Politecnico di Torino

Master's Degree in Cybersecurity
LM32 - Computer Engineering

Academic Year 2024-2025
October 2025

Towards Energy-Aware Network Security Automation in Edge-to-Cloud Environments

Supervisors

Prof. Daniele BRINGHENTI
Prof. Fulvio VALENZA
Prof. Riccardo SISTO

Candidate

Federico CAGNAZZO

Summary

The rapid evolution of digital infrastructures, accelerated by the proliferation of the Internet of Things (IoT) and virtualization, has increased the complexity of securing distributed environments while raising concerns about energy efficiency. Traditional manual configuration of security controls is no longer feasible in dynamic networks, where even small errors can compromise protection. Automation addresses this challenge by ensuring correctness and adaptability, but current approaches often overlook the energy consumption of security operations.

This thesis focuses on the integration of network security automation with energy-aware strategies in Edge-to-Cloud (E2C) environments, where resources are dynamically distributed between endpoints, edge devices, and cloud infrastructures. The work builds upon GreenShield, an energy-aware framework for automated firewall configuration, designed to enforce security policies with strict correctness while reducing energy consumption. The thesis extends GreenShield to support the hierarchical nature of E2C systems, introducing novel constraints to guide the placement and activation of security functions with explicit consideration for power efficiency. A key contribution is the analysis of the power consumption of virtualization technologies, i.e., virtual machines (VMs) and containers, that are at the basis of E2C platforms. Finally, new constraints are proposed to integrate GreenShield in the E2C context.

In the second part, to evaluate scalability and practical feasibility, the thesis develops a dedicated test generator integrated into Verefoo, the security automation framework that serves as the foundation for GreenShield. The generator is adapted to the CESNET3 research and education network, based on real encrypted TLS traffic traces, taken by the CESNET-TLS-Year22 dataset. It systematically produces realistic test cases by statistically modeling traffic distributions and generating reachability and isolation policies.

In the final part, the thesis presents a comprehensive validation of the tests generated for the CESNET3 network. The results are evaluated under different configurations, including various solver versions, both with a MaxSMT and a heuristic approach, and varying policies complexity. The findings demonstrate the effectiveness of the proposed extensions in maintaining security guarantees, while also providing a means to study the scalability of the test generator and of Verefoo, on a real network such as CESNET3. As a future work, the thesis suggests further extension of the test generator to GreenShield, enabling the evaluation of energy-aware security automation in contexts that closely resemble real-world scenarios like CESNET3 network.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The rapid evolution of digital infrastructures has fundamentally transformed the way modern societies generate, exchange, and protect information. The proliferation of the Internet of Things (IoT), together with the widespread adoption of virtualization technologies and cloud-native paradigms, has dramatically increased both the scale and complexity of networked systems. These transformations, while enabling extraordinary levels of connectivity and innovation, have also intensified the challenges of securing distributed environments against increasingly sophisticated cyber threats. At the same time, the growing energy demands of large-scale networks raise critical challenges about sustainability, making energy efficiency a critical design objective for future digital infrastructures.

In this context, network security automation is now an integral part of many strong infrastructures. Traditional manual security configuration is becoming increasingly infeasible in large and dynamic networks with the potential that even small errors can expose the system to critical vulnerabilities. Automating security policy deployment and enforcement not only reduces the risk of misconfigurations but also enhances agility, enabling organizations to respond rapidly to evolving threats. However, existing approaches to automation typically prioritize correctness and performance, without considering their energy power consumption.

At the same time, the Edge-to-Cloud Computing (E2C) paradigm has gained popularity as a response to the limitations of centralized cloud architectures. Providing computation and storage closer to data sources, E2C systems address the low-latency, high-bandwidth, and reliability requirements of emerging IoT and real-time applications. However, the distributed and heterogeneous nature of E2C infrastructures introduces new challenges for both security and energy management that demand dynamic coordination of capabilities across many layers while allowing compliances with policy and sustainability.

This thesis brings together two important areas: network security automation and energy efficiency in E2C environments. The main objective is to make security automation not only correct and reliable, but also more sustainable. To achieve this, the work extends GreenShield, a framework originally designed to automatically configure firewalls in an optimized way. GreenShield already combines formal policy enforcement with optimization techniques, but in this thesis it is further improved so that it can also take into account the unique characteristics of Edge-to-Cloud

systems. In particular, the new version of GreenShield introduces additional rules and constraints that make it capable of deciding where and when to activate security functions, while also reducing the overall power consumption of the network.

To make these improvements meaningful, the thesis also analyzes the energy usage of virtualization technologies, such as virtual machines (VMs) and containers, which form the foundation of most modern edge and cloud platforms. These technologies behave differently in terms of resource allocation and power consumption, so understanding their energy profiles is crucial. By analyzing how they perform under different conditions, it becomes possible to identify opportunities for saving energy without compromising the security of the system.

In addition to theoretical analysis, the thesis also develops a test generator that is currently programmed for the Verefoo framework. The generator has been specifically developed to apply Verefoo to the CESNET3 network, the national research and education network of the Czech Republic. This allows the evaluation of optimization strategies in a real and complex environment using real traffic traces. It is able to simulate realistic network traffic scenarios and evaluate the scalability, accuracy, and execution time of the optimization engine under different conditions. Although its first implementation is dedicated to Verefoo, the long-term goal is to extend the generator to GreenShield, so that the proposed energy-aware techniques can be tested directly within the enhanced framework as well.

Moreover, this thesis includes a validation phase in which the execution times of the generator are compared while varying different input parameters. This makes it possible to analyze how the system behaves under different configurations, providing insights into its feasibility for real-world use and its scalability when applied to larger and more complex scenarios. These tests represent an essential step in understanding the practical limits of the approach and verifying that the optimization engine can operate efficiently even in demanding environments.

In conclusion, this thesis addresses the integration of network security automation with Edge-to-Cloud infrastructures by extending the GreenShield framework to operate effectively in such environments. It also presents the development of a test generator for the CESNET3 network, implemented within the Verefoo framework, and validates its performance by analyzing scalability, execution time, and the realism of generated traffic scenarios. Through this validation, the work demonstrates that the proposed tools can reliably support the evaluation of automated security strategies in real-world and large networks.

1.1 Thesis description

This thesis is organized to provide a logical structure starting from the background topics and gradually advances toward the actual analysis. **Chapter 2** focuses on network security automation, discussing its importance in modern infrastructures and the challenges it faces. It also reviews existing solutions and highlights their limitations, particularly in terms of energy efficiency, presenting GreenShield as a possible solution to these challenges. **Chapter 3** introduces the Edge-to-Cloud

Computing paradigm, explaining its architecture, benefits, and the specific challenges it poses for security and energy management. **Chapter 4** presents the objectives of the thesis, detailing the specific research questions and goals that guide the work. **Chapter 5** presents a detailed analysis of the virtualization technologies power consumptions, and describes the proposed extensions to the GreenShield framework, including the new hard and soft constraints designed to optimize energy consumption in E2C environments. **Chapter 6** presents the analysis about the dataset used for the evaluation of the proposed methods and techniques, including a description of the data collection process and the key characteristics of the dataset. It also details the implementation of the CESNET Test Generator, including its design, architecture, and integration with the existing Verefoo framework. **Chapter 7** presents the validation of the implemented test generator by comparing its performance against baseline scenarios and evaluating its effectiveness in generating realistic traffic patterns. Finally, **Chapter 8** summarizes the key findings and contributions of the thesis, and outlines potential directions for future research.

Chapter 2

Network Security Automation

In modern enterprise and critical infrastructure networks, the size and complexity of computer networks are growing, as a consequence of novel architectural paradigms such as the Internet of Things (IoT) and network virtualization. Consequently, the growing scale and complexity of cybersecurity policy management have made manual configuration increasingly impractical in an environment where cyber attacks can dramatically exploit breaches related to any minimum configuration error. As a result, automated network security has become essential for maintaining robust security postures and ensuring compliance with evolving security requirements.

Network Security Automation involves a broad suite of methodologies, tools, and frameworks that focus on automating management, configuration, validation, and monitoring of core security elements like firewalls, Access Control Lists (ACLs), and Intrusion Detection Systems (IDS). This allows organizations to have strong security measures in place while avoiding the risks of human error and delayed responses [1].

The introduction of network softwarization in its two variants, Network Functions Virtualization (NFV) and Software-Defined Networking (SDN), has further enhanced network agility. On the one hand, NFV enables allocating virtual functions instead of manually installed physical middleboxes, whereas SDN decouples the network data plane from the control plane, leading to a centralization of the orchestration operations [2]. On the other hand, they allow users to dynamically request the creation of virtual Service Graphs (SGs), a generalized and more flexible form of the traditional Service Function Chain (SFC). SGs represent logical, virtualized network topologies that are abstracted from the underlying physical infrastructure, thanks to the decoupling of computing and networking resources.

A key challenge in this environment is the enforcement of Network Security Requirements (NSRs), such as data protection and traffic isolation, within these virtual service graphs. Traditionally, defining and enforcing such requirements is the responsibility of a security manager, who operates independently from the network manager responsible for designing the SG topology. This separation of roles can result in miscommunication, gaps in domain expertise, and ultimately, the misconfiguration of security controls. Moreover, the manual configuration of Network Security Functions (NSFs), which include virtual firewalls, IDS/IPS, and other protections, introduces delays in response to threats and increases the likelihood of

errors. For instance, failing to define just one of hundreds of rules required for a complete isolation policy could leave the network vulnerable, despite intentions to enforce strict segmentation [3, 4].

By automating these processes, organizations can accelerate the deployment of security policies, ensure consistent and accurate enforcement of security policies, and significantly reduce the potential for human error. Overall, network security automation both maximizes operations and secures the environment with fast mitigation of threats and continuous policy enforcement [5].

An examination of the Verizon Data Breach Investigations Reports from 2013 to 2021, highlights two key findings that underscore the significance of security configuration issues. Among the various causes of security breaches, both misconfigurations and their broader category, miscellaneous errors, have shown a consistent upward trend. Within this group, misconfigurations have grown from representing 42% of incidents, making them the leading contributor to breaches in this category. Similarly, the overall proportion of incidents attributed to miscellaneous errors has risen, increasing from 5% to 13%. In 2021 alone, Verizon recorded 23,896 security incidents, of which more than 1,300 were the result of misconfigurations (about 5%). This large number illustrates the critical impact of configuration issues on cybersecurity [2].

2.1 Objectives of Network Security Automation

Automation, by definition, is a technique that "emphasizes efficiency, productivity, quality, and reliability, focusing on systems that operate autonomously, often in structured environments over extended periods, and on the explicit structuring of such environments".

The main objective of Network Security Automation is to provide as automatic configuration as possible of security services to minimize human intervention. In other words, when the system receives an external input from a human being or from another system, it should be able to work without requiring other external contributions. In general, a fundamental requirement for automatic configuration is agility, i.e., whenever the current state changes, the system must be able to automatically adapt to the new conditions as fast as possible, to avoid inconsistencies. The absence of agility in traditional computer networks represented one of the main reasons why automation had not already been fully introduced in the past in this engineering field. In addition, the other objectives of Network Security Automation include [2, 6]:

- **Improve Accuracy and Reduce Errors**: it minimizes the risk of human error in security configurations, ensuring that policies are applied consistently and correctly across the network.
- **Enhanced Security Posture:** by automating the deployment, enforcement, and monitoring of security controls, organizations can maintain a resilient security posture and reduce risks of misconfiguration, breaches, and non-compliance.

- **Operational Efficiency:** automation streamlines security operations, reducing the burden of manual configuration and enabling teams to focus on high-value tasks. This leads to faster incident responses and lower operational overhead.

- **Optimize Resource Usage:** with the automation of configuration and management of security operations, organizations improve performance by minimizing unnecessary use of bandwidth, compute, and storage through optimization-aware configuration.

- **Scalability and Management of Complexity:** as networks become more dynamic and complex, automation allows organizations to scale security policies and controls, supporting large, distributed, and multi-tenant environments.

- **Continuous Compliance:** automated monitoring and policy verification help ensure continuous compliance with internal security standards and external regulations, including the ability to generate real-time audit reports.

- **Rapid Threat Mitigation:** integration with threat intelligence and automated response mechanisms allows for the immediate detection, mitigation, and remediation of security threats, reducing Mean Time To Resolution (MTTR).

