Politecnico di Torino

Master of Science in Computer Engineering

Master Degree Thesis

Towards automation of TLS-based VPN configuration

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

Candidate
Luca Bianconi

Academic Year 2022-2023

Summary

The advancements in networking technologies, namely Network Functions Virtualization (NFV) and Software-Defined Networking (SDN), have significantly enhanced the flexibility and efficiency of building Service Function Chains. NFV enables the implementation of specific network functions, such as NAT or proxy servers, on standard servers, eliminating the need for dedicated hardware devices. This utilization of standard servers allows for the consolidation of multiple network functions, optimizing resource utilization and enabling the addition of new functions without the need for additional physical devices.

SDN, on the other hand, provides the ability to create tailored routes for different types of traffic or users, adding greater flexibility to Service Function Chain construction. Through controller programming, the path taken by a packet can be dynamically modified and customized as it traverses various network devices.

However, the manual configuration of network devices in the creation of Service Function Chains presents challenges. Incorrect configurations can lead to serious security breaches or unwanted traffic acceptance. Additionally, manual configuration can result in significant latency during updates or maintenance of the security system. Network Automation offers a solution by automating the configuration of network security devices, reducing human errors and minimizing latency associated with configuration changes.

An example of a Network Automation framework is VEREFOO (VErified REFinement and Optimized Orchestration). By inputting a Service Graph and Network Security Requirements, VEREFOO generates an optimized Service Graph solution that identifies a physical network with automatically allocated and configured network security functions. This ensures the best alignment with the provided Network Security Requirements.

The primary focus of this thesis work was on selecting the most suitable VPN technology. The key contribution of this research was the enhancement and expansion of the VEREFOO framework, which was originally designed to address network security requirements by managing the allocation and configuration of Firewalls and Channel Protection Systems. However, the framework lacked the capability to choose between different VPN technologies. The thesis work addressed this limitation and after an extensive work of research and modelization, introduced the ability to select the optimal VPN technology within the VEREFOO framework. Furthermore, aspects of the previous version of the framework were improved, with

a specific focus on scalability and performance enhancements.

To validate the effectiveness of the framework and highlight the factors that significantly impact its scalability, performance tests were conducted. These tests were carried out by varying numerous input values, such as the Allocation Points and Network Security Requirements, both in number and in requested property. The performance tests aimed to verify the accuracy, scalability, and overall improvements achieved by the framework.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Thesis objective

Over the past few years, two new technologies in networking have emerged, namely Network Functions Virtualization (NFV) and Software-Defined Networks (SDN). NFV refers to the capability of running network functions on standard hardware, with the help of computing virtualization to optimize resource utilization, allowing for the flexible deployment of network functions to form a Service Function Chain. On the other hand, SDN allows the creation of packet paths inside the physical network using software processes. Prior to these technologies, service graphs were typically composed of specific hardware devices designed for implementing specific services like NAT or proxy servers. Nowadays, virtualizing the network functions composing the graph is more prevalent, as it allows for efficient use of hardware resources, sharing physical resources by installing more than one service function on the same server. Adding new functions is also simpler, and it no longer requires buying expensive, dedicated hardware. Instead, it can be installed on already available general purpose servers. These concepts can be applied not only to network service functions, but also to Network Security Functions (NSFs) such as packet filter firewalls and VPN Gateways.

One challenge that arises when creating a Service Graph and configuring the virtual network functions is that these tasks are typically performed manually. This is particularly critical when allocating and configuring security functions, which are intended to provide security services such as communication protection or the ability to deny/allow certain network traffic. Misconfigurations of such functions can easily result in severe and dangerous incidents such as security breaches. Additionally, performing these operations manually is slow, which can lead to delays in updating security defences according to changing or additional security requirements. For this reason, Network Automation is a valuable alternative as it enables the automation of Network and Security Management, managing configuration changes automatically with lower latency and without the risk of human errors.

The thesis objective is to contribute to the design and implementation of VEREFOO, which stands for VErified REFinement and Optimized Orchestration. The

framework is designed to automate the allocation and configuration of Network Security Functions on a Service Graph, in order to fulfil a set of network security requirements expressed by the security administrator using a high-level security language through a refinement process. This thesis focuses on extending the functionality of VEREFOO to ensure the optimal choice of VPN technology between IPsec and TLS based technologies.

1.2 Thesis description

The rest of the thesis is organized in the following way:

- Chapter 2 is about the different VPN technology. It starts with the presentation of three different way to implement TLS based VPN, highlighting commonalities and differences. Then a brief introduction to IPsec is performed so that is possible to perform a comparison between these two different technologies. The last part of the chapter is focused on the comparison of the security properties provided by both IPsec and TLS.

- Chapter 3 opens by introducing the concept of a Service Function Chain (SFC), which is a chain of multiple network functions designed to provide services for a specific communication. The chapter then proceeds to introduce two important advancements in computer networks: Software Defined Networking (SDN) and Network Function Virtualization (NFV). These advancements are discussed in relation to their benefits for SFCs. The final section of the chapter focuses on the concept of automating network and security management using the aforementioned advancements and an enhanced SFC. It highlights the advantages of implementing automatic configurations and management through the utilization of these innovations. this section also presents the framework on which part of the thesis work was carried out.

- Chapter 4 focuses on presenting the main objective of the thesis work and how it was accomplished. The chapter begins by discussing how the main objective was divided into smaller, essential objectives that needed to be achieved to fulfil the overall objective. The four objectives are then described in detail, along with the potential approaches that were pursued to attain them.

- Chapter 5 is entirely dedicated to the presentation of the approach used to carry out the thesis work. The first section addresses the study of the VPN technologies, in the specific IPsec and TLS. Then is presented the approach taken for the modelization work, both the introduction of new function and the modification of already present one, like Communication Protection Policies. In the last section, the focus is on the approach used in the implementation part of the thesis work, which is based on a partial weighted MaxSMT (Maximum Satisfiability) problem. The section describes the underlying principles of this approach, highlighting the distinction between hard and soft constraints.

- Chapter 6 open with the reformulation of Communication Protection Policies and Rule, necessary to implement the new functionality necessary to achieve the goal of this thesis. In the following sections are presented the model for the function: USED_TECHNOLOGY, MODE, ARCHITECTURE. In each one of this section in addition are presented a series of hard and soft constraints needed for the selection of the correct VPN technology given the requested security requirements.

- Chapter 7 begins with the presentation of five different use cases to show the newly introduced functionality. For each example are presented the relative section of input and output of the framework and the explanation of the reason a certain solution has been selected.

- Chapter 8 is dedicated to the conclusion of the thesis work, the achieved objectives are summarized and are presented some possible work to further improve the framework.

Chapter 2

VPN Technology

This chapter performs an in-depth analysis of the TLS protocol when used to create VPN tunnels. Since unlike IPsec there is no standard for implementing TLS-based VPNs in the first section of the chapter we will compare three different protocols: OpenVPN, TINC, SSTP.

The second section, after a brief introduction to IPsec, will focus on the differences and similarities between TLS-based VPNs and IPsec.

The third section will look in detail at the differences between the security requirements offered by these security technologies.

2.1 TLS protocols comparison

Transport Layer Security (TLS) is a cryptographic protocol that ensures communication security over a computer network. It is commonly used in email, instant messaging, and voice over IP applications, but is most well-known for securing HTTPS. The main objective of TLS is to provide security by ensuring confidentiality, integrity, and authenticity through the use of cryptography between two or more communicating computer applications.

Confidentiality and integrity are enforced on the header of the original packet, while the integrity on the header of the encapsulating packet is not possible. The TLS protocol consists of two layers: the TLS record and the TLS handshake protocols.

TLS can also be used for creating a Virtual Private Network (VPN) by tunnelling an entire network stack. Compared to traditional IPsec VPN technologies, TLS has inherent advantages in firewall and NAT traversal, making it easier to administer for large remote-access populations.

In this work we have specifically analysed three different protocols: OpenVPN, TINC, SSTP.

2.1.1 OpenVPN

is a widely used open-source virtual private network (VPN) technology that offers a secure and flexible solution for establishing encrypted connections over the internet. Its robust set of properties makes it an excellent choice for both individuals and organizations seeking to protect their online privacy, secure their data, and establish remote access capabilities.

Another important property of OpenVPN is its versatility. It is compatible with various operating systems, including Windows, macOS, Linux, Android, and iOS, making it accessible to a wide range of users. Additionally, OpenVPN supports multiple connection types, such as TCP and UDP, and can utilize various network protocols, including Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). This flexibility allows OpenVPN to adapt to different network environments and provide reliable connectivity across diverse platforms.

The structure of the VPN packets is described in OpenVPN documentation [22] as follows:

- packet length (16 bits, unsigned) [TCP-mode only]: always sent as plain text. Since TCP is a stream protocol, this packet length defines the packetization of the stream.
- packet opcode and key_id (8 bits) [TLS-mode only]:
  - package message type (high 5 bits): there are numerous type available divided in Control channel messages and Data channel messages.
  - key_id (low 3 bits): the key_id refers to an already negotiated TLS session. OpenVPN seamlessly renegotiates the TLS session by using a new key_id for the new session. Overlap (controlled by user definable parameters) between old and new TLS sessions is allowed, providing a seamless transition during tunnel operation.
- payload (n bytes)

OpenVPN also boasts excellent scalability and configurability. It can handle large-scale deployments and accommodate numerous clients and servers, making it suitable for organizations with extensive network infrastructure and multiple users. Furthermore, OpenVPN offers a comprehensive set of configuration options, allowing users to fine-tune settings to meet their specific requirements. This level of customization empowers users to optimize performance, prioritize traffic, and implement advanced security features based on their individual needs.

Moreover, OpenVPN is an open-source technology, which means its source code

is freely available to the public. This fosters transparency and encourages community involvement in its development and improvement. The open nature of OpenVPN enables security audits, peer reviews, and contributions from a global network of experts, reducing the risk of known vulnerabilities remaining unpatched for significant periods or backdoors being allowed to exist, enhancing its overall reliability and security.

OpenVPN utilizes the OpenSSL library to implement a combination of symmetric encryption algorithms like AES (Advanced Encryption Standard) and asymmetric encryption techniques such as RSA (Rivest-Shamir-Adleman), including widely adopted and recommended algorithms like AES 256. This approach ensures the confidentiality, integrity, and authenticity of the transmitted data.

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate, and the server must authenticate the client certificate before mutual trust is established.

As OpenVPN uses the TLS channel over TCP port 443, the traffic is indistinguishable from regular HTTPS traffic. It cannot be identified by Internet Service Providers (ISPs) or firewalls looking to block VPN traffic.

In summary, OpenVPN is a powerful VPN solution that combines robust security measures, versatility, scalability, configurability, and open-source benefits. With its wide compatibility and customization options, OpenVPN offers a flexible solution for various network environments and user requirements. Its open-source nature ensures continuous improvement and fosters a strong community-driven development process.

2.1.2 Tinc

is a lightweight, open-source VPN software that provides secure and efficient connectivity solutions for creating virtual private networks. With its unique set of properties, Tinc offers a versatile and reliable platform for establishing encrypted connections across different network topologies.

One of the key properties of Tinc is its peer-to-peer (P2P) nature. Unlike traditional VPN solutions that rely on a centralized server, Tinc employs a mesh routing architecture. This means that each node in the network can act as both a client and a server, allowing for a distributed and resilient infrastructure. The decentralized design enhances fault tolerance, as the network can automatically reroute traffic in case of node failures, ensuring uninterrupted connectivity.

Another notable property of Tinc is its support for various operating systems. It is compatible with a wide range of platforms, including Windows, macOS, Linux, FreeBSD, and even embedded systems like routers and IoT devices. This cross-platform compatibility enables seamless integration into diverse network environments, making Tinc a versatile choice for different deployment scenarios.

Tinc also excels in terms of performance and scalability. It utilizes efficient encryption algorithms and compression techniques to minimize latency and maximize throughput. Additionally, Tinc can scale to accommodate large networks with hundreds or even thousands of nodes, making it suitable for both small-scale deployments and enterprise-level infrastructures.

Tinc shares some similarities with OpenVPN. For instance, they both operate optimally over UDP, but TCP capability is provided. To ensure the confidentiality and integrity of transmitted data, Tinc also uses the OpenSSL library, complemented by the LibreSSL library. This means that Tinc employs strong cryptographic protocols, such as RSA and AES. Additionally, Tinc supports bidirectional authentication, where both the client and server must authenticate.

One difference with the other protocol analyzed in this section is that Tinc, by default, uses UDP port 655 (for both IPv4 and IPv6) to exchange routing and connectivity information between nodes in the network. However, this can be changed in the configuration file to use a different port or protocol if necessary.

The topic of network packet encryption is addressed in the documentation [24] as follows:
A data packet can only be sent if the encryption key is known to both parties, and the connection is activated. If the encryption key is not known, a request is sent to the destination to retrieve it. The packet is stored in a queue while waiting for the key to arrive. The UDP packet containing the network packet from the VPN has the following layout:

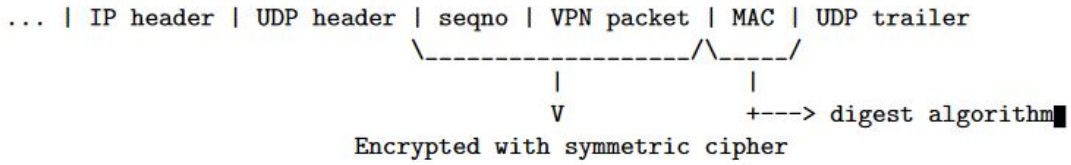

Figure 2.1: Tinc data packet structure

So, the entire VPN packet is encrypted using a symmetric cipher, including a 32 bits sequence number that is added in front of the actual VPN packet, to act as a unique IV for each packet and to prevent replay attacks. A message authentication code is added to the UDP packet to prevent alteration of packets. By default the first 4 bytes of the digest are used for this, but this can be changed using the MACLength configuration variable.

An additional advantage of Tinc is its simplicity and ease of use. It features a straightforward configuration process, allowing users to set up VPN connections with minimal effort. Tinc also supports dynamic routing, enabling automatic discovery and establishment of connections between nodes, further simplifying network management and reducing administrative overhead.

In summary, Tinc is a lightweight, decentralized VPN software that offers secure and efficient connectivity solutions. Its properties include a decentralized mesh routing architecture, cross-platform compatibility, excellent performance and scalability, robust security features, and ease of use. With Tinc, users can establish encrypted connections across different network topologies, ensuring reliable and private communication. Whether for small-scale deployments or large-scale infrastructures, Tinc provides a versatile and reliable platform for creating virtual private networks.

2.1.3 Secure Socket Tunnelling Protocol (SSTP)

is a VPN protocol developed by Microsoft that offers a secure and reliable solution for establishing encrypted connections over the internet. SSTP possesses several properties that make it a valuable choice for individuals and organizations seeking a secure VPN solution.

One of the key properties of SSTP is its strong security measures. It utilizes the TLS protocol to encrypt the data transmitted between the client and the server, ensuring the confidentiality and integrity of the communication. By leveraging the same encryption standards used in secure websites, SSTP effectively protects against eavesdropping and data manipulation by malicious entities, providing robust security for sensitive information during transit.

The following diagram shows the format of the SSTP packet when is sent on the HTTPS connection as shown by the Microsoft provided documentation [23].

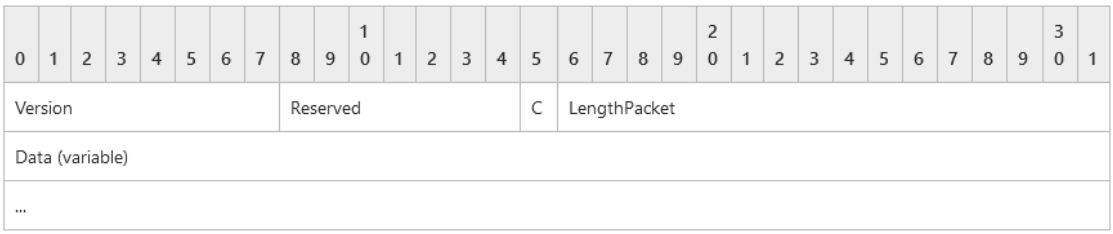

Figure 2.2: SSTP Packet Structure

- Version (1 byte): An 8-bit field utilized for communicating and negotiating the SSTP version in use.
- Reserved (7 bits): This 7-bit field is currently reserved for future purposes. It must be set to zero during transmission and disregarded upon receipt.
- C (1 bit): A 1-bit field indicating whether the packet is an SSTP control packet (set to 1) or an SSTP data packet (set to 0).
- LengthPacket (2 bytes): A 16-bit unsigned integer, represented in network byte order, that combines data for two fields. The first field, R, is a 4-bit

reserved area set to zero during transmission and ignored upon receipt. The second field, Length, is a 12-bit unsigned integer, represented in network byte order, which MUST specify the length of the entire packet (including the 4-byte SSTP header) in bytes.

- Data (variable): This field has a variable length equal to the value of the Length field minus 4. It contains either the SSTP control message (when C is set to 1) or the payload from a higher-layer protocol (when C is set to 0).

Another important property of SSTP is its wide compatibility. It is natively supported by Windows operating systems, making it an ideal choice for users within the Microsoft ecosystem. Additionally, SSTP can be configured on other platforms, including Linux and macOS, using third-party software. This compatibility allows for seamless integration into existing network environments, enabling users to establish VPN connections across different devices and operating systems.

SSTP, similar to OpenVPN, uses port 443 as the standard port, allowing it to traverse firewalls and network address translation (NAT) devices. This capability makes SSTP suitable for establishing secure connections even in restrictive network environments.

Moreover, SSTP is known for its high stability and robustness. Operating exclusively over TCP, SSTP ensures that packets are delivered in the correct order and without loss, providing reliable and uninterrupted connectivity for applications that require consistent performance.

SSTP is more flexible in the bidirectional authentication, in fact while the servers must be authenticated, the client authentication is optional.

Furthermore, SSTP is easy to configure and use. As a native protocol in Windows, it can be set up with a few simple steps, eliminating the need for additional software or complex configurations. This user-friendly nature makes SSTP accessible to both novice and experienced users, reducing the barrier to implementing secure VPN connections.

Additionally, SSTP supports a wide range of authentication methods, including username/password, smart card, and certificate-based authentication. This flexibility allows users to choose the most suitable authentication mechanism based on their specific security requirements and infrastructure.

In summary, SSTP is a secure and reliable VPN protocol with properties that make it a valuable choice for users seeking encrypted connections. Its strong security measures, wide compatibility, excellent performance and reliability, stability, and ease of use contribute to its effectiveness as a VPN solution. Whether for individual users or organizations, SSTP provides a secure and accessible means of establishing encrypted connections over the internet, safeguarding sensitive information, and ensuring privacy. In conclusion, considering the variety and spread of the protocols analysed, it is possible to highlight the similarities of the different

VPN implementations based on TLS. Therefore, below we are summarizing the characteristics of greater interest for our work:

- All the protocols studied provide the same security property, using the most modern cipher. Like shown in the picture, confidentiality and integrity are enforced on the header of the original packet, while the integrity on the header of the encapsulating packet is not possible.

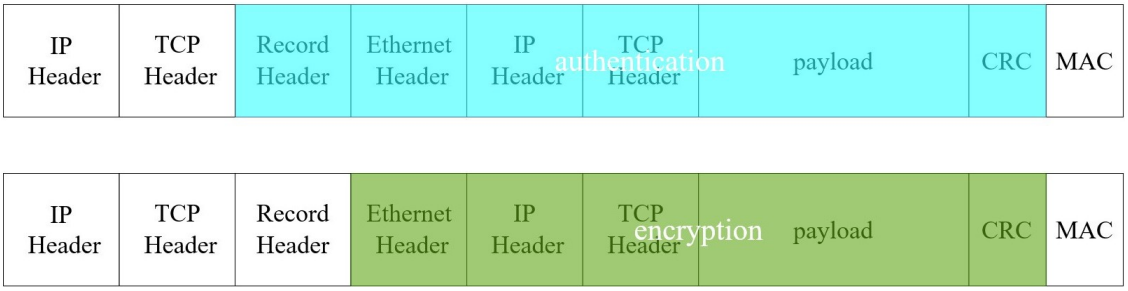

Figure 2.3: TLS security properties

- TLS has inherent advantages in firewall and NAT traversal.
- By default, there is peer authentication between client and server (SSTP is more flexible, so optionally the client may not be authenticated by the server)
- All the protocol have a great versatility, being compatible with various operating systems.

2.2 TLS and IPsec comparison

Before moving on to a comparison between TLS and IPsec, for the sake of completeness let us proceed with a brief summary of IPsec's properties.

