

Politecnico di Torino

Master of science in Cybersecurity

Master Degree Thesis

Toward a Heuristic Approach for Automated VPN Configuration

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

Candidate
Gaia Casellato

Academic Year 2025-2026

This work is subject to the Creative Commons Licence

Summary

Modern network infrastructures increasingly rely on Virtual Private Networks (VPNs) to ensure data confidentiality and integrity across untrusted domains. While formal automated configuration tools, such as the VEREFOO framework, provide mathematically verified and optimal security policies using Maximum Satisfiability Modulo Theories (MaxSMT) solvers, they suffer from severe scalability limitations. The exponential complexity of MaxSMT makes real-time configuration impossible for large enterprise networks.

To bridge the gap between formal assurance and practical performance, this thesis proposes a novel, polynomial-time heuristic algorithm for the automated allocation and configuration of VPN gateways. The heuristic operates through a two-phase architecture: an initial "Safety-First" local pre-allocation that immediately isolates untrusted network segments, followed by an iterative global consolidation phase utilizing a dynamic delta update mechanism to resolve redundant cryptographic tunnels. Furthermore, the algorithm actively resolves complex topological constraints, smoothly handling dynamic protocol switching for middlebox traversal and fallback mechanisms for Inspector Nodes.

Experimental validation and scalability stress tests demonstrate a massive performance improvement. While traditional MaxSMT solvers exhaust computational resources on topologies approaching 100 nodes, the heuristic orchestrator successfully optimizes 50 complex communication flows across a 1000-node infrastructure in under 7 seconds. The results confirm that the algorithm guarantees formal correctness-by-construction while enabling highly scalable, automated VPN deployments.

Acknowledgements

I would like to express my sincere appreciation to Politecnico di Torino, a place that not only shaped my academic background but also pushed me to step out of my comfort zone and grow as a person. I am deeply indebted to my supervisors for their fundamental role in this achievement.
Thank you to Professor Fulvio Valenza for his mentorship, endless encouragement, and for constantly inspiring me.
My immense gratitude goes to Professor Daniele Bringhenti for his extreme availability, precious advice, and for guiding me step-by-step through the complexities of this work.
I also would like to thank Professor Riccardo Sisto for believing in my potential and supporting this research.
The most special thanks go to my family.
To my parents: thank you for giving me the freedom to chase my ambitions and for your unconditional love. Everything I have achieved is because of the foundation you built for me.
To my sister: you are my pillar. Thank you for always listening, understanding my struggles, and reminding me of who I am.
This manuscript is the culmination of hard work, passion, and persistence. It is my sincere hope that these pages may serve as a valuable stepping stone for future developments in this field.

Contents

List of Figures VI

List of Tables VII

Listings VIII

1 Introduction 1

1.1 Motivation . . . 1

1.2 Objectives of the Thesis . . . 2

1.3 Thesis Organization . . . 2

2 Background and State of the Art 4

2.1 Virtual Private Networks and IPsec . . . 4

2.1.1 VPN Topologies and Deployment Models . . . 5

2.1.2 The IPsec Protocol Suite . . . 7

2.1.3 TLS-based VPNs as an Alternative Protocol . . . 8

2.1.4 Security Associations and Policy Management . . . 8

2.1.5 VPN Gateway Positioning and Topological Anomalies . . . 10

2.2 The VEREFOO Framework for VPN Automation . . . 10

2.2.1 The MaxSMT Formal Approach . . . 11

2.2.2 Scalability and Performance Challenges . . . 12

2.3 State of the Art in Automated Network Security . . . 13

3 The Heuristic VPN Configuration Algorithm 16

3.1 From Firewall to VPN Heuristics: Challenges and Differences . . . 16

3.1.1 Single-Node vs. Pairwise Allocation . . . 17

3.1.2 Middlebox Interferences and Protocol Switching . . . 17

3.1.3 Fallback Mechanisms and Inspector Exceptions . . . 17

3.2 Algorithm Overview and Strategy . . . 18
3.2.1 The "Site-to-Site First" Strategy . . . 18
3.2.2 Conceptual Workflow of the Heuristic Approach . . . 19
3.3 Mathematical Formalization . . . 19
3.3.1 Formal Model of Communication Protection Policies . . . 19
3.3.2 Definitions and Formal Rules . . . 20
3.4 Detailed Algorithmic Execution . . . 22
3.4.1 Phase I: Handling Multiple Danger Zones and Inspectors . . . 22
3.4.2 Phase II: Iterative Consolidation and Dynamic Delta Update 22
3.5 Heuristic Algorithm Specification . . . 23
3.5.1 Algorithm 1: Local Pre-allocation . . . 23
3.5.2 Algorithm 2: Global Consolidation . . . 25
3.6 Formal Assurance and Mathematical Consistency . . . 26

4 Design and Implementation 28
4.1 System Architecture . . . 28
4.2 Data Model and Formalization . . . 29
4.3 Translation Module . . . 31
4.4 Heuristic Logic Implementation . . . 32
4.5 Software Integration Testing . . . 34

5 Validation and Performance Analysis 35
5.1 Use Cases and Correctness Verification . . . 35
5.1.1 Use Case 1: Inspector Node and Fallback Mechanism . . . 35
5.1.2 Use Case 2: Middlebox Interference and Protocol Switch . . . 37
5.1.3 Use Case 3: Intersecting Paths and Global Consolidation . . . 38
5.1.4 Use Case 4: Site-to-Site First Strategy (IoT Constraints) . . . 40
5.1.5 Use Case 5: Cascading Tunnels and Multi-Iteration Delta Update . . . 42
5.2 Performance Comparison . . . 44
5.2.1 Heuristic vs MaxSMT . . . 44
5.2.2 Scalability Test . . . 44

6 Conclusions and Future Work 47
6.1 Final Remarks . . . 47
6.2 Future Developments . . . 47

Bibliography 49

List of Figures

2.1 Site-to-Site VPN architecture utilizing IPsec in Tunnel Mode. . . . 5

2.2 End-to-End VPN architecture and IoT deployment constraints. . . 6

2.3 Remote Access VPN architecture and client connections. 7

3.1 Heuristic mechanisms for Fallback mitigation and dynamic Protocol Switching. 18

5.1 Use Case 1: Phase I execution. The fallback mechanism splits the tunnel to guarantee clear-text visibility for the IDS strictly across the defined danger zone. 37

5.2 Use Case 2: Dynamic protocol switch adapting the tunnel encapsulation to traverse a restrictive middlebox. 38

5.3 Use Case 3: A concrete Y-topology where intersecting paths are dynamically merged. The consolidated SA_{new} tunnel bridges the danger zone safely while halving gateway overhead. 40

5.4 Use Case 4: The heuristic algorithm bypassing resource-constrained endpoints to establish an optimized Site-to-Site VPN architecture. . 41

5.5 Use Case 5: The dynamic Delta Update managing a multi-iteration consolidation. The algorithm terminates after resolving total overlaps, bypassing inclusion conflicts to preserve end-to-end encapsulation. 43

5.6 Scalability stress test results: Computation time of the heuristic orchestrator as a function of the network topology size. 46

List of Tables

5.1 Scalability stress test results: Execution time of the heuristic orchestrator as a function of the network topology size. 45

Listings

4.1 The `MergeCandidate` internal structure and its evaluation logic. . . 30
4.2 Core optimization loop executing the Global Consolidation Phase. . 32
5.1 JUnit test excerpt validating the IDS fallback mechanism. 36
5.2 JUnit test excerpt validating the dynamic protocol switch. 38
5.3 JUnit test excerpt validating the Y-Topology consolidation. 39
5.4 JUnit test excerpt validating the Site-to-Site gateway selection. . . 41
5.5 JUnit test excerpt validating the Early Exit mechanism on cascading tunnels. 42

Chapter 1

Introduction

1.1 Motivation

In modern network infrastructures, security and policy-based management are strictly correlated with the deployment of Virtual Private Networks (VPNs). Traditionally, the configuration of these networks is performed manually by network administrators. However, as network complexity grows, this manual process becomes increasingly error-prone, potentially introducing vulnerabilities that expose the infrastructure to cyber-attacks. The inherent limitations of manual configuration and the pressing need for automation represent widely debated challenges in the scientific community; recent comprehensive surveys [1] highlight how human-driven management of security policies is rapidly becoming unsustainable in large-scale infrastructures.

The protection of traffic crossing a VPN is delegated to specialized entities known as Communication Protection Systems (CPS), such as dedicated VPN gateways or secure communication endpoints. Ensuring the correct configuration of these CPSs is essential for maintaining the integrity and confidentiality of data.

While automated approaches have been proposed to mitigate human error, particularly through formal methods such as Maximum Satisfiability Modulo Theories (MaxSMT) solvers, these solutions often face significant scalability challenges. Although MaxSMT provides provably correct and optimized configurations, the computational time required becomes prohibitive as the number of network nodes increases.

To address these limitations, this thesis explores a different approach: the development of a heuristic algorithm. By leveraging earlier studies on heuristic configurations for packet-filtering firewalls, the goal is to define a strategy capable of providing near-optimal VPN configurations in a fraction of the time, effectively bridging the gap between formal assurance and practical scalability.

1.2 Objectives of the Thesis

The main objective of this thesis is to overcome the scalability limits of the existing MaxSMT solvers, creating a novel heuristic algorithm capable of reaching a near-optimal VPN configuration, taking into account both network security and performance.

In order to do that, a deep inspection of the current MaxSMT approach within the VEREFOO framework has been conducted. By combining the existing formal concepts with the application of a heuristic strategy, originally designed for packet-filtering firewalls, this research led to the realization of a highly efficient solution tailored for VPNs.

Specifically, this thesis aims to achieve the following goals:

- **Algorithm Design:** Formalizing a polynomial-time iterative greedy heuristic that correctly allocates VPN gateways and rules, ensuring a "Safety-First" compliance.
- **Implementation:** Integrating the proposed heuristic logic into the open-source VEREFOO framework.
- **Validation:** Satisfying the necessity of an algorithm that scales efficiently with the network size, providing a resource-optimized and formally secure configuration. This will be proved by comparing its execution times and optimization quality against the traditional MaxSMT formulation.

1.3 Thesis Organization

This thesis is organized as follows:

- **Chapter 2 - Background and State of the Art:** introduces the fundamental concepts underlying this work. It focuses on Communication Protection Systems (specifically VPN architectures, the IPsec protocol suite, and TLS), detailing Security Associations and the challenges of policy anomalies. Furthermore, it introduces the VEREFOO framework, outlining its current MaxSMT-based approach for automated VPN configuration and its scalability limits.
- **Chapter 3 - The Heuristic VPN Configuration Algorithm:** presents the core theoretical contribution of the thesis. It details the proposed two-phase heuristic algorithm, explaining the "Safety-First" local placement strategy (Phase I) and the subsequent iterative consolidation based on the Intersection Matrix (Phase II), highlighting how it manages complex elements like Inspector Nodes.
- **Chapter 4 - Design and Implementation:** describes the practical integration of the proposed algorithm into the VEREFOO framework. It illustrates

the system architecture, the XML-based data model, the translation module for concrete firewall rules, and the software design choices made to implement the heuristic logic.

- **Chapter 5 - Validation and Performance Analysis:** evaluates the proposed solution. It presents a set of specific use cases to verify the formal correctness of the generated configurations and provides a comprehensive performance comparison between the new heuristic approach and the traditional MaxSMT solver, focusing on execution time and scalability.
- **Chapter 6 - Conclusions and Future Work:** summarizes the results achieved in this work and outlines potential directions for future research and development.

Chapter 2

Background and State of the Art

This chapter introduces the fundamental concepts and state-of-the-art technologies directly related to the core contribution of this thesis. It provides a detailed overview of Communication Protection Systems, focusing on Virtual Private Networks (VPNs) and the IPsec protocol suite, followed by an analysis of the VEREFOO framework and its current formal approach to automated VPN configuration.

2.1 Virtual Private Networks and IPsec

Protecting data in transit across shared or untrusted infrastructures is a critical requirement in modern telecommunications. A notable mechanism to achieve this is the Virtual Private Network (VPN), which employs both hardware and/or software to create a private network while utilizing shared or otherwise untrusted channels and transmission devices.

There are three primary techniques for creating a VPN: VPN via Private Addresses, VPN via IP Tunnel, and VPN via Secure IP Tunnel. The first two methods offer minimal protection; for instance, a VPN via Private Addresses provides a level of security close to zero and can be compromised if an attacker sniffs packets or accesses communication devices. Similarly, a standard IP Tunnel simply encapsulates traffic and can be bypassed by anyone managing a router. The third approach, however, provides robust cryptographic protection and is often referred to as an S-VPN (Secure VPN). Prior to encapsulation, an S-VPN protects packets using a Message Authentication Code (MAC) for integrity and authentication, encryption for confidentiality, and sequence numbering to guard against replay attacks. Due to performance constraints, digital signatures are not employed. If robust cryptographic algorithms are selected for the S-VPN, the only viable attack is to disrupt communications via a Denial of Service (DoS).

A VPN establishes secure connectivity allowing security policies to be enforced as if the communication were occurring within a private, isolated network. The key components of an S-VPN are the cryptographic tunnel, which securely encapsulates traffic while in transit, and the VPN Gateway, which serves as the interface between the public network and the protected corporate infrastructure.