- **Policy-Aware Service Graph Design:** automation allows embedding security requirements directly into the design of virtualized service graphs, ensuring that security is an integral part of the network design process.

2.2 Problems in Manual Security Configuration

Manual configuration of security policies within networked environments presents a variety of issues that can generate significant security risks and operational inefficiencies. In such a situation, the security manager has to collect the security requirements that have been specified by the network users and manually assign and configure the corresponding NSFs in a way that the requirements are fulfilled. For example, in the event of a requirement for blocking all traffic destined to a particular website, the security manager needs to specify the corresponding filtering rule explicitly on a firewall. However, the process has always been more complex than configuring non-security-related network functions. That is because the failures or mistakes in the process have drastic implications, including exposing network users to malicious attacks.

The main reason is that the configuration of security functions, e.g., firewalls, or anti-spam filters, is traditionally performed manually, with a trial-and-error approach: whenever an attack is detected, the configuration is modified accordingly. This work paradigm is not scalable, and it is prone to several errors due to the human fallibility [7]. If done incorrectly, manual NSF configuration can introduce

anomalies, which are defined by the literature as an incorrect specification introduced during the configuration of an NSF by the security manager. Anomalies are typically classified into three broad categories [6]:

- **Conflicts**, such as where two sets of firewall filtering rules have identical condition sets but effect opposite actions.

  Moreover, conflict anomalies can be classified as:

  - **Contraddiction**, if two rules match the same traffic, but have opposite actions.
  - **Shadowing Conflict**, where two rules match the same traffic, but one is more specific than the other and the one with less priority matches the same traffic of the more specific one. In addition, they have opposite actions.
  - **Correlation**, where two rules specify different actions and some packets are matched by both rules, but there are other packets that are matched by one rule but not the other.

Figure 2.1: Conflict Anomalies - Shadowing Conflict and Correlation

- **Errors**, like when a VPN gateway is instructed to use a cryptographic algorithm that the gateway does not have in its cipher suite.
- **Sub-optimizations**, such as firewall rules that do not trigger because more specific rules already match the same traffic conditions.

  Sub-optimizations can be further classified as:

  - **Irrelevance**, if the firewall rule does not match any packet.
  - **Duplication**, where two rules match the same traffic, but one is more specific than the other.
  - **Shadowing Redundancy**, if two rules have the same action, but one is more specific than the other and the one with less priority matches the same traffic of the more specific one.
  - **Unnecessary**, where a rule is not needed because it does not match any traffic, or because it is already covered by another rule.

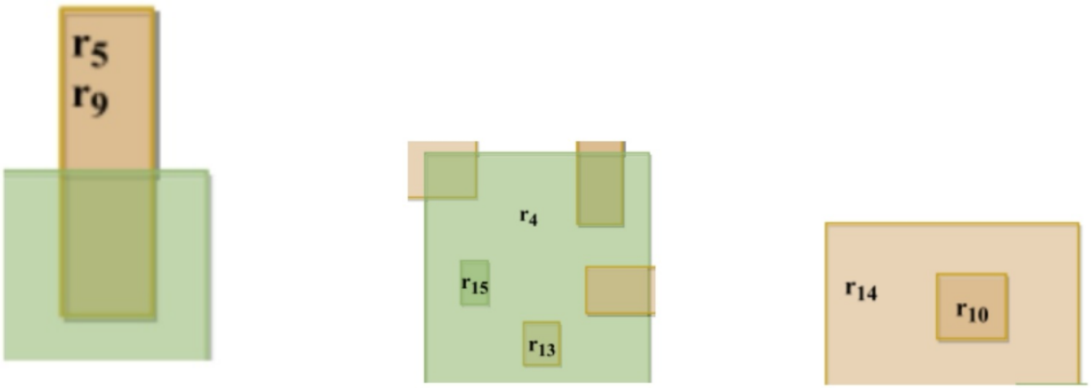

Figure 2.2: Sub-Optimization Anomalies - Duplication, Shadowing Redundancy, and Unnecessary

In addition to configuration issues, human intervention also introduces challenges to orchestrate security controls. For instance, during a transition from one security state of a network to another, i.e., a security transient, it is important that all required changes be quickly effected and in an organized fashion to avoid vulnerabilities arising. Sadly, this speed and precision are usually lacking in human operators, particularly when working under tight time deadlines, raising the risk of mistakes. Similarly, if a user has to enter a predefined security parameter manually into a network orchestrator, there will always be some possibility of the incorrect parameters being entered by mistake, even if the configuration is theoretically correct.

The following outlines the key reasons why manual security orchestration and configuration have become impractical in today's computer networks [2, 6]:

- **Human Error:** manual configuration is inherently error-prone, with common mistakes including misconfigured firewall rules, misplaced access controls, and misdirected routing policies. Even a single error can leave valuable assets open to exploitation or critical services interrupted. All of these are generally amplified by the utilization of trial-and-error techniques, where configurations are modified reactively according to attack needs. This accumulates to ever more complex and unmanageable configurations over time.

- **Role Separation and Lack of Communication:** the duty segregation between security managers and network admins inclines towards poor coordination and weak visibility, which can cause misconfigurations with disastrous consequences. For example, if the network administrator does not provide the security manager full information about the network settings, then the latter could make incorrect assumptions when starting to design the security architecture for the network service defined by the former.

- **Inconsistent Policy Enforcement:** the enforcement of security policies uniformly across a network by hand is hard in practice. Consequently, organizations typically experience policy drift and configuration inconsistencies, where conflicting or overlapping rules cause unintended behavior or security

vulnerabilities. These are crucial in heterogeneous environments that include equipment and software from multiple vendors, each having different forms for configuration and hardware/software management tools, hence minimizing complexity and risk of inconsistency.

- **Lack of Agility:** manually altering security settings to respond to new threats or developing network requirements is slow and labor-intensive. Such incapacity to respond paralyzes the organization's capacity to respond to risks in a timely manner. Agility is likewise disabled when security managers are not constantly informed regarding evolving threats or emerging protection technologies. Without current information, their ability to respond appropriately and promptly is limited.

- **Poor Scalability:** as network sizes and sophistication grow, driven by technologies such as IoT, cloud, and virtualized services, manual configuration remains increasingly impractical. Endpoints, services, and applications that require security action currently far outnumber those that can be practically managed manually. Organizations continue to report this trend and anticipate it will continue, putting more pressure on traditional manual solutions.

- **Increasing Network Sophistication and Diversity:** next-generation networks require the introduction of new services and security features, such as deep packet inspection and threat detection with AI. While these technologies are vital to combat threats of the current generation, they significantly increase the level of complexity for manual management as well. Presence of multi-vendor solutions and virtualized NSFs introduces heterogeneity as well. Each solution may have distinct syntax or config interfaces, so it is extremely important to maintain consistency and security across the entire network.

- **Limited Coordination Between Functions:** in the majority of firms, network management and security administration are carried out by different individuals or teams. Although both functions handle the same infrastructure, absence of frequent interaction and common understanding typically results in uncoordinated decision-making. For instance, if a security administrator is unaware of recent structural evolution in the network, they could set their configurations on the basis of misguided assumptions, inadvertently opening up vulnerabilities.

- **Outdated Training and Knowledge:** security managers are not necessarily brought up to speed with new cyber threats or latest defense mechanisms. Although recent efforts have improved update and training frequency, a vast majority of organizations fall behind. Lack of continuous learning can lead to obsolete or unusable configurations, weakening the overall security posture of the organization.

2.3 Pros and Cons of Automated Network Security Configuration

Automated network security configuration offers several advantages and its introduction is motivated by its ability to overcome most of the limitations listed in the previous section [8].

The main advantages of automated network security configuration include [2]:

- **Enhancing Network Security Through Automation:** trained security professionals are scarce and costly. Therefore, most organizations utilize staff with generic networking skills, supervised by some security professionals. Using automated tools, such teams can counterbalance their limited technical proficiency. Even though human oversight is still necessary.
- **Management of the Scale and Diversity of Modern Networks:** modern computer networks are vast and heterogeneous, making manual handling less feasible. Automatic orchestrators could provide a coherent, global perspective of the network, with centralized and consistent decision-making that would be impossible for a person to manage. Additionally, by using an abstraction layer between the orchestrator and diverse security functions, the system can automatically convert high-level setups into commands specific to the device. This minimizes the necessity to have deep knowledge of every vendor's implementation and makes deployments faster and more accurate.
- **Automatically Optimizing Security Configuration:** as opposed to human methods that are often trial-and-error-oriented, automated orchestration can identify correct and best solutions from the start. Optimization mechanisms can be applied to prevent middleboxes and unnecessary resources, reducing overhead. The systems can also optimize and maximize security guards. It would be extremely hard to manually achieve the same correctness and efficiency.

Despite all these benefits, there are some potential drawbacks that are normally mentioned. These are, however, largely psychological and not technical in nature and based on human bias or prejudice:

- **Fear and suspicion users may have towards automated tools.**

  Historically, automation has generated fear, mainly because users have been ignorant of the workings of these tools. This lack of knowledge has resulted in the belief that automated procedures will cause more harm than human actions.

- **The sense that automated decision-making is untrustworthy or obscure.**

  However, automation can actually improve reliability through the incorporation of formal verification methods, which provide extremely strong correctness assurance, much harder to achieve with manual configuration.

- **The challenge of understanding or debugging automated tools.**

  Remember that the tools are designed by humans, and the developers must ensure that the tools have complete documentation. The documentation enables users to grasp the behavior of the tool and how to solve issues that can occur.

- **The design of the tools and how well they are monitored.**

  Both of these tasks, design and monitoring, are human activities. Therefore, any negative consequences of automation are ultimately the result of human factors, not the automation process itself.

2.4 Sustainability in Firewall Configuration

Sustainability in Firewall Configuration refers to the ability to maintain and evolve automated security practices over time while minimizing environmental impact and resource consumption. It is an increasingly critical design feature for modern networks. However, green objectives related to energy savings are affected by the application of approximate cybersecurity management techniques. In particular, their impact is evident in distributed firewall configuration, where traditional manual approaches create redundant architectures, leading to avoidable power consumption.

Overall, operating networks with energy efficiency has been crucial for two primary purposes: economic and social. From an economic perspective, the continuously increasing size of networks compels providers to limit power usage in a bid to reduce operational costs. Socially, there is a rising global emphasis on environmental sustainability across all industries, compelling network operators to adopt "green" strategies that align with the expectations of both current users and future customers who increasingly value green initiatives [9].

One of the main challenges in sustainable network design is managing the power consumption associated with cybersecurity measures. Ensuring robust protection against a wide variety of attack types is crucial for next-generation networks, where threats evolve rapidly, exploit vulnerabilities in shorter timeframes, and may originate from multiple concurrent sources. The concept of defense in depth, a core principle of security by design, is frequently employed to protect network resources. This method depends on the use of frequency, varied security mechanisms, which all complement one another to create layers of protection. However, what really happens in actuality is that administrators compromise by using similar devices with duplicated configurations, hoping this is adequate. This method not only fails to achieve the real meaning of defense in depth, but also causes unnecessary redundancy, contrary to power-saving, environmentally friendly network administration principles, as every added security function raises the overall power consumption of the network.

A network security function where this issue is becoming increasingly evident is the distributed packet-filtering firewall, which remains the most widely adopted core defense mechanism against prevalent cyber threats. The operation of a distributed firewall impacts network sustainability in two key ways. First, each firewall

instance within a distributed architecture introduces its own baseline power consumption simply by being active. Second, this use of energy rises with the amount of traffic handled by the firewall. The cause of this inefficiency is most often the traditional ways of configuring firewalls, which are even to this day heavily manual, inexact, and trial-and-error-based. As a result of the huge sizes of contemporary networks, human administrators already have a very hard time getting even functionally correct configuration, typically adding misconfigurations that subvert designed firewall functionality. Additionally, they tend to deploy security layers with identical configurations, which undermines the defense in depth strategy and rarely results in an energy-optimized setup.

In order to overcome these limitations, a few research efforts have explored automated approaches to firewall configuration. These have shown promise in reducing human error and, in some cases, have led to more efficient configurations, such as minimizing the number of firewalls and filtering rules. However, while valuable, such optimizations fail to provide a substantial breakthrough for environmentally sustainable networks.