Internet Protocol Security (IPsec) is a widely used protocol suite that provides a secure and encrypted communication framework for IP networks. Known for its robust security features, IPsec offers a range of properties that make it a popular choice for establishing virtual private networks and ensuring, data origin authentication, data integrity, data confidentiality through encryption, and protection from replay attack.

One of the key properties of IPsec is its strong encryption capabilities. To encrypt data packets at the IP layer IPsec employs various encryption algorithms, such as AES (Advanced Encryption Standard), 3DES (Triple-DES). This ensures that all data transmitted over an IPsec-secured connection is protected from unauthorized access and eavesdropping. By encrypting the data, IPsec provides a high level of confidentiality, safeguarding sensitive information from interception.

Another important property of IPsec is its flexibility in terms of authentication

and key management. IPsec supports multiple authentication methods, including pre-shared keys, digital certificates, and public key infrastructure (PKI). These authentication mechanisms verify the identity of communicating parties, ensuring that data is exchanged between trusted endpoints. IPsec also provides secure key exchange protocols, such as Internet Key Exchange (IKE), for establishing and managing cryptographic keys used for encryption and decryption. This flexibility and robustness in authentication and key management make IPsec a reliable and secure protocol suite.

IPsec is also known for its compatibility with various network topologies and protocols. It can be implemented in host-to-host, site-to-site, and remote access VPN configurations. Furthermore, IPsec operates at the IP layer, making it transparent to higher-level protocols and applications. This means that IPsec can secure all types of IP traffic, including TCP, UDP, and ICMP, without requiring modifications to existing applications or network infrastructure. The compatibility and transparency of IPsec enable seamless integration into diverse network environments, making it a versatile choice for VPN deployments.

Moreover, IPsec provides strong data integrity through the use of integrity algorithms, such as HMAC (Hash-based Message Authentication Code). These algorithms generate hash values that ensure the integrity of transmitted data by detecting any tampering or modification during transit. This property ensures that data remains unaltered and authenticates the source of the information.

Additionally, IPsec is highly scalable and can support large-scale deployments. It can handle high volumes of traffic and accommodate many VPN connections simultaneously. IPsec's scalability makes it suitable for enterprises and organizations with extensive network infrastructure and a large number of users.

IPsec uses the following protocols to perform various functions:

- Authentication Headers (AH) provides data integrity and data origin authentication for IP datagrams and protects against replay attacks.
- Encapsulating Security Payloads (ESP) offers confidentiality, connectionless data integrity, data origin authentication, a form of partial sequence integrity called anti-replay service, and limited traffic-flow confidentiality.
- Internet Security Association and Key Management Protocol (ISAKMP) establish a framework for authentication and key exchange.

The AH and ESP protocols of IPsec can be applied either in a host-to-host transport mode or in a network tunnelling mode.

In transport mode, only the IP packet's payload is usually encrypted or authenticated, and the routing is preserved since the IP header is not altered or encrypted. However, using the authentication header hinders network address translation since it invalidates the hash value when IP addresses are modified. The hash secures the

transport and application layers, preventing any modifications, such as port number translations. To enable NAT traversal, RFC documents have defined a way to encapsulate IPsec messages, known as the NAT-T mechanism.

In tunnel mode, the complete IP packet is encrypted and authenticated, and then enclosed within a new IP packet with a fresh IP header.

The figures also depict these security properties:

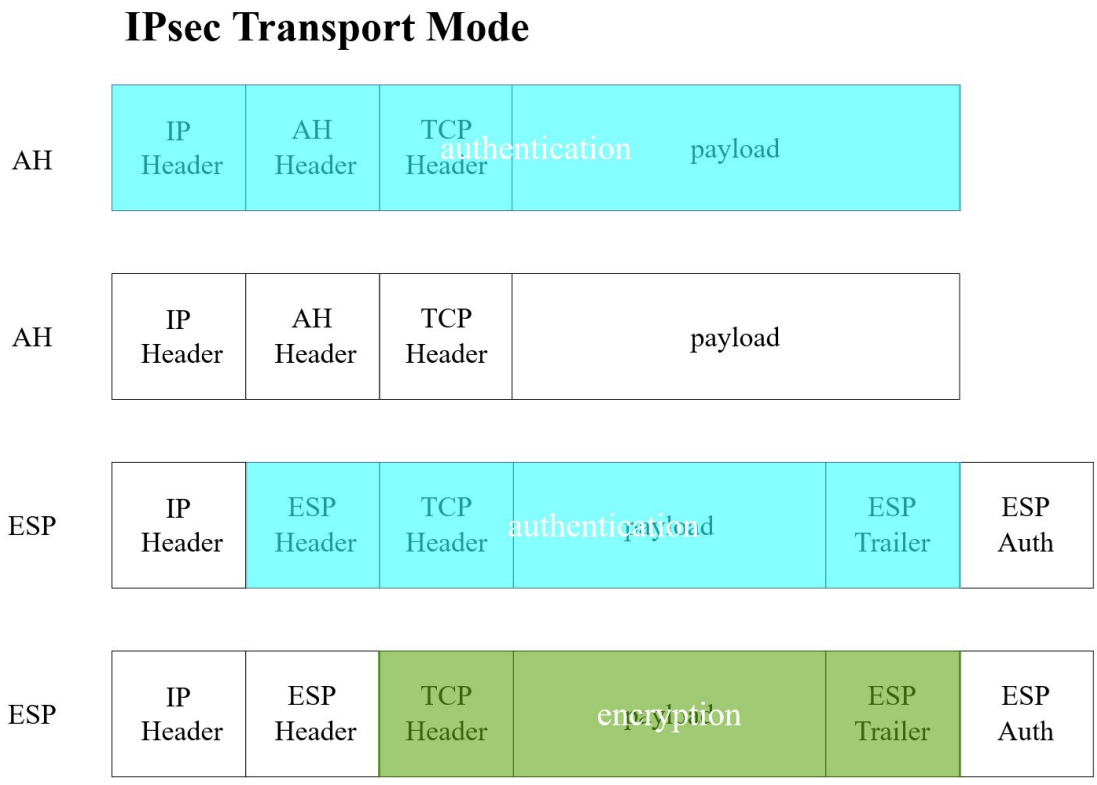

Figure 2.4: Ipsec Transport Mode security properties.jpg

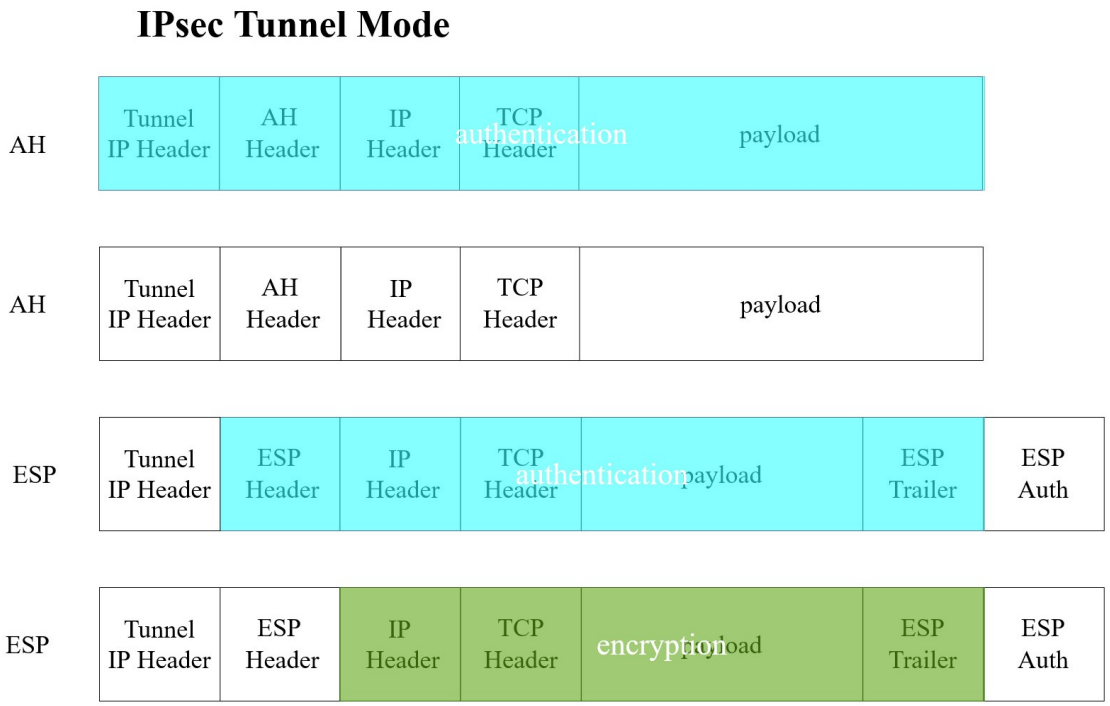

Figure 2.5: Ipsec Tunnel Mode security properties.jpg

In summary, IPsec is a powerful protocol suite that offers strong encryption, authentication, and key management capabilities. Its properties include robust security measures, flexibility in authentication and key management, compatibility with various network topologies and protocols, data integrity, and scalability.

The major distinction between an IPsec VPN and an TLS VPN is primarily related to the network layers where encryption and authentication are carried out. IPsec works at the network layer and can encrypt data exchanged between any systems that can be identified by IP addresses. On the other hand, TLS VPNs operate at the transport layer and can encrypt data sent between any two processes recognized by port numbers on network-connected hosts. For this reason, TLS VPNs have an inherent advantage in firewall and NAT traversal.

Both IPsec and TLS VPNs support various user authentication methods. IPsec employs Internet Key Exchange (IKE) version 1 or 2 and can use digital certificates or preshared secrets for two-way authentication. In contrast, TLS VPNs always use digital certificates for authentication.

Access control differs between these technologies. IPsec VPNs allow users to remotely connect to an entire network and all its applications, whereas TLS VPNs give users remote tunnelling access to a specific system or application on the network. If attackers gain access to the IPsec secured tunnel, they could potentially access anything on the private network. Is also possible to implement different access level using IPsec, but this requires organizations to set up multiple VPNs, increasing considerably the configuration difficulty.

TLS VPNs can use a web browser or a dedicated client software for accessing to secure systems, while IPsec VPNs require a dedicated client software. TLS VPNs can be considered more user-friendly than IPsec VPNs since they do not require the installation of any client software.

2.3 Security requirements

Having analysed the security properties offered by TLS-based VPNs and IPsec-based VPNs, in this section we summarise all the properties that an administrator may wish to implement when establishing a VPN.These security properties are also summarised in the following table:

	SSL/TLS VPN	IPsec Transport mode			IPsec Tunnel mode		
		ESP	AH	ESP+AH	ESP	AH	ESP+AH
Confidentiality Internal Header	USER CHOICE				USER CHOICE		USER CHOICE
Integrity Internal Header	MANDATORY		MANDATORY	MANDATORY	USER CHOICE	MANDATORY	MANDATORY
Integrity Internal Payload	MANDATORY	USER CHOICE	MANDATORY	MANDATORY	USER CHOICE	MANDATORY	MANDATORY
Integrity Additional Header						MANDATORY	MANDATORY
Server authentication	MANDATORY						
Client authentication	USER CHOICE						

Figure 2.6: Security Properties Table

For each requirement, is indicated whether it is implemented or not for each VPN technology and whether the implementation is mandatory or configurable by the end user.

- *Confidentiality of the internal header* is a user choice for the TLS VPNs, is also configurable for IPsec ESP and ESP+AH in tunnel mode, is not supported in all the other cases.

- *Integrity of the internal header* is mandatory for the TLS VPNs, likewise for IPsec AH/ESP+AH both in transport and tunnel mode. Is a user choice for IPsec ESP in tunnel mode. Is not supported for IPsec ESP in transport mode.

- *Integrity of the internal payload* is mandatory for the TLS VPNs, likewise for IPsec AH/ESP+AH both in transport and tunnel mode. Is a user choice for IPsec ESP in both tunnel and transport mode.
- *Integrity of the additional header* is implementable and mandatory only for IPsec AH or ESP+AH on tunnel mode.
- *Peer authentication*, as we have seen in the previous sections is possible to implement using only TLS based VPNs. Client authentication is not mandatory.

Chapter 3

Network Automation

This chapter begins by introducing the concept of Service Function Chain (SFC), which is a series of network functions designed to provide communication services. However, the limitations of this concept are highlighted, particularly its lack of flexibility and agility due to the underlying network architecture principles.

To overcome these limitations, two important innovations in computer networks are introduced: Software-Defined Network (SDN) and Network Function Virtualization (NFV). SDN allows for dynamic decision-making in the packet forwarding process using software, resulting in a more agile and simpler to manage SFC. NFV, on the other hand, utilizes standard hardware to run network function images, making the SFC more flexible.

The final section focuses on automating network and security management using the aforementioned innovations and an improved SFC. This approach provides several benefits, especially in the security field where rapid response to cyberattacks is critical. An example of a framework that implements this approach is also presented, along with its workflow.

3.1 Service Function Chain

To implement an end-to-end service, the service designer must carefully select a suitable set of functions and ensure that traffic flows through them in a predetermined order to fulfil user requests.

The formal definitions of Service Function and Service Function Chain have been outlined in RFC 7665 [1]:

> **Service Function** is responsible for the specific processing of received packets. It can operate at different layers of a protocol stack, such as the network layer or other OSI layers. The service function can be implemented as a virtual component or integrated into a physical network element. It is possible

to embed one or more service functions within a single network element. Additionally, multiple instances of the service function can coexist within the same administrative domain.

Service Function Chain establishes a predefined sequence of abstract service functions along with ordering constraints. These functions and constraints are applied to packets, frames, and flows that have been selected based on classification.

The correct working of the end-to-end service depends on the fixed positioning of network functions in the chain, so packets flow through them in a specific and predetermined order.

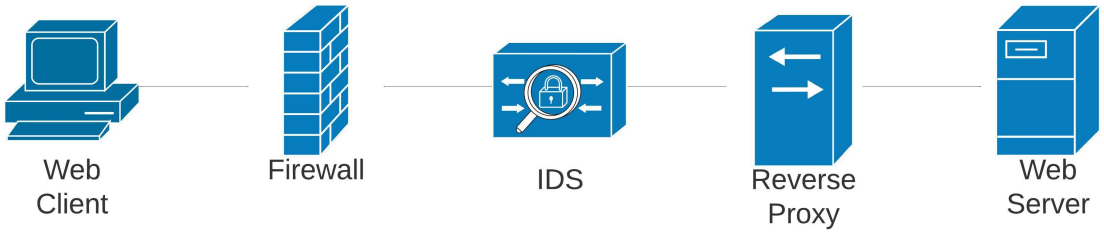

Figure 3.1: Example of Service Function Chain

Historically, network functions were physical devices dedicated to specific purposes. These devices were designed and built to fulfill their respective functions. Consequently, network function chains composed of such devices faced significant limitations:

- Suboptimal utilization of hardware resources: Physical appliances often experienced imbalanced resource usage. Some devices may handle heavy workloads while others remain underutilized. Since each function was implemented on a separate physical box with dedicated hardware, it was not feasible to share hardware resources among different services.

- Service disruption during modification: Making changes to a service chain, such as adding or removing a function, resulted in service disruption. This was because physical cables needed to be unplugged, functions added or removed, and cables reconnected. This process caused interruptions in service delivery.

- Cost and time-consuming installation: When new functions needed to be added to a service function chain, dedicated hardware had to be purchased for each function. Additionally, the installation process was often time-consuming and complex.

- Difficulty in differentiating services for users: Users typically require different services, but traditional service function chains lacked the necessary flexibility to accommodate this. Managing user-specific configurations and privileges,

such as per-application configuration, within a service function chain was challenging and not always achievable.

However, new paradigms have emerged in recent years to overcome these issues and implement more flexible and performant solutions.

3.2 Software Defined Network and Network Function Virtualization

Software-defined networking (SDN) technology provides a new approach to network architectures in cloud computing, which enables improved performance by simplifying their configuration and administration.
SDNs are networks consisting of network devices, both physical and virtual, where packet forwarding is determined by software. This software has the ability to influence network paths, making it a dynamic process that can change packet forwarding based on certain events. SDNs are built on three key concepts:

- Decoupling of the data plane and control plane: The control plane encompasses all the functions and processes responsible for determining the path to be used for packet or frame transmission. On the other hand, the data plane includes all the functions and processes that forward packets from one interface to another based on the logic defined in the control plane.

- Centralized control plane: This is where the intelligence of SDN technology resides. The control plane can be logically centralized or physically centralized. It is preferable to have a logically centralized control plane to avoid single points of failure and scalability issues.

- Southbound and northbound interfaces: The southbound interface enables communication between the SDN controller and the forwarding infrastructure. It allows the controller to manage and control the network devices. The northbound interface, on the other hand, facilitates interaction between the SDN controller and user-level applications or higher-level controllers. It provides a means for the controller to receive instructions or information from applications or other controllers.

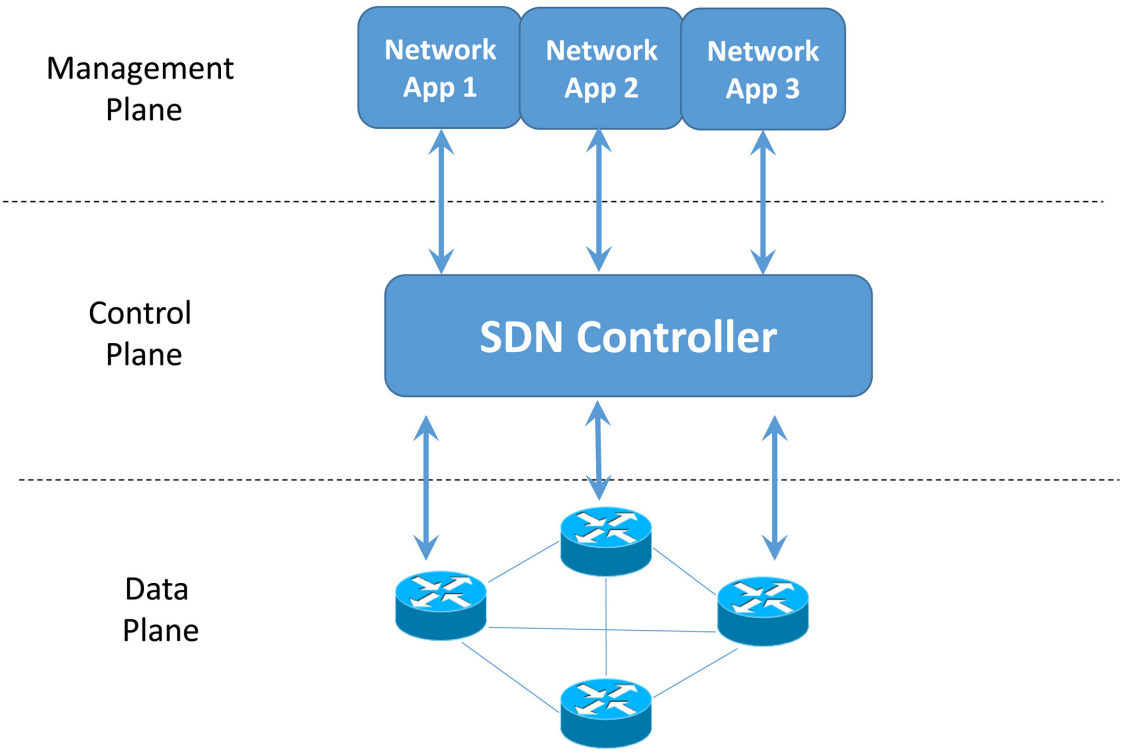

Figure 3.2: SDN Architecture

The picture shows how the SDN controller communicates with the forwarding infrastructure through the southbound interface and interacts with the application at the user level or a higher-level controller through the northbound interface.Is important to remark that while centralizing network intelligence can address most of the limitations of traditional network architectures, this approach has some disadvantages related to security, scalability, and elasticity, which are the major concerns of SDN technology.

By utilizing SDN technology, the limitations of service function chains as previously described can be addressed. The service functions comprising an SFC are akin to hardware appliances and can be interconnected via an SDN switch managed entirely by the controller. Consequently, the controller would be capable of directing packet flow and controlling the sequence of middlebox traversal.
Despite the many benefits of this SFC implementation, such as enhanced agility in delivering new services, greater reliability, maintenance, and the capacity to differentiate service chains for each user, remains the challenge of partitioning physical appliances. Since these devices remain hardware-based, it is still impractical to share resources between them, and the installation of new functions remains a slow process.

An improvement to the architecture of a service function chain, assuming that each service function corresponds to a hardware appliance, could be achieved by linking them using an SDN approach. This would enable the forwarding behaviour of each hardware appliance to be controlled entirely by the SDN controller.