2.1.1 VPN Topologies and Deployment Models

Depending on the architectural requirements, the scale of the network, and the specific entities involved in the communication, VPNs can be deployed using different topological models. The most used and known type of architectures include:

- **Site-to-Site VPNs:** This model securely connects entire isolated networks over a public infrastructure, such as linking a remote branch office to a corporate headquarters. The cryptographic operations are handled entirely by dedicated VPN gateways (or routers) located at the borders of each respective network. The individual hosts within the local networks do not require specialized VPN software, as the gateways transparently intercept, encrypt, and route the traffic across the tunnel.
  This approach is particularly suitable for interconnecting geographically distributed sites while maintaining a unified and secure corporate network infrastructure. It allows organizations to extend internal services across multiple locations without exposing sensitive data to the public Internet. As illustrated in Figure 2.1, two isolated networks (such as a Branch Office and Headquarters) can be securely connected over the public Internet, with their respective gateways utilizing IPsec in Tunnel Mode to ensure the entire original IP packet is completely encapsulated and protected. However, Site-to-Site VPNs generally require careful configuration and management of the gateway devices, as well as well-defined security policies to ensure proper segmentation and access control between interconnected networks.

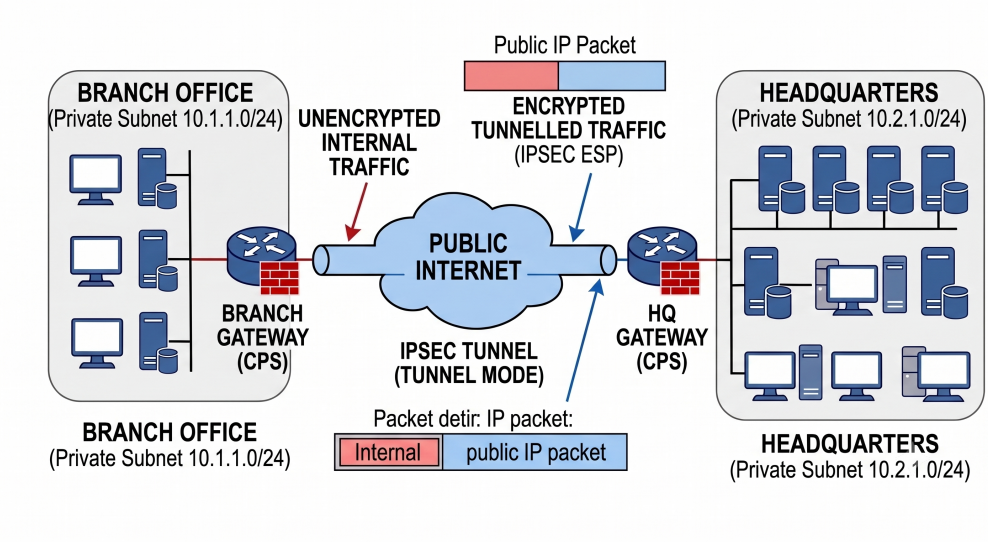


Figure 2.1. Site-to-Site VPN architecture utilizing IPsec in Tunnel Mode.

- **End-to-End (Host-to-Host) VPNs:** In this architecture, the secure tunnel is established directly between the two communicating endpoints (e.g., two specific data center servers or end-user devices). This approach requires both host machines to possess built-in VPN capabilities or dedicated client

software. While it ensures that data remains strictly encrypted across the entire path, including the internal local area networks (LANs), it significantly increases the configuration overhead and computational load on the individual hosts.
There is a possible disadvantage that can emerge in choosing End-to-End VPNs, involving IoT devices and resource-constrained systems. As illustrated in Figure 2.2, while this approach protects communication directly between end hosts, many IoT devices do not natively support VPN protocols due to limited computational power, memory constraints, or the absence of advanced networking capabilities. As a result, deploying end-to-end encryption across all devices may introduce compatibility issues, increased configuration complexity, and higher management overhead. Furthermore, maintaining VPN clients and cryptographic operations on a large number of low-power devices can negatively impact performance, battery life, and scalability, making centralized or gateway-based VPN architectures a more practical solution in certain contexts.

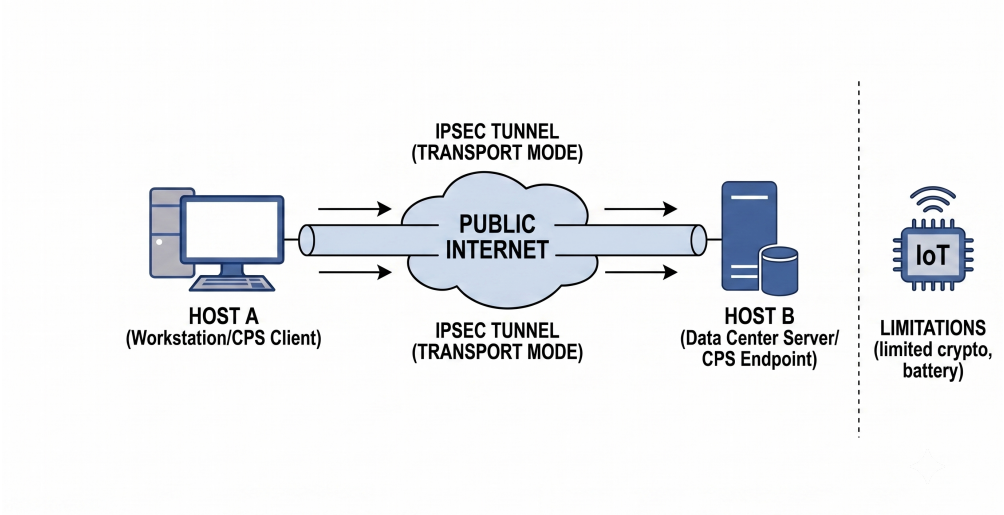


Figure 2.2. End-to-End VPN architecture and IoT deployment constraints.

- **Remote Access VPNs:** This model enables individual, distributed users (e.g., telecommuters or mobile workers) to securely connect to a centralized corporate network. A remote client establishes a secure tunnel directly with the corporate VPN gateway, granting the user authenticated access to internal resources as if they were physically connected to the local network.
As depicted in Figure 2.3, mobile and distributed users (whether from a home office or a cafe) establish direct encrypted connections with the corporate VPN gateway. In addition, this architecture typically relies on strong authentication mechanisms such as multi-factor authentication (MFA) and encryption protocols (e.g., IPSec or SSL/TLS) to ensure confidentiality and integrity of data in transit. Remote Access VPNs are particularly useful in modern distributed work environments, as they provide flexibility and mobility while maintaining centralized control over security policies and resource access.

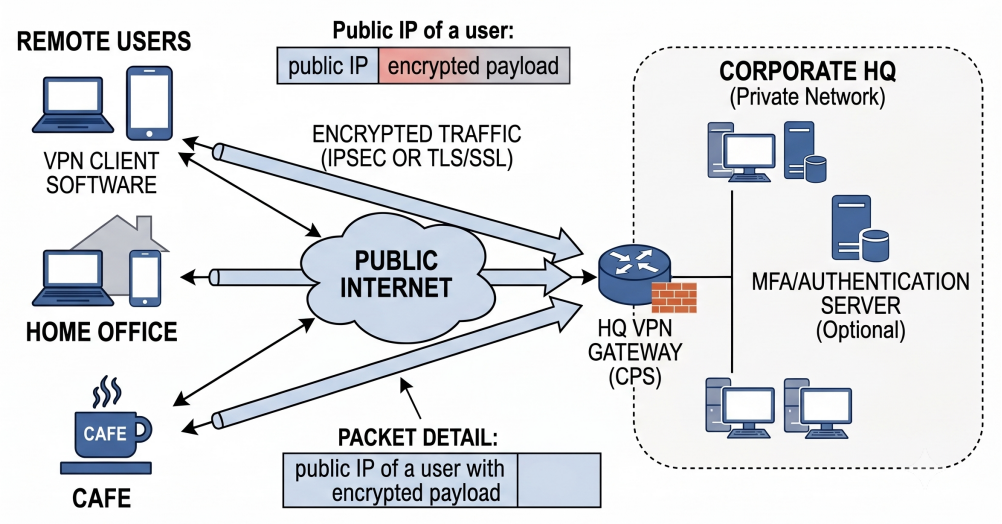


Figure 2.3. Remote Access VPN architecture and client connections.

2.1.2 The IPsec Protocol Suite

To ensure these secure tunnels, VPNs heavily rely on IPsec (Internet Protocol Security), which is the IETF (Internet Engineering Task Force) architecture for Layer 3 security in IPv4/IPv6 designed to create a Secure VPN (S-VPN) over untrusted networks and establish end-to-end secure packet flows. IPsec provides security at the network layer through robust authentication and encryption mechanisms.

This is achieved through the definition of two specific packet types:

- **Authentication Header (AH):** Provides integrity, authentication, and partial protection against replay attacks.
- **Encapsulating Security Payload (ESP):** Offers functions similar to AH, with the critical addition of payload confidentiality. It is crucial to emphasize that confidentiality can only be provided for the payload; it is never possible to encrypt the IP header, as intermediate systems would otherwise be unable to process and route the packets.

Additionally, IPsec relies on a dedicated protocol named IKE (Internet Key Exchange) to securely create and distribute keys across IP networks.

IPsec operates in two primary modalities, which closely align with the deployment models described above:

- **Transport Mode**: Typically utilized for *End-to-End* communications between two specific hosts, rather than gateways. It authenticates or encrypts only the payload of the IP packet, leaving the original IP header intact and visible for routing. While it is computationally light, it presents the disadvantage of offering no protection for the header variable fields.

- **Tunnel Mode**: The standard choice for network-to-network (*Site-to-Site*) or *Remote Access* communications, typically deployed among gateways. It protects both the entire original IP header and the payload by encapsulating the original end-to-end packet within a new outer IP tunnel packet, effectively hiding the internal routing details from external observers. While it provides complete protection of the packet (including header variable fields), it is computationally heavy.

2.1.3 TLS-based VPNs as an Alternative Protocol

While IPsec operates at the network layer and is considered the standard for Site-to-Site topologies, modern VPN deployments also heavily rely on the Transport Layer Security (TLS) protocol (formerly SSL). TLS-based VPNs operate at the transport and application layers, securing the communication channel directly between a client application (such as a web browser or a dedicated client) and a remote server or gateway.

TLS VPNs offer several distinct architectural advantages that make them a crucial alternative or complement to IPsec:

- **Firewall and NAT Traversal:** Because TLS traffic is typically encapsulated within standard HTTPS traffic (TCP port 443), it easily passes through most Network Address Translation (NAT) devices and restrictive firewalls. IPsec, which relies on specific protocols like ESP (IP protocol 50) and IKE (UDP port 500), is frequently blocked or corrupted by standard network middleboxes unless specific NAT-Traversal mechanisms are configured.
- **Granular Access Control:** TLS VPNs inherently provide application-level access control, allowing administrators to grant users access to specific internal web applications or services without exposing the entire underlying network layer, which is typical of IPsec tunnels.

In the context of automated network orchestration, the coexistence of IPsec and TLS is fundamentally important. An advanced orchestrator must be capable of dynamically selecting the most appropriate protocol based on the specific constraints of the network topology. For instance, if an intermediate middlebox strictly blocks IPsec traffic, the orchestration system should possess the capability to perform an automated protocol switch to TLS, ensuring continuous and secure connectivity without human intervention.

2.1.4 Security Associations and Policy Management

The fundamental building block of any IPsec-based communication is the Security Association (SA). An SA is a unidirectional logical connection between two IPsec systems that grants specific security services to the traffic carried on it. Because SAs are strictly unidirectional, establishing a standard bidirectional secure VPN exchange requires the creation of at least two separate security associations.

A Security Association is uniquely identified by three distinct parameters:

1. **Security Parameters Index (SPI)**: A unique identifier included in the IPsec header, used by the receiving node to search and match the correct SA in the database.
2. **IP Destination Address**: The address of the destination endpoint terminating the SA.
3. **Security Protocol Identifier**: An indicator specifying whether the association utilizes the Authentication Header (AH) for integrity or the Encapsulating Security Payload (ESP) protocol for both confidentiality and integrity.

To manage these connections and determine exactly which traffic requires protection, IPsec relies on Policy-based Network Management through two local databases:

- **Security Policy Database (SPD):** The SPD contains an ordered list of security rules and policies to apply to different packet flows. It specifies how incoming and outgoing IP traffic should be processed, specifically whether a packet should be permitted in clear text (bypass), dropped, or protected via an SA. The SPD can be configured a-priori manually or connected to an automatic system.
- **Security Association Database (SAD):** The SAD is a runtime database that contains the list of currently active SAs and their specific characteristics (such as algorithms, keys, and parameters) required to create protected traffic for that specific SA.

In complex scenarios, an individual SA might not be sufficient, and multiple SAs can be employed for the same traffic flow to achieve the desired services. This is known as a security association bundle, which can be implemented via *Transport adjacency* (applying more than one security protocol to the same IP packet without invoking tunneling) or *Iterated tunneling* (the application of multiple layers of security protocols effected through IP tunneling).

However, managing these SPD rules manually across multiple distributed VPN gateways often leads to severe policy rules anomalies. These anomalies occur when a set of policies simultaneously satisfies the same traffic, and they are generally categorized into:

- **Inter-Policy Anomalies:** Conflicts arising among rules of different, distributed firewalls or gateways.
- **Intra-Policy Anomalies:** Anomalies occurring among rules of the same device. These are further divided into:
  - *Conflicts:* Situations where the effect of one policy is influenced or altered by another. This includes *Contradiction* (rules matching the same

packets but specifying different actions), *Shadowing* (a higher priority rule completely hiding a lower priority rule), and *Correlation* (partially overlapping rules with different actions).

– *Sub-optimizations:* Situations where redundant rules degrade system performance, including *Irrelevance*, *Duplication*, *Shadow Redundancy*, and *Unnecessary* rules.