Finally, there is a lack of detailed assessments of firewall energy consumption. Since different firewall implementations may vary in power usage depending on the features they support, such variability must be considered for energy-aware optimization across the network. These should be as few middle nodes as possible to handle as little traffic as possible for getting close to peak power levels. This objective requires placing firewalls close to the sources of traffic [9].

2.5 GreenShield: Optimizing Firewall Configuration for Sustainable Networks

Nowadays, sustainability is recognized as a critical design factor, since current practices in security, particularly setting up firewalls, neglect green factors. Hand-built classical distributed firewall designs have a tendency to add redundancy that is power-intensive. Even the automated solutions implemented primarily focus on security, correctness, and performance, while lacking energy-focused optimization.

GreenShield, a Java framework that integrates security enforcement and sustainability goals, addresses this gap. GreenShield aims to reduce the energy usage of network firewalls by minimizing the number of active devices required to enforce a given security policy. Doing so, it enhances traffic processing efficiency by rendering unwanted traffic filtered as close to its source as possible, reducing unnecessary forwarding and load across the network.

GreenShield is built on an optimization-based orchestration engine that ensures administrator-specified security policies are enforced with confidence while also optimizing for energy savings. This methodology achieves automation by following a principle named policy-based management, where human administrators specify the desired security related to blocked or allowed communications as sentences named policies, which are expressed with a user-friendly language, and later are refined into the concrete network security configuration.

GreenShield combines automation with two other features: optimization and formal verification. The optimization component ensures that the firewall configuration is not only functionally correct but also energy-efficient, minimizing the number of active firewalls and the amount of traffic they process. The formal verification component guarantees that the generated configurations meet the specified security policies, providing a high level of assurance against misconfigurations [9].

2.5.1 Objectives of GreenShield

GreenShield achieves two green-oriented objectives [9]:

- **Energy-efficient firewall activation:** it activates the firewalls of the distributed architecture in a way to minimize the overall average power consumption of the network. From an operational point of view, the savings achieved by GreenShield are related to the power consumption during network operations, and therefore with a constant environmental impact.
- **Traffic-aware firewall placement:** it configures the firewalls so as to block communications as closely as possible to their sources, thus reducing the number of traffic flows processed by each network middlebox.

In addition, GreenShield provides formal assurance that the computed firewall configuration is correct and compliant with the requests of human administrators. All these features are embedded in the methodology of GreenShield, by formulating the configuration problem as a Maximum Satisfiability Modulo Theory (MaxSMT) problem based on constraint programming.

2.5.2 Network Graph and Security Policies

The input definition is the only manual activity that is requested for the user. This does not impact the automation of all next operations, because this task is merely descriptive and does not require computational complex operations.

In particular, GreenShield requires two specific inputs [9]:

- **Network Graph (NG):** representing the computer network where the distributed firewall must be automatically configured in a green-oriented way. It should provide information about how network functions are interconnected and how they are configured. The NG includes three possible types of nodes:
  - **Endpoints**, which represent the source and destination of traffic flows.
  - **Middleboxes**, providing service functionalities, but these do not execute firewalling functionalities.
  - **Firewalls**, instances of the distributed firewalling architecture. Each firewall is initially inactive, and it is activated only if it is needed to enforce the security policies. We assume that an inactive firewall does not perform traffic filtering. Each firewall is associated with a weight, representing the average power consumption.

- **Network Security Policies (NSP):** describing which traffic flows must be blocked because potentially malicious, and which other ones must be able to reach their destination to guarantee the availability of some services. Each NSP is composed of:
  - **Action**, which specifies how the firewalling architecture must manage packets satisfying the condition, and it also discriminates NSPs into two classes: isolation NSP, characterized by a deny action, and reachability NSP, characterized by an allow action.
  - **Condition**, which is used to identify the packets to which the action must be applied. In particular, it specifies the IP 5-tuple of the prohibited or allowed flows, where the 5-tuple is composed of IP source address, IP destination address, source port, destination port, and protocol.

2.5.3 Firewall Configuration Problem Formalization

Upon receiving the administrator input, GreenShield uses it as input to a partial weighted MaxSMT problem formulation. It is a generalization of the traditional SMT problem. The reason for its partiality is the discrimination of two kinds of constraints [9]:

- **Hard Constraints**, whose satisfaction is always required.
- **Soft Constraints**, whose satisfaction can be relaxed because it is not mandatory.

Rather, its weighted character lies in the fact that every soft constraint has a weight, and one wants to maximize the sum of the weights of the satisfied soft constraints.

For what concerns formal verification, if a solution is determinable, the MaxSMT formulation guarantees that every hard constraints are satisfied in such a solution. Such correctness-by-construction assurance is, however, naturally provided as long as basic building blocks of the constraints are formally defined so that their formal models satisfy the properties of their actual counterparts and include all the information potentially influencing the correctness of the solution.

On the other hand, the optimization is taken care of by the partial weightedness of the MaxSMT problem. As the soft constraints do not have to be fully complied with, they are well suited to express GreenShield's optimization objectives. In particular, if some of them cannot be met with a solution, then only it means that all optimization objectives cannot be achieved, but correctness is guaranteed. In summary, soft constraints are used to express the two optimization purposes of GreenShield, i.e., reduction of the average power consumption of the active firewalls, and blocking of traffic flows as close as possible to sources [10].

Lastly, in the formulation of all these constraints, there remain some predicates respectively free because solving the problem of firewall configuration is to find an eventual assignment for them. As a representative example of free predicates

are those specifying the choice of whether or not to enable each firewall for which the human administrator did not explicitly impose a strict requirement, and those specifying the choices of the filtering rules configuration they must enforce [9].

2.5.4 Automatic Output Computation

Once the MaxSMT problem has been formulated, GreenShield runs an automatic solver to search for the existence of an optimal solution and, if so, to produce the solution that optimizes the green goals as much as possible [11].

Whenever GreenShield calls the solver, this returns always in a decidable way, i.e., if there does not exist any solution for all hard constraints, the solver signals to GreenShield the problem's unsolvability, or else it returns the optimal solution. Decidability nature of the constructed MaxSMT is motivated by the observation that we used only a finite subset of theories to construct it. If GreenShield informs the network administrator that the solver has been unable to find any solution which satisfies all hard constraints, then the administrator must alter the input so it becomes solvable, i.e., the used firewalls are not enough to satisfy all NSPs, even when all of them are activated.

Instead, if there exists at least one valid solution, the MaxSMT solver returns GreenShield the assignment of variables and the predicate for the solution that maximizes the aggregation of weights of the soft constraints satisfied. With this result, GreenShield can directly infer the two outputs of the activation scheme, i.e., the set of firewalls that should be activated and the filtering rules to be configured, so that it can block or allow traffic flows as desired by the NSPs [9].

Chapter 3

Edge-to-Cloud Computing

The rise of the Internet of Things (IoT) has led to an explosion of networked devices producing unimaginable volumes of data in real time. They are everywhere in our daily environment, from homes and medical monitoring to industrial sensors, and require robust computing architectures to process and react to this data with quick and clever efficiency. Additionally, the increased complexity of IoT applications has created the need for scalable, responsive, and intelligent computing infrastructures. However, traditional cloud computing, which relies on remote data center processing, increasingly fails to meet the low-latency, high-bandwidth, and reliable connectivity requirements of modern IoT applications. To enable this attribution, the confluence of IoT and Cloud Computing (CC) as a new paradigm with advanced services exclusive to aggregating, storing, and processing data produced by IoT, called Edge-to-Cloud Computing (E2C), has been developed [12].

To overcome the limitations of traditional cloud-based architectures in handling IoT requirements, new paradigms such as Edge and Fog Computing (FC) have emerged. In such a model, computational and storage resources are pooled not just in central cloud data centers but also close to the data source, thus at the edge. This hierarchical and collaborative edge-to-cloud model has the significant advantage of providing intelligent and distributed processing. It enables optimized performance in terms of latency and energy efficiency optimization. Moreover, the integration of Edge Computing (EC) and Fog Computing (FC) into the cloud-IoT paradigm represents a major breakthrough, reshaping both current and future IoT solutions by significantly reducing latency in mission-critical applications and more effectively managing the massive volume of data generated by IoT devices. However, the integration of fog/edge and cloud computing takes a commanding position in providing greater potential for IoT applications, particularly those relying on Artificial Intelligence (AI) and Machine Learning (ML) [13].

Edge and fog-enabled network infrastructure enables edge storage and processing capacity beyond the cloud. These models were introduced to address the gap between the IoT endpoint devices and the centralized cloud infrastructure. Through the integration of the functionalities of end devices, edge nodes, and cloud servers, a hierarchical Internet of Things (IoT) architecture, or edge-fog-cloud is formed. This architecture enhances the system performance, optimizes the utilization of resources, and maximizes the Quality of Experience (QoE) for various IoT services.

Rather than operating in separate layers, IoT applications these days increasingly leverage interoperable coordination among the edge, fog, and cloud layers to deliver reliable services addressing a wide range of time-sensitive and location-dependent requirements. However, the very distributed nature of the edge-fog-cloud setup poses a variety of problems, such as offloading work, locating services, as well as security and privacy. This architecture is furthermore very heterogeneous and dynamic, comprised of a huge variety of devices with varying levels of computational capability. A few of these include wearable technology, sensors, smartphones, cars, gateways, base stations, servers, and other networked nodes that all combine to make up the collective computing power of the system [13].

Moreover, since most data is generated at the edge while computationally intensive processing typically occurs in centralized cloud infrastructures, a flexible interconnection between all participating entities is required to bring the edge closer to the cloud and vice versa. Together with cloud capabilities, edge computing is pushing the boundaries of traditional centralized solutions by enabling efficient data processing and storage, as well as low-latency service execution. This paradigm shift centers around the dynamic, intelligent, and seamless integration of IoT devices, edge resources, and cloud infrastructure into a unified computing environment, often referred to as the computing continuum. The goal of this synergy is to deliver advanced services and applications to end users. This is further empowered by innovations in networking, such as Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [14].

Hence, Edge-to-Cloud Computing (E2C) should address various technical challenges, such as: (i) the distributed data management; (ii) continuum infrastructure virtualization and diverse network connectivity; (iii) optimized and scalable service execution and performance; (iv) guaranteed trust, security, and privacy; (v) reliability and trustworthiness; and (vi) support of scalability, extensibility, and adoption of open-source frameworks [14]. These challenges are crucial for the successful implementation of E2C systems, as they ensure that the architecture can handle the complexities and demands of modern IoT applications while providing a seamless user experience (Figure 3.1).

3.1 Main Elements of the Edge-to-Cloud Architecture

The result of this new paradigm is the fusion of IoT devices and Cloud Computing (CC) into a unified computing environment. Generally, the entities within IoT can be categorized into five main elements: end devices, gateways, applications, cloud, and administrative monitoring tools and offices. This classification features the cloud as a unique entity, without distinguish the entities within it [15].

In this context, there are some characteristics shared by the models beyond the cloud [12]:

- They support computation and storage offloading between the endpoints and edge nodes, cloud devices, or other endpoints.

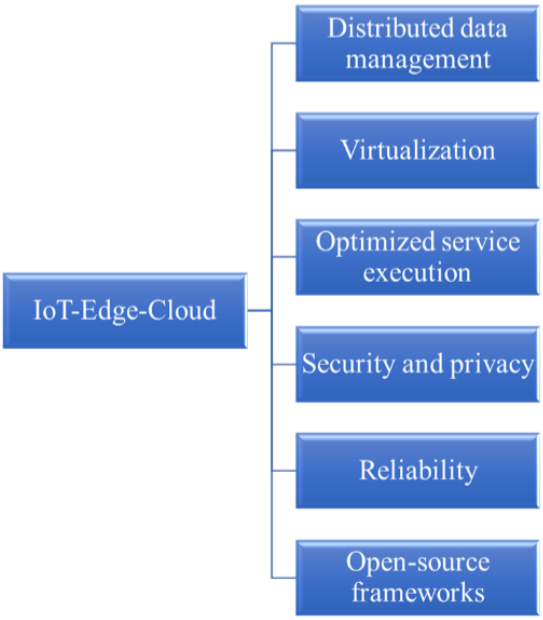

Figure 3.1: Challenges of Edge-to-Cloud Computing

- The offload sources support either local or data processing to decrease the amount of data sent over the network.
- The offload targets are capable of handling data streams from multiple endpoints simultaneously.
- The computing devices involved have varying resource constraints.
- They use a lot of networking technologies such as Ethernet, Wi-Fi, and mobile broadband.
- A centralized controller in the cloud is used to manage the entire system.