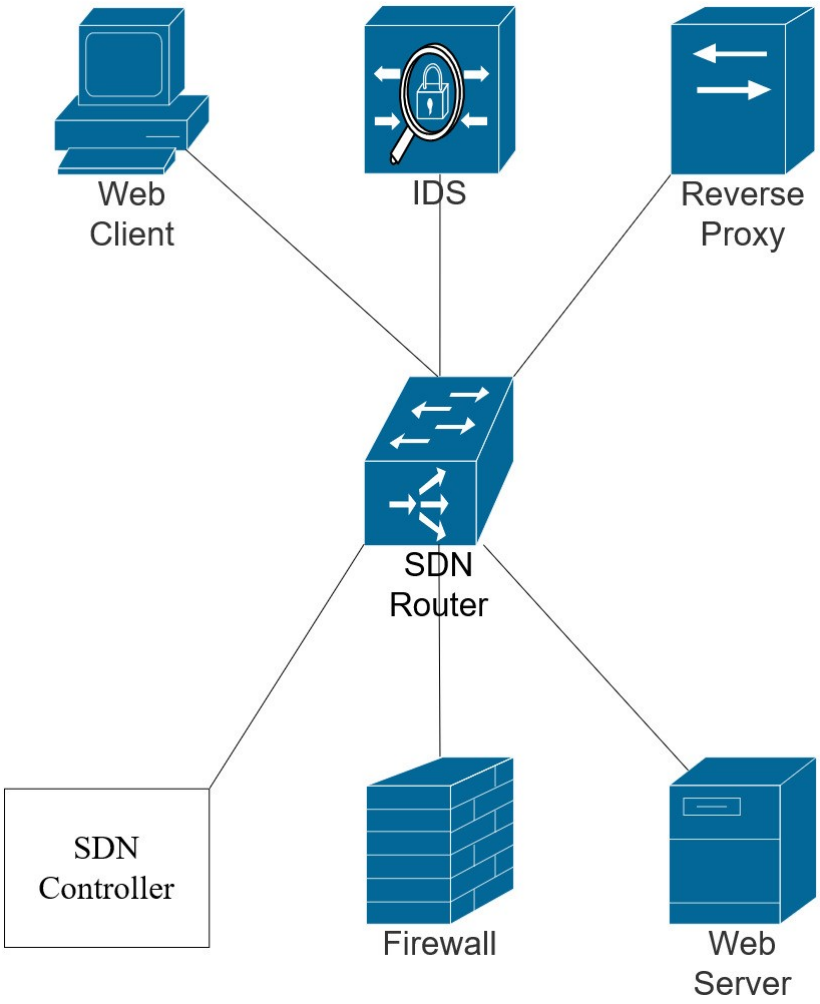

Figure 3.3: Example of SFC using SDN

The Figure illustrates how the service function chain depicted in Figure 2.1 can be modeled within an SDN architecture. In this setup, the SDN controller determines the path that the network functions, connected to the SDN switch, should follow.
Implementing a service function chain using this approach offers several advantages:

- Agility in provisioning new services: With software-based routing instead of physical wiring, new services can be provisioned quickly after installing the box.
- Maintenance and reliability: Cabling is only required once, simplifying maintenance and improving reliability.
- Differentiated service chains for users: Software-based routing allows for dynamic decision-making based on various parameters, such as application layer content. This enables the creation of different service chains for different users.

However, there are still some challenges to address. Partitioning physical appliances among different tenants remains difficult, and computational resources cannot be

shared among hardware-based appliances. Additionally, installing a new function in this setup is a slow operation.

In order to overcome the limitations of SFC, the next step involves virtualizing the network and devices. Network Function Virtualization (NFV) leverages virtualization to create virtual structures for network functions, allowing software to be separated from hardware. there are several literature article about this topic, such as: [2]

From the above literature we can identify four main components that define NFVs:

- Fast standard hardware: NFV utilizes off-the-shelf hardware that meets industry standards. This hardware is capable of efficiently running network functions.

- Software-based network function: Network functions are implemented as software images running on standard servers, enabling them to be added and removed quickly in response to user requests.

- Computing virtualization: Computing virtualization enables a single server to host multiple network functions concurrently. By virtualizing the resources of the server, multiple instances of network functions can be deployed and managed independently.

- Standard API: NFV relies on standardized Application Programming Interfaces (APIs) to enable interoperability and facilitate communication between different network functions. These APIs define the interfaces and protocols for interaction and integration between various network functions in the NFV environment.

An important feature of NFV is the capability of auto-scaling Virtualized Network Functions (VNFs) when additional resources are required. Two different strategies can be adopted for auto-scaling:

Scale Up: This strategy involves allocating more resources to a VNF when it needs them, such as increasing CPU, memory, or disk space. However, there are two important considerations when implementing this scaling approach. Firstly, it may not always be feasible if the physical server hosting the VNF has reached its resource limits. Secondly, simply assigning more resources may not necessarily improve the situation. For example, if the software is not designed to utilize multiple threads, adding more CPUs will not necessarily solve performance issues.

Scale Out: In this strategy, the VNF is duplicated multiple times to create additional instances of the same function. This approach offers parallelization and flexibility. The workload of each instance is often monitored and balanced by a load balancer, ensuring efficient distribution of traffic among the instances.

In summury NFV offers several advantages, including increased flexibility, scalability, and cost savings:

- Flexibility: By separating software from hardware, different activities can be performed on the same hardware device. Providers can select devices from various vendors, giving them more flexibility in choosing the best hardware for their network architecture.

- Scalability: The network can scale to meet current demands by allocating more CPU, memory, or bandwidth to devices, reducing service implementation times.

- Cost: Running functions on standard hardware devices reduces network costs, making it easier for network administrators to create hardware devices as efficiently as possible to meet network needs.

3.3 Automatizing Network and Security Management

The new paradigms introduced so far have placed significant emphasis on the principle of Network Automation, which is often used in conjunction with network function virtualization and software-defined network principles.

Network Automation is the process of using software to automate network and security provisioning and management to continuously maximize network efficiency and functionality. The decision system is capable of reacting to new requirements introduced into the system and network events, taking into account the current state and behaviour of the underlying physical level.

Traditionally, IT security has been largely hardware-based and required manual provisioning and management, limiting its ability to cope with cyber attacks. The speed of reaction to cyber security incidents is critical to mitigate a growing range of unpredictable attacks.

Manually reconfiguring a set of security devices in a timely manner is almost impossible without errors. Automating network and security management allows for automatic configuration changes with lower latency.

The goal is to increase network resiliency by avoiding manual interruption caused by human reconfiguration in the event of an incident.
The software can close the loop of reactions on its own, retrieve information from the network, and promptly use it to allow the network to converge on normal behaviour. To have a properly working automatic system that can reconfigure the network upon the occurrence of different scenarios, it is crucial that the system has a comprehensive overview of the entire infrastructure.

3.3.1 VEREFOO

The VEREFOO framework, which proposes an approach to overcome the limitations of a traditional Service Function Chain, is an example of a framework that implements this approach. The framework is designed with the primary objectives of achieving optimal automatic allocation and configuration of Network Security Functions on a Service Graph. Its purpose is to fulfill a set of network security requirements specified by the security administrator using a high-level security language. This is accomplished through a refinement process. Additionally, the framework facilitates the placement of the necessary network security functions onto the servers of a physical underlying network to ensure compliance with the security constraints.

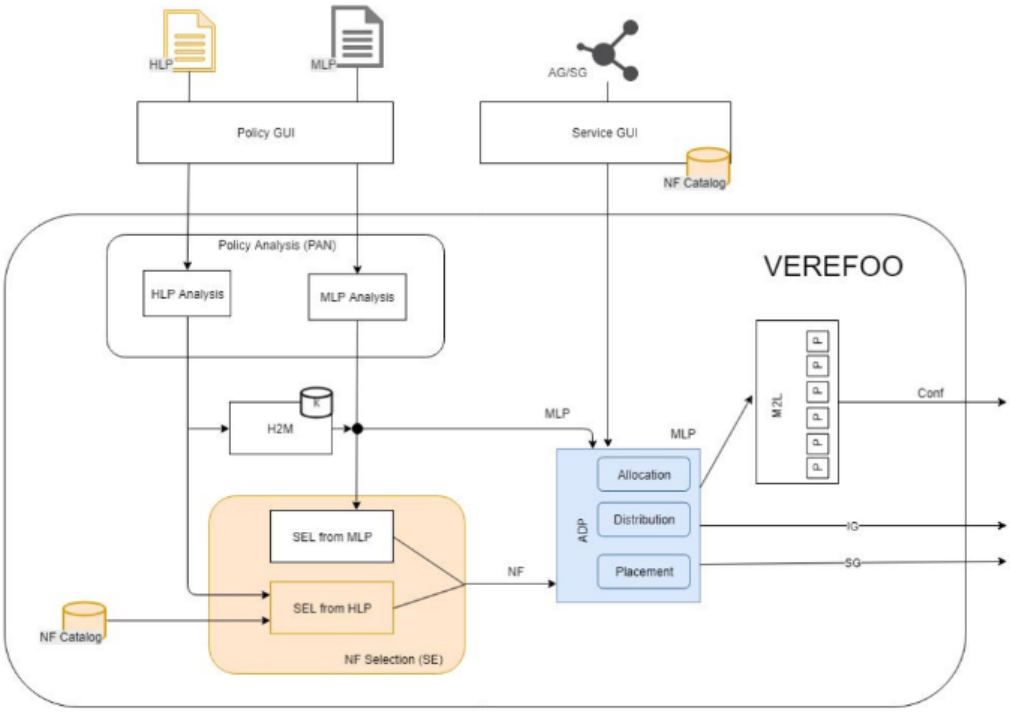

Figure 3.4: VEREFOO Workflow

The main components of the VEREFOO framework are presented in the figure to describe the complete workflow of the framework. To use VEREFOO, a user must provide two inputs: a set of Network Security Requirements and a Service Graph or an Allocation Graph.

The Policy ANalysis (PAN) module performs a conflict analysis of the Network Security Requirements to detect conflicts among requirements and create the minimal set of constraints that must be respected in the network. If an error is detected, a report is generated.

The High-to-Medium (H2M) module refines the high-level Network Security Requirements into a set of medium-level requirements containing all the necessary

information for the creation of the policies of the Network Security Functions automatically allocated on the graph and the low-level configuration of the VNFs placed on the underlying network.

The NF Selection module selects the appropriate Network Security Functions to satisfy the input Security Requirements.

At the heart of the VEREFOO architecture lies the Allocation, Distribution, and Placement (ADP) module. This module serves as the central component responsible for various tasks. It takes a collection of medium-level network security requirements, the chosen Network Security Functions, and either a Service Graph or an Allocation Graph as input. Subsequently, it generates a fresh Service Graph that incorporates the newly allocated network security functions, in addition to those already existing in the original input graph.

Before the changes introduced by this thesis work, as shown in the following articles: [3], [4], [5], [6], [7], VEREFOO already had the capability to ensure the fulfillment of properties like isolation, reachability, and protection.

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

The framework was also able to perform conflict analysis between the requested properties, as shown in the following articles: [8], [9], [10].
This thesis work focused on expanding the protection properties that an administrator could require, extending the range of potential security properties that can be requested, with a particular focus of the packet security. As a result of this thesis work, the framework is now capable of not only selecting the appropriate allocation place for the VPN gateway but also choosing the best VPN technology to implement.

Chapter 4

Thesis Objective

As discussed in the preceding sections, ensuring network security in computer systems is a critical yet highly complex task that demands careful management. Numerous articles, such as: [11],[12],[13] highlight the challenges faced by network administrators when it comes to manually configuring network security features like firewalls and channel protection systems. These tasks demand specific expertise and a high level of competency from administrators. Moreover, often are encountered issues arising from manual configuration or conflicts between policies of the same or different devices. To overcome these challenges, innovations such as Software Defined Network and Network Function Virtualization can be used to automate processes and enhance flexibility, agility, and network resilience. By adopting this approach, administrators can streamline network security management and mitigate potential issues.
Hence, the aim of this thesis is to model and implement a solution capable of fulfilling a set of Network Security Requirements that ensure secure communications through the establishment of Virtual Private Networks (VPNs) between end-to-end, site-to-site, and secure gateways. Specifically, the focus has been on extending the functionality of the ADP module of VEREFOO developing a new algorithm capable of selecting the best VPN technology between TLS and IPsec.
To achieve the aim of the thesis the preliminarty work performed in this article [14] was fundamental.

The ultimate goal of this thesis work can be seen as a subset of steps that were crucial for achieving the overall aim:

- First, a detailed study of different TLS-based VPN implementations was conducted, including OpenVPN, Tinc, and SSTP. This research was particularly important because unlike IPsec, there is no standardized protocol for implementing VPNs using TLS. Therefore, identifying the similarity and differences of each protocol was essential to develop a comprehensive solution.

- Exploring the differences between TLS-based and IPsec-based VPNs in detail was the next step. This was done with a focus on identifying the different types of security that each approach could provide.

- After this initial study phase, we moved on to a modelling and implementation phase. Based on the results of the previous research, we reformulated the Communication Protection Policies and Rule models by introducing new fields to represent various security properties that network administrators could use to safeguard the traffic or indicate a particular technology to be used. We defined a series of hard and soft constraints to enable the framework to indicate the best VPN solution based on factors such as security level, network speed, and user preferences.

- The previous phase involved a substantial amount of coding and testing to ensure that the model was robust and accurate. To evaluate the model's performance, we developed a code that generated a series of diverse tests, each with a different number and type of requested protection properties. A work of code analysis was performed along the all framework to try different approach to improve the performance and scalability of the program.

Chapter 5

Approach

In the previous chapter, we described the objective of the thesis. In this chapter, we will outline the approach taken to achieve this objective. Firstly, we will describe how we selected and compared various TLS-based VPN implementations and the following confrontation with IPsec-based VPNs.

In the second part we will describe how the problem of remodelling Communication Protection Policies was addressed.

The final section discusses how the approach utilizes a formulation of the methodology using a partial weighted MaxSMT problem. It describes the underlying principles and highlights the distinction between hard and soft constraints.

5.1 VPN Technology study

A crucial first step in this research was conducting a thorough analysis of various protocols for implementing VPNs based on TLS technology. The aim was to identify similarities and differences between them, especially regarding the security properties they provide. This was necessary because unlike IPsec, there is no standardized protocol for implementing VPNs using TLS.

The following protocols were selected for analysis: OpenVPN, Tinc, and Microsoft SSTP. These were chosen because they are widely used and provide a representative sample to generalize the characteristics of a TLS-based VPN. Moreover, OpenVPN and Tinc are open source, while Microsoft SSTP has detailed documentation, enabling an in-depth investigation of their characteristics.

After identifying the most interesting properties of each protocol, a general model was extracted. The results were compared with IPsec-based VPNs. Information on IPsec was readily available from existing literature that standardized the protocol implementation, including RFC 4301 [15], 4302 [16], and 4303 [17].

The comparison of these technologies was a crucial step before we could proceed

with the modelling work as it provided us with all the knowledge we needed to extend the capabilities of the VEREFOO framework to ensure that an administrator has a wide and meaningful selection of choices regarding the security properties required of a VPN.

5.2 Modelling Phase

After completing the study phase, the results obtained were utilized to modify existing models within the framework, effectively expanding the range of protection options available to administrators. The most significant additions were made to the Communication Protection Policies Model, which enables the definition of security requirements for communication protection, including the establishment of secure communication between two endpoints and the creation of one or more secure channels, depending on the location of untrusted nodes and inspector nodes within the network.

The Communication Protection Policies (CPP) model to be remodelled is the following:

Let P be the set of the CPPs. Each $p \in P$ is modelled as a tuple $p = (C, A^c, A^i, S, W)$.

- C is the condition set, which identifies the communications for which the policy must be enforced. It is formalized as a conjunction of five predicates on the fields of the IP 5-tuple, written as *C = (IPSrc, IPDst, pSrc, pDst, tProto)*.
- A^c and A^i represent the cipher algorithms that, among all the possible algorithms represented by the A^c and A^i sets, may be used to respectively enforce confidentiality and integrity on the traffic.
- S represents the information about how the security properties must be applied on the traffic. It is modeled as a tuple $S = s_{ci}, s_{ii}, s_{ia}$, where each component can be a ternary value: true, false, or d.c. (i.e., "don't care"). For traffic t, s_{ci} states if confidentiality must be enforced on the original internal header *t.hi* , s_{ii} states if integrity must be enforced on *t.hi* and s_{ia} states if integrity must be enforced on the additional header *t.ha* , if present.
- W represents the information about the trustworthiness of network nodes and links.

In this thesis work, the previous model has been expanded in such a way that each component of S can also take on the values prefTrue and prefFalse, to indicate a preference that does not necessarily have to be satisfied for the solution to be considered valid. This gives to the user a greater flexibility in the definition of the requirements. In addition, new components have been added to S so that it is possible to require or deny integrity on the internal payload, server peer authentication and client peer authentication.

The model used for the configuration decision has also been updated. The rules of each possible Communication Protection System, along with their associated details, are represented by a set of free variables that need to be determined by the MaxSMT solver to find their actual values. More precisely each rule r ∈ R is modelled as the following tuple:

Let R be the set of the Rules. Each $r \in R$ is modelled as a tuple $r = (C, A^c, A^i, S, m, act)$.

- *C = (IPSrc, IPDst, pSrc, pDst, tProto)* expresses the conditions that identify the traffic that rule r refers to.
- A^c and A^i respectively specify the confidentiality and integrity algorithms that are allowed to use to enforce protection properties on the traffic identified by C.
- S specifies how the security properties must be enforced by the CPS on the same traffic, and it is modelled as the S element of a policy $p \in P$.
- m is a Boolean value that expresses if the traffic satisfying C must be encapsulated through a tunnel-based VPN (i.e., when m = true) or if communication protection is enforced without adding an additional header (i.e., when m = false).
- *act*, finally, specifies if the security properties expressed by p must be applied to t (i.e., when act = protect), or must be removed from it because not necessary anymore (i.e., when act = unprotect).

Furthermore, the introduction of completely new models was necessary to incorporate the additional functionality in this thesis work. These models were designed specifically to manage the selection of the appropriate VPN technology, including its architecture and mode. The inclusion of new functions and the addition of soft and hard constraints were integral to this process.

5.3 Required protection implementation

For the main part of the thesis work, various constraints for the correct configuration of the CPP were identified and implemented. This approach is based on formulating the methodology using the Maximum Satisfiability Modulo Theories problem (MaxSMT), which extends the SMT problem to an optimization context. In MaxSMT, a set of predicate clauses with predicate variables is given, and the goal is to find optimal values for these variables that maximize the satisfiability of the clauses.

Similar to the SMT problem, MaxSMT is NP-complete in terms of worst-case computational complexity. However, the main difference is that each clause is assigned a weight, which is unitary in the standard version. Therefore, it is not enough to find a solution that satisfies the predicate clauses, but the chosen solution must

also maximize the number of satisfied clauses among all possible solutions.

There are several variants of MaxSMT, including weighted MaxSMT, partial MaxSMT, and weighted partial MaxSMT. In this thesis, the weighted partial MaxSMT variant is adopted, which involves defining two categories of constraints:

- Hard Constraints: These are non-relaxable constraints that must be fulfilled to obtain a satisfiable solution. Hard constraints are used to implement the behaviour of the components of the ADP module, ensuring the proper functioning of the entire module. For example, statements such as "the end-host can protect only its own traffic" must be implemented as hard constraints because they must always be guaranteed.

- Soft Constraints: These are relaxable constraints, and their fulfilment is not mandatory for obtaining a satisfiable solution. Since a weighted MaxSMT problem is adopted, each soft constraint is assigned a specific weight that represents its priority. The solver then selects the solution with the highest sum of all satisfied soft constraints, considering their weights. Soft constraints are used to handle preferences in reaching an optimal solution. For example, the statement "use TLS-based VPN to enforce secure communications if a NAT is present between the source and the destination" implemented as a soft constraint means that if a NAT is present in the flow, the solver tries to find a solution using TLS for secure communication, but if no solution is found, it will try to find a satisfiable solution that use another VPN technology

Chapter 6

Model

The first part of this chapter will show all the necessary changes that have been apported to VEREFOO's model, so that the framework will be capable of selecting the most appropriate VPN technology based on the preferences expressed by the administrator.
The second part will instead focus on the presentation of the model of the constraint that was necessary to add to the framework to achieve the thesis goals.

6.1 Reformulation of Communication Protection Policies and Rule

After the research work seen in the previous chapter, the first step of this work is the updating of the existing model, so that the capability of VEREFOO can be extended. In the specific the model of Communication Protection Policies and the model of the Rule has been modified respect the one already present.

The following was the original Communication Protection Policies model:

Let P be the set of the CPPs. Each $p \in P$ is modelled as a tuple $p = (C, A^c, A^i, S, W)$.