2.1.5 VPN Gateway Positioning and Topological Anomalies

VPN management complexity is not limited to rule inconsistencies, it also involves the strategic physical and logical positioning of the gateways The positioning of a VPN Gateway relative to other security appliances (like firewalls or access routers) profoundly impacts network security. Common positioning strategies include:

- **Inside:** The VPN gateway is placed behind the firewall, meaning there is no initial inspection of VPN traffic by the firewall, but the gateway itself is protected from direct external attacks.
- **Outside:** The VPN gateway is placed outside the main firewall, often protected only by an access router.
- **Parallel:** The VPN gateway is placed parallel to the firewall, which can lead to potential uncontrolled access paths into the network.
- **Integrated:** The VPN capabilities are integrated directly into the firewall device, offering maximum flexibility.

Incorrect topological placement or flawed automated orchestration can lead to specific VPN-related vulnerabilities:

- **Monitorability Anomaly:** Occurs when some nodes at the channel junctions can improperly "see" the exchanged data in clear text when they should not.
- **Skewed Channel Anomaly:** Occurs when a wrong tunnel overlapping accidentally removes the confidentiality in a segment of the communication path.

The presence of these complex policy and topological anomalies highlights the crucial need for advanced, automated configuration tools.

2.2 The VEREFOO Framework for VPN Automation

The manual configuration of Communication Protection Systems (CPSs), such as the aforementioned VPN gateways, is a highly complex and error-prone task. Incorrect rule definition or suboptimal placement of CPSs can lead to severe security

vulnerabilities, such as unencrypted traffic crossing untrusted network zones, or network inefficiencies due to redundant encryption operations.

To address these issues, the **VEREFOO** (VErifier and REconfigurator for Out-Sourcing) framework was conceived and developed by the **NetGroup research team at Politecnico di Torino** [2]. Driven by continuous academic research, the framework proposes an advanced automated policy refinement process capable of computing both the optimal allocation scheme and the specific protection rules of distributed CPSs.

By taking high-level security goals as input, VEREFOO eliminates the need for manual, low-level device configuration. The methodology requires three main inputs:

- **Allocation Graph (AG):** A formal representation of the physical or virtual network topology, including endpoints, specific middleboxes, and Allocation Places (APs) which act as placeholder positions where new CPSs may be deployed.
- **Communication Protection Policies (CPPs):** A set of high-level, disjoint security requirements. Each CPP specifies the traffic conditions, the required cryptographic algorithms for confidentiality and integrity, the accepted VPN protocols, and the trustworthiness of the network nodes and links involved in the communication path.
- **Optimization Profile:** A criterion used to guide the solver toward the best possible configuration among all the valid ones.

2.2.1 The MaxSMT Formal Approach

The core of the VEREFOO methodology consists of translating the auto-configuration problem into a Maximum Satisfiability Modulo Theories (MaxSMT) problem [2]. Preliminary studies successfully applied this formal strategy to virtualized networks, aiming to automatically allocate and configure Channel Protection Systems (CPSs) [3]. By modeling the policy refinement process as a MaxSMT problem, this foundational approach uniquely combines automation, formal verification (through a correctness-by-construction paradigm), and optimization criteria into a single technique [3]. A MaxSMT formulation extends a traditional Boolean Satisfiability (SAT) problem by evaluating logical formulas over various mathematical theories (e.g., integers, uninterpreted functions) and by supporting two distinct types of constraints: hard and soft constraints.

As detailed in the foundational work on automated VPN configuration [2], the framework models the network topology as an Allocation Graph and translates the high-level Communication Protection Policies (CPPs) into explicit logical clauses. These clauses are then processed by a state-of-the-art solver (such as Microsoft's Z3).

Hard constraints are mandatory conditions that must be entirely satisfied to generate a valid solution. In the context of VPN orchestration, these constraints

guarantee the formal *correctness-by-construction* of the solution [2]. They mathematically ensure that the generated VPN tunnels respect all the input CPPs. For instance, they force the traffic to be encrypted before crossing a node designated as untrusted, or compel the traffic to be extracted into plain text when crossing a designated inspector node (e.g., an Intrusion Detection System). If a network topology makes it physically impossible to satisfy all hard constraints simultaneously, the solver correctly flags the problem as unsatisfiable (UNSAT), preventing the deployment of a flawed security architecture.

Soft constraints, on the other hand, are associated with a specific numerical weight, and the solver aims to maximize the sum of the weights of the satisfied clauses. Soft constraints enable the achievement of optimization objectives defined by the network administrator. Two primary optimization profiles are commonly implemented [2]:

- **Min-allocation:** Aims to minimize the total number of allocated VPN gateways, typically favoring end-to-end VPNs over intermediate site-to-site tunnels to reduce the hardware footprint and licensing costs.
- **Min-bandwidth:** Aims to improve network performance by avoiding unnecessary encryption overhead, preferring the allocation of CPSs in intermediate nodes to keep the protected, encapsulated paths as short as possible.

2.2.2 Scalability and Performance Challenges

While the MaxSMT formulation successfully provides full automation, formal correctness, and optimized deployments, its main limitation resides in its computational complexity. The MaxSMT problem is a generalization of the SAT problem, making it inherently NP-complete [2].

State-of-the-art solvers (such as Z3) employ advanced algorithms to solve these instances in polynomial time on average for networks of limited sizes. To illustrate this scalability boundary concretely, the experimental validation that was conducted [2] provides a comprehensive performance evaluation on both synthetic and real-world network topologies. In the analysis, the MaxSMT solver successfully orchestrated synthetic environments of increasing complexity, scaling from a baseline of 35 nodes up to a maximum tested scenario of 260 network nodes, 100 Allocation Places, and 400 Communication Protection Policies (CPPs). For this highly complex case, the solver computed the optimal allocation and ruleset in approximately 300 seconds [2].

Furthermore, the authors tested the framework on highly ramified production networks, such as the European GÉANT and the Asian APAN topologies. The results highlighted that network ramification significantly impacts performance: because the number of free variables and logical constraints strictly depends on the number of concurrent traffic flows and possible routing paths, highly interconnected networks cause a severe spike in computational complexity [2]. For instance,

despite GÉANT having fewer overall vertices than APAN, its highly ramified structure generated significantly more traffic flows, leading to higher computation times (approximately 150 seconds to enforce 200 CPPs) [2].

While these execution times represent a massive improvement over error-prone manual configuration, they also expose the inevitable "computational wall" of exhaustive formal solvers. As explicitly acknowledged in the foundational study, the MaxSMT approach "cannot manage computer networks composed of thousands of nodes" [2]. Because the underlying problem is NP-complete, the execution time faces an exponential growth trend as the problem size expands toward massive scales [2].

In large-scale, dynamic scenarios such as modern cloud infrastructures, where thousands of nodes are dynamically instantiated and network orchestrators require real-time responsiveness, waiting several minutes to compute a security response or configure a new tenant is unacceptable. This strict scalability limitation inherently motivates the exploration of alternative approaches, such as polynomial-time heuristic algorithms, capable of bridging the gap between the rigorous formal assurance of MaxSMT models and the practical, real-time performance requirements of massive real-world networks.

Recent research has actively begun addressing these specific performance overheads and memory consumption issues within the VEREFOO architecture. For instance, enhanced MaxSMT strategies have been proposed to minimize the number of configured security rules and streamline the generation of formal constraints, aiming to improve computational efficiency without sacrificing correctness-by-construction [4]. However, despite these significant algorithmic optimizations, the inherent exponential complexity of exact mathematical solvers persists as the problem size expands toward massive scales [1]. In large-scale, dynamic scenarios such as modern cloud infrastructures, where thousands of nodes are dynamically instantiated and network orchestrators require real-time responsiveness, waiting to compute a security response is unacceptable. This strict scalability limitation inherently motivates the exploration of alternative approaches, such as polynomial-time heuristic algorithms.

2.3 State of the Art in Automated Network Security

The automation of network security configurations, particularly concerning cryptographic tunnels and policy enforcement, sits at the intersection of several distinct research domains. To properly contextualize the heuristic orchestrator proposed in this thesis, it is necessary to examine how both the academic community and the networking industry approach the challenges of formal verification, automated orchestration, and scalable function placement.

Before addressing VPN orchestration, the formal approach based on MaxSMT was extensively validated for the automated allocation and configuration of distributed firewalls [5]. Subsequently, the need to adapt to highly dynamic virtual

networks led to the extension of these models to optimize the reconfiguration phases of packet filters. To counteract modern fast-mutating attacks, these automated solutions update existing deployments in significantly reduced timeframes, strictly avoiding the computational overhead of regenerating the entire network configuration from scratch [6], while safely managing state transients to avoid temporary network vulnerabilities [7]. Building upon these rapid reconfigurations, fully autonomous threat response mechanisms have been successfully developed [8]. These advanced architectures eliminate the need for human intervention by continuously monitoring Intrusion Detection System (IDS) logs, automatically extracting new security requirements, and deploying the optimized firewall reconfigurations in real-time to mitigate ongoing cyber attacks [8].

More recently, this research trajectory has embraced environmental sustainability, focusing on energy-aware deployments. A notable advancement in this domain is GreenShield, a formal methodology that optimizes the automated configuration of distributed firewalls to explicitly minimize network power consumption [9]. By expressing green-oriented objectives as soft constraints within a MaxSMT formulation, GreenShield ensures formal policy compliance while activating the least energy-consuming combination of firewalls and actively blocking undesired traffic as close to its source as possible [9]. This foundational concept has subsequently been extended to the edge-to-cloud continuum with GreenShield-E2C [10]. This enhanced version manages the complex heterogeneity of multi-layer infrastructures by dynamically optimizing the allocation and configuration of both physical and virtual firewalls, effectively minimizing overall power consumption while explicitly accounting for the energy impact of the underlying virtualization technologies across the edge, fog, and cloud layers [10].

Beyond energy awareness, the continuous evolution toward highly distributed infrastructures has expanded the investigation into cloud-native orchestration. To manage the complex multi-tenant dynamics and cross-cluster communications of the edge-to-cloud continuum, intent-driven methodologies have been proposed to automate network isolation [11]. These advanced orchestrators seamlessly harmonize conflicting security objectives across different administrative domains, translating high-level user intents directly into actionable Kubernetes Network Policies [11].

Furthermore, contemporary research has highlighted that performing the automated configuration of a single security function type at a time can be highly restrictive and error-prone [12]. For instance, the disjoint configuration of packet filtering firewalls and Channel Protection Systems (CPSs) can lead to severe inter-function conflict anomalies; the encryption rules applied by a VPN gateway might encapsulate traffic and directly invalidate the filtering conditions previously established by a firewall [12]. To mitigate these structural anomalies, recent studies have begun exploring the simultaneous, joint configuration of multiple security functions using MaxSMT formulations, ensuring that the deployment models and rule generations remain globally consistent across the virtualized infrastructure [12].

However, the transition from classic firewalls to VPNs introduces a level of

non-linear topological complexity (such as traffic encapsulation and fallback mechanisms for Inspector Nodes) that intensifies the computational limits of MaxSMT, making the development of a dedicated heuristic approach necessary.

Chapter 3

The Heuristic VPN Configuration Algorithm

This chapter explains and highlights the primary theoretical contribution of this thesis: a polynomial-time iterative greedy heuristic algorithm designed for the automated allocation and configuration of distributed Virtual Private Network (VPN) gateways. The algorithm aims to overcome the strict scalability barriers of exact mathematical formulations, such as Maximum Satisfiability Modulo Theories (MaxSMT) models, while structurally preserving the formal correctness-by-construction of the generated security configurations.

3.1 From Firewall to VPN Heuristics: Challenges and Differences

The automation of network security configuration has traditionally focused on packet-filtering firewalls, with numerous studies establishing formal methods and MaxSMT problem formulations for their optimal allocation and rule generation [13]. However, building practical rules for decentralized crypto systems takes a completely different approach than standard access control. Although both firewalls and Virtual Private Networks (VPNs) are used to enforce network security policies, the underlying mechanics of how they achieve compliance are inherently different. Specifically, filtering operations performed by firewalls only impact the forwarding behavior of a network, whereas the encryption and authentication mechanisms employed by VPN gateways actively modify the structure and encapsulation of the traffic itself. The combination of this architectural distinction and the necessity to evaluate the trustworthiness of nearby network elements makes the automatic allocation of VPN gateways a significantly more complex operational challenge.

3.1.1 Single-Node vs. Pairwise Allocation

In the context of firewalls, the allocation logic can be successfully modeled using a single-node evaluation scheme. Heuristic approaches for packet filters typically compute an independent weight for each Allocation Place (AP) based on the number of traffic flows crossing it and the specific type of network security requirements associated with those flows. Each AP is scored and selected independently to intercept unauthorized traffic flows as close to the source as possible.

In contrast, a VPN function operates inherently as a distributed, point-to-point architecture. Deploying an encryption gateway at a given Allocation Place (AP) requires by nature a corresponding decryption gateway downstream; failing to provision both simultaneously disrupts the network flow. Furthermore, deciding the allocation scheme requires a continuous topological analysis of the trustworthiness of the nodes and links between the two endpoints. The heuristic must therefore abandon the single-node scoring system and adopt a strict *pairwise evaluation* paradigm, where candidate tunnels (SA_{in} and SA_{out}) are generated, scored, and instantiated as cohesive logical entities.

3.1.2 Middlebox Interferences and Protocol Switching

Modern networks are heavily populated by pre-existing network functions, such as Network Address Translators (NATs) or traditional packet filters, which strictly regulate the traffic allowed to pass through specific segments. While a packet filter simply drops or allows independent packets without altering their structural profile, a VPN encapsulates the original data inside a new cryptographic protocol header (e.g., ESP for IPsec or a TLS record).