The reference architecture consists of three main layers: Endpoint, Edge, and Cloud. This separation is made from a data processing perspective: data is generated by users at the Endpoint layer and is processed either locally or offloaded for remote processing, either to resource-constrained edge devices near the user or to large-scale cloud infrastructures located far away.

- **Endpoint Layer:** this layer consists of the endpoint devices that are typically operated by a single tenant, which generate data via interactions with users or sensors, and are positioned at the end of the network. Examples of endpoint devices are smartphones and IoT sensors.

  They are made up of the following features [12]:

  - **Data Preprocessing:** the endpoint devices implement data preprocessing capabilities to reduce the amount of data that needs to be processed in the cloud. This can include filtering, aggregation, and compression of data.

  - **Application:** user-defined logic determines how to handle data generated at the endpoint. It evaluates whether to offload the data to the edge or cloud, whether additional preprocessing is needed, or if local processing is more appropriate, based on application-level metrics.
  - **Operating System and Resource Management:** the device management layer serves as a bridge between applications and the underlying endpoint hardware. Operating systems like TinyOS and Android are commonly tailored to accommodate the resource constraints of endpoint devices and are capable of supporting specialized hardware.
  - **Infrastructure:** this encompasses all physical and virtual resources available on an endpoint device, including the CPU, memory, virtual machines, and containers. Unlike cloud infrastructure managed by a service provider, these resources are directly accessible to users.

- **Edge Layer:** this layer enables data processing near the user, directly in the field, to satisfy the low-latency and privacy requirements of applications, needs that centralized cloud offloading often cannot fulfill. Notably, edge and cloud environments are generally more distributed and may support multiple tenants, unlike endpoint devices. To support applications in such distributed settings, edge and cloud systems include an operating services layer that is absent in endpoints.

  Edge devices are capable of locally running on bare-metal hardware or can be virtualized with containers or virtual machines (VMs), enabling flexible deployment and efficient resource isolation. These systems are designed to support multi-tenancy, allowing multiple applications or users to share the same underlying infrastructure securely and efficiently.

  The edge layer is made up of the following components [12]:

  - **Application:** developers of user-facing applications on edge systems must determine whether the data offloaded from endpoints should be processed locally at the edge or further forwarded to the cloud.
  - **Back-end:** supports applications designed for resource-constrained environments. These back-end systems are often specialized for a specific domain, for instance, TensorFlow Lite for machine learning tasks.
  - **Resource Manager:** manages physical and virtual resource allocation and distribution across edge applications that may extend over multiple tenants. The resource manager also shapes the selection of the computing models that are available within the edge-cloud continuum.
  - **Operating Services:** facilitates the operation of applications within the highly distributed, heterogeneous, and complex edge environment. These services include support for communication, metadata management, consensus protocols, and more.
  - **Infrastructure:** includes all physical and virtual components available on edge devices, similar to endpoint infrastructure. However, unlike endpoints, which are typically user-managed, whereas edge systems may be operated by service providers. This management model can mirror

cloud environments, where users interact with virtualized resources while providers maintain the underlying physical infrastructure.

The edge layer is composed of both edge devices and fog nodes. The fog is composed of additional layers to reflect the distribution of Fog Nodes (FNs) throughout the network infrastructure. These FNs can communicate both horizontally (with each other) and vertically (with higher or lower layers), depending on system load and application demands. This communication and coordination are typically managed by Fog Orchestration Nodes (FONs), which form the control layer within fog clusters. Depending on the application, fog infrastructures may also include Fog Gateway Nodes (FGNs), gateways or access points that act as entry points into the fog environment. Additionally, Fog Computing Nodes (FCNs) serve as general-purpose computing units responsible for executing tasks delegated by the FONs. To increase flexibility in the E2C architecture, the model can be extended to support edge-device-driven resource orchestration. This allows an edge device to autonomously trigger Service Placement (SP), which helps prevent resource starvation in multi-user scenarios where multiple devices may compete for access to the same service [13].

- **Cloud Layer:** this layer is a parallel and distributed system consisting of virtualized nodes owned by cloud providers and can be provisioned on demand to different consumers. Both the edge and cloud offer users access to physical and virtual resources, utilize operating services and resource managers to hide the complexity of their distributed nature, and support applications through dedicated back-end systems that execute user-defined logic [12].

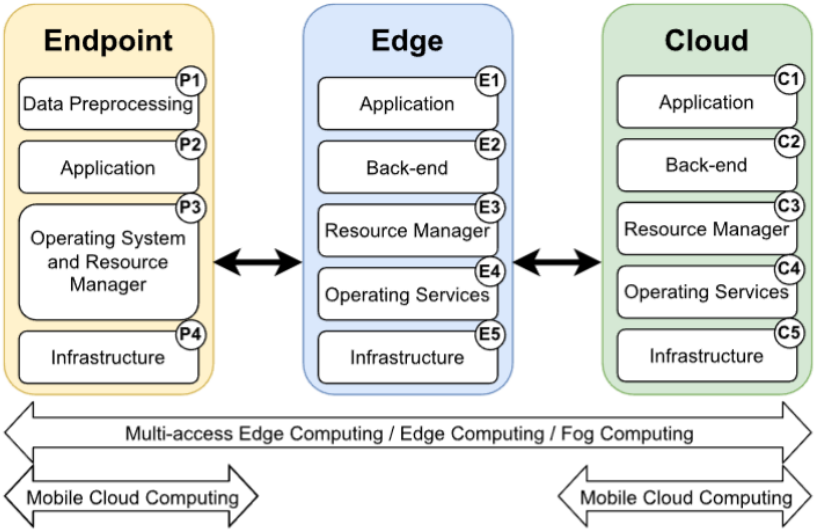

Figure 3.2: Edge-to-Cloud Computing Architecture

The conceptual reference architecture is made up of a horizontal dimension as well as a vertical dimension [13]:

- **Vertical Dimension**, made of 3 main layers, and each layer contains different types of nodes required to handle component tasks. In addition, the edge/fog layer can also consist of multiple sub-layers.

- **Horizontal Dimension**, which captures the following aspects of the architecture:

  - **Node Perspective**, including accelerators, computation, sensors, and storage nodes. Edge/Fog servers, gateways, or devices can stand alone or be connected to the IoT devices, and they can be virtualized or bare-metal.

  - **Communication Perspective**, providing flexibility, scalability, and availability required of communication processes as well as the Quality of Service (QoS) needed to handle delivery of low latency or important data. The connectivity model depends upon the node's location and function.

  - **Control and Management Perspective**, which captures lifecycle management, registration, provisioning, automated discovery, offloading, load balancing, task placement, task migration, and resource allocation.

  - **Data Processing and Analytics Perspective**, which includes how to effectively process, analyze, and manipulate the tsunami of multi-scale and distributed IoT data.

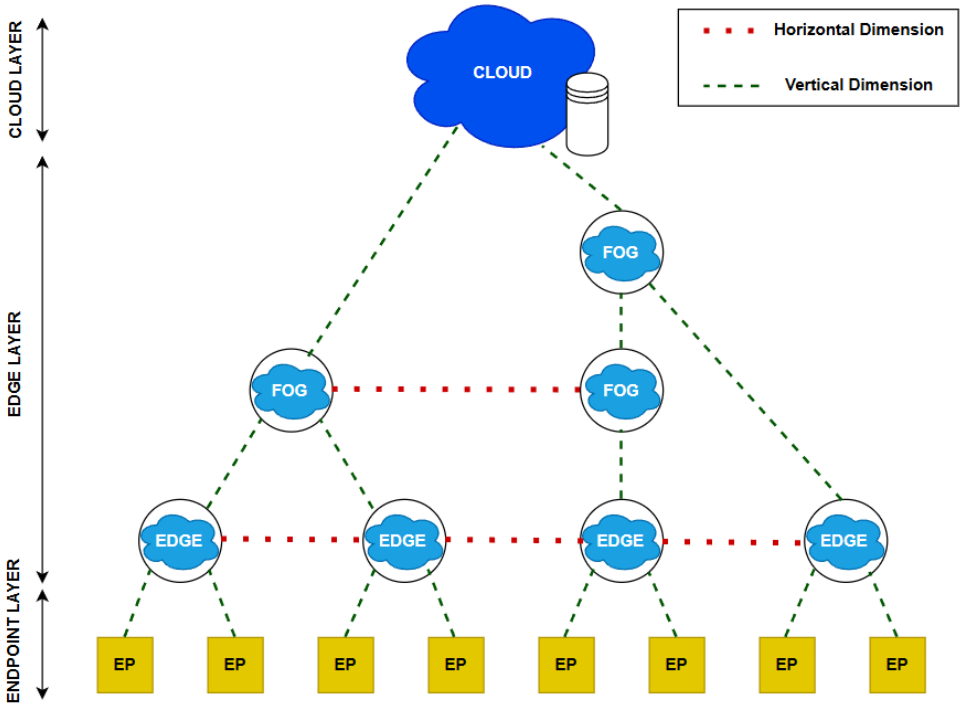

Figure 3.3: Edge-to-Cloud Computing Architecture Dimensions

3.1.1 Connectivity and Integration in Edge-Cloud Systems

The hardware within the Edge-to-Cloud (E2C) is highly heterogeneous. It may come in all sizes, from huge data centers to the smallest single-purpose network-connected sensors and microcontrollers. Essentially, any device with minimal computational power and the ability to connect to a network can participate in the E2C ecosystem.

Software platforms running on that hardware are also very heterogeneous, and they can be categorized into: device-specific firmware without an OS, real-time OS, language runtime, full OS, e.g., Linux, App OS, e.g., Android Wear, server OS, e.g., Linux and Node.js, and container OS, e.g., Linux and Docker.

Networks are crucial in E2C systems, as all devices need to be somehow connected in order to be a part of the same continuum. These networks encompass both hardware and software components, with technologies like Software-Defined Networking (SDN) and Network Function Virtualization (NFV) recognized as key enablers of E2C systems. The technologies and standards used across E2C are also very diverse. Common wireless protocols include WiFi, Bluetooth, and NFC, while less commonly adopted protocols such as ZigBee, Z-Wave, and MiWi are also used. This diversity stems partly from the current lack of universal standards in the still-evolving IoT and ECC landscape. However, efforts are underway to address this, with new interoperable standards being developed and supported by major IoT platform providers [15].

3.2 Edge-to-Cloud Interplay

The widely distributed nature of Edge-to-Cloud (E2C) systems, and the adoption of its infrastructure, has led to the reduction on endpoint device local storage demands and it has also enabled the edge devices to offload computationally intensive tasks. On the other hand, fog computing has enabled the endpoints to offload tasks with lower transmission latency and greater resilience in case of heavy traffic on the network. Moreover, their well-orchestrated collaboration allows endpoint devices to leverage the strengths of both approaches [13]:

- **Reduced Network Load:** processing data streams at the edge layer, by placing computing resources closer to the data sources, helps to balance the load more evenly across the network. As a result, the demand for higher backhaul data rates is minimized, and long-distance data links can be reserved for services that truly require them.

- **Latency-aware Computing:** reducing the network load is only one of the many benefits of redistributing processing resources across the network. By bringing computation closer to data sources, edge devices benefit from faster task execution due to shorter data propagation times. As a result, the Quality of Service (QoS) is enhanced and latency-sensitive applications can be supported more effectively.

- **Native Support for Mobility:** bringing resources closer to the data sources also means the network can responde to the mobility of the users and devices more effectively, because the edge and fog nodes can communicate directly with one another, due to the horizontal dimension of the E2C architecture.
- **Providing Context:** resources near to the users can provide them with specific content related to that geographical area. If the FNs are aware of the absolute position of the resources, they can provide context-aware services to the users.
- **No Single Point of Failure:** the distribution of resources throughout the network also means that if a certain data link is congested or fails, the system can still operate effectively. The edge and fog nodes can communicate with each other, allowing for alternative paths and redundancy in data processing.
- **Low Energy Consumption:** due to the fact that data is significantly processed in or close to the site of origins, the need for long-distance, multi-hop transmissions is minimized, resulting in lower overall energy usage. Additionally, the distributed power demand can be better managed through existing energy infrastructure and renewable sources.
- **Support Heavy Loads:** despite the strength of fog computing, the cloud components remain essential for handling peak loads. When a FN is overwhelmed, tasks can be offloaded to the cloud, which offers extensive computational capacity. This hybrid capability ensures the system remains scalable under heavy workloads.
- **Infinite Storage:** avoiding the need to integrate additional storage capacities into the space-constrained edge and fog nodes, the edge devices can rely on the cloud's ability to almost infinitely expand its resources and meet their storage demands.