- C is the condition set, which identifies the communications for which the policy must be enforced. It is formalized as a conjunction of five predicates on the fields of the IP 5-tuple, written as *C = (IPSrc, IPDst, pSrc, pDst, tProto)*.
- A^c and A^i represent the cipher algorithms that, among all the possible algorithms represented by the A^c and A^i sets, may be used to respectively enforce confidentiality and integrity on the traffic.
- S represents the information about how the security properties must be applied on the traffic. It is modeled as a tuple $S = (s_{ci}, s_{ii}, s_{ia})$, where each component can be a ternary value: true, false, or d.c. (i.e., "don't care"). For traffic t, s_{ci} states if confidentiality must be enforced on the original internal

header $t.h^i$, , s_{ii} states if integrity must be enforced on $t.h^i$ and s_{ia} states if integrity must be enforced on the additional header $t.h^a$, if present.

- W represents the information about the trustworthiness of network nodes and links. Specifically, W is a tuple $W = (NU,NT,LU)$, where:
  - $NU \subseteq N$ is the set of untrustworthy nodes.
  - $NU \subseteq N$ is the set of inspector nodes.
  - $NU \subseteq N$ is the set of untrustworthy links.

Based on the result of the previous phase, to extend the capability of the framework, was necessary the introduction of new fields in the tuple p and the modification of the field S.

The updated model for Communication Protection Policies model is the following:

Let P be the set of the CPPs.
Each $p \in P$ is modelled as a tuple $p = (C, A^c, A^i, S, , ST, M, Ar, W)$.

The new optional fields are:

- ST that represents the security technology, in the case an administrator wants to implement a specific VPN technology. This field can be either TLS or IPsec.
- M represents the mode of the VPN, can assume these values: transport, tunnel, null or dontCare.
- Ar represents the architecture of the VPN and can be one of these values: esp, ah, esp+ah, null or dontCare.

S still represents the information about how the security properties must be applied on the traffic. It is modelled as a tuple $S = (s_{ci}, s_{ii}, s_{ia}, s_{iip}, s_{sa}, s_{ca})$, where each component can be one of the following values: true, false, d.c. (i.e., "don't care"), prefTrue, prefFalse. The first three component are still the same, the new components are:

- s_{iip} that states if integrity must be enforced on the internal payload $t.h^i$.
- s_{sa} states if server authentication must be enforced.
- s_{ca} states if client authentication must be enforced.

This is the new model for the Rules: each rule $r \in R_n$ is modelled as a tuple $r = (C, A^c, A^i, S, m, act)$, on the basis of the most common secure VPN solutions. In particular:

- *C = (IPSrc, IPDst, pSrc, pDst, tProto)* expresses the conditions that identify the traffic that rule r refers to.
- A^c and A^i respectively specify the confidentiality and integrity algorithms that are allowed to use to enforce protection properties on the traffic identified by C.
- S specifies how the security properties must be enforced by the CPS on the same traffic, and it is modelled as the S element of a policy $p \in P$.
- m is a Boolean value that expresses if the traffic satisfying C must be encapsulated through a tunnel-based VPN (i.e., when m = true) or if communication protection is enforced without adding an additional header (i.e., when m = false).
- *vProto* specifies the VPN technology.
- *arch* specifies the VPN architecture.
- *mode* specifies the VPN mode.
- *act*, finally, specifies if the security properties expressed by p must be applied to t (i.e., when act = protect), or must be removed from it because not necessary anymore (i.e., when act = unprotect).

To extend the capability of the framework was necessary the addition of the field vProto to the tuple r, to represent the chosen VPN protocol, the addition of the field arch and mode to represent the chosen architecture and mode of the VPN protocol.

6.2 VPN technology Constraints Model

In the following sections are presented a series of models of function definition, hard and soft constraints that have been implemented to extend the capabilities of VEREFOO, to achieve the thesis goal.

This section in the specific will first show the function usedTechnology, which formulation is needed to select the VPN technology. After that a series of constraints that are needed to assure that the correct choice is taken are presented.

$USED\_TECHNOLOGY(n, f, t) \Rightarrow Boolean\ with\ n \in N, f \in F, t \in T$

where:

- N is the set of all network nodes.
- F is the set of all network flows.
- T is the set of all virtual private network technologies

The first example of constraint that has been introduced is the following, which assures that the chosen algorithms for confidentiality and integrity are both available in the cipher suite of the selected VPN protocol.

CS_i and CS_t respectively indicate the cipher suite of the IPsec and TLS VPN protocol.

$$a_c \in CS_i \wedge a_i \in CS_i \vee a_c \in CS_t \wedge a_i \in CS_t \tag{6.1}$$

The next constraint states that if the current node VpnCapIsUsed is equal to true, vProto must be equal to securityTechnology set by the user except if securityTechnology is equal to d.c.

$$\begin{gathered} securityTechnology \neq d.c \wedge VpnCapIsUsed = true \\ \Rightarrow vProto = securityTechnology \end{gathered} \tag{6.2}$$

where VpnCapIsUsed is a Boolean Expression that is used extensively in the following constraints. When is true indicates that the current node VPN capabilities are actually used, on the contrary assume the value false.

The following constraint express the necessity for vProto to be equal to IPSEC or TLS in case VpnCapIsUsed is true.

$$VpnCapIsUsed = true \Rightarrow vProto = IPSEC \vee vProto = TLS \tag{6.3}$$

The following constraints will focus particularly on the implementation of the security requirements highlighted in the last section of the previous chapter.
The first one is the following: if the current node VpnCapIsUsed is equal to true and IntegrityAdditionalHeader is equal to true then vProto must be IPSEC.

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ IntegrityAdditionalHeader = true \\ \Rightarrow vProto = IPSEC \end{gathered} \tag{6.4}$$

In case IntegrityAdditionalHeader is equal to prefTrue the previous constraint will be a soft constraints instead of a hard one.

The next constraint states that if the current node VpnCapIsUsed is equal to true and ServerAuthentication is equal to false then vProto cannot be TLS. Like in the previous case there is a soft constraint similar to the hard one if ServerAuthentication is equal to prefFalse

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ ServerAuthentication = false \\ \Rightarrow vProto \neq TLS \end{gathered} \tag{6.5}$$

The following constraints make so that if the current node VpnCapIsUsed is equal to true and IntegrityInternalHeader is equal to false then vProto cannot be TLS.

Also, for this constraint there is a soft one in case IntegrityInternalHeader is equal to prefFalse.

$$VpnCapIsUsed = true \ \wedge \ IntegrityInternalHeader = false \Rightarrow vProto \neq TLS \tag{6.6}$$

Another constraints assert that if the current node VpnCapIsUsed is equal to true and IntegrityInternalPayload is equal to false then vProto cannot be TLS. Like the previous one there is a similar soft constraint in case IntegrityInternalPayload is equal to prefFalse.

$$VpnCapIsUsed = true \ \wedge \ IntegrityInternalPayload = false \Rightarrow vProto \neq TLS \tag{6.7}$$

In summary, in this section, we have defined a set of hard and soft constraints for selecting the appropriate VPN technology. These constraints ensure that the selected technology supports the required encryption and authentication algorithms, allow the administrator to request a specific technology, mandate the use of a technology, and guarantee that the chosen technology satisfies the user's security requirements.

6.3 Mode Constraints Model

In this section we will continue to showcase the modelization work focusing on the constraint needed to select the correct mode based on the security properties requested. We begin presenting the function mode, which formulation is needed to select the VPN mode.

$MODE(n, f, m) \Rightarrow Boolean \ with \ n \in N, f \in F, m \in M$
where:

- N is the set of all network nodes.
- F is the set of all network flows.
- M is the set of all the possible mode a VPN can be set. In the case of this work transport, tunnel, null.

The first constraint asserts that if the current node VpnCapIsUsed is equal to true and IntegrityAdditionalHeader is equal to true then mode cannot be TRANSPORT. In case IntegrityAdditionalHeader is equal to prefTrue the constraint will be a soft constraints instead

$$VpnCapIsUsed = true \ \wedge \ IntegrityAdditionalHeader = true \Rightarrow mode \neq TRANSPORT \tag{6.8}$$

The succeeding constraint states that if the current node VpnCapIsUsed is equal to true and ConfidentialityInternalHeader is equal to true then mode cannot be

TRANSPORT. Also in this case there is a corresponding soft constraint in case ConfidentialityInternalHeader is equal to prefTrue.

$$VpnCapIsUsed = true \ \wedge \ ConfidentialityInternalHeader = true \Rightarrow mode \neq TRANSPORT \tag{6.9}$$

The following constraints affirms that if the current node VpnCapIsUsed is equal to true and requestedMode is equal to d.c. and or ConfidentialityInternalHeader is equal to true or integrityAdditionalHeader is true or securityTechnology is equal to TLS then mode cannot be TRANSPORT.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedMode = d.c \wedge \\ (ConfidentialityInternalHeader = true \vee \\ integrityAdditionalHeader = true \vee \\ securityTechnology = TLS) \\ \Rightarrow mode \neq TRANSPORT \end{gathered} \tag{6.10}$$

This constraint instead indicates that if the current node VpnCapIsUsed is equal to true and requestedMode is not equal to d.c and requestedMode is not equal to TRANSPORT then mode cannot be TRANSPORT.

$$VpnCapIsUsed = true \wedge requestedMode = d.c \wedge requestedMode \neq TRANSPORT \Rightarrow mode \neq TRANSPORT \tag{6.11}$$

The succeeding constraint asserts that if the current node VpnCapIsUsed is equal to true and requestedMode is equal d.c. and securityTechnology is equal to TLS then mode cannot be TUNNEL.

$$VpnCapIsUsed = true \wedge requestedMode = d.c \wedge securityTechnology = TLS \Rightarrow mode \neq TUNNEL \tag{6.12}$$

The following constraints states that if the current node VpnCapIsUsed is equal to true and requestedMode is not equal to d.c. and requestedMode is not equal to TUNNEL then mode cannot be TUNNEL.

$$VpnCapIsUsed = true \wedge requestedMode = d.c \wedge requestedMode \neq TUNNEL \Rightarrow mode \neq TUNNEL \tag{6.13}$$

This constraint instead affirms that if the current node VpnCapIsUsed is equal to true and requestedMode is equal to d.c. and securityTechnology is equal to IPSEC then mode cannot be NULL.

$$VpnCapIsUsed = true \wedge requestedMode = d.c \wedge securityTechnology = IPSEC \Rightarrow mode \neq NULL \tag{6.14}$$

The next constraints asserts that if the current node VpnCapIsUsed is equal to true

and requestedMode is not to d.c. and requestedMode is not equal to NULL then mode cannot be NULL.

$$VpnCapIsUsed = true \land requestedMode = d.c \land requestedMode \neq NULL \Rightarrow mode \neq NULL \tag{6.15}$$

The constraint that comes after states that if the current node VpnCapIsUsed is equal to true and securityTechnology is equal to TLS then mode cannot be TRANSPORT and mode cannot be TUNNEL.

$$VpnCapIsUsed = true \land securityTechnology = TLS \Rightarrow mode \neq TRANSPORT \land mode \neq TUNNEL \tag{6.16}$$

The following constraints affirms if the current node VpnCapIsUsed is equal to true and securityTechnology is equal to IPSEC then mode cannot be NULL.

$$VpnCapIsUsed = true \land securityTechnology = IPSEC \Rightarrow mode \neq NULL \tag{6.17}$$

This constraint instead indicates that If the current node VpnCapIsUsed is equal to true then mode must be TRANSPORT or mode must be TUNNEL or mode must be NULL.

$$VpnCapIsUsed = true \Rightarrow mode = TRANSPORT \lor mode = TUNNEL \lor mode = NULL \tag{6.18}$$

In summary, in this section, we have defined a set of hard and soft constraints for selecting the appropriate mode for implementing the VPN. These constraints ensure that the chosen mode is compatible with the selected VPN technology, allow the administrator to request a specific mode, require the selection of a mode, and guarantee that the chosen mode meets the user's security requirements.

6.4 Architecture Constraints Model

In this section, our focus remains on showcasing the modelization work, specifically highlighting the constraint required for selecting the appropriate architecture based on the requested security properties. We commence by introducing the architecture function, which plays a pivotal role in determining the VPN architecture to be selected.

$ARCHITETECTURE(n, f, a) \Rightarrow Boolean\ with\ n \in N, f \in F, a \in A$

where:

- N is the set of all network nodes.
- F is the set of all network flows.

- A is the set of all the possible architecture a VPN can be set. In the case of this work esp, ah,esp+ah, null.

The first constraint asserts that if the current node VpnCapIsUsed is equal to true and IntegrityAdditionalHeader is equal to true then architecture cannot be ESP. In case IntegrityAdditionalHeader is equal to prefTrue the constraint will be a soft constraint instead.

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ IntegrityAdditionalHeader = true \\ \Rightarrow arch \neq ESP \end{gathered} \tag{6.19}$$

The succeeding constraint states that if the current node VpnCapIsUsed is equal to true and IntegrityInternalHeader is equal to false then architecture cannot be AH and cannot be ESP+AH. Also for this constraint exist a similar soft constraint in case IntegrityInternalHeader is equal to prefFalse.

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ IntegrityInternalHeader = false \\ \Rightarrow arch \neq AH \ \wedge \ arch \neq ESP + AH \end{gathered} \tag{6.20}$$

The following constraints affirms that if the current node VpnCapIsUsed is equal to true and IntegrityInternalPayload is equal to false then architecture cannot be AH and cannot be ESP+AH. Like in the previous cases there is a soft constraint if IntegrityInternalPayload is equal to prefFalse.

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ IntegrityInternalPayload = false \\ \Rightarrow arch \neq AH \ \wedge \ arch \neq ESP + AH \end{gathered} \tag{6.21}$$

This constraint instead indicates that if the current node VpnCapIsUsed is equal to true and ConfidentialityInternalHeader is equal to true then architecture cannot be AH. In a similar way there is a corresponding soft constraint in case ConfidentialityInternalHeader is equal to prefTrue.

$$\begin{gathered} VpnCapIsUsed = true \ \wedge \ ConfidentialityInternalHeader = true \\ \Rightarrow arch \neq AH \end{gathered} \tag{6.22}$$

The succeeding constraint asserts that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is equal d.c. and or IntegrityAdditionalHeader is equal to true or securityTechnology is equal to TLS then architecture cannot be ESP.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ (IntegrityAdditionalHeader = true \vee securityTechnology = TLS) \\ \Rightarrow architecture \neq ESP \end{gathered} \tag{6.23}$$

The following constraints states that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is not equal to d.c and requestedArchitecture is not equal to ESP then architecture cannot be .

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ requestedArchitecture \neq ESP \\ \Rightarrow architecture \neq ESP \end{gathered} \tag{6.24}$$

This constraint instead affirms that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is equal to d.c. and or ConfidentialityInternalHeader is equal to true or IntegrityInternalHeader is equal to false or IntegrityInternalPayload is equal to false or securityTechnology is equal to TLS then architecture cannot be AH.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ (ConfidentialityInternalHeader = true \vee \\ IntegrityInternalHeader = false \vee IntegrityInternalPayload = false \vee \\ securityTechnology = TLS) \\ \Rightarrow architecture \neq AH \end{gathered} \tag{6.25}$$

The next constraints asserts that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is not equal to d.c. and requestedArchitecture is not equal to AH then architecture cannot be AH.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ requestedArchitecture \neq AH \\ \Rightarrow architecture \neq AH \end{gathered} \tag{6.26}$$

The constraint that comes after states that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is equal d.c. and or IntegrityInternalHeader is equal to false or IntegrityInternalPayload is equal to false or securityTechnology is equal to TLS then architecture cannot be ESP+AH.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ (IntegrityInternalHeader = false \vee IntegrityInternalPayload = false \vee \\ securityTechnology = TLS) \Rightarrow architecture \neq ESP + AH \end{gathered} \tag{6.27}$$

The following constraints affirms if the current node VpnCapIsUsed is equal to true and requestedArchitecture is not equal to d.c. and requestedArchitecture is not equal to ESP+AH then architecture cannot be ESP+AH.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ requestedArchitecture \neq ESP + AH \\ \Rightarrow architecture \neq ESP + AH \end{gathered} \tag{6.28}$$

The succeeding constraint states that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is equal d.c. and or IntegrityInternalHeader is equal to false or IntegrityInternalPayload is equal to false or securityTechnology is equal to IPSEC then architecture cannot be NULL.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ (IntegrityInternalHeader = false \vee IntegrityInternalPayload = false \vee \\ securityTechnology = IPSEC) \\ \Rightarrow architecture \neq NULL \end{gathered} \tag{6.29}$$

This constraint instead affirms that if the current node VpnCapIsUsed is equal to true and requestedArchitecture is not equal to d.c. and requestedArchitecture is not equal to NULL then architecture cannot be NULL.

$$\begin{gathered} VpnCapIsUsed = true \wedge requestedArchitecture = d.c \wedge \\ requestedArchitecture \neq NULL \\ \Rightarrow architecture \neq NULL \end{gathered} \tag{6.30}$$

The next constraints asserts that if the current node VpnCapIsUsed is equal to true and securityTechnology is equal to TLS then architecture cannot be ESP and architecture cannot be AH and architecture cannot be ESP+AH.

$$\begin{gathered} VpnCapIsUsed = true \wedge securityTechnology = TLS \\ \Rightarrow architecture \neq ESP \wedge architecture \neq AH \wedge architecture \neq ESP + AH \end{gathered} \tag{6.31}$$

The constraint that comes after states If the current node VpnCapIsUsed is equal to true and securityTechnology is equal to IPSEC then architecture cannot be NULL.

$$\begin{gathered} VpnCapIsUsed = true \wedge securityTechnology = IPSEC \\ \Rightarrow architecture \neq NULL \end{gathered} \tag{6.32}$$

The following constraints affirms if the current node VpnCapIsUsed is equal to true then architecture must be ESP or architecture must be AH or architecture must be ESP+AH or architecture must be NULL.

$$\begin{gathered} VpnCapIsUsed = true \\ \Rightarrow architecture = ESP \vee architecture = AH \vee \\ architecture = ESP + AH \vee architecture = NULL \end{gathered} \tag{6.33}$$

The rest of constraints presented in this section are always about the correct choice of VPN technology mode and architecture, but they affect more than one of those fields at the time.

The first constraint asserts that if the current node VpnCapIsUsed is equal to true and IntegrityAdditionalHeader is equal to false then or mode is not TUNNEL or architecture is not AH. In case IntegrityAdditionalHeader is equal to prefFalse the constraint will be a soft constraint instead.

$$\begin{gathered} VpnCapIsUsed = true \wedge (IntegrityAdditionalHeader = false \\ \Rightarrow mode \neq TUNNEL \vee (architecture \neq AH \wedge architecture \neq ESP + AH) \end{gathered} \tag{6.34}$$

The constraint that comes after states that if the current node VpnCapIsUsed is equal to true and IntegrityInternalHeader is equal to true then or mode is not TRANSPORT or architecture is not ESP. Also, for this constraint exist a similar soft constraint in case IntegrityInternalHeader is equal to prefTrue.

$$\begin{gathered} VpnCapIsUsed = true \wedge IntegrityInternalHeader = true \\ \Rightarrow mode \neq TRANSPOSRT \vee architecture \neq ESP \end{gathered} \tag{6.35}$$

The last constraint states that both end of the VPN must have the same value for vProto, for mode and for architecture. Using i and j to indicate the start and end node properties:

$$vProto_i = vProto_j \wedge mode_i = mode_j \wedge architecture_i = architecture_j \quad (6.36)$$

In conclusion, this section has introduced a range of hard and soft constraints for the selection of the appropriate architecture in implementing the VPN. These constraints ensure compatibility between the chosen architecture and the VPN technology, provide flexibility for the administrator to specify a preferred architecture, enforce the selection of an architecture, and ensure that the chosen architecture satisfies the required security standards. Furthermore, constraints have been defined to maintain consistency across the entire VPN, ensuring that the identified technology, mode, and architecture remain uniform at both ends of the network.

Chapter 7

Implementation, Validation and Performance Analysis

This chapter examines the functionality of the framework and assesses its computational performance under varying conditions of Allocation Places and Network Security Requirements. The initial segment focuses on the implementation of the new functionalities, the second section focuses on testing the accuracy of the implementation through specific tests that verify the outputs obtained given the inputs provided. These tests are illustrated using five distinct Use Case examples.

Subsequently, the last part of the chapter evaluates the computational performance of the framework, taking into account the modifications made as part of this thesis work. It identifies the parameters that have the greatest impact on the system's scalability. The results of these tests are described and depicted through graphs that consider the total execution time of the framework.

7.1 Implementation

The entire development of this project utilized Java as the programming language, the Z3 Java library for implementing the partial weighted MaxSMT problem, and the XML markup language for structuring the communication protection requirements and CPSs configurations.