If a pre-existing firewall located between the two chosen VPN endpoints is configured to block IPsec traffic, the newly created tunnel will be divided. To overcome this, the VPN heuristic must preventively evaluate the predicates of every intermediate network function encompassed within the candidate tunnel path $\pi(f)$. As illustrated in the lower diagram of Figure 3.1, if a protocol like IPsec cannot successfully traverse an intermediate middlebox, the algorithm must dynamically react by performing a Protocol Switch, automatically attempting to establish the tunnel using an alternative allowed protocol, such as TLS.

3.1.3 Fallback Mechanisms and Inspector Exceptions

An additional layer of complexity arises from the mathematical necessity to accommodate traffic inspection. Corporate security policies frequently contradict each other: traffic must be encrypted across untrusted zones, yet it must remain fully visible to designated entities known as Inspector Nodes (N^I), such as Intrusion Detection Systems (IDS), which require clear-text data for deep packet inspection. A firewall heuristic simply drops malicious traffic, permanently terminating the flow.

A VPN heuristic, however, must protect the flow without blinding these intermediate inspection systems. As depicted in the upper diagram of Figure 3.1, if an active inspector node is detected inside a proposed tunnel boundary, it must implement a Fallback mechanism. This forces the algorithm to prematurely terminate the cryptographic tunnel immediately before the inspector node, allow the system to perform signature matching on clear-text data, and initiate a new, consecutive tunnel immediately following the inspection boundary.

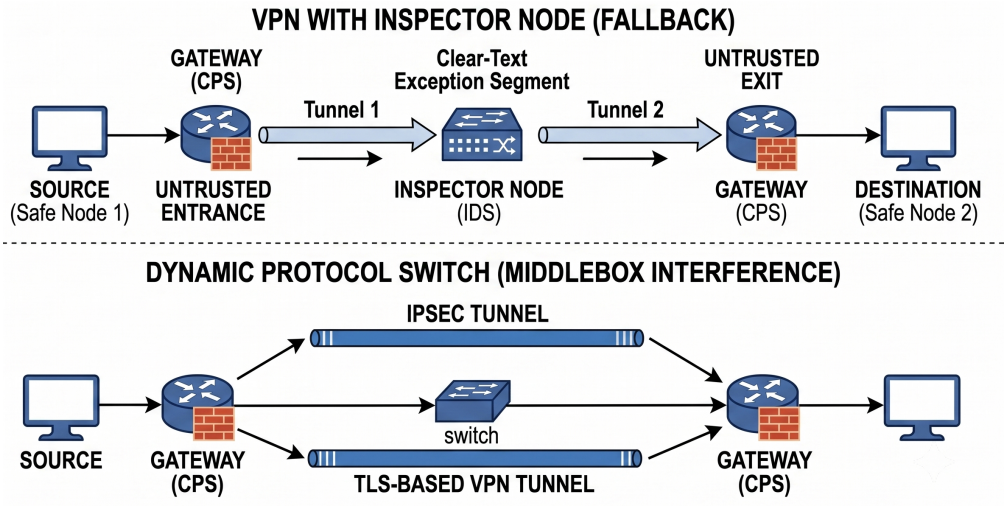


Figure 3.1. Heuristic mechanisms for Fallback mitigation and dynamic Protocol Switching.

3.2 Algorithm Overview and Strategy

The proposed heuristic framework operates via a split execution architecture driven by specific strategic paradigms, in order to navigate this complex solution space efficiently, bypassing the overhead of an exhaustive and exponential search.

3.2.1 The "Site-to-Site First" Strategy

To guarantee a resource-optimized topology, the heuristic is driven by a *Min-Bandwidth* optimization profile. The fundamental strategy adopted is the "Site-to-Site First" approach.

Instead of defaulting to End-to-End VPNs (which deploy gateways directly at the source and destination hosts), the algorithm prioritizes the allocation of Communication Protection Systems (CPS) at intermediate nodes, as close to the critical danger zones as possible. By restricting the length of the tunnel strictly to the untrustworthy segments, the algorithm minimizes the total network path where traffic travels encapsulated, drastically reducing the bandwidth overhead and the encryption/decryption processing burden on the network infrastructure.

3.2.2 Conceptual Workflow of the Heuristic Approach

Conceiving an algorithm for network security requires reaching a delicate balance: it must be mathematically rigorous, yet practically scalable. Before detailing the formal rules, it is useful to visualize the algorithm's logic. Instead of trying to find the mathematically "perfect" configuration by analyzing every possible combination (which causes the MaxSMT scalability issues), the algorithm uses a **Greedy approach**, making the best possible local choice step-by-step. It operates in two distinct logical phases:

1. **Phase I: Local Pre-allocation (The "Safety-First" Approach):** The algorithm acts as a rapid first responder. It scans the path of a traffic flow to identify the *Maximal Contiguous Danger Zone*. To shield the data, it looks mathematically backward to find the last safe node before the danger (entrance) and forward to find the first safe node after the danger (exit). Furthermore, if a VPN tunnel completely covers an Inspector Node, the algorithm actively splits the tunnel in two, allowing clear-text analysis in the middle. By the end of Phase I, the network is formally secure, but highly inefficient due to redundant tunnels.
2. **Phase II: Global Consolidation (The "Resource Optimization" Phase):** To resolve redundancies, the algorithm creates an Intersection Matrix comparing every single tunnel against every other tunnel. It iteratively finds the pair of tunnels with the absolute highest path overlap and merges them into a single, larger tunnel, saving gateway resources. Crucially, it employs a *Dynamic Delta Update*: once a merge happens, it updates only the matrix cells related to the new tunnel, rather than recalculating the entire grid. This cycle repeats until no more feasible merges exist.

3.3 Mathematical Formalization

To ensure academic rigor and avoid informal procedural descriptions, the heuristic relies strictly on set theory and path sequence operations.

3.3.1 Formal Model of Communication Protection Policies

To guarantee the formal correctness of the heuristic execution, it is necessary to first define the mathematical structure of the input policies. Let P be the set of Communication Protection Policies (CPPs). Each policy $p \in P$ is mathematically modeled as a 7-tuple:

$$p = (C, A^c, A^i, A^a, V, S, W) \tag{3.1}$$

where each element formally dictates a specific constraint that the orchestrator must enforce:

- C **(Traffic Conditions):** Identifies the specific data flow subject to the policy. This is typically defined through standard packet header matching fields, such as Source IP, Destination IP, Transport Protocol, and Port ranges.
- A^c, A^i, A^a **(Security Actions):** Define the mandatory enforcement levels regarding the fundamental security triad. Specifically, they dictate the requirements for **C**onfidentiality (e.g., data encryption), **I**ntegrity (e.g., tampering prevention via hashing), and **A**uthentication (e.g., endpoint verification).
- V **(VPN Constraints):** Specifies the allowable architectural models and topological constraints for the tunnel (e.g., enforcing a Site-to-Site rather than a Host-to-Host configuration).
- S **(Cryptographic Suites):** Outlines the set of acceptable security protocols and algorithmic parameters (e.g., IPsec with ESP, TLS, AES-256) required to satisfy the assigned security actions.
- W **(Trustworthiness Information):** Provides the topological security context of the underlying Allocation Graph, defined as the sub-tuple $W = (N^U, N^T, N^I, L^U, L^T)$.

Specifically, within the trustworthiness parameter W, the topological elements are classified as follows:

- N^U **and** L^U define the sets of *Untrustworthy Nodes and Links*. These represent the physical or virtual danger zones, dictating exactly the boundaries where the traffic must be cryptographically encapsulated.
- N^T **and** L^T define the sets of *Trustworthy Nodes and Links*, representing secure infrastructure where clear-text traffic is safely permitted without violating the policy.
- N^I identifies the set of *Inspector Nodes* (e.g., Intrusion Detection Systems or Deep Packet Inspectors). These middleboxes explicitly require traffic to be delivered in clear-text for security analysis, structurally forcing the heuristic to evaluate tunnel splitting and fallback mechanisms.

3.3.2 Definitions and Formal Rules

To efficiently manage the state-space of the underlying formal models, the representation of network traffic must be highly optimized. State-of-the-art methodologies demonstrate that aggregating packet classes into unified traffic flow models, such as Maximal Flows, drastically reduces the number of variables the solver must evaluate, thereby improving the overall scalability of the network security management process [14]. Building upon these modeling principles, let $\pi(f) = [n_1, n_2, \ldots, n_k]$ be the ordered sequence of nodes traversed by a flow f.

Let $\pi(f) = [n_1, n_2, \ldots, n_k]$ be the ordered sequence of nodes traversed by a flow f. The precedence operator $\prec_\pi$ indicates that a node appears before another in the path.

- **Sequence Successor:** To define active links strictly along a flow, we denote $n_y = \text{succ}(n_x, \pi(f))$ if n_y immediately follows n_x in $\pi(f)$.
- **Security Association Path ($\pi(SA)$):** A Security Association $SA(n_{in}, n_{out}, f)$ protects a specific segment of a flow. $\pi(SA)$ is the contiguous sub-sequence of $\pi(f)$ starting at n_{in} and ending at n_{out}.
- **Path Extension Set (E):** When evaluating the consolidation of two tunnels, E represents the set of additional nodes that a tunnel must traverse to unify its endpoints with the other. For instance, in an inclusion (Case 2) where $\pi(SA_i) \subset \pi(SA_j)$, the required extension for SA_i is explicitly defined by the set difference $E = \pi(SA_j) \setminus \pi(SA_i)$.
- **Shielding Gateway Selection:** To ensure mathematical consistency when evaluating the precedence operator $\prec_\pi$ (which is strictly defined over the node sequence), a contiguous danger zone $z \in Z$ is bounded by its first and last untrustworthy topological elements. Let n_{d_first} be the first node of zone z (or the source node if the element is a link) and n_{d_last} be the final node of zone z.

  It is formally assumed that the source n_s and destination n_d of any flow are implicitly considered valid Allocation Places ($n_s, n_d \in N^A$). This assumption mathematically guarantees that the sets evaluated for gateway selection are never empty. The optimal entry and exit gateways are extracted as:

$$n_{in}(z) = \arg\max_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid n \prec_\pi n_{d_first}\} \tag{3.2}$$

$$n_{out}(z) = \arg\min_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid n_{d_last} \prec_\pi n\} \tag{3.3}$$

- **Topological Evaluation:** For any pair of active SAs, the overlap length ($L_{i,j}$) and topological case ($C_{i,j}$) are evaluated via set intersections:

$$L_{i,j} = |\pi(SA_i) \cap \pi(SA_j)| \tag{3.4}$$

$$C_{i,j} = \begin{cases} 1 & \text{if } \pi(SA_i) = \pi(SA_j) \\ 2 & \text{if } \pi(SA_i) \subset \pi(SA_j) \vee \pi(SA_j) \subset \pi(SA_i) \\ 3 & \text{if } (\pi(SA_i) \cap \pi(SA_j) \neq \emptyset) \wedge \text{not Case 1, 2} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

- **Feasibility Constraint Function Φ:** A merge is feasible ($\Phi = \text{true}$) if and only if the newly generated unified path ($\pi(SA_{new}) = \pi(SA_i) \cup \pi(SA_j)$) does not encapsulate any active inspector node. Checking only the path extensions is insufficient, as merging two adjacent tunnels might inadvertently cover an inspector originally located between them, thus violating a separation established in Phase I. Therefore, absolute safety is guaranteed by verifying the complete sequence:

$$\Phi(SA_i, SA_j, C_{i,j}) = \begin{cases} \text{true} & \text{if } C_{i,j} = 1 \\ \text{true} & \text{if } C_{i,j} \in \{2, 3\} \wedge (\pi(SA_{new}) \cap p.W.N^I = \emptyset) \\ \text{false} & \text{otherwise} \end{cases} \tag{3.6}$$

3.4 Detailed Algorithmic Execution

3.4.1 Phase I: Handling Multiple Danger Zones and Inspectors

In complex network topologies, a single traffic flow may traverse multiple, distinct untrustworthy domains separated by trusted infrastructure. To handle this, Phase I does not stop at the first violation. Instead, it identifies all *Maximal Contiguous Danger Zones*, sub-paths composed entirely of untrusted nodes or links. For each distinct zone, an independent pre-allocation process is executed.

Furthermore, the algorithm accommodates multiple Inspector Nodes (N^I) within a single protected segment. Once a shielding tunnel is mapped, the algorithm extracts the ordered set of all enclosed inspectors. The tunnel is iteratively split around each inspector. If multiple inspectors are strictly adjacent, the clear-text segment is naturally expanded to cover the entire sequence; if they are separated by trusted nodes, intermediate cryptographic tunnels are established between them.

3.4.2 Phase II: Iterative Consolidation and Dynamic Delta Update

Phase II systematically resolves the inefficiencies of the initial placement. Instead of rebuilding the Intersection Matrix M entirely at each step, the algorithm employs a **Dynamic Delta Update** optimization. The full matrix M is initialized only once. During the continuous loop, the algorithm selects the feasible pair with the maximum overlap ($\arg\max M_{i,j}$.length), merges them into SA_{new}, and removes their respective rows/columns. Crucially, the algorithm computes the new intersections *only* between SA_{new} and the remaining active SAs. This cache-based approach reduces the *intersection recalculation time* from $\mathcal{O}(|SA|^2)$ to $\mathcal{O}(|SA|)$ per iteration, leaving the global maximum selection over the active candidates as the primary computational operation.

3.5 Heuristic Algorithm Specification

The overall heuristic approach has been structured into two distinct operational algorithms to clearly separate the logical boundaries between the security-first allocation and the subsequent topological optimization.

3.5.1 Algorithm 1: Local Pre-allocation

The first algorithm processes the network flows independently to establish a fully compliant, unoptimized, baseline of cryptographic tunnels.

Algorithm 1 Heuristic CPS Allocation - Phase I: Local Pre-allocation

Require: $G = (N, L)$: Allocation Graph; P: Set of Policies (CPPs).
Ensure: SA_{list}: Initial unoptimized set of Security Associations.

