology with several branches between nodes was proposed. The generated network was designed to simulate the behavior of modern networks, allowing the definition of a set of protection policies for each endpoint in the network that must be respected in the final solution. This enables an evaluation of the network in a realistic situation, where several traffic flows with different lengths and directions must be protected.

Subsequently, the generator was used to evaluate the performance of VEREFOO to find a final network configuration that satisfies the input protection requirements. During this process, complete automation and correctness of the final solution were ensured. All tests were conducted using the two traffic flow models already implemented within VEREFOO, Atomic Flows, and Maximal flows. This allowed the two models to be compared under the same conditions, which one was more advantageous. The results showed that, despite a slightly lower initial performance, the Maximal Flows model was preferable. This advantage was observed both in the execution time required by the framework to compute the solution and in the number of constraints passed to the solver for the MaxSMT problem. Despite these differences, both models still demonstrated good performance even with increasing network topology complexity and input requirements.

In conclusion, the thesis demonstrated the effectiveness of the framework in automatically configuring and allocating virtual functions to protect communication, showing scalability even as the network topology becomes more complex. However, the work has some limitations. In particular, the focus was mainly on secure configuration using VPN with IPSec, without considering the possibility of other technologies and protocols. Furthermore, other types of nodes such as load balancers or NAT were not considered in the generated network structure.

In the future, it would be interesting to explore how VEREFOO adapts to these new changes, also attempting to evaluate the performance of end-to-end or side-to-side VPNs. In addition, changes will need to be introduced to the models in order to avoid an exponential deterioration of performance when the number of security requirements and related constraints exceeds a certain threshold.

Bibliography

[1] R. Doriguzzi-Corin, S. Scott-Hayward, D. Siracusa, M. Savi, and E. Salvadori, "Dynamic and application-aware provisioning of chained virtual security network functions," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 294–307, 2019.

[2] D. Bringhenti, J. Yusupov, A. M. Zarca, F. Valenza, R. Sisto, J. B. Bernabé, and A. F. Skarmeta, "Automatic, verifiable and optimized policy-based security enforcement for sdn-aware iot networks," *Comput. Networks*, vol. 213, p. 109123, 2022. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109123

[3] D. Bringhenti, F. Valenza, and C. Basile, "Toward cybersecurity personalization in smart homes," *IEEE Secur. Priv.*, vol. 20, no. 1, pp. 45–53, 2022. [Online]. Available: https://doi.org/10.1109/MSEC.2021.3117471

[4] 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

[5] D. Bringhenti, R. Sisto, and F. Valenza, "A novel abstraction for security configuration in virtual networks," *Comput. Networks*, vol. 228, p. 109745, 2023. [Online]. Available: https://doi.org/10.1016/j.comnet.2023.109745

[6] 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

[7] ——, "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

[8] ——, "Automated firewall configuration in virtual networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1559–1576, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3160293

[9] S. Bussa, R. Sisto, and F. Valenza, "Security automation using traffic flow modeling," in *8th IEEE International Conference on Network Softwarization, NetSoft 2022, Milan, Italy, June 27 - July 1, 2022*, A. Clemm,

G. Maier, C. M. Machuca, K. K. Ramakrishnan, F. Risso, P. Chemouil, and N. Limam, Eds. IEEE, 2022, pp. 486–491. [Online]. Available: https://doi.org/10.1109/NetSoft54395.2022.9844025

[10] 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

[11] F. Valenza and M. Cheminod, "An optimized firewall anomaly resolution," *J. Internet Serv. Inf. Secur.*, vol. 10, no. 1, pp. 22–37, 2020. [Online]. Available: https://doi.org/10.22667/JISIS.2020.02.29.022

[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] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *CYSARM@CCS '20: Proceedings of the 2nd Workshop on Cyber-Security Arms Race, Virtual Event, USA, November, 2020*, L. Chen, C. J. Mitchell, T. Giannetsos, and D. Sgandurra, Eds. ACM, 2020, pp. 25–30. [Online]. Available: https://doi.org/10.1145/3411505.3418439

[14] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[15] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE communications magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[16] E. Demaria, A. Pinnola, and N. Santinelli, "La virtualizzazione di rete: lo standard nfv," *Telecom Italia*, 2015.

[17] "Network functions virtualisation (nfv); infrastructure overview. gs nfv-inf 001 v1.1.1," ETSI, Tech. Rep, Tech. Rep., 2015.

[18] "Network functions virtualisation (nfv); management and orchestration; report on management and orchestration framework. gr nfv-man 001 v1.2.1," *Group specification, ETSI*, 2021.

[19] R. Patil and C. Modi, "An exhaustive survey on security concerns and solutions at different components of virtualization," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[20] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[21] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–7.

[22] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM*

Comput. Commun. Rev., vol. 43, no. 4, p. 3–14, aug 2013. [Online]. Available: https://doi.org/10.1145/2534169.2486019

[23] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[24] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Tech. Rep., 2015.