7.1.1 XML schema

The XML schema includes a section called "propertyDefinition" that is dedicated to network security requirements. This section can accommodate anywhere from one to "n" Property elements. Each Property element corresponds to a distinct security requirement specified by the security administrator. The structure of the property element has been expanded and modified to allow the implementation of the new functionality; the final structure is the following:

- **name** is the kind of requirement (i.e. protection);

- **src** is the identifier of the source node of the requirement;
- **dst** is the identifier of the destination node of the requirement;
- **src_port** is the number or interval of numbers for the source port;
- **dst_port** is the number or interval of numbers for the destination port;
- **l4_proto** is the type of layer 4 protocol
- **protectionInfo** contains additional information about the requested protection properties. Is composed by the following field:
  - **authenticationAlgorithm** is the authentication algorithm;
  - **encryptionAlgorithm** is the encryption algorithm;
  - **untrustedNode** contains all the untrusted nodes of the requirement;
  - **inspectorNode** contains all the inspector nodes of the requirement;
  - **untrustedLink** contains all the untrusted link of the requirement;
  - **securityTechnology** contains the security technologies which can be used (i.e. IPsec or TLS).
  - **securityTechnologyInfo** contains additional information about the used security technology. is composed of the field mode (i.e. Tunnel, Transport, Null) and the field architecture (i.e. ESP, AH, ecc.);
  - **requiredProtection** contains all the protection on the packet that the admin can request (i.e. Confidentiality Internal Header, Integrity Internal Header, ecc.);

A series of example of this schema are presented in the following section where five different use cases are presented.

7.1.2 MaxSMT Problem

Z3, developed by Microsoft Research, is an open-source Satisfiability Modulo Theories (SMT) solver. It is designed to address problems that arise in software verification and software analysis. Z3 is available under the MIT license and is compatible with Windows, OSX, Linux, and FreeBSD operating systems. It provides a variety of APIs, including C, C++, Java, .Net, OCaml, and Python, allowing users to invoke Z3 programmatically.

The Z3 java library has been exploited to set up the MaxSMT problem and implement the long series of constraints that have been presented in the previous chapter. In the java class NetContext the functions usedTechnology, vpnModeFinal, vpnArchFinal have been defined as presented in the following listing.

```
usedTechnology = ctx.mkFuncDecl("usedTechnology", new Sort[]{ nodeType,
    ctx.mkIntSort(), ctx.mkStringSort()},ctx.mkBoolSort());
vpnArchFinal = ctx.mkFuncDecl("vpnArchFinal", new Sort[]{nodeType, ctx.mkIntSort(),
    ctx.mkStringSort()},ctx.mkBoolSort());
vpnModeFinal = ctx.mkFuncDecl("vpnModeFinal", new Sort[]{nodeType, ctx.mkIntSort(),
    ctx.mkStringSort()},ctx.mkBoolSort());
```

Listing 7.1: usedTechnology, vpnModeFinal, vpnArchFinal Function Definition

Considering the first function, the code specifies that a predicate named usedTechnology is declared and can assume a different boolean value for each tuple [nodeType, Int, String], where the nodeType represents a node in the considered topology, the int type represents the FlowId of the considered flow, and the String type represent the value of the VPN technology considered.

The constraints where instead implemented in the Checker class inside the function createProtectionConstraints. As an example is presented the implementation of the hard constraints that link the architecture of the VPN to the correct VPN technology.

In the previous chapter, the constraint model for this scenario was presented, and it was established that only the IPsec technology can have ESP, AH, or ESP+AH as architecture options. Therefore, the code above ensures that if the current node utilizes its VPN capacity and the VPN technology is TLS, the architecture cannot assume any of the mentioned values.

```
//hard constraint to link arch type to the correct vpn technology
constraintList.add(ctx.mkImplies( ctx.mkAnd(node.getPlacedNF().getVpnCapIsUsed(),
    usedTecIPSECTrue), ctx.mkEq(nctx.vpnArchFinal.apply(node.getZ3Name(),
    ctx.mkInt(flow.getIdFlow()), ctx.mkString("NULL")), ctx.mkFalse())));
constraintList.add(ctx.mkImplies( ctx.mkAnd( node.getPlacedNF().getVpnCapIsUsed(),
    usedTecTLSTrue), ctx.mkAnd(ctx.mkEq(nctx.vpnArchFinal.apply(node.getZ3Name(),
    ctx.mkInt(flow.getIdFlow()), ctx.mkString("ESP")), ctx.mkFalse()),
    ctx.mkEq(nctx.vpnArchFinal.apply(node.getZ3Name(), ctx.mkInt(flow.getIdFlow()),
    ctx.mkString("AH")), ctx.mkFalse()),
    ctx.mkEq(nctx.vpnArchFinal.apply(node.getZ3Name(),
    ctx.mkInt(flow.getIdFlow()),ctx.mkString("ESP_AH")), ctx.mkFalse()))));
```

Listing 7.2: Hard Constraint Example

7.2 Validation

Numerous JUnit tests were defined to verify the correct implementation of the code added in this thesis work. Numerous network topology and all the various possible combinations of security properties that an admin may require were considered.
In this section five use cases are presented to showcase all the explained features and provide a practical understanding of how the framework operates. These use cases encompass all the features discussed earlier, providing a comprehensive understanding of how the ADP module operates.

Since the main focus of this thesis work is the selection of the optimal VPN technology and its associated characteristics, the presented use cases utilize a simplified allocation graph as input. The graph, as depicted in the accompanying picture, consists of a client and a server situated at the network endpoints, an untrusted node in between, and two allocation places where the VPN gateway will be allocated.

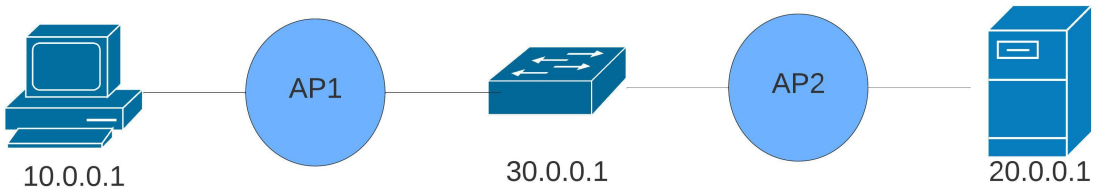

Figure 7.1: Input Service Graph

7.2.1 Use Case 1

In the first use case, we require the confidentiality of the internal header, without specifying any other constraints, to do so we use the communication protection requirement defined as follows:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
   dst="20.0.0.1">
   <protectionInfo encryptionAlgorithm="AES_256_CBC"
      authenticationAlgorithm="SHA2_256">
      <untrustedNode node="30.0.0.1"></untrustedNode>
      <SecurityTechnologyInfo>
         <mode>dontCare</mode>
         <architecture>dontCare</architecture>
      </SecurityTechnologyInfo>
      <requiredProtection>

    <confidentialityInternalHeader>true</confidentialityInternalHeader>
         <integrityInternalHeader></integrityInternalHeader>
         <integrityAdditionalHeader></integrityAdditionalHeader>
         <integrityInternalPayload></integrityInternalPayload>
         <serverAuthentication></serverAuthentication>
         <clientAuthentication></clientAuthentication>
      </requiredProtection>
   </protectionInfo>
</Property>
```

Listing 7.3: Input Security Requirements Use Case 1

The following listing show the section of the output relative to the client node, the server one is analogous.

```
<securityAssociation>
   <behavior>ACCESS</behavior>
   <startChannel></startChannel>
   <endChannel></endChannel>
```

```
    <source>10.0.0.1</source>
    <destination>20.0.0.1</destination>
    <protocol>ANY</protocol>
     <src_port>*</src_port>
    <dst_port>*</dst_port>
    <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
    <encryptionAlgorithm>AES_256_CBC</encryptionAlgorithm>
    <SecurityTechnology>IPSEC</SecurityTechnology>
    <SecurityTechnologyInfo>
       <mode>TUNNEL</mode>
       <architecture>ESP</architecture>
    </SecurityTechnologyInfo>
</securityAssociation>
```

Listing 7.4: Output Example Use Case 1

We can see that Verefoo suggests using IPsec ESP in Tunnel mode, because it is the fastest alternative among those that satisfy the required constraints.

7.2.2 Use Case 2

In the second use case several preferences are requested, we would like to have integrity of the internal header and server authentication, but we prefer to not have confidentiality of the internal header and we would like to avoid integrity for the internal payload. However, it is not possible to satisfy all these preferences that are shown the communication protection requirement defined as follows:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
    dst="20.0.0.1">
    <protectionInfo encryptionAlgorithm="AES_256_CBC"
       authenticationAlgorithm="SHA2_256">
       <untrustedNode node="30.0.0.1"></untrustedNode>
       <SecurityTechnologyInfo>
          <mode>dontCare</mode>
          <architecture>dontCare</architecture>
       </SecurityTechnologyInfo>
       <requiredProtection>

    <confidentialityInternalHeader>prefFalse</confidentialityInternalHeader>
          <integrityInternalHeader>prefTrue</integrityInternalHeader>
          <integrityAdditionalHeader></integrityAdditionalHeader>

    <integrityInternalPayload>prefFalse</integrityInternalPayload>
          <serverAuthentication>prefTrue</serverAuthentication>
          <clientAuthentication></clientAuthentication>
       </requiredProtection>
    </protectionInfo>
</Property>
```

Listing 7.5: Input Security Requirements Use Case 2

In fact, VEREFOO shows as the solution that satisfies the greatest number of requests. In this case the solution proposed by the framework is the use of IPsec ESP in Tunnel mode. This solution, satisfy all the previously expressed preferences

with the exception of the server authentication. The output relative to the VPN is shown in this extract:

```
<securityAssociation>
   <behavior>ACCESS</behavior>
   <startChannel></startChannel>
   <endChannel></endChannel>
   <source>10.0.0.1</source>
   <destination>20.0.0.1</destination>
   <protocol>ANY</protocol>
    <src_port>*</src_port>
   <dst_port>*</dst_port>
   <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
   <encryptionAlgorithm>AES_256_CBC</encryptionAlgorithm>
   <SecurityTechnology>IPSEC</SecurityTechnology>
   <SecurityTechnologyInfo>
      <mode>TUNNEL</mode>
      <architecture>ESP</architecture>
   </SecurityTechnologyInfo>
</securityAssociation>
```

Listing 7.6: Output Example Use Case 2

7.2.3 Use Case 3

The third use case is similar to the second one, in fact we request the same properties, but this time we require the implementation of server authentication. So, in this case the communication protection requirement is defined as follows:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
   dst="20.0.0.1">
   <protectionInfo encryptionAlgorithm="AES_256_CBC"
      authenticationAlgorithm="SHA2_256">
      <untrustedNode node="30.0.0.1"></untrustedNode>
      <SecurityTechnologyInfo>
         <mode>dontCare</mode>
         <architecture>dontCare</architecture>
      </SecurityTechnologyInfo>
      <requiredProtection>

   <confidentialityInternalHeader>prefFalse</confidentialityInternalHeader>
         <integrityInternalHeader>prefTrue</integrityInternalHeader>
         <integrityAdditionalHeader></integrityAdditionalHeader>

   <integrityInternalPayload>prefFalse</integrityInternalPayload>
         <serverAuthentication>true</serverAuthentication>
         <clientAuthentication></clientAuthentication>
      </requiredProtection>
   </protectionInfo>
</Property>
```

Listing 7.7: Input Security Requirements Use Case 3

Since we asked the framework to guarantee server authentication, in the solution we can notice that this time VEREFOO identifies a TLS based VPN as the solution, like shown in this extract:

```
<securityAssociation>
   <behavior>ACCESS</behavior>
   <startChannel></startChannel>
   <endChannel></endChannel>
   <source>10.0.0.1</source>
   <destination>20.0.0.1</destination>
   <protocol>ANY</protocol>
    <src_port>*</src_port>
   <dst_port>*</dst_port>
   <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
   <encryptionAlgorithm>AES_256_CBC</encryptionAlgorithm>
   <SecurityTechnology>TLS</SecurityTechnology>
   <SecurityTechnologyInfo>
      <mode>NULL</mode>
      <architecture>NULL</architecture>
   </SecurityTechnologyInfo>
</securityAssociation>
```

Listing 7.8: Output Example Use Case 3

7.2.4 Use Case 4

The following example requires the ESP+AH architecture to be used and the integrity of the additional header to be guaranteed, the communication protection requirement is defined as follows:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
   dst="20.0.0.1">
   <protectionInfo encryptionAlgorithm="AES_256_CBC"
      authenticationAlgorithm="SHA2_256">
      <untrustedNode node="30.0.0.1"></untrustedNode>
      <SecurityTechnologyInfo>
         <mode>dontCare</mode>
         <architecture>ESP+AH</architecture>
      </SecurityTechnologyInfo>
      <requiredProtection>

    <confidentialityInternalHeader></confidentialityInternalHeader>
         <integrityInternalHeader></integrityInternalHeader>
         <integrityAdditionalHeader>true</integrityAdditionalHeader>
         <integrityInternalPayload></integrityInternalPayload>
         <serverAuthentication></serverAuthentication>
         <clientAuthentication></clientAuthentication>
      </requiredProtection>
   </protectionInfo>
</Property>
```

Listing 7.9: Input Security Requirements Use Case 4

As expected, to meet the specified demand, the solution is IPsec ESP+AH in Tunnel mode. We can notice that the fastest solution is IPsec AH, but since we have requested a specific mode the output is the following:

```
<securityAssociation>
   <behavior>ACCESS</behavior>
   <startChannel></startChannel>
   <endChannel></endChannel>
   <source>10.0.0.1</source>
   <destination>20.0.0.1</destination>
   <protocol>ANY</protocol>
    <src_port>*</src_port>
   <dst_port>*</dst_port>
   <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
   <encryptionAlgorithm>AES_256_CBC</encryptionAlgorithm>
   <SecurityTechnology>IPSEC</SecurityTechnology>
   <SecurityTechnologyInfo>
      <mode>TUNNEL</mode>
      <architecture>ESP+AH</architecture>
   </SecurityTechnologyInfo>
</securityAssociation>
```

Listing 7.10: Output Example Use Case 4

7.2.5 Use Case 5

For the last use case we use a different network, the one shown in the picture below, because we want to analyse the behaviour of the framework in the presence of a NAT.

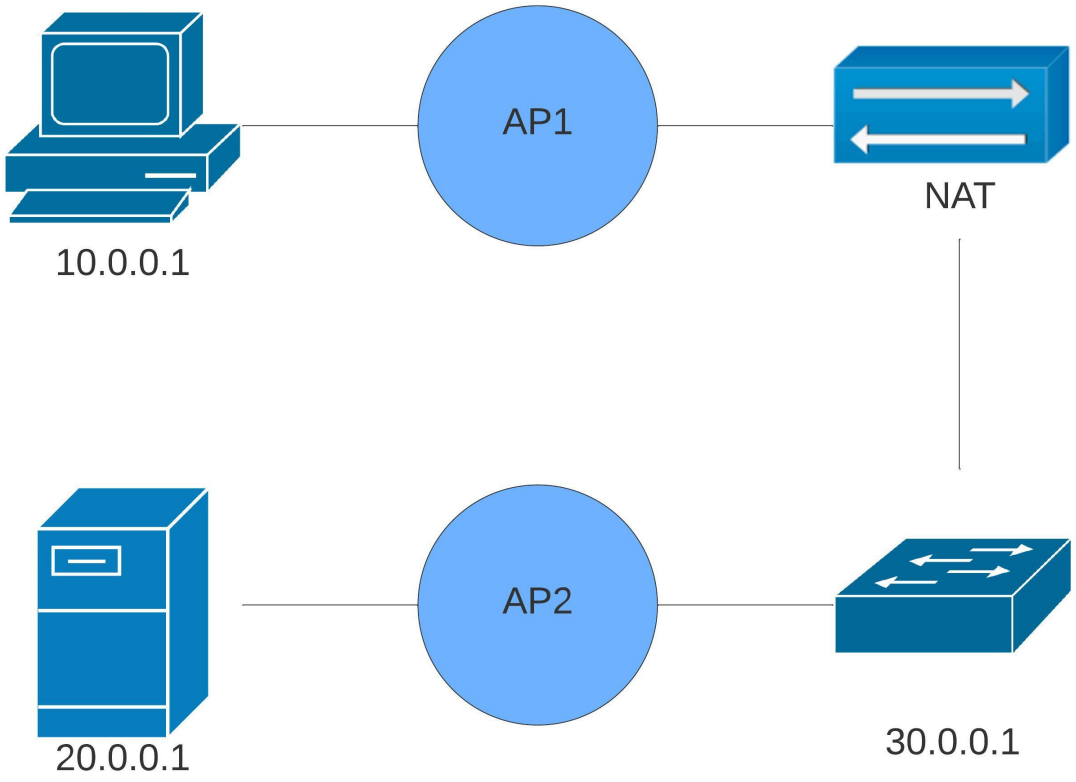

Figure 7.2: Input Service Graph with NAT

After inserting the NAT in the network, we define the communication protection requirement as follows:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
   dst="20.0.0.1">
   <protectionInfo encryptionAlgorithm="AES_256_CBC"
      authenticationAlgorithm="SHA2_256">
      <untrustedNode node="30.0.0.1"></untrustedNode>
      <SecurityTechnologyInfo>
         <mode>dontCare</mode>
         <architecture>dontCare</architecture>
      </SecurityTechnologyInfo>
      <requiredProtection>

   <confidentialityInternalHeader>true</confidentialityInternalHeader>
         <integrityInternalHeader></integrityInternalHeader>
         <integrityAdditionalHeader></integrityAdditionalHeader>
         <integrityInternalPayload></integrityInternalPayload>
         <serverAuthentication></serverAuthentication>
         <clientAuthentication></clientAuthentication>
      </requiredProtection>
   </protectionInfo>
</Property>
```

Listing 7.11: Input Security Requirements Use Case 5

As expected, given the presence of a NAT along the route, the solution suggests the use of a TLS-based VPN, as shown in the picture.