$SA_{list} \leftarrow \emptyset$
for all $p \in P, f \in \phi(p)$ **do**
  Let $\pi(f) = [n_s, \ldots, n_d]$ be the ordered path sequence of f
  // Identify all disjoint Maximal Contiguous Danger Zones
  Let $Z = \{z_1, z_2, \ldots, z_m\}$ be the set of contiguous sub-paths in $\pi(f)$ where every element satisfies $(n_k \in p.W.N^U) \vee (l_{x,y} \in p.W.L^U \wedge n_y = \text{succ}(n_x, \pi(f)))$
  for all $z \in Z$ **do**
    Let d_{first} and d_{last} be the first and last elements of zone z
    $n_{in} \leftarrow \arg\max_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid n \prec_\pi d_{first}\}$
    $n_{out} \leftarrow \arg\min_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid d_{last} \prec_\pi n\}$
    // Multiple Inspector Nodes Evaluation (Eq. 11 Compliance)
    Let $I_z = \{n_i \in p.W.N^I \mid n_{in} \prec_\pi n_i \prec_\pi n_{out}\}$ ordered by $\prec_\pi$
    if $I_z \neq \emptyset$ **then**
      Let $n_{curr} = n_{in}$
      for all $n_i \in I_z$ **do**
        $n_{split} \leftarrow \arg\max_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid n \prec_\pi n_i\}$
        if $n_{curr} \prec_\pi n_{split}$ **then** ▷ Prevents zero-length SAs
          $SA_{list} \leftarrow SA_{list} \cup \{SA(n_{curr}, n_{split}, f)\}$
        end if
        $n_{curr} \leftarrow \arg\min_{\prec_\pi} \{n \in \pi(f) \cap N^A \mid n_i \prec_\pi n\}$
      end for
      $SA_{list} \leftarrow SA_{list} \cup \{SA(n_{curr}, n_{out}, f)\}$ ▷ Final segment
    else
      $SA_{list} \leftarrow SA_{list} \cup \{SA(n_{in}, n_{out}, f)\}$
    end if
  end for
end for
return SA_{list}

Phase I Walkthrough:
The execution logic of Algorithm 1 can be detailed step-by-step as follows:

- **Lines 1-3:** The algorithm initializes the SA_{list} and iterates through each required security policy f, retrieving its corresponding ordered routing path $\pi(f)$.
- **Lines 4-5:** The procedure identifies all the contiguous danger zones Z within the path by checking the trustworthiness of each topological element (nodes and links).
- **Line 6:** A loop initiates to process each distinct danger zone $z \in Z$ independently.
- **Lines 7-9:** The optimal gateway selection is performed. The boundaries d_{first} and d_{last} are extracted, and the gateways n_{in} and n_{out} are calculated using the arg max and arg min operators to find the closest safe Allocation Places before and after the danger.
- **Lines 10-12:** The algorithm identifies any active inspector nodes (I_z) enclosed between the newly selected gateways n_{in} and n_{out}, and evaluates if this subset is empty.
- **Lines 13-21:** If one or more inspectors are present, the fallback mechanism for middlebox visibility is triggered. The algorithm iterates over each inspector n_i (**Lines 14-20**). At **Line 15**, it calculates n_{split}. **Lines 16-18** enforce the zero-length SA prevention, adding the partial tunnel to SA_{list} only if n_{curr} strictly precedes n_{split}. At **Line 19**, the starting point n_{curr} is updated. After processing all inspectors, the final segment bridging to n_{out} is appended (**Line 21**).
- **Lines 22-24:** If no inspectors are detected within the zone, the standard procedure is applied in the **else** branch: a single Security Association spanning from n_{in} to n_{out} is instantiated to cover the entire danger zone.
- **Line 27:** The algorithm returns the fully populated, unoptimized SA_{list}, concluding Phase I.

3.5.2 Algorithm 2: Global Consolidation

The second algorithm takes the highly redundant output of Phase I (SA_{list}) and applies the Intersection Matrix logic coupled with the dynamic delta update to minimize resource overhead.

Algorithm 2 Heuristic CPS Allocation - Phase II: Global Consolidation

Require: SA_{list}: Initial set of Security Associations from Phase I; P: Set of Policies.
Ensure: SA_{final}: Optimized set of Security Associations.

1: *// 1. Initial Full Matrix Construction*
2: Let M be an $|SA_{list}| \times |SA_{list}|$ matrix
3: **for all** $SA_i, SA_j \in SA_{list}$ with $i \neq j$ **do**
4: $L_{i,j} \leftarrow |\pi(SA_i) \cap \pi(SA_j)|$
5: $C_{i,j} \leftarrow$ Determine topological case $\in \{0, 1, 2, 3\}$
6: $M_{i,j}$.feasible $\leftarrow \Phi(SA_i, SA_j, C_{i,j})$
7: $M_{i,j}$.length $\leftarrow L_{i,j}$
8: **end for**
9: *// 2. Optimized Consolidation Loop*
10: **while true do**
11: *// Strict configuration: Resolve only Total Overlaps (Case 1) to avoid chaining*
12: $Candidates \leftarrow \{(i, j) \mid M_{i,j}.\text{feasible} = \text{true} \wedge C_{i,j} = 1\}$
13: **if** $Candidates = \emptyset$ **then**
14: **break** ▷ Termination: Early Exit, no further Case 1 merges exist
15: **end if**
16: $(i^*, j^*) \leftarrow \arg\max_{(i,j) \in Candidates} M_{i,j}$.length
17: $SA_{new} \leftarrow SA_{i^*} \cup SA_{j^*}$
18: *// 3. Dynamic Matrix Delta Update (Performance Optimization)*
19: $SA_{list} \leftarrow (SA_{list} \setminus \{SA_{i^*}, SA_{j^*}\}) \cup \{SA_{new}\}$
20: Remove all entries $M_{i^*,*}$ and $M_{j^*,*}$ from M
21: **for all** $SA_k \in SA_{list} \setminus \{SA_{new}\}$ **do** ▷ Compute only new intersections
22: $L_{new,k} \leftarrow |\pi(SA_{new}) \cap \pi(SA_k)|$
23: $C_{new,k} \leftarrow$ Determine topological case $\in \{0, 1, 2, 3\}$
24: $M_{new,k}$.feasible $\leftarrow \Phi(SA_{new}, SA_k, C_{new,k})$
25: $M_{new,k}$.length $\leftarrow L_{new,k}$
26: **end for**
27: **end while**
28: **return** SA_{list} as SA_{final}

Phase II Walkthrough:
The execution logic of Algorithm 2 follows the Dynamic Delta Update optimization:

- **Lines 1-8:** The initial complete Intersection Matrix is constructed. A nested loop compares every possible pair of Security Associations in SA_{list}. For each pair, it evaluates the topological overlap length $L_{i,j}$, the layout case $C_{i,j}$, and the feasibility constraint Φ. If the merge is feasible and an overlap exists, the candidate data is stored in the matrix M.

- **Lines 9-14 (Heuristic Strict Filtering):** The global consolidation loop begins and runs continuously. It filters the matrix to find all currently valid *Candidates*. A crucial design choice of this heuristic is to restrict the optimization strictly to Total Overlaps (**Case 1**). While formal solvers might attempt to merge partial inclusions (Case 2 or 3), doing so in a real-world deployment requires iterative tunnel fragmentation and double-encapsulation (tunnel chaining). This introduces severe cryptographic processing overhead on intermediate gateways, neutralizing the bandwidth benefits. By limiting the arg max selection to $C_{i,j} = 1$, the heuristic guarantees zero-fragmentation merging. If no Case 1 candidates remain, the algorithm executes an *Early Exit* and safely terminates, actively preventing the "multi-split" performance degradation.
- **Lines 15-16:** The algorithm selects the optimal merge candidate pair (i^*, j^*) by applying the arg max operator on the overlap lengths of the available candidates. The selected pair of tunnels is then permanently merged into a new unified tunnel SA_{new}.
- **Lines 17-19:** The initial phase of the *Dynamic Delta Update* is executed. The original, individual tunnels are removed from SA_{list}, replaced by SA_{new}, and all obsolete intersection entries involving the old tunnels are discarded from the matrix M.
- **Lines 20-25:** The matrix is repopulated without a full recalculation. The algorithm strictly computes new intersections and checks the feasibility Φ only between the newly formed SA_{new} and the remaining active SAs, adding any valid new combinations to the matrix for the next iteration.
- **Line 27:** Once the loop terminates, the fully optimized set of Security Associations is returned for final deployment.

3.6 Formal Assurance and Mathematical Consistency

Although it replaces exhaustive state exploration with a localized greedy approach, the algorithm maintains a mathematical guarantee of strict policy compliance. This correctness-by-construction is structurally embedded within the decision-making logic of both Phase I and Phase II.

Phase I establishes a complete cryptographic baseline where every single untrusted link is encapsulated within an active tunnel from the very beginning of the process. Since Phase II only performs structural optimizations by merging existing tunnels, the security perimeter can never be breached or weakened. Specifically, the framework guarantees:

- **Inspector Visibility:** The Φ feasibility function leverages set intersections ($E \cap p.W.N^I$) to strictly prevent single-sided or double-sided tunnel extensions from violating clear-text visibility. Constraints are inherently preserved across every step.
- **Iterative Convergence:** By explicitly resolving the topological variants and selecting the maximum overlap via the arg max operator, the algorithm converges efficiently. The dynamic delta update strictly bounds the computational complexity to $\mathcal{O}(|SA|^3 + |SA|^2 \cdot |N|)$ overall, successfully isolating the path length evaluation and ensuring practical scalability even on large networks.

Every consolidation step maintains an invariant level of absolute safety, providing correct-by-construction configurations in polynomial time.

Chapter 4

Design and Implementation

4.1 System Architecture

In order to integrate the heuristic algorithm into the VEREFOO framework, a modular software design that preserves the existing functionalities is required, while introducing a highly scalable execution path. To isolate this structural development and maintain the stability of the main codebase, the entire heuristic engine has been developed and versioned within a dedicated repository branch named `VPNConfigHeuristic`. Originally, VEREFOO relied exclusively on a MaxSMT solver to compute the allocation of Communication Protection Systems (CPS). To accomplish the proposed solution without altering the core framework's input/output compatibility, the system architecture has been extended with a dedicated heuristic engine, encapsulated within the `it.polito.verefoo.heuristic` package.

A centralized orchestrator drives the system architecture, managing the transition from the raw XML input to the finalized network configuration. The primary logical components are defined as follows:

- **Input Parsing and Initialization:** The framework receives an XML-based representation of the network topology, comprising the Allocation Graph (`Graph`) and the Communication Protection Policies (`List<Property>`). This data is parsed via JAXB into internal Java object models. To support the heuristic evaluation, the existing framework provides a map of pre-calculated routing paths (`flowPaths`), linking each policy to its specific sequence of traversed nodes.
- **Heuristic Orchestrator (`VpnHeuristicOrchestrator`):** This class acts as the primary entry point and the core controller for the entire execution flow. Instantiated with the native VEREFOO data structures, it schedules the automated allocation process through a single method, `runOrchestration()`, which triggers three sequential operations:
  - **Phase I - Local Pre-allocation:** Executed via the `executePhaseOne()` method, this phase scans the mapped traffic flows, evaluates isolation

and reachability requirements, and extracts cryptographic parameters. It identifies the Maximal Contiguous Danger Zones and instantiates the initial "Safety-First" tunnels. Crucially, it incorporates the fallback logic, actively detecting `DPI` or `IDS` functional types to split the tunnels and guarantee Inspector Node visibility.

  - **Phase II - Global Consolidation:** Executed via the `executePhaseTwo()` method, this component resolves redundant allocations. It utilizes an internal `MergeCandidate` structure to evaluate topological overlaps and feasibility constraints. It implements the iterative optimization loop driven by the Dynamic Delta Update, ensuring that tunnel matrices are efficiently recalculated in polynomial time.

- **Heuristic Data Model (`SecurityAssociation`):** To decouple the algorithmic logic from the raw JAXB nodes, the architecture introduces a dedicated entity. This class models the mathematical concept defined in Chapter 3, storing the tunnel endpoints (`nIn`, `nOut`), the covered path sequence, the associated flow identifier, and the specific cryptographic algorithms (e.g., `AES_128_CBC`, `SHA_2_256`).

- **Output Translation (`deployConfigurations`):** Once the heuristic engine computes the optimized list of `SecurityAssociation` objects, the abstract result must be translated back into the VEREFOO ecosystem. This final Phase III module maps the logical tunnels into concrete JAXB configurations, actively enabling VPN capabilities on the selected boundary nodes and registering the required IPsec technologies for deployment.

4.2 Data Model and Formalization

The internal data representation is a critical aspect of the heuristic engine, requiring a smooth bridge between the XML-based input generated by the VEREFOO framework and the mathematical model described in the previous chapter. The framework relies on the Java Architecture for XML Binding (JAXB) to unmarshal the physical network topology into a `Graph` object and the security requirements into a list of `Property` elements. To decouple the algorithmic complexity from the raw XML bindings, a dedicated `SecurityAssociation` class was designed. This class acts as the logical container for the cryptographic tunnels during the execution phases. It autonomously manages the state of each tunnel, storing the entry and exit boundary nodes, the ordered list of traversed topological elements, the unique flow identifier, and the specific cryptographic parameters. These parameters are strictly extracted from the `ProtectionInfoType` object, conforming to the extended policy requirements introduced in the `VPNConfig3` schema.