3.3 Applications of Edge-to-Cloud Computing

The Edge-to-Cloud paradigm is being increasingly adopted across various scenarios. It is particularly beneficial in applications that require real-time data processing, low latency, and high reliability. Some of the key application areas include:

- **Smart Cities**, which have to ingest and analyze sensor data in real-time. E2C could assist with smart streets, or mapping of noise pollution. In this type of scenario, various types of data are collected and processed from different types of diverse sources, and the goal is to facilitate everyday tasks and improve citizens' life quality. Moreover, data collected directly from sensing IoT devices can be offloaded at edge servers for latency-critical applications, while big data analytics can be used in the cloud infrastructure to analyze long-period time data and reconfigure infrastructure deployment when necessary [13, 14].

  In this scenario, we can point out some components of the E2C architecture:

  - **Endpoints Devices:** these include IoT sensors, surveillance cameras, and environmental sensors.
  - **Edge Devices:** smart gateways in buildings or city blocks, edge servers located in street cabinets, and traffic signal controllers with local compute.
  - **Cloud Infrastructure:** analytics platforms hosted in data centers for energy usage trends or pollution analysis, ML models for large-scale predictions, and data storage and long-term archiving.

- **Healthcare**, where E2C computing addresses several big data challenges in the healthcare sector, such as the dispersed locations of healthcare providers, limited interoperability between systems, and stringent data privacy and security regulations concerning patient information. By enabling efficient data processing, E2C helps convert massive volumes of raw data into actionable insights or smart data. It also facilitates real-time patient monitoring through the integration of sensor technologies [13].

  In the healthcare scenario, we point out the following E2C components:

  - **Endpoints Devices:** patient monitoring devices, e.g., blood pressure monitors or glucometers, imaging equipment such as X-ray machines, and mobile health apps.
  - **Edge Devices:** on-premise hospital servers, smart medical gateways for patient device data aggregating, and edge analytics boxes in clinics for real-time alerting, e.g., detecting anomalies.
  - **Cloud Infrastructure:** healthcare analytics platforms, or AI-based diagnostic services for analyzing medical images, and long-term patient data storage.

- **Smart Transportation Systems**, where E2C enables real-time monitoring of public transit systems like buses, trains, and subways, offering live updates on vehicle locations and delays. It enhances traffic management through technologies such as intelligent traffic lights, surveillance cameras, and vehicle-mounted sensors. Additionally, smart vehicles can interact with roadside infrastructure and E2C platforms to analyze current traffic conditions, calculate optimal routes, and prevent collisions [13].

  In the smart transportation systems scenario, we can identify the following E2C components:

  - **Endpoints Devices:** roadside cameras, inductive loop traffic sensors for vehicle detection, and mobile traffic apps used by drivers.
  - **Edge Devices:** local traffic control centers, and edge devices processing video streams, e.g., for license plate recognition.
  - **Cloud Infrastructure:** centralized traffic management system, integration with public transport systems, and predictive maintenance of traffic infrastructure.

Chapter 4

Thesis Objective

In the evolving landscape of network security and automation, two innovations stand out as particularly relevant: the integration of network security automation in the context of the Edge-to-Cloud Computing (E2C) paradigm, and the analysis of the power consumption of the main involved technologies. This chapter outlines the objectives of the thesis, focusing on these two key areas. Specifically, it highlights the challenges and opportunities they introduce, proposes solutions to address them, and finally presents the implementation and evaluation of the developed solutions.

The GreenShield framework represents an important step forward in the direction of energy-efficient network security. Its main goal is to reduce the energy usage of network firewalls by minimizing the number of active devices required to enforce a given security policy. By doing so, it not only reduces energy consumption, but also enhances traffic processing efficiency. In practice, unwanted traffic is filtered as close to its source as possible, which avoids unnecessary forwarding and reduces the load across the network. The framework is built on top of an optimization-based orchestration engine that ensures the administrator-specified security policies are always enforced with strict correctness, while at the same time applying energy-saving optimizations in a transparent and automated manner. This dual focus on policy enforcement and energy efficiency makes GreenShield particularly suited to future large-scale, distributed network environments.

At the same time, in order to overcome the limitations presented by traditional cloud-based architectures in meeting the diversified requirements of Internet of Things (IoT) applications, a new paradigm known as Edge-to-Cloud Computing (E2C) has emerged. Unlike classical models, where computation and storage are mainly centralized in remote cloud data centers, the E2C paradigm extends these resources closer to the data sources, i.e., at the edge of the network. This hierarchical and collaborative edge-to-cloud model has the significant advantage of enabling distributed intelligence, maximizing responsiveness, and reducing the amount of raw data transmitted across the network. As a result, it is able to provide optimized performance not only in terms of latency reduction, but also in terms of energy efficiency, thanks to more localized processing and adaptive resource allocation strategies. This paradigm shift is especially critical for modern applications that demand real-time processing, such as smart cities, industrial IoT, and cybersecurity monitoring.

To create a connection between these two areas, this thesis proposes a set of novel features to be integrated into the GreenShield framework, enabling it to fully support the Edge-to-Cloud Computing paradigm. The proposed features extend the optimization engine by introducing new hard and soft constraints, designed to account for both the hierarchical structure of the edge-to-cloud architecture and the energy consumption of the devices involved. These extensions will allow GreenShield not only to enforce security policies in a distributed environment, but also to optimize the placement and utilization of security functions with explicit consideration for energy efficiency. A key challenge in this context is the detailed analysis of the power consumption of the virtualization technologies that support edge and cloud infrastructures. Understanding their consumption profiles is fundamental to evaluate whether GreenShield can be effectively and sustainably deployed in E2C scenarios.

Finally, to evaluate the scalability and performance of the proposed solutions, this thesis presents the development of a test generator to simulate representative use cases and measure execution performance under different conditions. The test generator is designed for VEREFOO, a framework for network security automation that constitutes the basis of GreenShield. Specifically, the generator is adapted to the CESNET3 network, a Czech research and education network, and it is extensively used for anomaly detection and traffic classification tasks. The evaluation leverages a dataset of real traffic traces, primarily composed of TLS traffic collected in 2022. By comparing the execution time of the optimization engine under different parameters and configurations, the study aims to assess the scalability and performance of the proposed approach. As a future work, the idea is to extend the test generator to support GreenShield, enabling large-scale evaluations of the proposed energy-aware solutions.

The thesis objectives can therefore be summarized as follows. First, to analyze the energy consumption of different virtualization technologies, which is crucial to understand the overall energy efficiency of the Edge-to-Cloud Computing paradigm and identifying opportunities for optimization. Second, to extend GreenShield with new hard and soft constraints that explicitly consider both the hierarchical nature of E2C architectures and their energy requirements. Third, to define a concrete use case where GreenShield is applied to an E2C scenario, demonstrating the practical benefits of the proposed features and optimizations. Lastly, to design and evaluate a test generator capable of simulating traffic scenarios and verifying the performance of the optimization engine in realistic conditions. The test generator is initially targeted to VEREFOO and the CESNET3 network.

In summary, this thesis aims to bridge the gap between energy-efficient network security automation and the emerging Edge-to-Cloud Computing paradigm. The work is structured around three main contributions. First, it investigates the power consumption of different virtualization technologies, providing insights that are essential for understanding and improving the energy efficiency of distributed computing infrastructures. Second, it extends the GreenShield framework by introducing new hard and soft optimization constraints that explicitly account for the hierarchical and collaborative nature of edge-to-cloud architectures, as well as their energy requirements. Third, it develops a dedicated test generator, integrated with VEREFOO and adapted to the CESNET3 network, in order to evaluate the

scalability and performance of the proposed solutions on real traffic traces.

Altogether, these contributions not only demonstrate the feasibility of applying GreenShield in an Edge-to-Cloud environment, but also highlight how energy-aware policy enforcement can enhance both the sustainability and the effectiveness of future network infrastructures. By combining theoretical analysis, architectural extensions, and practical evaluation, the thesis provides a solid foundation for further research and real-world deployment of secure and energy-efficient network automation frameworks.

Chapter 5

GreenShield in the Edge-to-Cloud Computing

This chapter discusses the integration of GreenShield into Edge-to-Cloud Computing (ECC) environments, focusing on its role in optimizing resource usage and enhancing security across distributed infrastructures.

The idea behind the integration in the Edge-to-Cloud (E2C) is to leverage the capabilities of GreenShield to monitor and manage the energy consumption and security of applications that span across edge devices and cloud resources, focusing also on the virtualization of technologies as firewalls and the power consumption of the used infrastructure.

The chapter first explores the motivations for the integration of GreenShield into E2C environments, highlighting the need for efficient resource management and enhanced security in distributed systems. Then, it performs an analysis of the power consumption of the different virtualization technologies, comparing their efficiency and impact on overall system performance. Finally, it proposes new hard and soft constraints to be implemented in the new version of GreenShield, and concludes with a use case demonstrating its application in a real-world E2C scenario.

5.1 Motivations for GreenShield Integration in Edge-to-Cloud Environments

Before the integration of GreenShield in Edge-to-Cloud (E2C) environments, security management relied mainly on traditional or semi-automated firewall configurations. These approaches often suffered from several limitations:

- **Redundancy and Inefficiency:** distributed firewalls were deployed without energy-awareness, leading to unnecessary activation of multiple devices, which increased power consumption.
- **Lack of Scalability:** as E2C environments combine highly heterogeneous devices, traditional security solutions struggled to scale and adapt to dynamic workloads.

- **Weak Alignment with Sustainability Goals:** classical security management focused primarily on correctness and protection, without considering the environmental and operational costs of excessive energy consumption.
- **Limited Responsiveness to Dynamic Threats:** traditional solutions could not rapidly adapt to evolving attack vectors in highly distributed contexts, where threats may emerge simultaneously at different layers of the E2C architecture.
- **Fragmented Orchestration:** firewall policies were often designed in isolation from resource allocation strategies, preventing a unified optimization of security and system performance.
- **High Operational Overhead:** maintaining security across multiple domains, i.e., edge, fog, and cloud, required significant manual intervention, increasing both complexity and costs.

The motivation for moving to a GreenShield-enabled scenario stems directly from these limitations. E2C environments require not only strict enforcement of security policies but also intelligent resource optimization across distributed infrastructures. By extending GreenShield to E2C, organizations can:

- **Enhance adaptability** in dynamic E2C contexts, where workloads, traffic patterns, and user demands constantly evolve.
- **Integrate seamlessly with other virtualization technologies** (containers, VMs), while ensuring that firewall deployment and optimization respect both performance and energy efficiency levels.
- **Support multi-tenancy and isolation** in shared infrastructures, therefore ensuring that security guarantees remain intact even when resources are dynamically reallocated among different users or services.
- **Enable context-aware enforcement** by leveraging edge proximity to data sources, which allows the system to filter malicious or unnecessary traffic at the earliest possible point, reducing backhaul congestion.
- **Facilitate compliance and auditing** by embedding formal verification in distributed environments, producing verifiable proofs of correctness that help organizations meet regulatory requirements.
- **Optimize cost-efficiency** by reducing not only energy consumption but also the operational expenses tied to manual intervention, redundant configurations, and over-provisioned infrastructures.

This evolution represents a transition from an energy-aware model, primarily focused on optimizing firewall activation and placement, to an extended model that explicitly considers the characteristics of E2C environments. In this new scenario, GreenShield not only enforces security and energy efficiency but also integrates cloud-related aspects such as scalability, multi-tenancy, virtualization overhead, and

distributed resource allocation. By addressing these additional challenges, GreenShield evolves into a comprehensive framework capable of balancing protection, efficiency, and sustainability across the entire E2C continuum, making it particularly suitable for large-scale, heterogeneous, and future-ready infrastructures.