```
<securityAssociation>
   <behavior>ACCESS</behavior>
   <startChannel></startChannel>
   <endChannel></endChannel>
   <source>10.0.0.1</source>
   <destination>20.0.0.1</destination>
   <protocol>ANY</protocol>
    <src_port>*</src_port>
   <dst_port>*</dst_port>
   <authenticationAlgorithm>SHA2_256</authenticationAlgorithm>
   <encryptionAlgorithm>AES_256_CBC</encryptionAlgorithm>
   <SecurityTechnology>TLS</SecurityTechnology>
   <SecurityTechnologyInfo>
      <mode>NULL</mode>
      <architecture>NULL</architecture>
   </SecurityTechnologyInfo>
</securityAssociation>
```

Listing 7.12: Output Example Use Case 5

7.3 Performance Analysis

The final part of the thesis work focused on evaluating and improving the performance of the framework. The first part of this section analyses attempts to enhance the system's performance. Subsequently, the results obtained are presented using graphs.

From the very first preliminary tests it was found that the performance of the framework, after the newly implemented functions were insufficient. Upon analysing the code, the primary cause of this performance and scalability issue was identified as the presence of various soft constraints. In particular the ones aimed to ensure the selection of the fastest VPN solution.

Soft constraints are constraints that are not strictly enforced, allowing some degree of violation or relaxation. Unlike hard constraints, which must be satisfied without exception, soft constraints can be violated to some extent based on a cost or penalty function associated with each violation. The solver's objective is to find an assignment that minimizes the total cost of constraint violations.

The reason why soft constraints have such an impact on the framework performance is primarily the increased complexity and search space involved. Solving soft constraints typically requires exploring a larger solution space to find an assignment that minimizes the overall cost. This search process involves exploring multiple possible assignments, evaluating the cost associated with each assignment, and iteratively refining the solution to minimize the total cost.

As a result, it was decided to modify the code to provide the administrator with the option to prioritize either the search for the fastest VPN solution or overall performance. In the latter case, the framework would still offer a valid solution based on the security property requirements. However, if multiple valid solutions exist, there is no longer a guarantee that the fastest solution would be selected.

After this choice to improve the scalability of the framework, further analysis was conducted to enhance its performance even further. To achieve these objective different approaches were taken.

Firstly, the function 'supportCap' was removed.

$SUPPORT\_CAP(n, f, s) \Rightarrow Boolean\ with\ n \in N, f \in F, s \in S$

where:

- N is the set of all network nodes.
- F is the set of all network flows.
- S is the set of all possible combination of virtual private network technologies, encryption algorithm and authentication algorithm.

This function aimed to link the VPN technology with the authentication and encryption algorithms using a single string element.
However, this approach became outdated due to the increasing number of properties that the administrator can now choose from. The new approach performs this check using the constraint defined for the function USED_TECHNOLOGY, as discussed in Chapter 6. The new solution is more modular, making it easier to maintain and enhance by expanding the supported technologies.

Another approach taken was to reduce the number of constraints passed to the z3 solver. This was achieved by implementing a check to ensure that no duplicate constraints were added to the solver. However, this solution did not yield the expected results. Despite a slight reduction of slightly above 3% in the total number of constraints, the performance of the framework only improved insignificantly. One possible explanation is that z3 automatically optimizes the received constraints and manages duplicate ones autonomously.

With this in mind, we proceeded to subdivide the concatenations of constraints linked by a logical 'and' into smaller constraints to be passed to z3. However, this approach, like the previous one, did not result in significant improvements. A possible reason is that although it allows the solver to optimize the constraints better, the number of constraints grows significantly.

Finally, analysing the code were identified some constraints needed to simplify the work of the translator to present the result of the solver in a user-friendly manner. Upon further investigation was discovered that these constraints were no longer necessary and removed. As shown in the following pictures, these constraints had a

significant impact on the overall performance of the framework. In fact, the performance improvement becomes more pronounced as the number of allocation points and policy requirements increases, as clearly shown in the table.

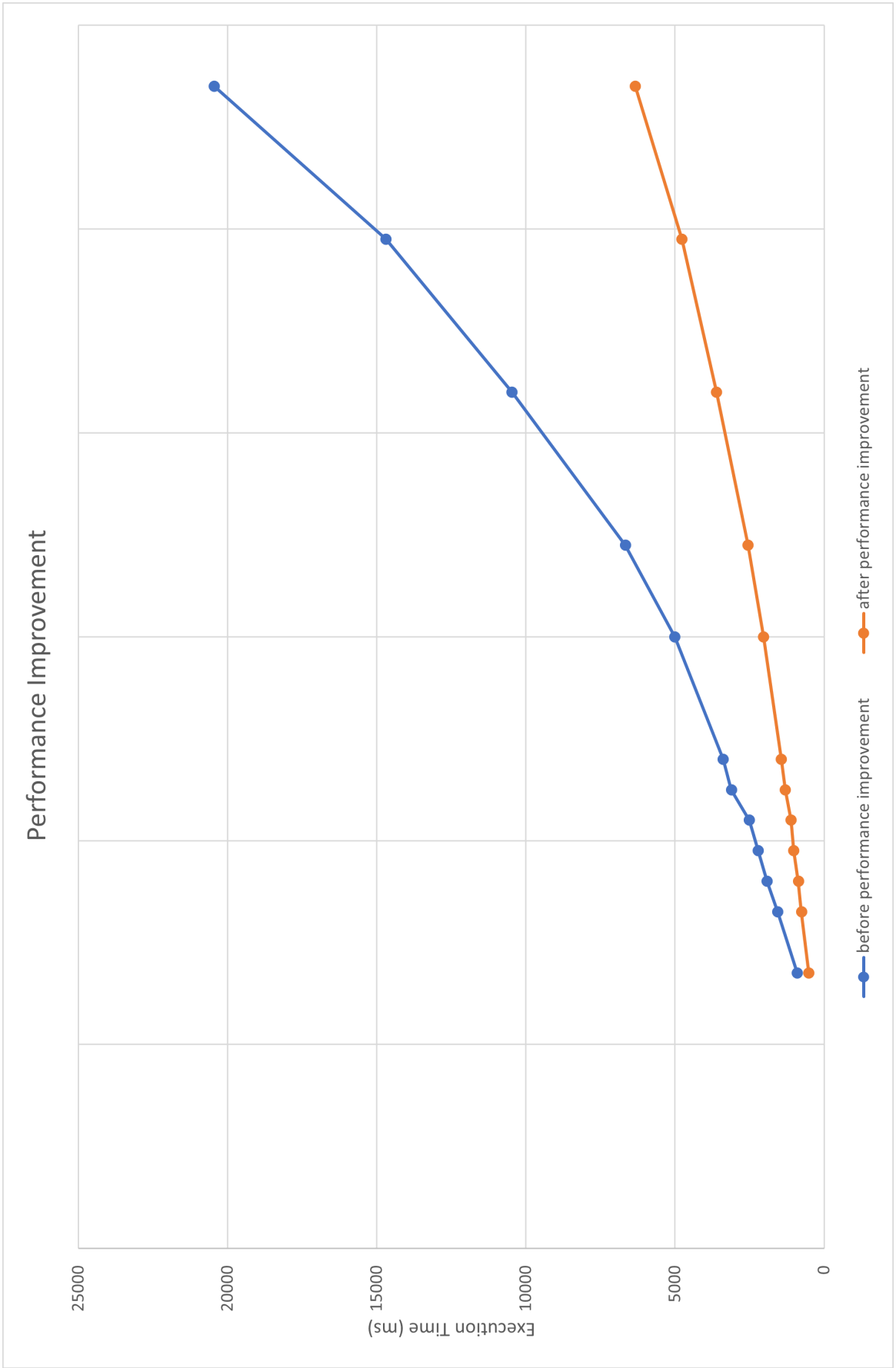

Figure 7.3: Performance Improvement Graph

Number Allocation Points	Number Security Requirements	Before Performance Improvement	After Performance Improvement	% time reduction
5	27	703	509	38,11%
8	36	1317	863	52,61%
10	42	1750	1103	58,66%
12	48	2220	1439	54,27%
16	60	3107	2028	53,21%
24	84	5717	3611	58,32%
34	114	11707	6328	85,00%
44	144	18142	10033	80,82%
54	174	26724	14142	88,97%
68	216	40161	22645	77,35%
84	264	60556	34330	76,40%
138	426	164747	96609	70,53%

Figure 7.4: Performance Improvement Table

The previous test and all the following were performed on a 4-core Intel i7-6700 3.40 GHz workstation with 32 GB of RAM and for each input value, each test was repeated 50 times. The networks used in the tests were automatically generated using a program. The program aimed to create a realistic network by inserting the number of clients and servers as input. The number of policies increased with the desired number of end hosts. The network generated by the program can be seen in the following pictures:

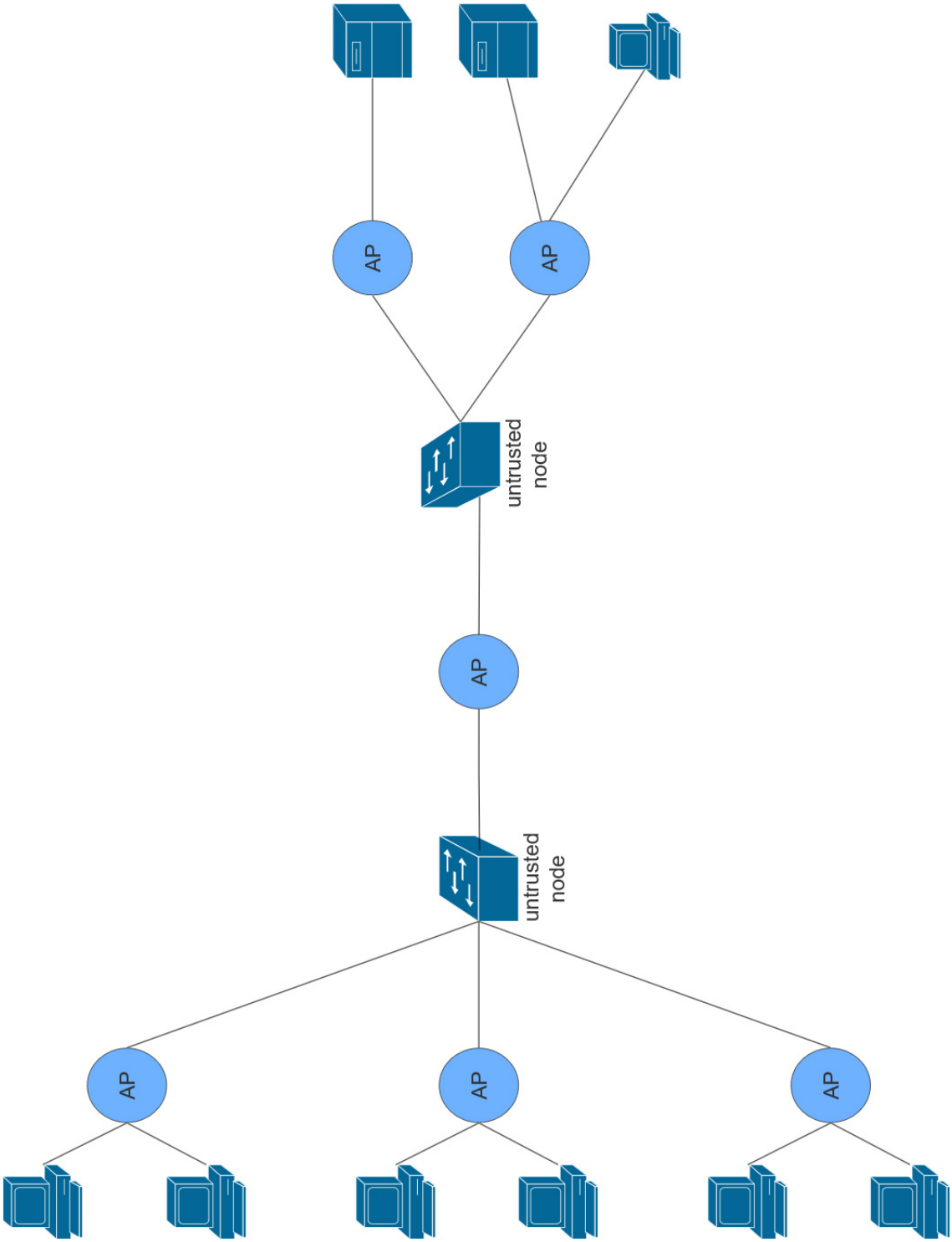

Figure 7.5: Example of Starting Test Network

The base structures are the one left and the one right of the central allocation point. Considering the requested number of client and server a base structure can be added linked to one of the untrusted nodes or allocation point, for example if twenty nine client and four servers are requested the generated network will be the following:

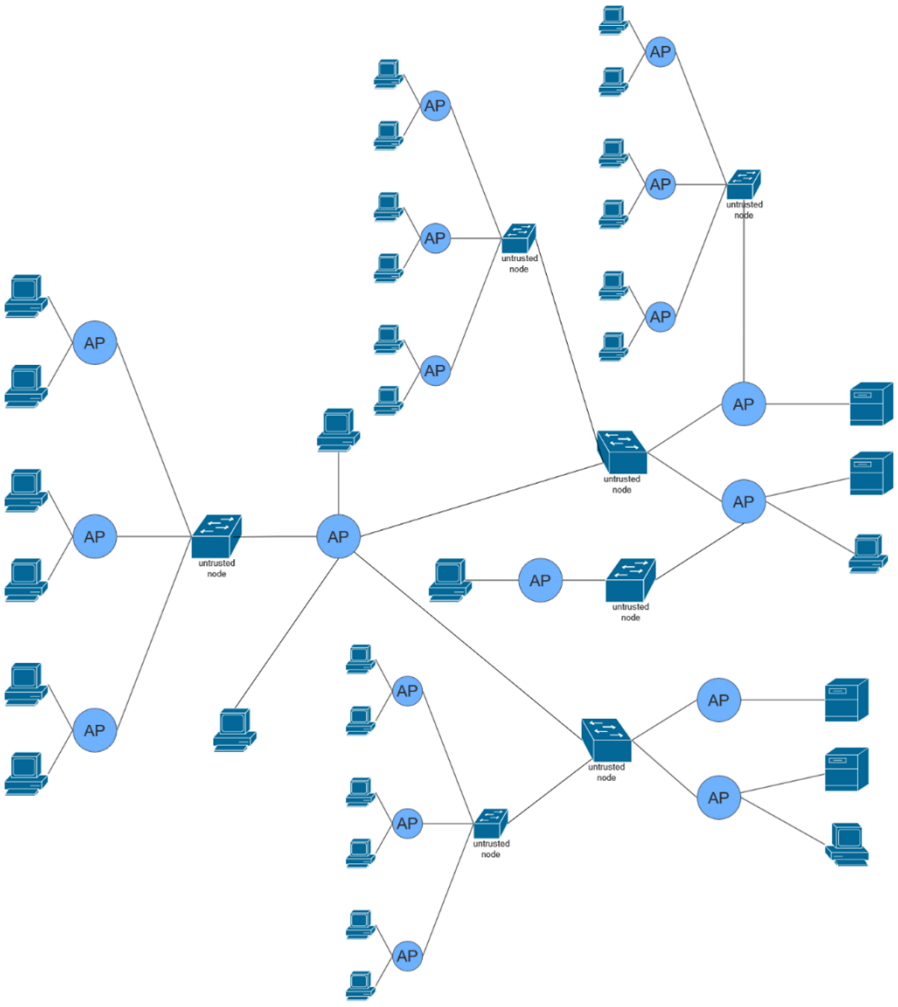

Figure 7.6: Example of Test Network with 29 Client and 4 Server

Different kinds of tests were performed, using various type of policies as input, to better understand the impact of the different part of the program on the performance of the framework.
First were evaluated very simple policies that request only the secure communication between two endpoints. An example of this policies is the following.

```
<PropertyDefinition>
      <Property graph="0" name="ProtectionProperty" src="10.0.0.1"
         dst="20.0.0.1">
            <protectionInfo encryptionAlgorithm="AES_256_CBC"
               authenticationAlgorithm="SHA2_256">
                  <untrustedNode node="30.0.0.1"></untrustedNode>
            </protectionInfo>
      </Property>
```

```
</PropertyDefinition>
```

Listing 7.13: Example Input Security Requirements

The achieved results are presented in the following graph. Upon examining the graph, becomes evident that the rise in execution time does not exhibit an exponential growth pattern. This finding holds significant importance, particularly when considering the computational expenses associated with the MaxSMT problem.

Figure 7.7: Basic Policy Performance Graph

In the following graph the number of Allocation Points remain constant and equal to six, the number of End Point is constant as well and is equal to nine, the number of Secuirty Policy instead grows.

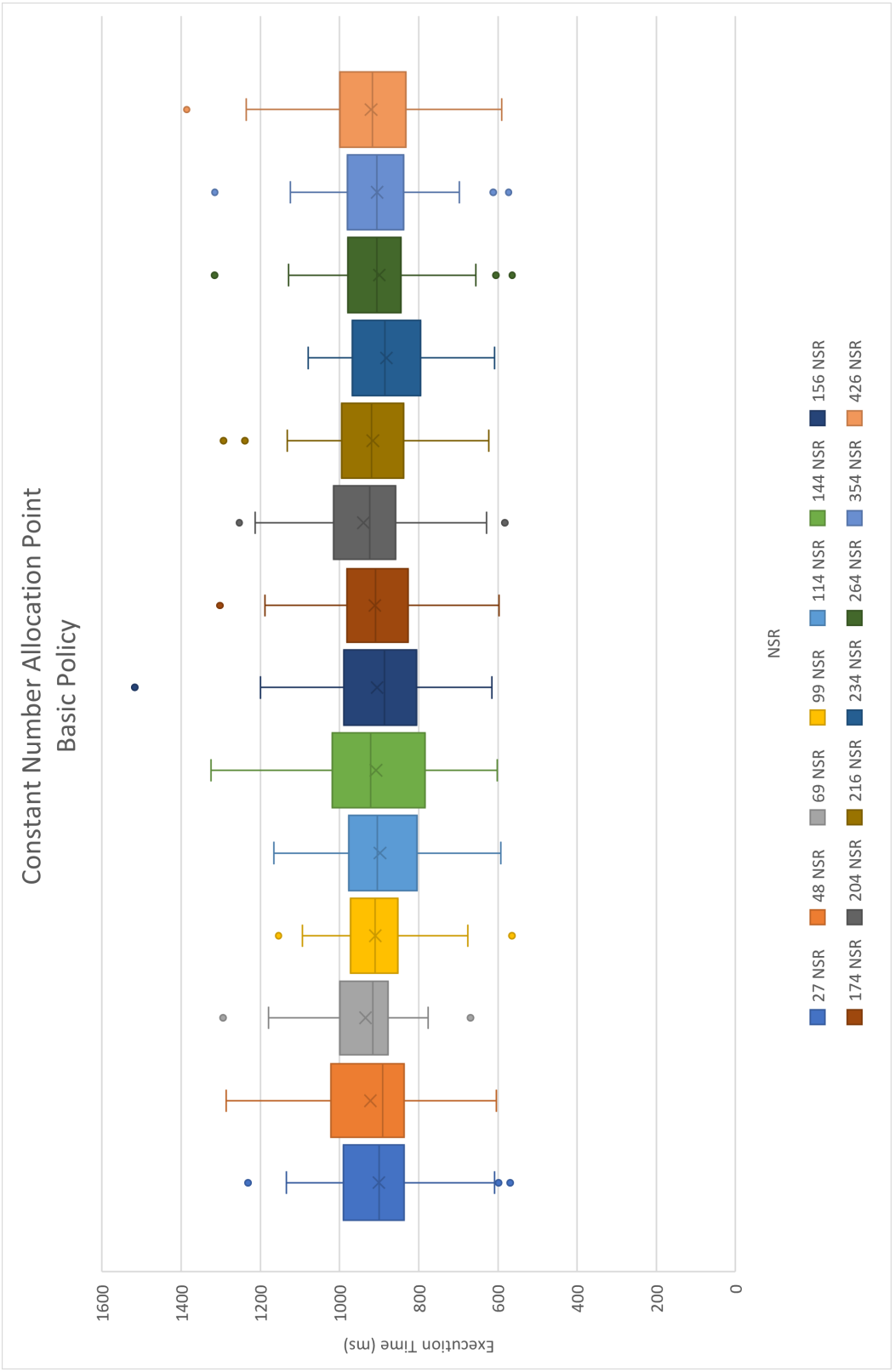

Figure 7.8: Constant AP and End Host

The next graph instead shows the performance in case of increasing number of Allocation Points and End Points with a constant number of Security Policy equals to one.

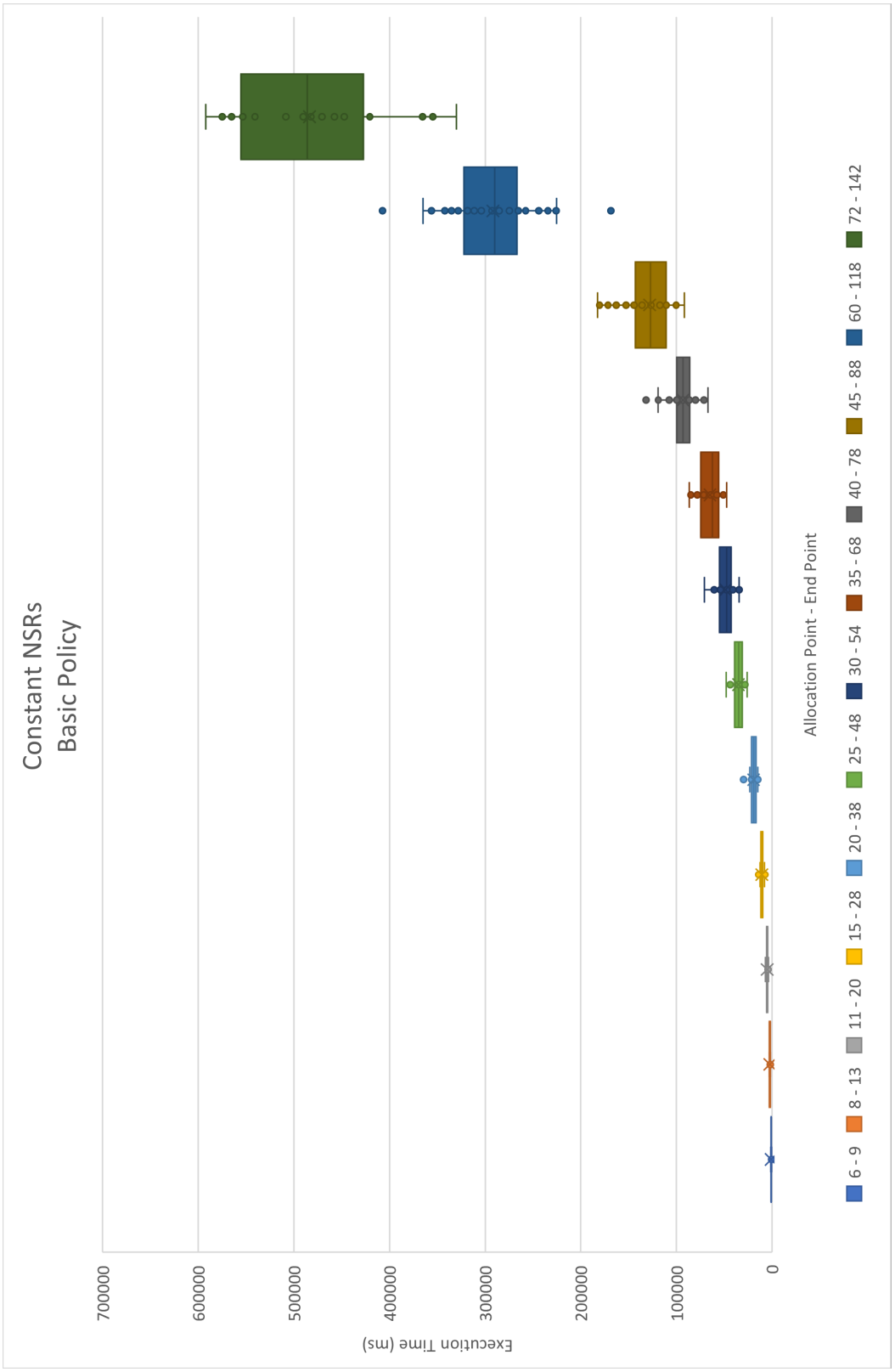

Figure 7.9: Constant NSRs

Is immediately possible to observe that the bigger impact on the framework performances is the number of End Points, this is caused by the solver requiring more time to find the correct path between the two ends of the VPN.

Other tests were performed requesting the use of a specific VPN technology, so the policy had the following form:

```
    <PropertyDefinition>
        <Property graph="0" name="ProtectionProperty" src="10.0.0.1"
           dst="20.0.0.1">
              <protectionInfo encryptionAlgorithm="AES_256_CBC"
                 authenticationAlgorithm="SHA2_256">
                  <untrustedNode node="30.0.0.1"></untrustedNode>
                    <securityTechnology>IPSEC</securityTechnology>
        </protectionInfo>
        </Property>
  </PropertyDefinition>
```

Listing 7.14: Example Input Security Requirements

As shown by the next graph the obtained performance are better than the previous one, this can be explained considering the formulation of the constraints that allowed the solver to discard constraint relative to the unsupported properties of each technology.

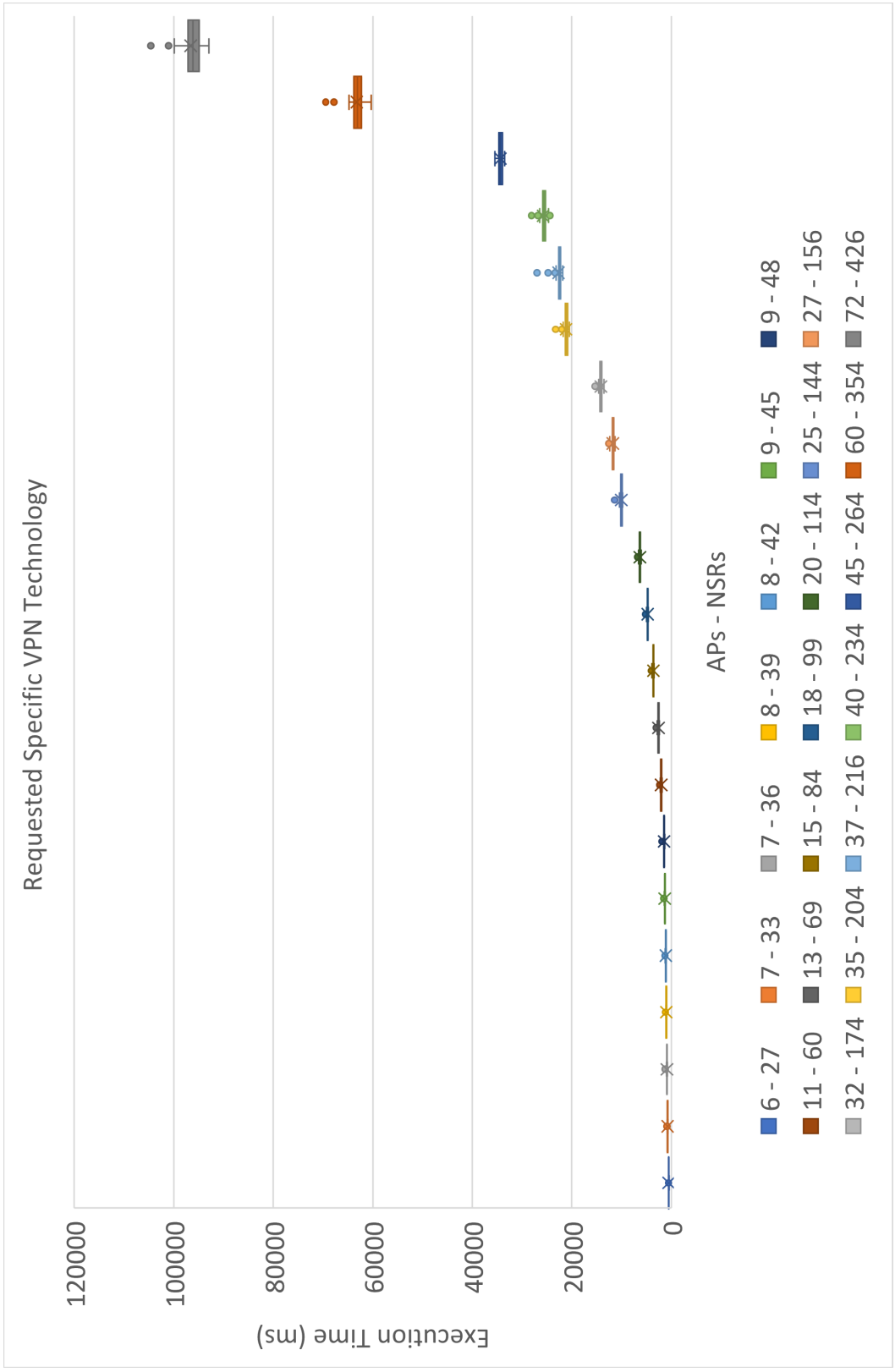

Figure 7.10: Requested Specific VPN Technology Performance Graph

The next graph shows again the performance in case the number of Allocation Points remain constant and equal to six, with an also constant number of End Point equal to nine, and with a growing number of Secuirty Policy.

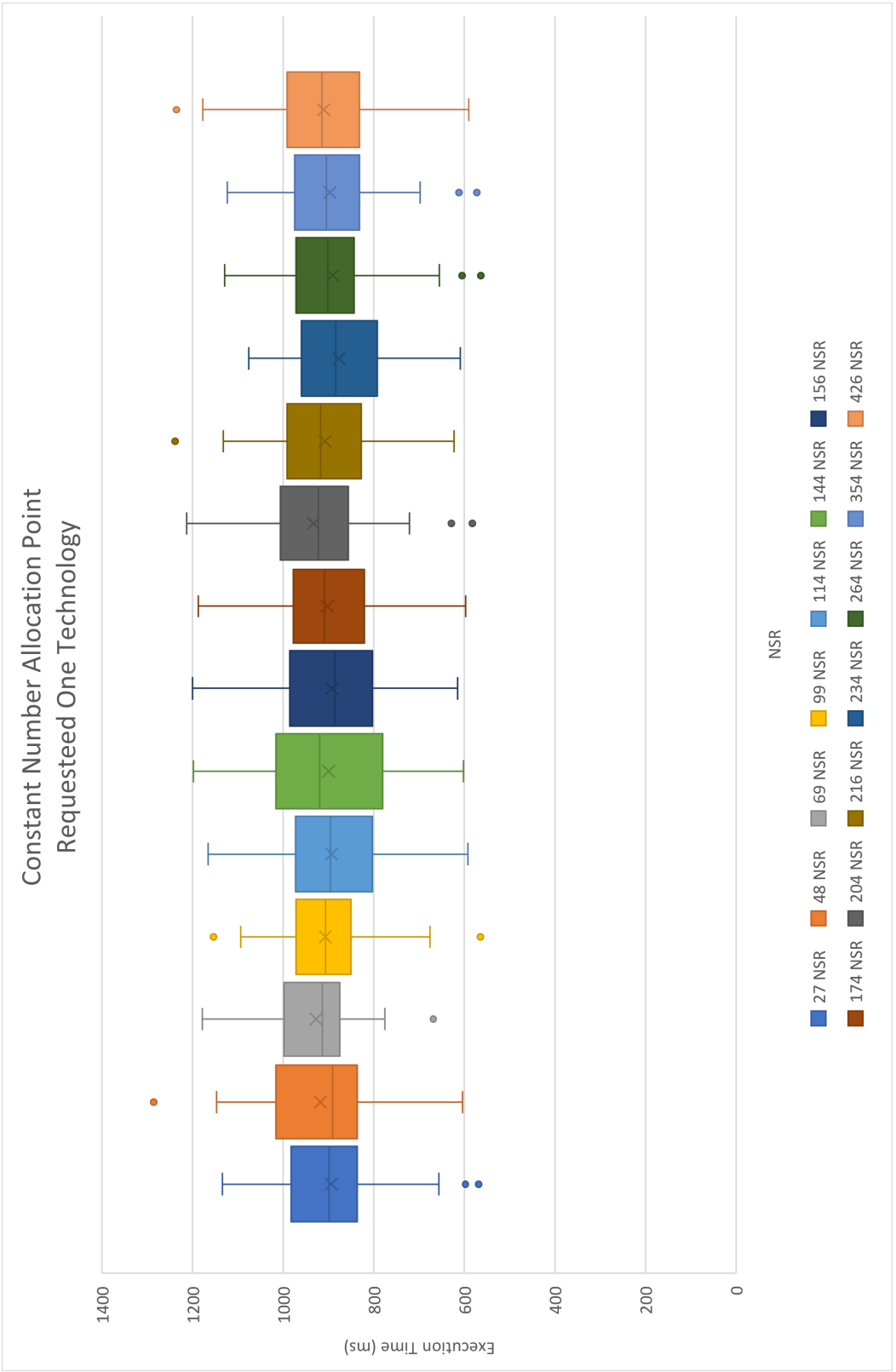

Figure 7.11: Constant AP and End Host

The next graph instead shows the performance in case of increasing number of Allocation Points and End Points with a constant number of Security Policy equals to one.

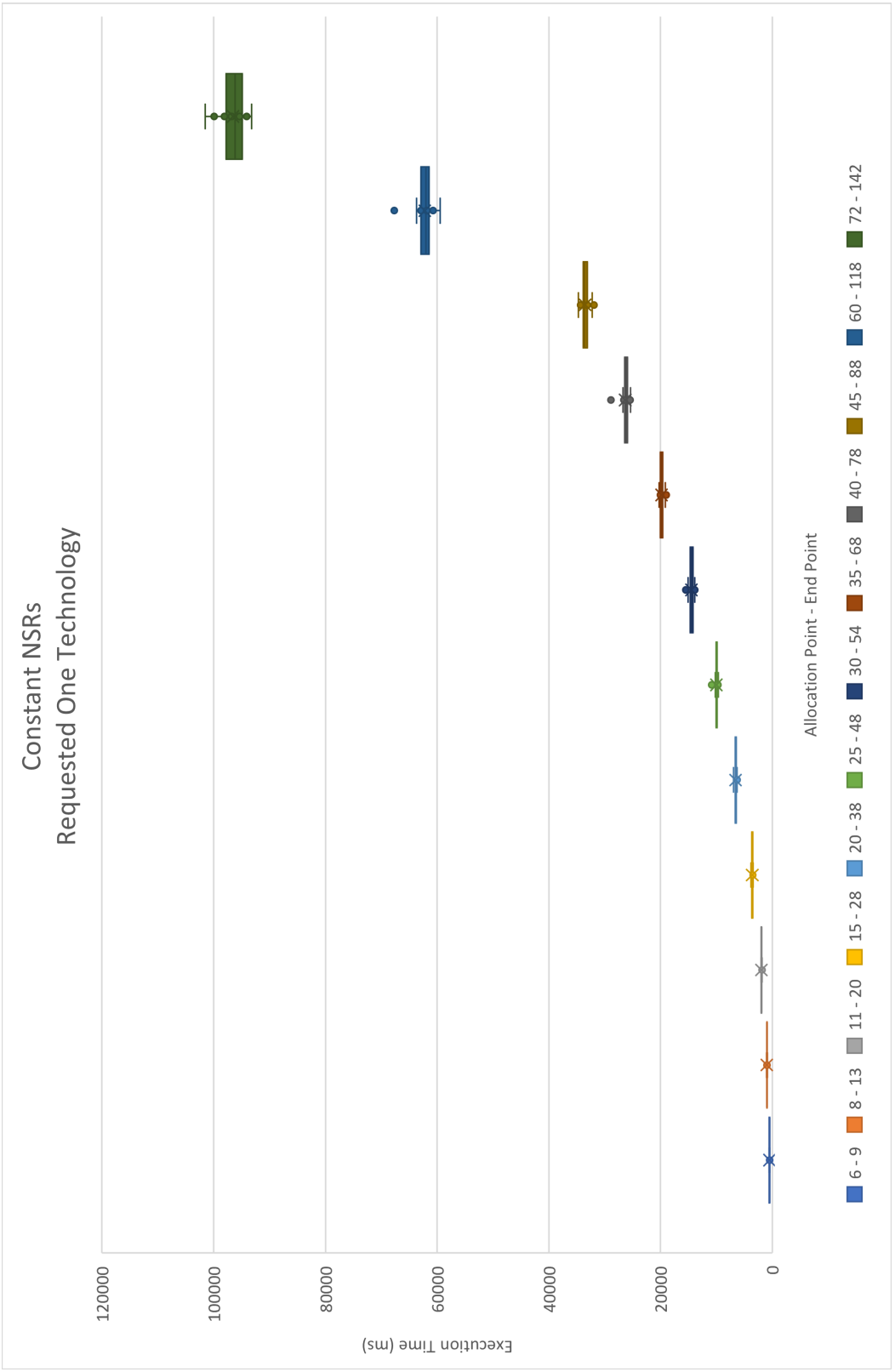

Figure 7.12: Constant NSRs

The general behaviour of the framework is the same of the previous case, so the number of End Points is still the main cause for the deterioration of the performance, but we can notice an improvement over the previous case.

Finally, the worst-case scenario was analysed, the one where all the field of the security properties are given a value, that is randomly selected from the set of all possible combinations that have a viable solution. A possible example can be the following:

```
<Property graph="0" name="ProtectionProperty" src="10.0.0.1"
    dst="20.0.0.1">
    <protectionInfo encryptionAlgorithm="AES_256_CBC"
        authenticationAlgorithm="SHA2_256">
        <untrustedNode node="30.0.0.1"></untrustedNode>
        <requiredProtection>