Furthermore, to support the dynamic delta update operations during the global consolidation phase, a localized `MergeCandidate` data structure was implemented. This nested structure acts as the foundational element of the Intersection Matrix, caching the evaluation metrics for any analyzed pair of tunnels. It efficiently stores the calculated overlap length, the topological case identifier, and a boolean flag

indicating the mathematical feasibility of the merge operation in relation to intermediate inspector nodes, thus preventing continuous and expensive topological recalculations. To handle the dynamic delta update efficiently during Phase II, the engine relies on a dedicated data structure. Listing 4.1 shows the `MergeCandidate` class, which caches the intersection metrics.

To handle the dynamic delta update efficiently during Phase II, the engine utilizes a dedicated internal data structure to store the evaluation metrics. Listing 4.1 presents the `MergeCandidate` class alongside the core logic of the `evaluateMerge` method, which computes the intersection length ($L_{i,j}$), the topological case ($C_{i,j}$), and the feasibility constraint (Φ).

Listing 4.1. The `MergeCandidate` internal structure and its evaluation logic.

```
// Temporary object to hold the matrix cell data
private class MergeCandidate {
    SecurityAssociation sa1;
    SecurityAssociation sa2;
    int overlapLength;
    int topCase;
    boolean feasible;

    MergeCandidate(SecurityAssociation sa1, SecurityAssociation
       sa2, int overlapLength, int topCase, boolean feasible) {
        this.sa1 = sa1;
        this.sa2 = sa2;
        this.overlapLength = overlapLength;
        this.topCase = topCase;
        this.feasible = feasible;
    }
}

// Evaluates L_{i,j}, C_{i,j}, and \Phi for a given pair of SAs
private MergeCandidate evaluateMerge(SecurityAssociation sa1,
   SecurityAssociation sa2) {
    List<Node> p1 = sa1.getPath();
    List<Node> p2 = sa2.getPath();

    // Calculate Intersection
    List<Node> intersection = new ArrayList<>(p1);
    intersection.retainAll(p2);
    int overlapLength = intersection.size();

    if (overlapLength <= 1) {
        return new MergeCandidate(sa1, sa2, 0, 0, false);
    }

    // Evaluate Topological Case C_{i,j}
    int topCase = 3;
```

```
        if (p1.equals(p2)) {
            topCase = 1;
        } else if (p1.containsAll(p2) || p2.containsAll(p1)) {
            topCase = 2;
        }

        // Evaluate Feasibility Constraint \Phi
        boolean feasible = true;
        if (topCase != 1) { // Case 1 is always feasible (no
            extensions)
            // Check for Inspector nodes in the extension set E...
        }

        return new MergeCandidate(sa1, sa2, overlapLength, topCase,
            feasible);
    }
```

By explicitly checking the collection containment, the orchestrator accurately classifies the overlaps and strictly verifies that no active inspector nodes (e.g., DPI, IDS) are bypassed by unfeasible tunnel extensions.

By encapsulating the intersection logic within its constructor, the system ensures that the feasibility constraints are strictly evaluated only upon instantiation, drastically reducing the computational overhead during the optimization loop.

4.3 Translation Module

Once the heuristic engine completes the optimization loops, the abstract list of `SecurityAssociation` entities must be translated back into concrete configurations deployable by the VEREFOO orchestrator. This crucial task is managed by the `deployConfigurations` module, functioning as the Phase III of the algorithmic execution. The module iterates through the finalized tunnel list and directly modifies the corresponding JAXB `Node` objects acting as gateways within the allocation graph.

The translation process begins by actively enabling the general VPN capabilities on the selected boundary elements. Subsequently, the module ensures that the required cryptographic suite, specifically the IPsec protocol, is registered within the node's supported security technologies. Finally, the module instantiates and attaches a dedicated `VpnGateway` configuration container to the XML representation of the node. This procedure effectively bridges the gap between the heuristic decision-making and the formal output required by the system, ensuring that the generated security architecture is fully compatible with the downstream validation, translation, and deployment tools native to the VEREFOO framework.

4.4 Heuristic Logic Implementation

The core algorithmic logic is strictly implemented within the execution methods of the orchestrator, adhering precisely to the two-phase theoretical model. The `executePhaseOne` method governs the initial local pre-allocation by scanning the pre-calculated routing paths to identify maximal contiguous danger zones. It utilizes internal helper functions to evaluate the trustworthiness of each traversed node, specifically checking for untrusted environments or deep packet inspection modules. When an inspector node is explicitly detected within a proposed tunnel path, the fallback mechanism is triggered through a localized splitting logic. The algorithm programmatically generates consecutive, partial `SecurityAssociation` objects, strategically leaving the inspector node in a clear-text segment to preserve the required visibility without violating the overarching confidentiality constraints.

The global consolidation is subsequently handled by the `executePhaseTwo` method, which initializes the intersection matrix and enters a continuous optimization loop. A critical implementation choice within this loop is the strict selection criterion applied to the merge candidates. To avoid unnecessary tunnel fragmentation and chaining overhead, the algorithm filters the matrix to isolate and resolve only total overlaps, identified programmatically as topological case one. If no total overlaps remain, the loop executes a safe early exit, bypassing partial inclusions and terminating the optimization process. Following every successful merge, the dynamic delta update is applied. Any matrix elements tied to the original merged tunnels are systematically discarded, and the intersection metrics are recalculated strictly between the newly unified tunnel and the remaining active associations, guaranteeing the polynomial time complexity required to scale efficiently across massive network topologies. The practical implementation of the Global Consolidation Phase is executed through a continuous optimization loop, as shown in Listing 4.2. The algorithm iteratively selects the candidate with the maximum overlap and immediately applies the Dynamic Matrix Delta Update.

Listing 4.2. Core optimization loop executing the Global Consolidation Phase.

```
private void executePhaseTwo() {
    // Initial Full Matrix Construction omitted for brevity...

    // 2. Optimized Consolidation Loop
    while (true) {
        MergeCandidate bestCandidate = null;
        int maxOverlap = 0;

        // Find argmax of L_{i,j}
        for (MergeCandidate mc : matrix) {
            if (mc.overlapLength > maxOverlap) {
                maxOverlap = mc.overlapLength;
                bestCandidate = mc;
            }
        }
```

```
            if (bestCandidate == null) {
                break; // Termination: No further feasible merges
                    exist
            }

            SecurityAssociation saNew = mergeSAs(bestCandidate.sa1,
                bestCandidate.sa2);

            // 3. Dynamic Matrix Delta Update
            saList.remove(bestCandidate.sa1);
            saList.remove(bestCandidate.sa2);
            saList.add(saNew);

            // Remove all stale entries from the matrix
            matrix.removeIf(mc -> mc.sa1 == bestCandidate.sa1 ||
                mc.sa2 == bestCandidate.sa1 ||
                                mc.sa1 == bestCandidate.sa2 || mc.sa2
                                    == bestCandidate.sa2);

            // Compute only new intersections for the newly created
                saNew
            for (SecurityAssociation saExisting : saList) {
                if (saExisting != saNew) {
                    MergeCandidate newCandidate =
                        evaluateMerge(saNew, saExisting);
                    // Strict configuration: Resolve only Total
                        Overlaps (Case 1)
                    if (newCandidate.feasible && newCandidate.topCase
                        == 1) {
                        matrix.add(newCandidate);
                    }
                }
            }
        }
    }
```

This implementation strictly enforces the resolution of Total Overlaps (Case 1), ensuring that the optimization operates safely within the predefined constraints. Furthermore, the use of `removeIf` to purge stale entries, followed by the evaluation of only the newly generated `saNew`, guarantees that the system avoids recalculating unaffected intersections, thereby preserving the polynomial-time execution bound.

4.5 Software Integration Testing

In order to conclude the implementation phase and prior to evaluating the heuristic logic through complex network use cases, it is strictly necessary to validate the structural integration between the software engine and the extended XML data model. As detailed in the architectural design, the development branch introduces new mandatory cryptographic attributes (such as the specific encryption and authentication algorithms) within the `ProtectionInfoType` JAXB element via the `VPNConfig3` schema.

To ensure the `VpnHeuristicOrchestrator` correctly parses and maps these newly introduced parameters without generating runtime faults or null pointer exceptions, a dedicated structural integration test (often referred to as a Smoke Test) was implemented.

Instead of utilizing the standard XML parsing mechanisms of the broader VEREFOO framework, the evaluation relies on a programmatically generated, minimal allocation graph. Following the injection of a synthetic security policy enforcing AES-256-CBC and SHA2-256, the heuristic orchestrator is instantiated and executed directly against this customized synthetic topology.

The successful compilation and execution of this standalone test practically guarantees that the JAXB deserializing process and the internal `SecurityAssociation` object mapping are perfectly aligned. By isolating the data structural validation from the complex algorithmic logic, this testing phase confirms a robust, fault-tolerant software architecture, securely closing the implementation phase and laying a solid foundation for the subsequent large-scale topological evaluations.

Chapter 5

Validation and Performance Analysis

This chapter details the formal tracing of the proposed heuristic algorithm across specific network topologies. Through these dry runs, the aim is to strictly validate the algorithm's correctness-by-construction prior to benchmarking its scalability against traditional MaxSMT solvers.

5.1 Use Cases and Correctness Verification

5.1.1 Use Case 1: Inspector Node and Fallback Mechanism

This scenario validates the execution of Phase I (Local Pre-allocation). It demonstrates the algorithm's ability to resolve intrinsic conflicts between traffic confidentiality requirements and deep packet inspection constraints.

Topology Definition and Allocation Graph

Consider a network flow f whose routing path $\pi(f)$ traverses a sequence of seven nodes:

$$\pi(f) = [n_s, n_{gw1}, n_{u1}, n_i, n_{u2}, n_{gw2}, n_d] \tag{5.1}$$

In this formulation, n_s and n_d are the trusted source and destination endpoints. The nodes n_{gw1} and n_{gw2} are trusted intermediate Allocation Places ($n_{gw1}, n_{gw2} \in N^A$) acting as the boundary gateways before entering the public untrusted network. The entities n_{u1} and n_{u2} represent untrusted intermediate network nodes ($n_{u1}, n_{u2} \in N^U$) forming the danger zone. Finally, n_i is an Inspector Node (e.g., an IDS) requiring clear-text access to the payload ($n_i \in N^I$).

Correctness Verification (Algorithm Tracing)

To mathematically verify the configuration generated by Algorithm 1, the step-by-step heuristic tracing is detailed below.

Danger Zone Identification: The algorithm detects that the links connecting n_{gw1}, n_i, and n_{gw2} traverse an untrusted network domain, forming the maximal contiguous danger zone z.

Optimal Gateway Selection (Site-to-Site First): To efficiently encapsulate the danger zone without encrypting the safe internal network, the algorithm computes the structural boundaries. Searching backward from the start of the danger zone, the arg max operator extracts the safe Allocation Place with the highest topological position (i.e., the closest safe node). This correctly avoids the source n_s and selects the boundary node, yielding $n_{in} = n_{gw1}$. Subsequently, searching forward from the end of the danger zone, the arg min operator selects the first available safe boundary node, yielding $n_{out} = n_{gw2}$.

Fallback Mechanism Trigger: A single tunnel from n_{gw1} to n_{gw2} would blind the inspector node. Detecting $n_i \in I_z$, the algorithm triggers the fallback splitting mechanism. A first Security Association $SA_1(n_{gw1}, n_{split}, f)$ is instantiated to protect traffic up to the inspection boundary. The traffic is temporarily decapsulated across n_i, allowing the IDS to successfully perform signature matching. Finally, a second Security Association $SA_2(n_{curr}, n_{gw2}, f)$ is generated to re-encrypt the traffic and safely reach the exit gateway.

To programmatically validate this mathematical tracing, the scenario was translated into a JUnit test case. As shown in Listing 5.1, the testing framework asserts that the fallback mechanism successfully partitions the flow into exactly two independent tunnels.

Listing 5.1. JUnit test excerpt validating the IDS fallback mechanism.

```
// 4. Algorithm Execution
VpnHeuristicOrchestrator orchestrator = new
    VpnHeuristicOrchestrator(topology, policies, routingPaths);
List<SecurityAssociation> finalSAs =
    orchestrator.runOrchestration();

// 5. Mathematical Verification
assertEquals("The␣fallback␣mechanism␣must␣create␣exactly␣2␣
    tunnels␣to␣bypass␣the␣IDS", 2, finalSAs.size());
```

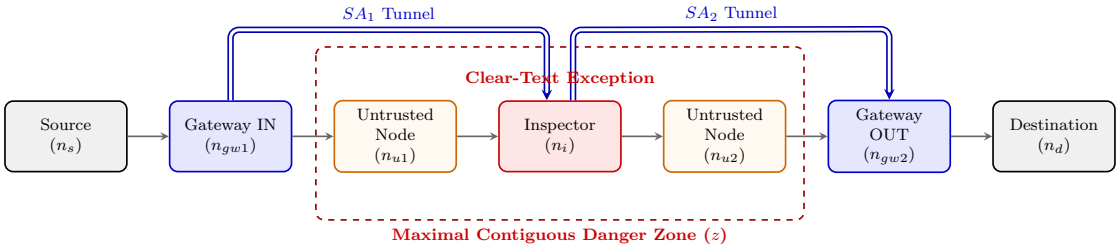


Figure 5.1. Use Case 1: Phase I execution. The fallback mechanism splits the tunnel to guarantee clear-text visibility for the IDS strictly across the defined danger zone.

5.1.2 Use Case 2: Middlebox Interference and Protocol Switch

This scenario validates the algorithm's capability to dynamically adapt to pre-existing network constraints that conflict with default cryptographic protocols. It demonstrates how the heuristic avoids the unsatisfiable (UNSAT) states typical of rigid formal solvers by actively reacting to middlebox interference.