5.2 Analysis of Power Consumption in Virtualization Technologies

Understanding power consumption patterns of virtualization technologies is crucial for optimizing energy efficiency in Edge-to-Cloud (E2C) environments. The analysis involves examining how different virtualization techniques, such as Virtual Machines (VMs) and Containers, impact overall energy use, taking into account workloads, resource allocation, and optimization strategies.

Each physical node in the system can be modeled as a logical node capable of virtualizing one or more additional nodes, enabling scalable and flexible service deployment. For GreenShield, this means that activated firewalls may be either physical or virtualized. In the case of virtualized firewalls, multiple instances can coexist within a single logical node, enabling multi-tenancy. This additional layer of virtualization introduces new patterns of power consumption that must be carefully analyzed and managed to ensure energy-efficient operation.

5.2.1 Physical vs Virtualized Firewalls

The choice between deploying physical or virtualized firewalls has a significant impact on both energy consumption and system performance.

On the one hand, **physical firewalls** provide dedicated hardware resources optimized for packet inspection and filtering, typically ensuring higher and more predictable performance under heavy traffic loads. They offer strong isolation and security guarantees, as they are not subject to multi-tenancy risks. At the same time, physical firewalls have limitations in terms of flexibility and scalability, as they require dedicated hardware for each instance, leading to higher capital and maintenance costs. They have also high and fixed power consumption due to always-on dedicated hardware.

On the other hand, **virtualized firewalls** uses the existing server infrastructure to run multiple instances on a single physical machine. This approach can lead to significant energy savings, as resources can be dynamically allocated based on demand, allowing for more efficient use of hardware. Virtualized firewalls are also highly scalable and flexible, enabling rapid deployment of new instances without the need for additional hardware. However, they may suffer from performance degradation due to resource contention and virtualization overhead, especially under high loads. Security isolation can also be weaker compared to dedicated hardware, depending on the virtualization technology used.

In summary, physical firewalls provide strong performance and isolation at the

cost of higher power consumption, while virtualized firewalls enable flexible, energy-efficient deployment but involve trade-offs in performance and security.

5.2.2 Power Consumption Analysis

Power consumption in virtualization can be studied by comparing hypervisor-based and container-based technologies:

- **Hypervisor-based virtualization** relies on a virtualization layer (the hypervisor) that emulates hardware resources, allowing multiple Virtual Machines (VMs) to run on a single physical host. Each VM includes its own operating system, libraries, and applications, which provides strong isolation and compatibility but introduces additional overhead in terms of memory and CPU usage.
- **Container-based virtualization**, on the other hand, leverages the host operating system's kernel to run multiple isolated user-space instances, known as containers. Unlike VMs, containers share the same OS kernel, making them more lightweight and efficient in terms of resource utilization. This reduces the startup time but may result in weaker isolation compared to VMs.

In general, hypervisor-based technologies tend to have higher power consumption compared to container-based technologies, primarily due to the overhead associated with running multiple full operating systems and the associated resource requirements. In the idle state, the differences are small and negligible, but with network- and CPU-intensive tasks, hypervisor-based technologies consume more power [16].

In the table 5.1, it is presented a clear distinction in power usage patterns between the two virtualization approaches. It summarizes the power consumption for some different platforms: KVM, Xen, which are hypervisor-based, Docker, LXC, which are container-based [16, 17].

Table 5.1: Power Consumption Comparison

Platform	**Idle State**	**CPU-intensive**
Native	123W	-
Docker	124W	176W
LXC	124W	177W
KVM	126W	184W
Xen	128W	201W

Moreover, two additional observations can be made from the power consumption data. First, the power consumption of container-based platforms (Docker and LXC) is consistently lower than that of hypervisor-based platforms (KVM and Xen) in CPU-intensive state, indicating a more efficient resource utilization in containerized environments. Second, the difference in power consumption between idle and CPU-intensive states is more pronounced in hypervisor-based platforms, suggesting that

these platforms may not scale as efficiently under load compared to their container-based counterparts.

At the same time, other element of comparison are the power consumption based on the number of active VMs/container and the memory power consumption [16]:

- **Number of Active VMs/Containers:** power consumption increases logarithmitically with the number of active VMs or containers. This means that while adding more instances does increase power usage, the rate of increase diminishes as more instances are added, indicating a level of efficiency in resource sharing. From this point of view, there are not consistent differences between the two types of virtualization technologies, and the power consumption are quite similar. At the same time, it is important to consider the fact that the number of virtual entities influences how resources are allocated to handle the traffic, since a poor allocation can cause energy waste, whereas skewed distribution can improve efficiency.

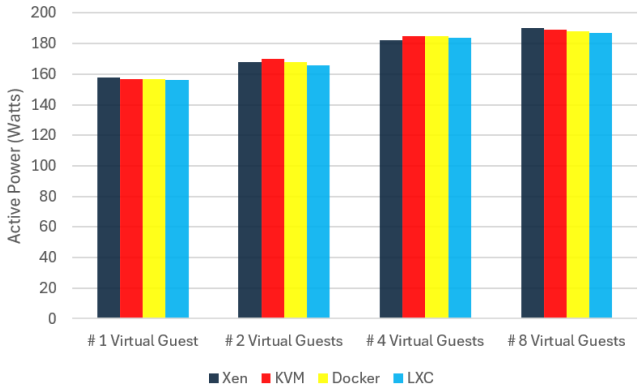

Figure 5.1: Power consumption with different number of active VMs/containers.

- **Memory Allocation:** more resources allocated to guests result in higher power consumption. In general, it is crucial to consider a trade-off between performance and power consumption. This implies that increasing memory allocation can lead to better performance but at the cost of higher power usage.

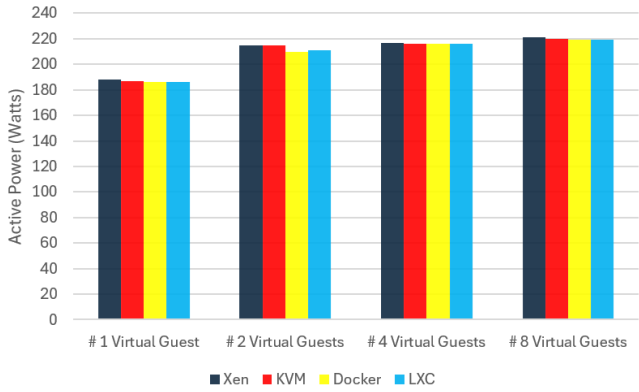

Figure 5.2: Memory power consumption with eight active VMs/containers.

Power consumption depends also on other factors, like the workload. From this point of view, it is important to analyze the specific characteristics of the workloads being executed in virtualized environments. Different workloads can have varying resource requirements and usage patterns, which can significantly impact power consumption. In general, the power consumption increases linearly with the workload, up to a certain point, after which it may plateau due to the saturation of resources [18, 19].

5.2.3 Optimization Techniques

To mitigate the power consumption challenges associated with virtualization, there exist two crucial optimization techniques: vCPU pinning and buffering of network packets.

- **vCPU Pinning:** this technique involves binding virtual CPUs (vCPUs) to specific physical CPU cores. By doing so, it reduces the overhead associated with CPU scheduling and context switching, leading to improved performance and lower power consumption. vCPU pinning can be particularly beneficial in scenarios with predictable workloads, as it allows for better cache utilization and reduced latency. It improves data locality, reduces core migration and the power consumption can be reduced of about 17%.
- **Buffering of Network Packets:** network I/O can be a significant source of power consumption in virtualized environments. By buffering network packets and optimizing their transmission, the frequency of context switches and the associated power overhead can be reduced. Techniques such as batch processing and traffic shaping help achieve this by allowing the CPU to handle multiple packets at once, rather than waking up repeatedly for individual packets. Processing larger batches of packets also minimizes context-switching overhead and cache misses, further improving energy efficiency. It can reduce the overall power consumption of the system of about the 16%.

5.3 Proposed Hard and Soft Constraints

Building on the motivations and power consumption analysis, the integration of GreenShield into Edge-to-Cloud (E2C) environments requires a formalization of constraints that can guide the management of security and resource allocation. As seen in Chapter 3, GreenShield leverages a MaxSMT formulation where hard constraints ensure strict correctness and compliance, while soft constraints encode optimization objectives that improve efficiency without compromising security guarantees.

Moreover, in the context of Edge-to-Cloud (E2C) environments, these constraints must also take into account the dynamic nature of both edge and cloud resources. Unlike traditional centralized systems, where the infrastructure tends to be more static and predictable, E2C ecosystems are composed of highly heterogeneous devices, ranging from resource-constrained IoT endpoints to powerful cloud

servers. As a result, constraint definition and enforcement cannot rely solely on static parameters, but must be designed to adapt dynamically to context changes. This includes an additional layer of complexity to the management of both resources and security policies, as orchestration strategies must continuously balance strict security guarantees with the need for efficient and sustainable use of computing and networking resources.

The proposed constraints aim to directly address these challenges by introducing mechanisms of flexibility and adaptability into the resource management processes. Instead of enforcing rigid placement or activation rules, the new model allows firewall instances and other security mechanisms to be dynamically allocated and reconfigured in response to real-time conditions, such as fluctuating workloads, mobility of users and devices, or unexpected peaks in traffic demand. In this way, the optimization process is not limited to static energy savings, but evolves towards a proactive and context-aware approach that maximizes efficiency without compromising security. By embedding these constraints into GreenShield, the placement and activation of security functions can be continuously adapted, achieving a balance between performance, sustainability, and regulatory compliance. Ultimately, this ensures that the distributed E2C infrastructure is resilient, adaptable, and efficient in terms of energy consumption, even when there is considerable variation and diversity.

- **Node Power Cap:** this hard constraint enforces that each node must not exceed a predefined average power consumption threshold during its operation. By setting such a limit, it becomes possible to avoid situations where nodes are overused, leading to inefficient energy expenditure and potential instability. In specific cases, such as nodes designed to handle specialized or high-priority traffic types, this limit can be selectively relaxed, provided that the hardware is dimensioned to tolerate higher energy usage. The application of this constraint is particularly relevant when multiple logical nodes are virtualized on the same physical host, as exceeding the aggregated power limit could lead to significant inefficiencies or even service disruption. Each logical node is associated with a virtual firewall instance, characterized by its own power consumption and specific features.

- **Virtualization Optimization:** this soft constraint encourages the deployment of firewall instances on nodes that support advanced virtualization features, such as vCPU pinning and network packet buffering. These optimizations help reduce the overhead of virtualization, minimizing context switching and improving cache locality, which in turn lead to both better performance and lower energy consumption. By favoring nodes equipped with these capabilities, GreenShield can ensure that the allocation of virtualized firewalls not only satisfies security requirements but also achieves sustainability and scalability compared to deployments on less optimized infrastructures.

- **Energy-Efficient Software Configuration:** this new soft constraint states that when a node offers multiple possible configurations for its software firewalls, the system should prefer the one that provides the best balance between

energy consumption and performance. In addition, the optimization can extend to the number of active VMs or containers within the same guest, since their proliferation can increase power usage disproportionately. By intelligently adjusting the quantity and configuration of these virtual instances, GreenShield can maintain the required level of protection while ensuring that the energy-to-performance ratio remains as efficient as possible.

5.4 Use Case: Smart City

To demonstrate the practical applicability of the proposed framework, this section presents a use case in which GreenShield is deployed within an Edge-to-Cloud (E2C) environment. The aim of the use case is to fill the gap between the theoretical formulation of constraints and optimization strategies, and their real-world implementation in a distributed infrastructure. By focusing on a representative E2C scenario, the use case illustrates how GreenShield can orchestrate firewall placement and configuration in a manner that simultaneously satisfies strict security requirements and minimizes energy consumption.

In modern urban contexts, and particularly within smart cities, the massive proliferation of IoT devices, ranging from traffic cameras to environmental monitoring systems, requires the design of robust, scalable, and secure network infrastructures. At the same time, the complexity and scale of these environments make manual security management impractical, highlighting the need for automated approaches capable of adapting to evolving conditions. Furthermore, the widespread use of edge computing introduces an additional challenge: ensuring that security mechanisms are not only effective but also energy-efficient, in order to mitigate the environmental impact of growing energy consumption.