[25] 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

[26] P. Quinn and T. Nadeau, "Problem statement for service function chaining," Tech. Rep., 2015.

[27] N. ISG, "Network functions virtualisation (nfv)-virtual network functions architecture," ETSI, Tech. Rep, Tech. Rep., 2013.

[28] S. Demirci, M. Demirci, and S. Sagiroglu, "Virtual security functions and their placement in software defined networks: A survey," *Gazi University Journal of Science*, vol. 32, no. 3, pp. 833–851, 2019.

[29] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *Proceedings of the 2nd Workshop on Cyber-Security Arms Race*, 2020, pp. 25–30.

[30] D. Bringhenti, "Network security automation," 2022.

[31] D. C. Verma, "Simplifying network administration using policy-based management," *IEEE network*, vol. 16, no. 2, pp. 20–26, 2002.

[32] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*. Springer, 2008, pp. 337–340.

[33] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 887–900, 2016.

[34] F. Valenza and A. Lioy, "User-oriented network security policy specification." *J. Internet Serv. Inf. Secur.*, vol. 8, no. 2, pp. 33–47, 2018.

Appendices

Appendix A

Setting Up the Generator for Testing

This section shows how the generator was set up for testing. For both models, Atomic Flows and Maximal Flows, the files used are named differently, but the content is the same.

Test 1-2

TestPerformanceScalabilityVPNwithAP1.java file

This file uses Atomic Flows model. For Maximal Flows tests, the file is the same but is named "TestPerformanceScalabilityVPNConfA.java".

Listing A.1. Configuration of parameters for running tests

```
public class TestPerformanceScalabilityVPNwithAP1 {
    public static void main(String[] args) {

        int times = 100;

        Random rand1 = new Random();
        int upperbound = 9999;
        for (int i = 0; i < times; i++) {
            System.gc();
            seed = rand1.nextInt(upperbound);
            numberClients = 138;
            numberClientsPerAP = 2;
            numberServers = 4;
            numberLB = 0;
            numberInspectorNodes = (numberServers > 2) ? 1 : 0;
            numberInspectorNodes = 0;

            testScalabilityPerformance();
        }
        ...
    }
    ...
}
```

Number of Executions

It indicates how many times the test is repeated with those inputs. After each execution, a result is computed and at the end an average value is obtained from these results.

```
int times = 100;
```

Number of Clients and Servers

These lines set the number of Clients and Servers there will be in the generated network structure. The minimum value of Clients is 5.

```
numberClients = 30;
numberServers = 8;
```

Number of Inspector Nodes

As described, there are two versions: one without inspector nodes in the network and another with the possibility of having an inspector node if the number of servers is greater than 2. In the latter case, the second line must be removed.

```
numberInspectorNodes = (numberServers > 2) ? 1 : 0;
numberInspectorNodes = 0;
```

All other parameters, such as the number of untrusted nodes and the number of APs, are calculated automatically.

TestCaseGeneratorVPNwithAP1.java file

Once all parameters have been set, the test can be executed. During the execution, this file will be called, which contains the generator responsible for creating the network structure and defining security policies.

Number of Network Security Policies

In the tests performed, the number of network security policies per endpoint was always set to 3. There is also the possibility to change this parameter in the generator

```
int numberPolicyEndpoint = 3;
```

Test 3 - Fixed APs

A separate file was created for this case. The general parameters remain the same, but the possibility of increasing client and server nodes while keeping the network structure unchanged has been added. This allows increasing client and server nodes without changing the number of AP and untrusted nodes. This is done by setting the values of two specific variables.

TestPerformanceScalabilityVPNwithAP2.java file

All parameters for network creation remain the same, except for two variables. Variable *numberClientsToAdd* allows for the addition of client nodes once the network structure has been created based on the previous parameter values. The same applies to variable *numberServersToAdd*, which adds server nodes. This file uses Atomic Flows model. For Maximal Flows tests, the file is the same but is named "TestPerformanceScalabilityVPNConfB.java".

```
numberClientsToAdd = 30;
numberServersToAdd = 10;
```

TestCaseGeneratorVPNwithAP2.java file

The generator defines the network in two steps. First, it creates the network structure based on the value of the user-defined variable *numberClients* and *numberServers*. In the second it only adds the number of clients (*numberClientsToAdd*) and servers (*numberServersToAdd*) to the network already created. In conclusion, the total number of clients will be *numberClients*+*numberClientsToAdd*, while *numberServers*+*numberServersToAdd* for servers.

Test 3 - Fixed Network Security Policies

To maintain the fixed number of network requirements, modifications were introduced in the generator file used in Test 1.

TestCaseGeneratorVPNwithAP1.java file

The parameter *numberPolicyFixed* specifies the total number of policies that need to be defined within the network.

```
int numberPolicyFixed = 100;
```