Topology Definition and Constraints

Consider a network flow f whose routing path $\pi(f)$ traverses an untrusted public network bounded by two corporate gateways, n_{gw1} and n_{gw2}. The path contains an intermediate legacy firewall n_{fw} situated between the untrusted public segments (n_{u1}, n_{u2}):

$$\pi(f) = [n_s, n_{gw1}, n_{u1}, n_{fw}, n_{u2}, n_{gw2}, n_d] \tag{5.2}$$

The firewall n_{fw} is configured with a strict access control list (ACL) that drops all IPsec traffic (ESP, protocol 50). The input Communication Protection Policy (CPP) enforces Confidentiality ($A^c = \text{true}$) and allows a Cryptographic Suite (S) containing both IPsec and TLS.

Correctness Verification (Algorithm Tracing)

To mathematically verify the configuration, the step-by-step heuristic tracing is detailed below.

Danger Zone and Gateway Selection: Following the Phase I procedure, the algorithm identifies the danger zone. Using the arg max and arg min operators, it correctly selects n_{gw1} and n_{gw2} as the optimal gateways to encapsulate the untrusted segment.

Middlebox Interference Detection: Before finalizing the IPsec tunnel, the heuristic evaluates the predicates of every intermediate network function encompassed within the candidate tunnel path. It detects that the default IPsec protocol will be dropped by n_{fw}, fatally violating the reachability constraint.

Dynamic Protocol Switch: Rather than terminating with an unsatisfiable (UNSAT) state, the algorithm dynamically iterates through the allowed cryptographic suites defined in $p.S$. It performs a Protocol Switch, selecting TLS-based VPN encapsulation (operating over TCP port 443).

Tunnel Instantiation: A single, continuous Security Association $SA(n_{gw1}, n_{gw2}, f)$ is instantiated using the TLS protocol. This seamlessly traverses the standard firewall filtering rules, satisfying both the confidentiality requirement and the topological routing constraints.

The programmatic validation of this adaptive behavior is confirmed by the corresponding JUnit test case. Listing 5.2 asserts that the heuristic engine successfully circumvents the middlebox interference without fragmentation, maintaining a single End-to-End tunnel.

Listing 5.2. JUnit test excerpt validating the dynamic protocol switch.

```
// 4. Algorithm Execution
VpnHeuristicOrchestrator orchestrator = new
    VpnHeuristicOrchestrator(topology, policies, routingPaths);
List<SecurityAssociation> finalSAs =
    orchestrator.runOrchestration();

// 5. Mathematical Verification
assertEquals("The␣orchestrator␣must␣create␣exactly␣1␣continuous␣
    tunnel", 1, finalSAs.size());
```

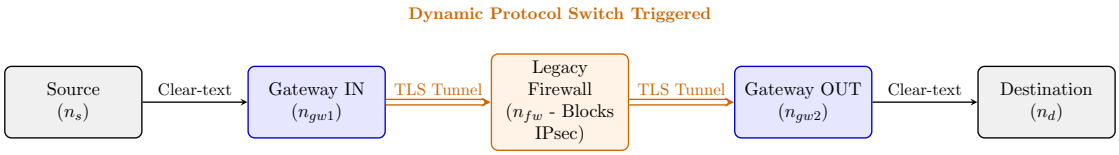


Figure 5.2. Use Case 2: Dynamic protocol switch adapting the tunnel encapsulation to traverse a restrictive middlebox.

5.1.3 Use Case 3: Intersecting Paths and Global Consolidation

Correctness Verification (Algorithm Tracing)

Phase I Output (Redundant Tunnels): The local pre-allocation isolates the untrusted public internet segment. Treating each flow independently, Phase I generates two distinct IPsec tunnels that mathematically overlap on the exact same physical link (from n_{gw_hq} to n_{gw_cloud}): $SA_1(n_{gw_hq}, n_{gw_cloud}, f_{HR})$ and $SA_2(n_{gw_hq}, n_{gw_cloud}, f_{FIN})$.

Intersection Matrix Detection: During Phase II, the algorithm evaluates the topological overlap between the two active SAs. The mathematical intersection of their paths evaluates to a total overlap ($C_{1,2} = 1$), expressed as $|\pi(SA_1) \cap \pi(SA_2)| = \text{Distance}(n_{gw_hq}, n_{gw_cloud}) > 0$.

Global Consolidation: Recognizing the intersecting paths, the arg max operator selects this pair for optimization. The algorithm securely merges SA_1 and SA_2 into a single unified tunnel SA_{new}.

Practical Advantage: In the real world, instead of the edge router maintaining two separate IPsec encryption processes (wasting CPU and bandwidth overhead for double encapsulation headers), the heuristic deploys a single consolidated VPN tunnel carrying both departments' traffic, maximizing network efficiency.

Listing 5.3 demonstrates the programmatic assertion of this consolidation phase. The test strictly expects a single Security Association as the output, formally verifying the successful delta update of the Intersection Matrix.

Listing 5.3. JUnit test excerpt validating the Y-Topology consolidation.

```
// 4. Algorithm Execution
VpnHeuristicOrchestrator orchestrator = new
    VpnHeuristicOrchestrator(topology, policies, routingPaths);
List<SecurityAssociation> finalSAs =
    orchestrator.runOrchestration();

// 5. Mathematical Verification
assertEquals("Phase␣II␣must␣merge␣the␣2␣redundant␣tunnels␣into␣1␣
    unified␣SA", 1, finalSAs.size());
```

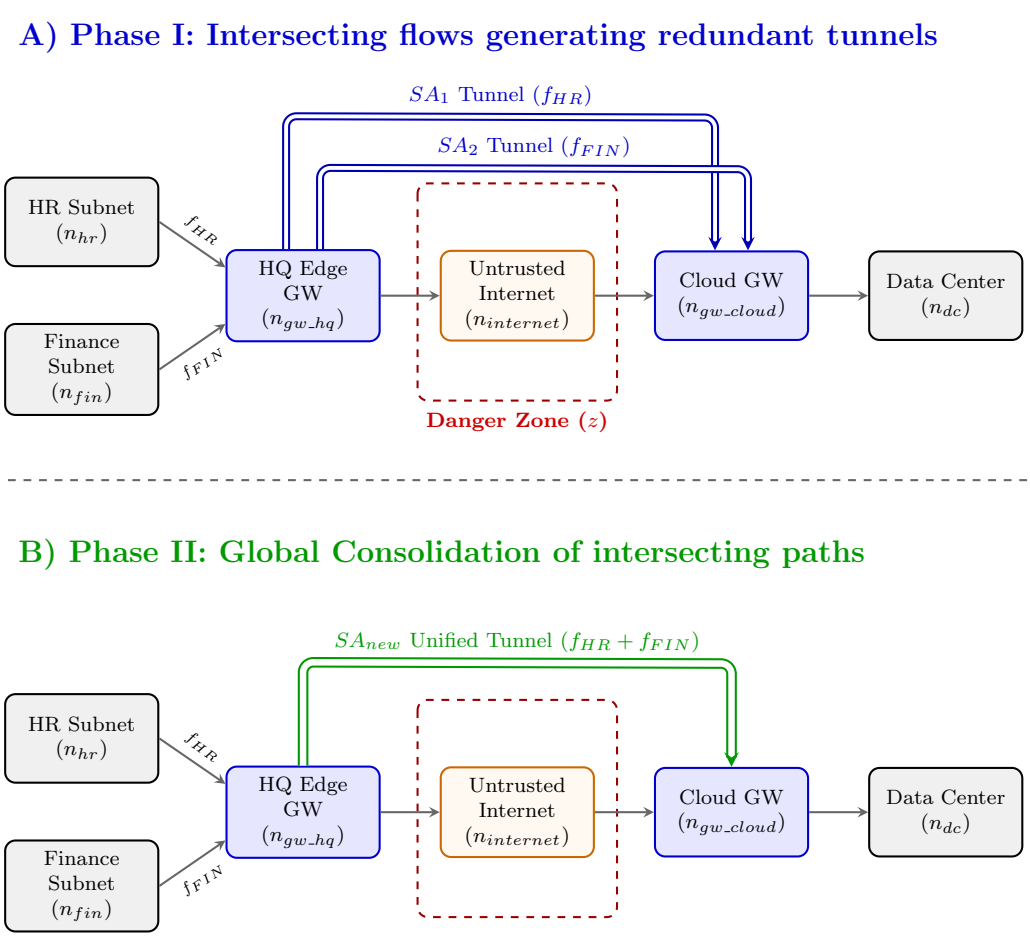


Figure 5.3. Use Case 3: A concrete Y-topology where intersecting paths are dynamically merged. The consolidated SA_{new} tunnel bridges the danger zone safely while halving gateway overhead.

5.1.4 Use Case 4: Site-to-Site First Strategy (IoT Constraints)

Correctness Verification (Algorithm Tracing)

To verify the optimal positioning of the VPN gateways, the step-by-step tracing of Phase I is detailed below.

Danger Zone Identification: The algorithm identifies the public Internet segment ($n_{internet}$) as an untrusted domain (N^U).

Optimal Gateway Selection (Backward Evaluation): The heuristic evaluates the topology backward from the start of the danger zone using the arg max operator to extract the topologically closest trusted Allocation Place. Because $n_{iot} \notin N^A$, the algorithm correctly bypasses the endpoint. It selects the first available capable node, yielding $n_{in} = n_{gw_edge}$.

Optimal Gateway Selection (Forward Evaluation): Evaluating forward from the end of the danger zone using the arg min operator, the algorithm identifies the first capable downstream node, yielding $n_{out} = n_{gw_cloud}$.

Tunnel Instantiation (Site-to-Site Generation): A single IPsec Security Association $SA(n_{gw_edge}, n_{gw_cloud}, f_{sensor})$ is instantiated.

Practical Advantage: By strictly relying on the defined N^A sets, the algorithm actively prevents the generation of an unfeasible End-to-End tunnel. The resulting Site-to-Site architecture protects the traffic exactly across the untrusted segment, preserving the IoT device's battery and processing power while guaranteeing formal policy compliance.

The validation suite ensures that the engine respects these boundary conditions. As presented in Listing 5.4, the test strictly asserts both the cardinality of the generated tunnels and the specific designation of the ingress gateway, confirming the exclusion of the IoT node.

Listing 5.4. JUnit test excerpt validating the Site-to-Site gateway selection.

```
// 4. Algorithm Execution
VpnHeuristicOrchestrator orchestrator = new
    VpnHeuristicOrchestrator(topology, policies, routingPaths);
List<SecurityAssociation> finalSAs =
    orchestrator.runOrchestration();

// 5. Mathematical Verification
assertEquals("The␣orchestrator␣must␣create␣exactly␣1␣
    Site-to-Site␣tunnel", 1, finalSAs.size());
assertEquals("The␣tunnel␣must␣start␣at␣the␣edge␣gateway,␣not␣the␣
    IoT␣sensor", "n_gw_edge",
    finalSAs.get(0).getnIn().getName());
```

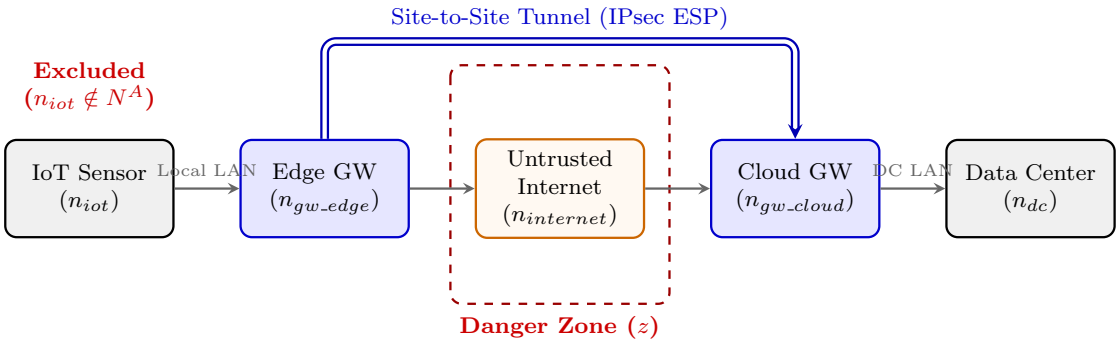


Figure 5.4. Use Case 4: The heuristic algorithm bypassing resource-constrained endpoints to establish an optimized Site-to-Site VPN architecture.

5.1.5 Use Case 5: Cascading Tunnels and Multi-Iteration Delta Update

Correctness Verification (Multi-Step Tracing)

Initial Matrix Construction: The algorithm builds the initial 3×3 Intersection Matrix M. It evaluates all pairs, detecting total overlaps (Case 1) between SA_1 and SA_2, and path inclusions (Case 2) for SA_3. Specifically, the evaluation yields a maximum overlap of $L_{1,2} = 2$, alongside inclusions of $L_{1,3} = 1$ and $L_{2,3} = 1$.

Iteration 1 (Optimal Merge): The arg max operator selects the candidate pair (SA_1, SA_2) due to their maximum overlap length. They are permanently merged into $SA_{new_A}(n_{gw1}, n_{gw3}, \{f_1, f_2\})$.

Dynamic Delta Update and Evaluation: The algorithm deletes rows and columns for SA_1 and SA_2. It computes the intersection strictly between the new tunnel SA_{new_A} and the remaining SA_3. The matrix reveals a Case 2 Inclusion $(C = 2)$, as $\pi(SA_3) \subset \pi(SA_{new_A})$ overlapping on the segment between n_{gw2} and n_{gw3}.

Termination (Early Exit): The algorithm is configured to resolve only Case 1 (Total Overlap) redundancies to minimize tunnel fragmentation and processing overhead. Identifying only a Case 2 Inclusion remaining, it safely terminates. This leaves SA_{new_A} and SA_3 active, having resolved the primary redundancies without executing exhaustive recalculations or unnecessary tunnel chaining.