The chosen scenario, a smart city, reflects the typical characteristics of modern E2C systems, in which different edge devices collaborate with centralized cloud resources to support IoT applications. Such environments are inherently dynamic, with fluctuating workloads, heterogeneous devices, and different requirements in terms of connectivity, computational power, and energy efficiency. In this context, traditional firewall management approaches prove insufficient, as they cannot adapt to variability while simultaneously meeting the strict sustainability goals that characterize large-scale, future-ready deployments.

5.4.1 The Architecture of the Smart City

The architectural model represents a comprehensive approach to securing and managing heterogeneous data flows within a smart city ecosystem. This system architecture is structured on a Edge-to-Cloud Network composed of three interdependent layers: Endpoint Layer, Edge Computing Layer, and Cloud Processing Layer, each playing a distinct role in the collection, processing, and protection of urban data.

- **Endpoint Layer:** composed of data-generating devices such as traffic cameras, SOS call stations, noise and humidity sensors, and operational offices.

These endpoints collect raw data and forward it for processing. In a real scenario, these endpoints help people in various ways, such as providing real-time traffic updates, monitoring environmental conditions, and facilitating emergency responses.

- **Edge Computing Layer:** composed of fog nodes that aggregate, pre-process, and filter data locally before sending it to the cloud. This layer also hosts both hardware and virtual firewalls, some managed by the city administration, others controlled by third parties. These nodes can be servers or dedicated appliances designed for specific tasks.
- **Cloud Processing Layer:** composed of containerized services and AI-based modules that perform advanced analytics on the data streams, supporting decision-making in real time. These layer is composed of virtualized resources that can be dynamically allocated based on demand, ensuring efficient use of computational power and energy. They are in charge of performing final computation, producing some statistics, and finally sending back to the endpoints and related offices for response handling.

Devices are interconnected through secure channels, with uni-directional flows, e.g., cameras and sensors sending data to the cloud, and bi-directional flows, e.g., offices interacting with both cloud services and sensors. In this architecture, firewalls play a key role in filtering traffic, protecting critical services, and ensuring compliance with security policies.

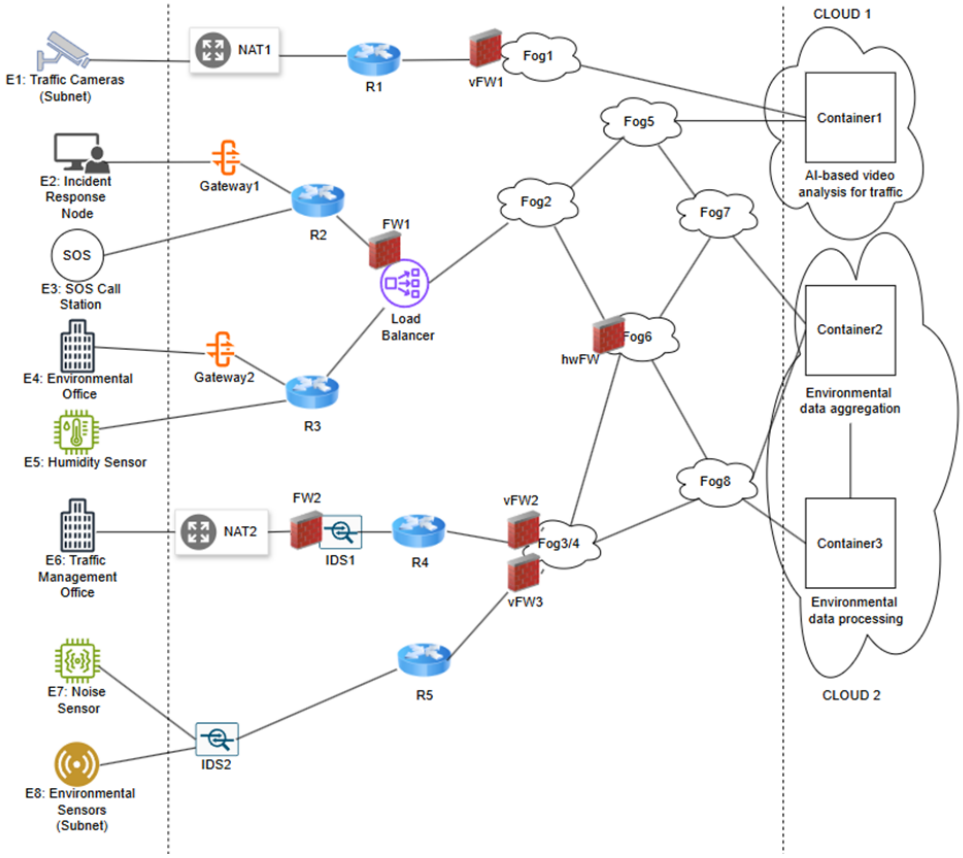

Figure 5.3: Architecture of the Smart City

In particular, the smart city, at the endpoint layer, is composed of the following nodes:

- **Traffic Cameras:** deployed at key intersections and roads, these cameras continuously capture video streams of traffic conditions. They contribute to traffic congestion monitoring, incident detection, vehicle counting, and real-time pattern analysis. The video data is forwarded to AI modules in the cloud for deeper semantic interpretation, e.g., anomaly detection on the public streets, or flow estimation. At the same time, they can also be used for security and surveillance purposes.
- **Incident Response Node:** this node acts as a central dispatcher and communication hub for emergency services. Upon receiving alerts from SOS stations or sensors, e.g., fire, or gas leak, it initiates appropriate response workflows, such as notifying the fire department, redirecting traffic, or activating public warning systems. It also has a microphone to speak through the SOS call station with the person who raised the alert. It interacts with both edge and cloud resources to ensure timely and effective action.
- **SOS Call Station:** installed in public areas, these emergency stations allow citizens to directly call for help. When activated, the station transmits metadata, e.g., location, time, along with a voice or video feed to the Incident Response Node, triggering immediate analysis and intervention procedures.
- **Environmental Office:** serves as the data collection and regulatory node for environmental metrics. It receives sensor inputs, e.g., humidity, noise, air quality, and provides initial validation and reporting. It may also publish alerts to other city departments.
- **Humidity Sensor:** measures the relative humidity. This data is used for weather prediction, urban agriculture monitoring, and optimizing HVAC systems in public buildings.
- **Traffic Management Office:** functions as the city's central traffic coordination hub. It ingests data from traffic cameras, sensors, and road condition reports, then generates optimized traffic light schedules, road use policies, and public notifications.
- **Noise Sensor:** continuously monitors ambient sound levels to detect violations of noise ordinances or to analyze patterns related to population density, traffic noise, or events.
- **Environmental Sensors:** includes a variety of devices measuring air quality, temperature, and light levels. These sensors support public health monitoring, urban planning, and smart building systems.

In general, both the cameras and the various sensors establish a mono-directional communication channel with their respective containers in the cloud. Their role is mainly to capture and forward raw data, such as video streams, environmental measurements, or emergency signals, without requiring feedback from the cloud.

This one-way communication pattern reflects their nature as data producers, designed primarily for continuous monitoring and information delivery rather than interaction.

On the other hand, the operational offices maintain bi-directional communication with the cloud containers. These entities do not simply handle processed data but also contribute actively to the system by updating or enriching the datasets used for analysis. For instance, an office can receive aggregated environmental reports while simultaneously adding new contextual information, such as incident reports or operational instructions, which refine the global decision-making process. Moreover, offices interact directly with sensors, enabling them to monitor or even reconfigure them if necessary. This implies the need for open and secure channels that allow both upstream and downstream data flows, ensuring that the offices can act not only as consumers but also as coordinators within the overall architecture.

To mitigate risks, improve resilience, and reduce communication latency, the **Edge Computing Layer** introduces a set of fog nodes that have localized processing and storage capabilities. These nodes act as intermediate points between the endpoints and the cloud, performing tasks such as data aggregation, filtering, and preliminary security checks before forwarding information for advanced analysis. By processing data closer to the source, fog nodes help reduce bandwidth consumption on backbone links and provide faster response times for applications with high latency. Each fog node can be equipped with firewalls, which may serve different administrative domains:

- **Administrator-controlled firewalls**, configured and maintained directly by the data or infrastructure owners, ensuring strong alignment with global security policies.
- **Third-party-controlled firewalls**, typically managed by external providers, which may impose additional constraints on configuration flexibility and optimization.

Figure 5.4: Hardware and Virtual Firewall

These firewalls can be implemented either as dedicated hardware appliances or as virtual instances running on shared infrastructure. Virtualization allows each fog node to host multiple firewall instances, enabling multi-tenancy and greater scalability, particularly in scenarios where traffic from different services or tenants must remain logically separated. The choice of firewall deployment type, hardware or virtual, depends on several factors, such as expected traffic load, performance requirements, applicable security policies, and available computational resources. For instance, critical nodes handling high-throughput video streams may favor

hardware-based firewalls for predictable performance, whereas flexible containerized firewalls may be preferable in dynamic, multi-service contexts.

At the top of the architecture there is the Cloud Computing Layer, which hosts a collection of containerized microservices responsible for advanced analytics and large-scale data processing. These services include AI-driven modules for real-time video analysis, sensor fusion for environmental monitoring, and decision-support tools capable of integrating data from heterogeneous sources. Containerization provides modularity, scalability, and ease of continuous deployment, features that are essential for coping with the ever changing requirements of smart city operations. In practice, the cloud layer receives data streams that have already been pre-processed by sensors and fog nodes, refines them through advanced algorithms, and outputs actionable insights. These results are then communicated back to the operational offices, which can take corrective measures, coordinate emergency responses, or update field devices. In this way, the cloud layer acts as the intelligence core of the smart city infrastructure, while still relying on the edge and fog layers to ensure timely, efficient, and secure data management.

In conclusion, the architecture of the endpoints, fog nodes, and cloud container is summarized in table 5.2:

Table 5.2: Smart City Nodes

Device	**Description**	**IP Address**
E1: Traffic Cameras	Subnet of Smart Cameras	10.0.1.*
E2: Incident Response Node	Office for Incident Response	10.0.2.1
E3: SOS Call Station	Station for SOS Calls	10.0.2.2
E4: Environmental Office	Office for Environmental Monitoring	40.40.41.*
E5: Humidity Sensor	Sensor for Humidity	40.40.42.1
E6: Traffic Management Office	Office for Traffic Management	10.0.3.*
E7: Noise Sensor	Sensor for Noise Monitoring	40.40.43.1
E8: Environmental Sensors	Subnet of Environmental Sensors	40.40.44.*
NAT1	Network Address Translation	100.64.10.1
NAT2	Network Address Translation	100.64.10.2
Gateway 1	Default Gateway	40.40.41.254
Gateway 2	Default Gateway	10.0.2.254
Router R1	Core Router	198.168.1.1
Router R2	Core Router	198.168.1.2
Router R3	Core Router	198.168.1.3
Load Balancer	Load Balancer	198.168.1.4
IDS1	Intrusion Detection System	198.168.1.5
Router R4	Core Router	198.168.1.6
IDS2	Intrusion Detection System	198.168.1.7
Router R5	Core Router	198.168.1.8
Container 1	AI-Based Video Analysis	198.51.100.1
Container 2	Data Aggregation	198.51.100.2
Container 3	Data Processing	198.51.100.3

Chapter 6

Implementation of the CESNET Test Generator

This chapter presents the implementation of the CESNET Test Generator, a feature developed to support the testing and validation of Verefoo in real-world scenarios, such as the CESNET3 network. The generator is designed to produce realistic traffic patterns and workloads that closely resemble real cases, enabling a thorough evaluation of Verefoo's performance and effectiveness. This work also serves as a preparatory step toward the broader goal of extending testing and validation to GreenShield.

The idea behind the implementation of the test generator is to leverage the capabilities of Verefoo to monitor and manage the security of applications distributed across network nodes. The main goal is to create a flexible and scalable testing framework that can adapt to various network conditions and workloads, ensuring comprehensive coverage of potential security threats. To achieve this, the generator must faithfully reproduce real traffic traces and incorporate all network nodes with their specific characteristics and peculiarities.

The chapter opens with a detailed description of the CESNET3 network architecture and design, followed by an analysis of the dataset used to characterize typical traffic patterns and behaviors within the network. This analysis is essential for generating realistic test scenarios that accurately reflect CESNET's operational environment. Finally, the chapter outlines the specific features and functionalities of the test generator, detailing first how it was implemented and integrated with the existing Verefoo framework, and then how it can be configured to test the performance of Verefoo under different conditions.