    <confidentialityInternalHeader>true</confidentialityInternalHeader>
            <integrityInternalHeader>true</integrityInternalHeader>
            <integrityAdditionalHeader>true</integrityAdditionalHeader>
            <integrityInternalPayload>false</integrityInternalPayload>
            <serverAuthentication>true</serverAuthentication>
            <clientAuthentication>false</clientAuthentication>
        </requiredProtection>
    </protectionInfo>
</Property>
```

Listing 7.15: Example Input Security Requirements

As we can see in the next graph the performance deteriorates faster than in all the previous case, and with 144 requested policies the time to obtain a solution became considerable.

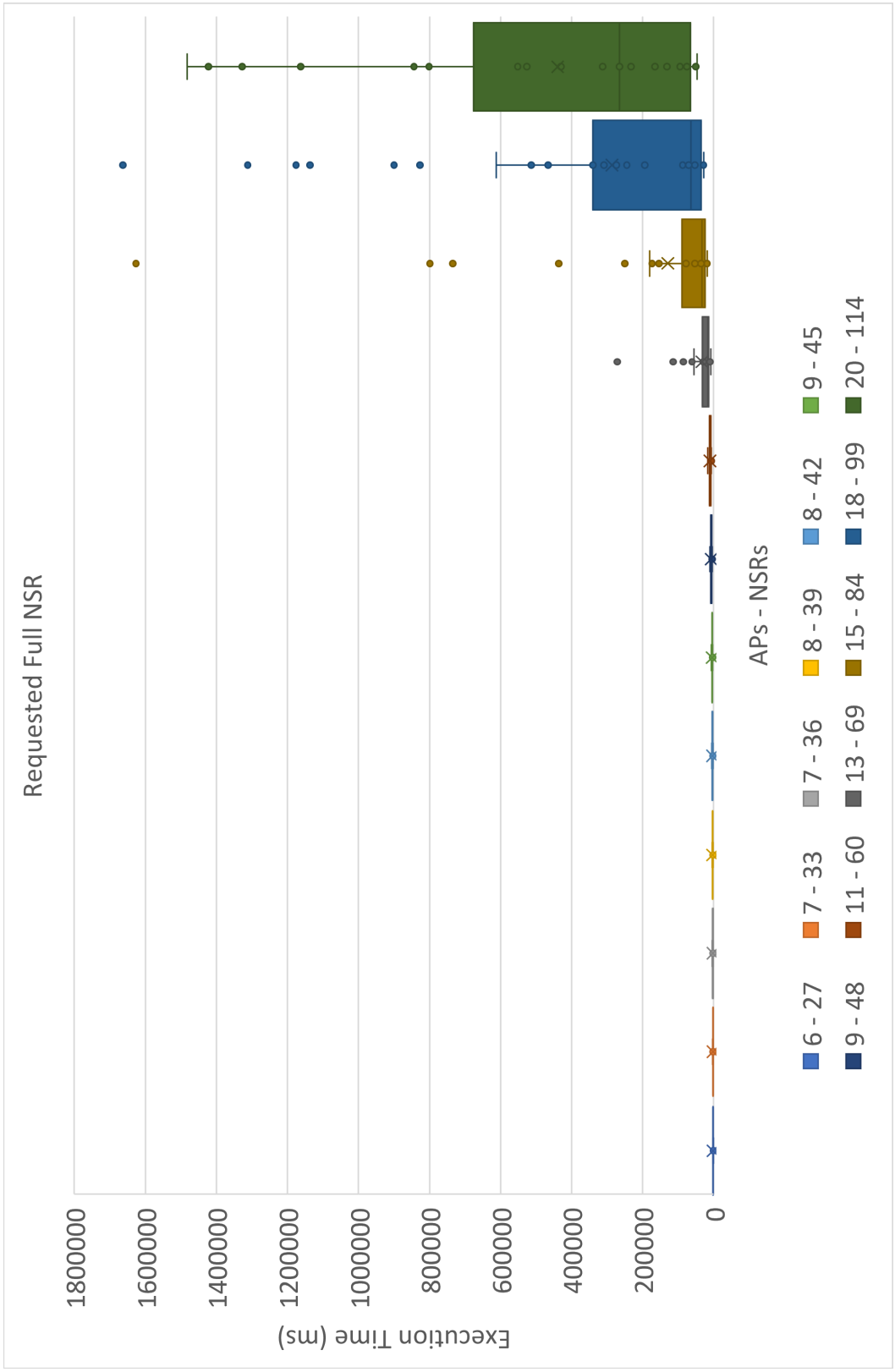

Figure 7.13: Requested Full NSR Performance Graph

As in the previous cases the next graph shows what happens to the performances with a constant number of Allocation Points equal to six, with an also constant number of End Point equal to nine, and with a growing number of Secuirty Policy.

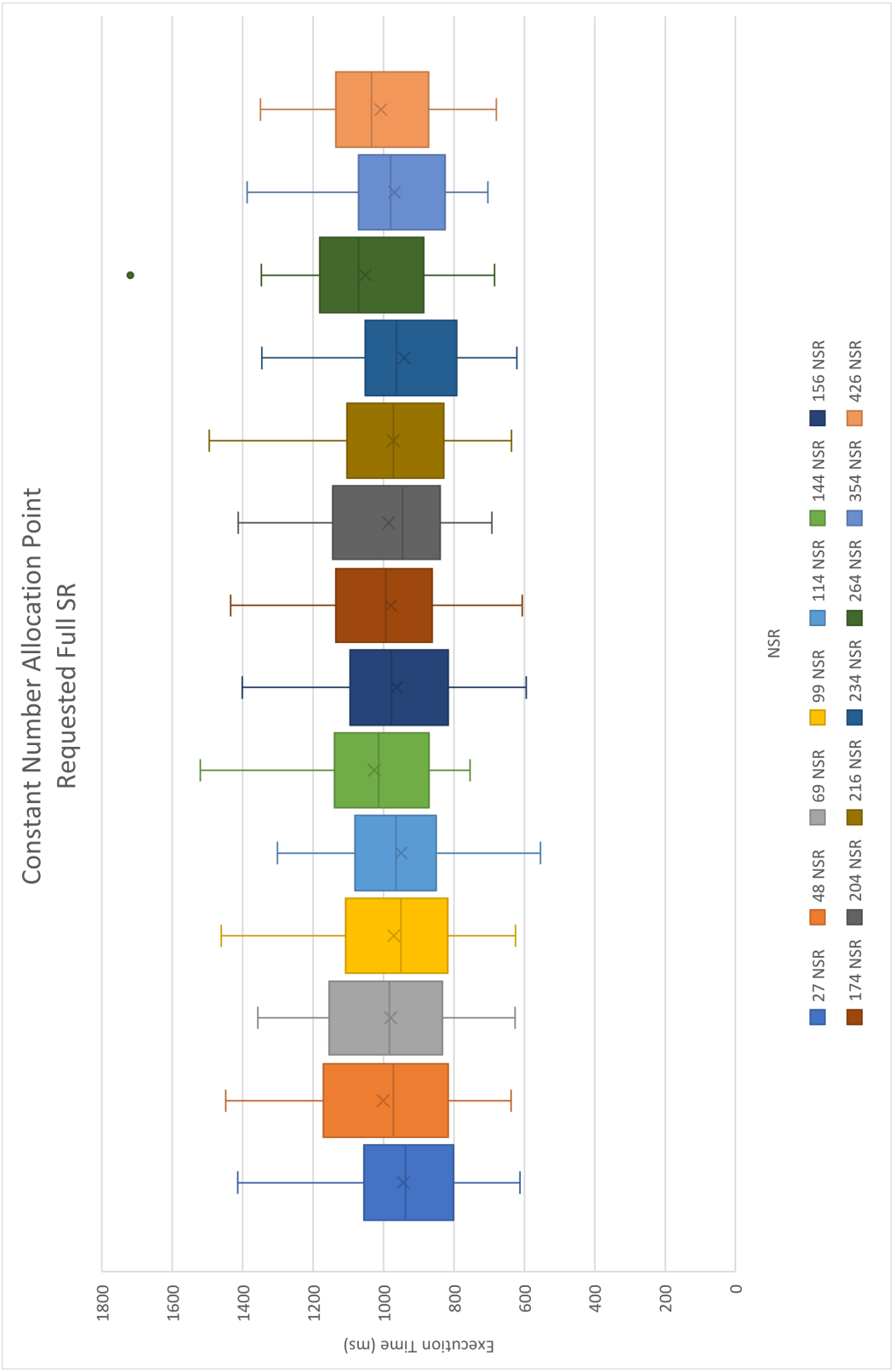

Figure 7.14: Constant AP and End Host

The next graph illustrates the performance when the number of Allocation Points and End Points is increased while keeping the number of Security Policies constant at one. Some outlier points, whose values sometimes exceed 2,000,000 ms,

have been omitted from the graph to ensure readability.

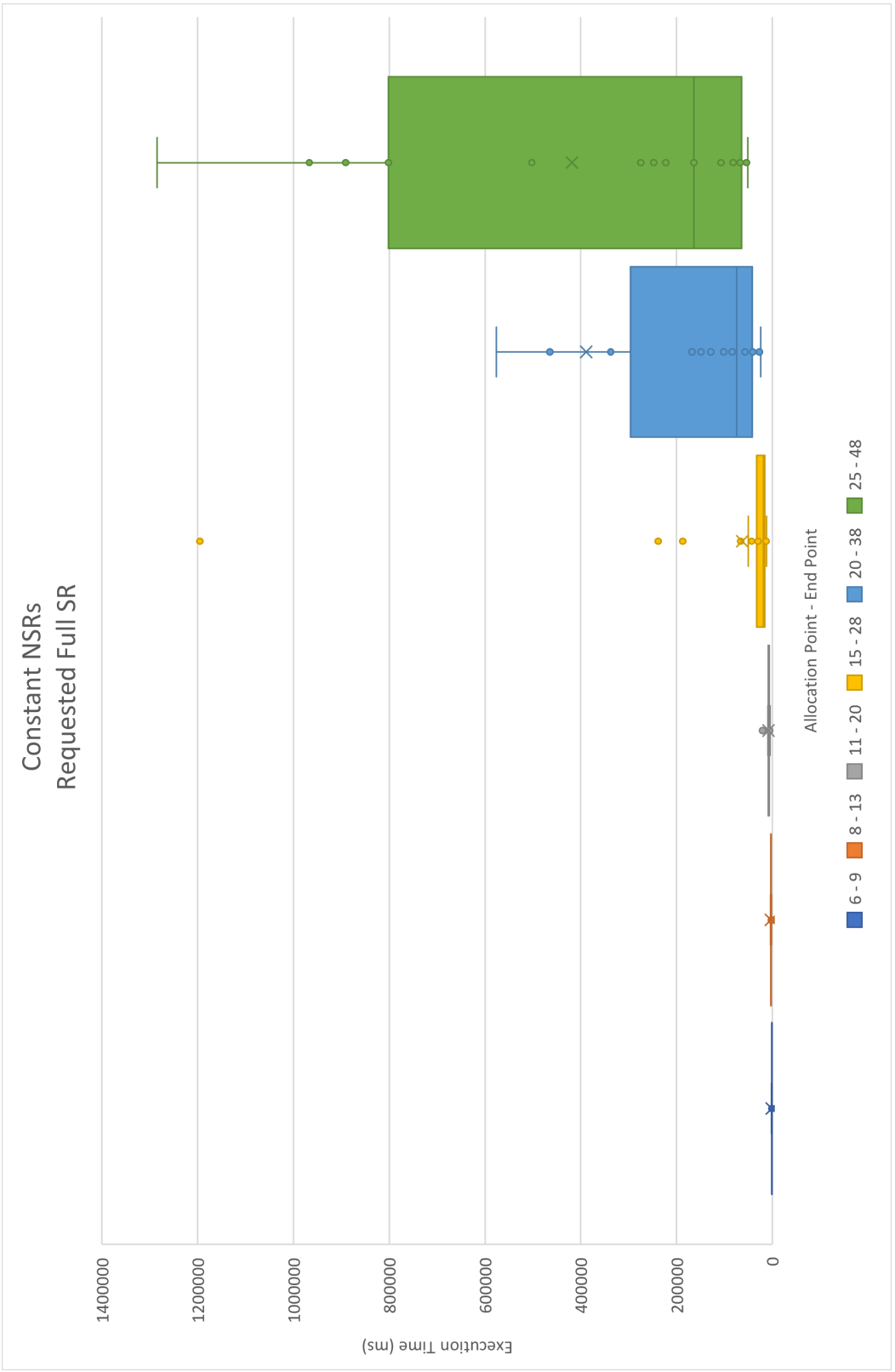

Figure 7.15: Constant NSRs

Another notable observation from the above graphs is that as the average performance deteriorates, the timing of the tests becomes increasingly varied despite having the same initial conditions. This phenomenon is particularly pronounced in the last case, where it is also exacerbated by the fact that the required protection properties are not consistent across all the tests. Instead, they are randomly selected from the set of all possible combinations that have a viable solution.

The following graphs provides a better comparison of the performance of the framework both before and after the introduction of the changes implemented in this thesis work. Notably, in the case of policies requiring a specific VPN technology, the performance is better than the initial one. This attests the successful achievement of the final objective of the thesis work, which effectively enhanced the performance of the framework.

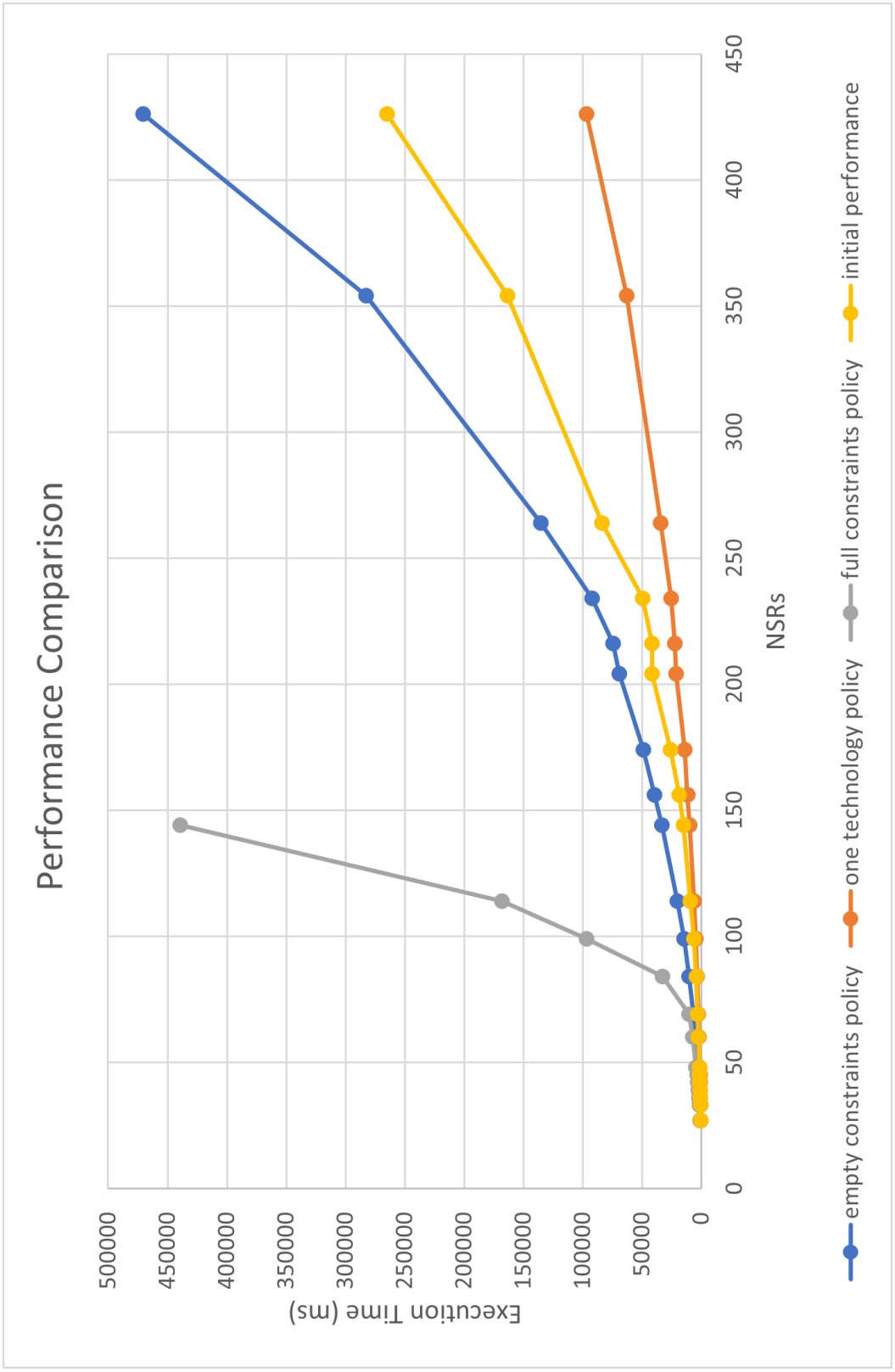

Figure 7.16: Performance Comparison Graph

Instead in the following graph and table we compare the number of total constraints that the framework must solve to find the solution.
Before analysing the graph, we can notice that the data from of the full constraints policy is represented until the performance of the framework begin to deteriorate to fast as seen previously.

An interesting fact that emerges from the observation of this data is that the new functionalities to be implemented required a significant increment in the number of hard and soft constraints to be solved.
This does not reflect directly on the performance of the framework. In fact, as shown in the previous graph the performance differs in a very significant way between the policy where a specific VPN technology is requested and the one where all the field of the security properties are given a value, even tough the total number of constraints is similar.
The same behaviour can be seen also if we focus our attention on the performance of the framework before the changes introduced by this thesis work. Indeed, in this case the total number of constraints is significantly lower than in the other cases analysed, but this does not automatically translate into better performance.
This testifies how important it is for performance how constraints are written and structured.

Number Allocation Points	Number Security Requirements	Starting Branch	Basic Policy	Requested Specific VPN Technology	Requested Full NSR
6	27	285	1751	1814	1907
7	33	379	2460	2547	2678
7	36	440	2718	2814	2951
8	39	506	3041	3148	3271
8	42	573	3299	3415	3599
9	45	648	3494	3617	3795
9	48	704	3752	3884	4116
11	60	982	4466	4624	4872
13	69	1144	5018	5196	5451
15	84	1513	6262	6483	6856
18	99	2045	7679	7948	8466
20	114	2588	9095	9412	9965
25	144	3553	11928	12341	12987
27	156	4162	13074	13526	
32	174	5161	14497	14999	
35	204	6518	16570	17146	
37	216	7211	17724	18339	
40	234	8235	19403	20075	
45	264	9929	22340	23111	
60	354	17087	29943	30977	
72	426	23875	36499	37757	

Figure 7.17: Comparison of Number of Constraints Table

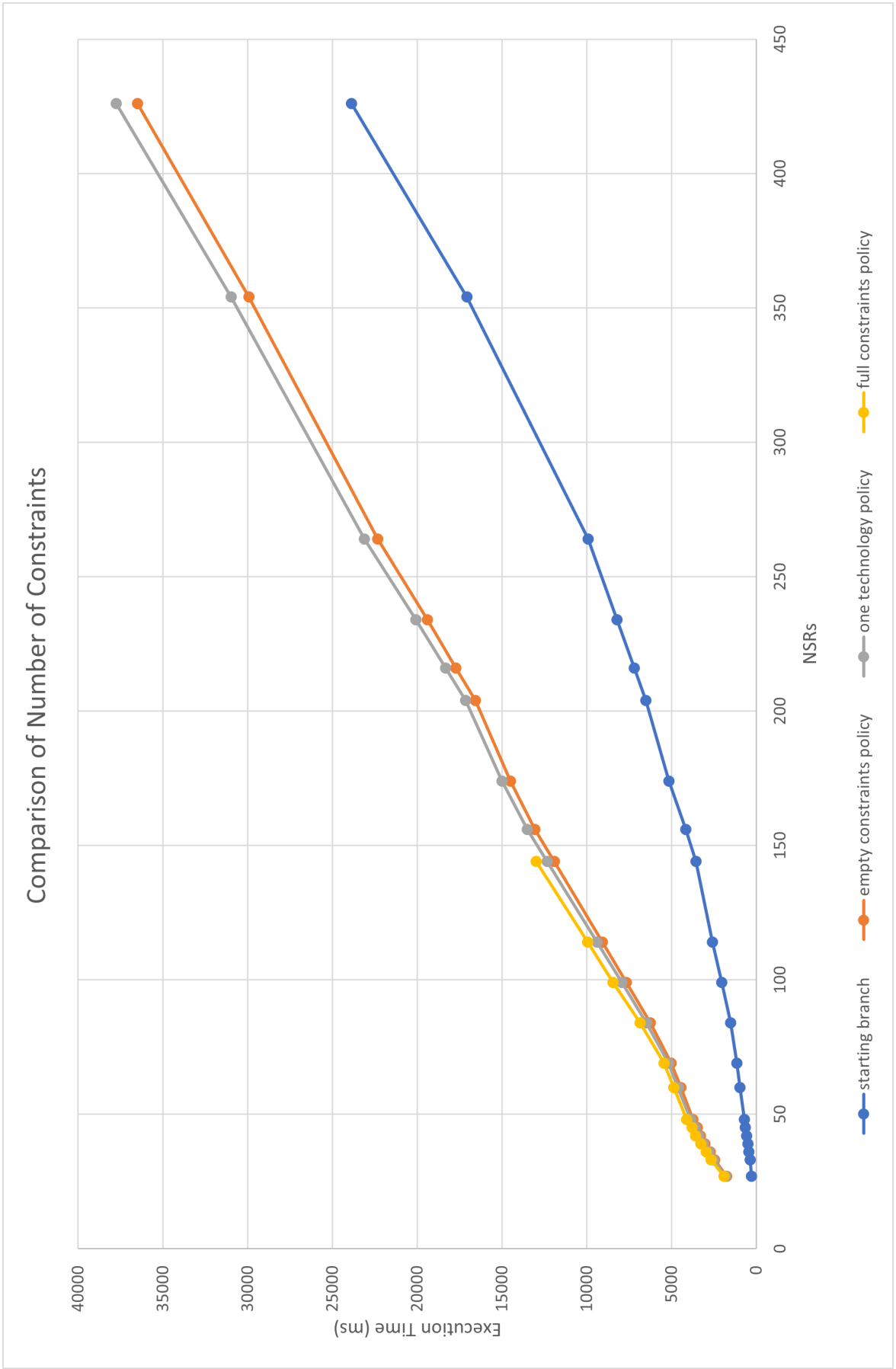

Figure 7.18: Comparison of Number of Constraints Graph

Chapter 8

Conclusions

In this thesis work, significant enhancements were made to VEREFOO, expanding its capabilities and functionalities. The framework underwent extensive development to not only automate the allocation and configuration of the Channel Protection System for network security requirements, ensuring secure communication enforcement, but also to enable the selection of the most optimal VPN technology for the task at hand.

To lay the foundation for these advancements, a comprehensive study was conducted, specifically focusing on various TLS-based VPNs. This investigation was crucial since, unlike IPsec, there is no standardized protocol for implementing TLS-based VPNs. Therefore, it was essential to identify the similarities and differences among each protocol to develop a comprehensive solution.

The next step was to conduct a thorough examination to find the similarities and differences between TLS-based and IPsec-base VPNs, with a particular focus on identifying the different types of security that each approach could provide.

Following this initial research phase, Communication Protection Policies and Rule models were updated with the introduction of new fields and possible values. A series of hard and soft constraints were introduced to enable the framework to choose the best possible VPN technology with its architecture and mode. The XML schema dedicated to the Network Security Requirements was expanded and modified to allow the implementation of the new functionality.

The implementation of this new functionalities was extensively tested thanks to the definition of numerous JUnit tests. Numerous network topology and all the various possible combinations of security properties that an admin may require were considered.

Research of way to improve the performance of VEREFOO was also conducted and the result achieved were presented. After completing all these tasks, a series of performance tests were conducted to assess the impact of the number of communication security requirements and the number of allocation points on the scalability of the system. Different types of scenarios were considered to allow a

deeper comprehension of what affected the most the performance of the framework.

There are several possible future works that can be considered to improve VEREFOO.
The definition of higher-level policy and the mapping from high to middle level policy and vice versa. The implementation of a non-enforceability report could also be an interesting addition to the functions of the framework. This way the tool could be more user friendly and be used by a wider audience.

Furthermore, there is a potential for further improvement in VEREFOO's performance, particularly by focusing on formulating constraints.

Bibliography

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

[2] D. Bringhenti, R. Sisto, and F. Valenza, "A novel abstraction for security configuration in virtual networks," *Comput. Networks*, vol. 228, p. 109745, 2023.

[3] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS*. IEEE, 2020, pp. 1–7.

[4] ——, "Automated firewall configuration in virtual networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1559–1576, 2023.

[5] S. Bussa, R. Sisto, and F. Valenza, "Security automation using traffic flow modeling," in *NetSoft*. IEEE, 2022, pp. 486–491.

[6] D. Bringhenti and F. Valenza, "Optimizing distributed firewall reconfiguration transients," *Comput. Networks*, vol. 215, p. 109183, 2022.

[7] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Introducing programmability and automation in the synthesis of virtual firewall rules," in *NetSoft*. IEEE, 2020, pp. 473–478.

[8] 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.

[9] F. Valenza and M. Cheminod, "An optimized firewall anomaly resolution," *J. Internet Serv. Inf. Secur.*, vol. 10, no. 1, pp. 22–37, 2020.

[10] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *CRiSIS*, ser. Lecture Notes in Computer Science, vol. 8924. Springer, 2014, pp. 148–163.

[11] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, 2017.

[12] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Towards a fully automated and optimized network security functions orchestration," in *ICCCS*. IEEE, 2019, pp. 1–7.

[13] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "A novel approach for security function graph configuration and deployment," in *NetSoft*. IEEE, 2021, pp. 457–463.

[14] ——, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *CYSARM@CCS*. ACM, 2020, pp. 25–30.

[15] K. Seo and S. Kent, "Security Architecture for the Internet Protocol," RFC 4301, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4301

[16] S. Kent, "IP Authentication Header," RFC 4302, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4302

[17] ——, "IP Encapsulating Security Payload (ESP)," RFC 4303, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4303

[18] I. Pedone, A. Lioy, and F. Valenza, "Towards an efficient management and orchestration framework for virtual network security functions," *Secur. Commun. Networks*, vol. 2019, pp. 2 425 983:1–2 425 983:11, 2019.

[19] 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.

[20] D. Bringhenti, F. Valenza, and C. Basile, "Toward cybersecurity personalization in smart homes," *IEEE Secur. Priv.*, vol. 20, no. 1, pp. 45–53, 2022.

[21] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019.

[22] "Openvpn protocol." [Online]. Available: https://openvpn.net/community-resources/openvpn-protocol/

[23] "Sstp packet." [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-sstp/2991892f-fefc-4129-adac-cd6a5d04bb48

[24] "Tinc encryption of network packets." [Online]. Available: https://www.tinc-vpn.org/documentation-1.1/Encryption-of-network-packets.html