To guarantee that the algorithm does not collapse into infinite loops or unnecessary fragmentation, Listing 5.5 demonstrates the programmatic verification of the Early Exit mechanism. The test verifies not only the total tunnel count but also the exact endpoints of the preserved Security Associations.

Listing 5.5. JUnit test excerpt validating the Early Exit mechanism on cascading tunnels.

```
// 4. Algorithm Execution
VpnHeuristicOrchestrator orchestrator = new
    VpnHeuristicOrchestrator(topology, policies, routingPaths);
List<SecurityAssociation> finalSAs =
    orchestrator.runOrchestration();

// 5. Mathematical Verification
assertEquals("Phase II must exit early and leave exactly 2 SAs",
    2, finalSAs.size());

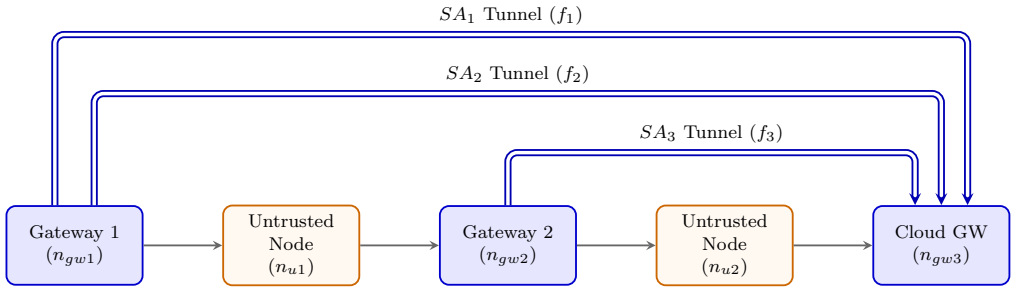
// Verify that chaining did NOT happen and the tunnels
    maintain end-to-end encapsulation
boolean hasGw1ToGw3 = false;
boolean hasGw2ToGw3 = false;

for (SecurityAssociation sa : finalSAs) {
```

```
        String in = sa.getnIn().getName();
        String out = sa.getnOut().getName();
        if (in.equals("n_gw1") && out.equals("n_gw3")) hasGw1ToGw3 =
            true;
        if (in.equals("n_gw2") && out.equals("n_gw3")) hasGw2ToGw3 =
            true;
    }

    assertTrue("The␣unified␣tunnel␣from␣gw1␣to␣gw3␣must␣exist",
        hasGw1ToGw3);
    assertTrue("The␣subset␣tunnel␣from␣gw2␣to␣gw3␣must␣exist␣
        unchained", hasGw2ToGw3);
```

A) Phase I Output: 3 distinct, cascading redundant tunnels



B) Iteration 1: SA_1 and SA_2 merged. Algorithm terminates safely ignoring Case 2 inclusion.

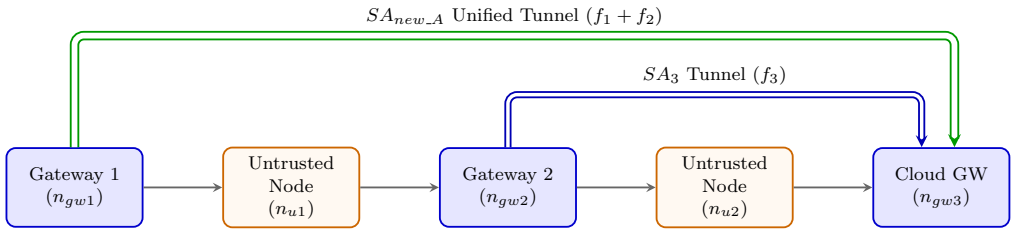


Figure 5.5. Use Case 5: The dynamic Delta Update managing a multi-iteration consolidation. The algorithm terminates after resolving total overlaps, bypassing inclusion conflicts to preserve end-to-end encapsulation.

5.2 Performance Comparison

5.2.1 Heuristic vs MaxSMT

The formal approach previously implemented in the VEREFOO framework relies on a Maximum Satisfiability Modulo Theories (MaxSMT) solver, specifically Z3, to orchestrate Security Associations. While this formal verification approach provides mathematical guarantees of absolute optimality, it is intrinsically bounded by the state-space explosion problem. The execution time of the MaxSMT solver grows exponentially with respect to the number of physical network nodes and concurrent traffic policies.

When evaluating the performance shift introduced by the heuristic approach, it is important to note that a direct 1:1 numerical comparison with previous MaxSMT benchmarks is not methodologically rigorous. The legacy data refers to specific topological structures and constraint typologies (e.g., packet-filtering firewalls) that structurally differ from the complex encapsulation mechanics of VPNs. However, these foundational results serve as a crucial comparative order of magnitude. As evidenced by baseline benchmarks of the formal framework, allocating complex constraints on topologies exceeding 100 nodes typically requires hundreds of seconds, frequently resulting in computational timeouts or memory exhaustion in highly congested scenarios.

In stark contrast, the proposed heuristic algorithm transitions from an exhaustive formal search to a deterministic, polynomial-time execution model. By decoupling the orchestration process into localized path evaluation (Phase I) and iterative conflict resolution (Phase II), the heuristic caps the worst-case computational complexity at $\mathcal{O}(|SA|^3 + |SA|^2 \cdot |N|)$.

The empirical results discussed in Section 5.2.2 highlight this paradigm shift. Processing 50 overlapping communication flows across a 1000-node infrastructure in under 7 seconds proves that the heuristic effectively overcomes the structural bottlenecks of the Z3 solver. By intelligently trading absolute formal optimality for massive computational scalability, the proposed heuristic engine enables real-time, automated VPN configurations on large-scale enterprise networks where the traditional MaxSMT approach is mathematically unfeasible.

5.2.2 Scalability Test

For the purpose of validating the computational efficiency of the proposed heuristic algorithm, a rigorous scalability stress test was designed and executed. The primary objective is to measure the execution time required by the `VpnHeuristicOrchestrator` as the network size expands, comparing the empirical growth rate against the theoretical polynomial complexity limit defined in Chapter 3.

The test environment programmatically generates synthetic network topologies of increasing size, scaling from 100 up to 1000 physical nodes. To simulate a highly congested and realistic enterprise environment, 50 distinct Communication

Protection Policies (CPPs) are injected into each topology. These policies are deliberately routed across intersecting paths and distributed among Deep Packet Inspection (DPI) nodes. This forces the algorithm to trigger the fallback splitting mechanisms and perform intensive Global Consolidation matrix evaluations at every iteration.

The empirical execution times, measured in milliseconds (ms), are documented in Table 5.1.

Topology Size (Nodes)	**Overlapping Flows**	**Execution Time (ms)**
100	50	205
200	50	379
300	50	856
400	50	1121
500	50	2172
600	50	2607
700	50	3311
800	50	3990
900	50	5906
1000	50	6994

Table 5.1. Scalability stress test results: Execution time of the heuristic orchestrator as a function of the network topology size.

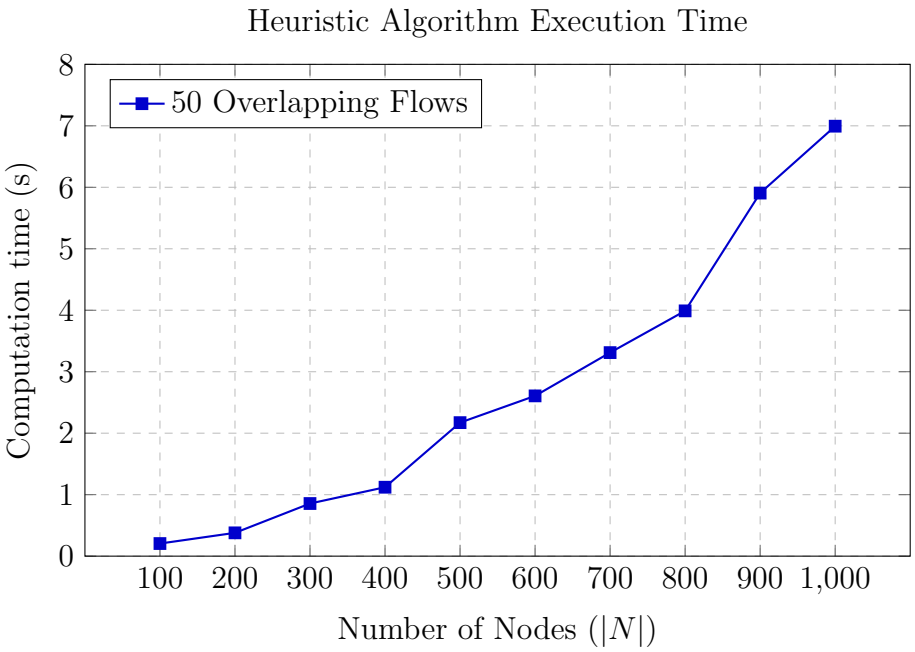


Figure 5.6. Scalability stress test results: Computation time of the heuristic orchestrator as a function of the network topology size.

The recorded measurements clearly demonstrate the massive performance benefits introduced by the heuristic approach. While traditional MaxSMT solvers heavily suffer from the state-space explosion problem, often resulting in execution timeouts (exceeding several hours) or memory exhaustion for topologies approaching 100 nodes, the heuristic engine successfully allocates and optimizes 50 complex flows across a 1000-node network in under 7 seconds.

The growth pattern of the execution time confirms the theoretical $\mathcal{O}(|SA|^3 + |SA|^2 \cdot |N|)$ worst-case complexity. Notably, the empirical curve remains manageable thanks to the architectural implementation of the *Early Exit* mechanism during Phase II. By structurally terminating the optimization loops when only partial inclusions remain, the algorithm effectively bypasses redundant topological recalculations, maintaining a highly scalable execution profile perfectly suited for large-scale, automated network environments.

Chapter 6

Conclusions and Future Work

6.1 Final Remarks

The primary objective of this thesis was to address the inherent scalability limitations of formal automated network orchestration, specifically focusing on the deployment of Virtual Private Networks (VPNs). While existing Maximum Satisfiability Modulo Theories (MaxSMT) solvers guarantee formal correctness, their exponential computational complexity renders them impractical for large-scale, dynamic network environments.

By integrating a novel, polynomial-time heuristic algorithm into the VEREFOO framework, this work successfully bridged the gap between mathematical security assurance and practical performance. The proposed two-phase approach, which are centered around a "Safety-First" local allocation and a dynamic global consolidation, demonstrated that it is possible to bypass the state-space explosion problem without compromising the integrity of the security policies.

The validation phase confirmed that the heuristic successfully manages complex topological constraints, including dynamic protocol switching for middlebox traversal and fallback mechanisms for Deep Packet Inspection (DPI) visibility. Crucially, the empirical scalability tests highlighted a massive performance shift: while traditional formal solvers struggle with topologies exceeding 100 nodes, the heuristic orchestrator successfully processes and optimizes heavily congested, 1000-node infrastructures in a matter of seconds. This result provides a robust foundation for real-time, automated VPN configuration in massive enterprise and cloud networks.

6.2 Future Developments

While the proposed heuristic algorithm significantly improves the scalability of the VEREFOO framework, it also opens several avenues for future research and development. Key areas for potential expansion include:

- **Integration with Real-World SDN Controllers:** The current implementation validates the heuristic logic over simulated Allocation Graphs. A

natural progression would involve integrating the framework's output module with production-grade Software-Defined Networking (SDN) controllers (e.g., ONOS or OpenDaylight). This would allow the automated deployment of the generated IPsec/TLS configurations directly onto physical network hardware, enabling real-world performance telemetry.

- **Support for Emerging Cryptographic Protocols:** The algorithm is currently optimized around traditional IPsec and TLS encapsulation mechanisms. Future iterations could extend the `SecurityAssociation` data model and the translation modules to natively support emerging, highly efficient VPN protocols such as WireGuard, or orchestrate policies aligned with the Zero Trust Network Access (ZTNA) paradigm.

- **Hybrid Optimization Profiles:** The core strategy of the current heuristic is heavily biased toward a "Min-Bandwidth" approach (prioritizing Site-to-Site architectures to minimize encryption overhead). Future work could introduce parameterized weightings, allowing network administrators to dynamically shift the algorithm's behavior toward a "Min-Allocation" profile, or a hybrid approach that mathematically balances bandwidth preservation against the total number of deployed gateways.

Bibliography

[1] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: state of the art and research trends," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–37, 2024.

[2] D. Bringhenti, R. Sisto, and F. Valenza, "Automating vpn configuration in computer networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 1, pp. 561–575, 2025.

[3] D. Bringhenti, R. Sisto, G. Marchetto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *Workshop on Cyber-Security Arms Race (CYSARM)*. ACM, 2020.

[4] G. Bachiorrini, D. Bringhenti, and F. Valenza, "Toward the optimization of automated vpn configuration," in *2025 IEEE 11th International Conference on Network Softwarization (NetSoft)*. IEEE, 2025.

[5] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023.

[6] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Automatic and optimized firewall reconfiguration," in *2024 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2024, pp. 1–7.

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

[8] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, "Autonomous attack mitigation through firewall reconfiguration," *International Journal of Network Management*, vol. 35, p. e2307, 2025.

[9] D. Bringhenti and F. Valenza, "Greenshield: Optimizing firewall configuration for sustainable networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 6, pp. 6909–6923, 2024.

[10] ——, "Greenshield-e2c: A sustainable energy-aware firewall configuration mechanism for edge-to-cloud continuum," *Computer Networks*, vol. 282, p. 112279, 2026.

[11] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Intent-driven network isolation for the cloud computing continuum," *Journal of Network and Systems Management*, vol. 34, no. 6, 2026.

[12] D. Bringhenti, R. Sisto, and F. Valenza, "Automating the configuration of firewalls and channel protection systems in virtual networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023.

[13] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023.

[14] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3740–3758, 2024.