6.1 CESNET3 Network Overview

The CESNET3 network is the Czech Republic's national research and education network (NREN), operated by CESNET Z.S.P.O., an association of Czech universities and Academy of Sciences. Since its establishment, CESNET has played a key role in bringing high-performance networking technologies to the academic and

research community, evolving from the country's first academic internet service into one of the most advanced European backbone infrastructures [20].

Nowadays, CESNET3 connects universities, research centers, and other institutions in the nation to nearly half a million users. It is based on redundant topological high-capacity optical fiber routes, which support scalability and redundancy. The architecture supports several link speeds between n x 10 Gbps and n x 100 Gbps to meet high-data-intensity scientific needs, such as high-energy physics experiments, climate modeling, bioinformatics, and distributed large-scale computing.

As can be seen in Figure 6.1, CESNET3 provides the entirety of the Czech Republic with an extensive network of high-density Points of Presence (PoPs). In particular, it is composed of 59 nodes distributed across the country, interconnected by a robust optical fiber infrastructure. Main hubs such as Prague, Brno, Ostrava, Plzeň, and Hradec Králové are utilized as focal aggregation and distribution points, with regional connections offering access to institutions throughout the nation. Three of these nodes, i.e., Prague, Brno, and Ostrava, are also used for monitoring and measurement purposes. CESNET3 has high-capacity connections to international research networks through GÉANT and to large commercial exchanges such as AMS-IX, thus well connecting the Czech research community with the global digital infrastructure.

A defining feature of CESNET3 is its dual role: serving as a stable production-grade service and simultaneously serving as an innovation testbed. It has pioneered technologies in network monitoring, cyber security, photonic networking, distributed data storage, and multimedia transfers throughout the years. Among its specialist services are e-infrastructures for scientific cloud computing, advanced data repositories, and enhanced videoconferencing systems, many of which are already adopted by international scientific collaborations.

In network monitoring and traffic observation, CESNET3 is perhaps best known for its backbone level measurement infrastructure, where multiple 100 Gbps lines are monitored around the clock through Prague, Brno, and Ostrava. These observation points enable mass-scale traffic analysis under real operational conditions, producing high-quality anonymized data for the research community. As an example, CESNET-TLS-Year22, a yearly dataset of encrypted TLS traffic, which is also used for the current analysis, was captured directly from CESNET backbone lines and provides important insights into the temporal evolution of traffic as well as helping to create robust machine learning models for encrypted traffic classification.

No less important is CESNET3's compliance with ethical research and privacy principles. Any monitoring processes are stringently anonymized and data protection policies applied, such that sensitive user information is never revealed without even permitting meaningful scientific results. Being able to strike such a fine balance between operational credibility, research worth, and privacy protection says much for CESNET3 as a credible infrastructure enabler and facilitator of research.

In conclusion, CESNET3 is more than a backbone network. It is a strategic enabler of Czech science and education, a testing ground for networking innovation, and a key player in the international research infrastructure. Its combination

of high-performance connectivity, experimentation capacity, and international connectivity makes it one of the pillars of European collaborative digital research [20].

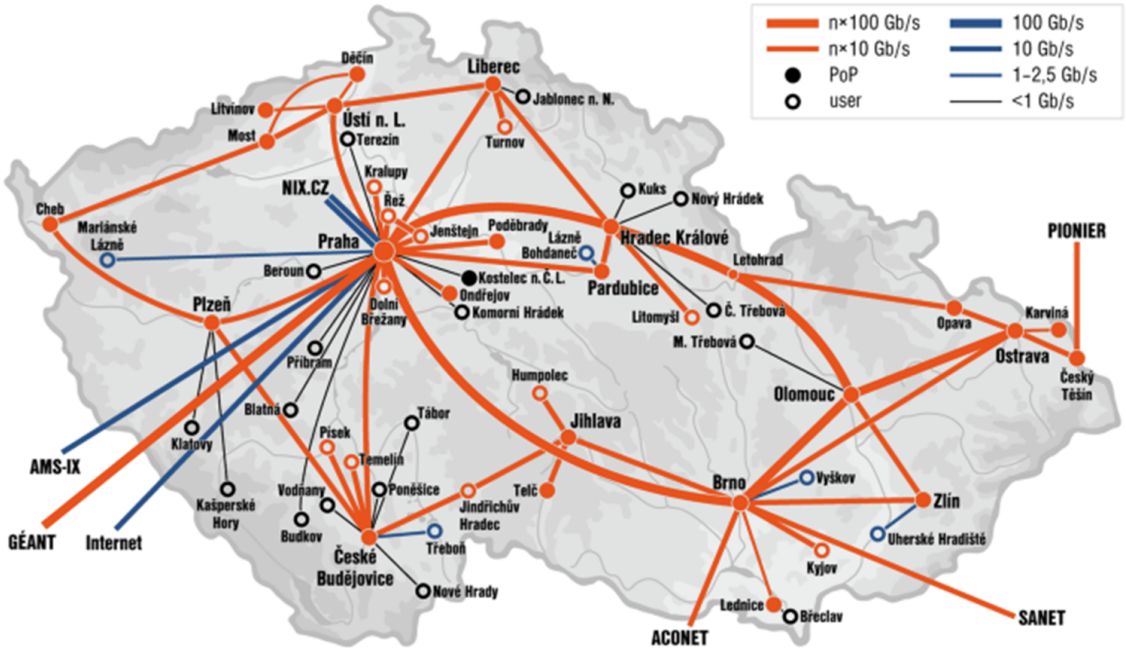

Figure 6.1: CESNET3 Network Architecture

6.2 Dataset Analysis

The CESNET-TLS-Year22 dataset is a large-scale collection of TLS network traffic captured from the backbone of the CESNET3 national research and education network during the entire year of 2022. It represents one of the most comprehensive publicly available datasets for encrypted traffic analysis, providing researchers with unique insight into the evolution of internet services and user behavior over a long time horizon. Unlike many existing datasets, which are often limited to short capture windows or lack temporal context, CESNET-TLS-Year22 spans 52 continuous weeks, making it possible to study trends, seasonal variations, and long-term changes in encrypted communication [20].

The dataset was collected at multiple vantage points on CESNET3's 100 Gbps backbone lines, with monitoring probes deployed in Prague, Brno, and Ostrava. These probes passively observed production traffic, ensuring that the dataset reflects authentic usage patterns from a diverse set of users. In total, CESNET3 serves around half a million users across universities, research institutions, hospitals, and public organizations, which guarantees that the captured data covers a wide variety of applications and services. Importantly, all data were anonymized using privacy-preserving methods such as IP hashing, timestamp rounding, and the removal of sensitive identifiers, ensuring that no individual user can be identified while still preserving the structural and statistical properties of traffic flows. In particular, there is the hashing of the source IP addresses and ports, for each network trace, while other information such as the requested domain and the destination IP address are in clear and accessible.

Each flow record in CESNET-TLS-Year22 describes a bidirectional TLS communication session, enriched with multiple types of metadata. These include packet sequences (capturing the size, direction, and timing of the first 30 packets), flow-level statistics (e.g., byte and packet counts, durations, histograms), and information extracted from the TLS handshake, such as the Server Name Indication (SNI) domain. In total, the dataset labels 180 distinct web services, grouped into 24 categories ranging from cloud platforms and streaming services to social media and communication tools. This diversity makes the dataset especially valuable for machine learning research on encrypted traffic classification, anomaly detection, and robustness against data drift.

By combining scale, diversity, and temporal continuity, CESNET-TLS-Year22 fills a major gap in the availability of open, real-world encrypted traffic datasets. It provides a reliable foundation for evaluating traffic analysis methods, developing robust machine learning models, and advancing the state of the art in encrypted traffic research [20].

6.2.1 Dataset Structure and Statistics

The CESNET-TLS-Year22 dataset is organized to provide researchers with easy access to traffic records while maintaining logical grouping by time and service. The data are stored in compressed CSV files, each file corresponding to a specific week of the year 2022. This weekly granularity enables time-consistent evaluation, allowing models to be trained on one period and tested on subsequent weeks to study the effects of traffic evolution and data drift.

Each CSV file contains one row per bidirectional TLS flow, enriched with both flow-level statistics and packet-level metadata. The attached JSON metadata files summarize the number of flows per week, as well as aggregated counts for individual services and for the entire dataset. This dual structure makes it possible to quickly explore dataset composition without needing to parse all the raw CSV records [20]. The most important dataset's features, also shown in Figure 6.2, are:

- **Source IP:** the hashed IP address of the client initiating the TLS connection.
- **Destination IP:** the IP address of the server hosting the TLS service. This address is not hashed and available in clear.
- **TLS Service Name Indication (SNI):** the hostname requested by the client during the TLS handshake. This field is important to filter the more interesting traffic from the unneeded, in particular to filter the Czech services from the rest.
- **Destination Autonomous System Number (ASN):** the ASN of the server's IP address, providing insight into the network provider. This field is important to filter the traffic related to the CESNET3 network from other providers.

In particular, the ASN related to the CESNET3 traffic is the 2852, which is assigned to CESNET Z.S.P.O. and includes all IP addresses within the network,

while the TLS SNI field used to filter the dataset is every Czech domain, ending with `.cz`. Doing so allows us to focus on the most relevant traffic for our analysis and to exclude all the unneeded traffic traces that can mislead our analysis and statistics about the CESNET3 traffic.

From the point of view of the source IP addresses, we can briefly prove their anonymization by using simple and free-to-use tool, such as [21], where you can insert the IP and obtain information about it, such as geographical location, organization, and more. Inserting the hashed IP address into the tool will not reveal the original IP, thus demonstrating the effectiveness of the anonymization process, like in the example shown in Figure 6.3.

Given the following entries extracted from the dataset:

	ID	SRC_IP	DST_IP	DST_ASN	DST_PORT	PROTOCOL	TLS_SNI	TLS_JA3	TIME_FIRST	TIME_LAST
58	101692285	145.181.82.234	147.32.210.168	2852	443	6	wimap.feld.cvut.cz	c279b0189edb9269da7bc43dea5e0c36	2022-03-20 22:00:00	2022-03-20 22:00:05.257316
87	101692450	7.73.238.153	195.113.180.204	2852	443	6	search.mlp.cz	26f25487242c869a283ccc2989f8ee79	2022-03-20 23:00:00	2022-03-20 23:00:05.979982

Anonymized Source IP Address

Figure 6.2: Entries extracted from the dataset

we observe that looking for the source IP address `7.73.238.153`, even if it refers to a Czech IP address, it does not yield any useful information, as it is hashed and results located in the USA.

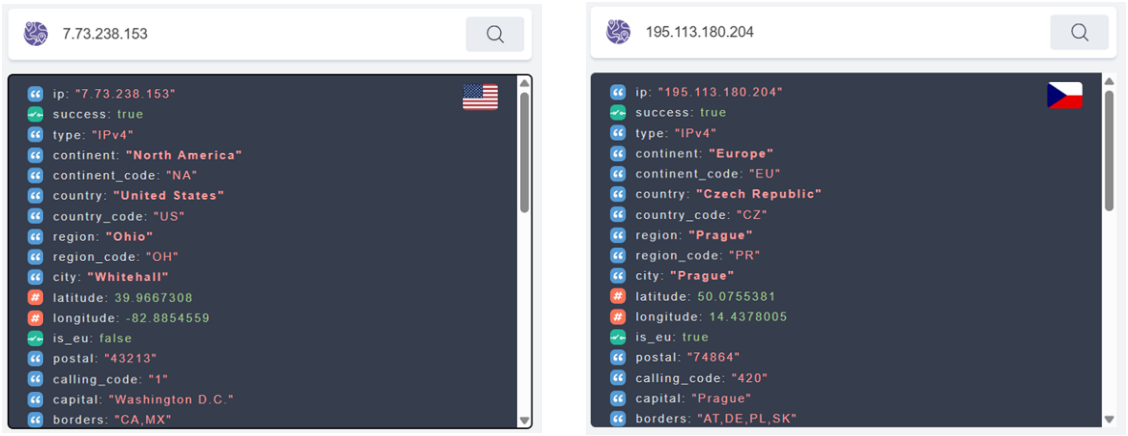

Figure 6.3: Hashed Source IP address and in clear Destination IP address

At the same time, since the destination IP address is in clear, we can easily retrieve information about it, such as geographical location, organization, and more. In this case, the destination IP address `195.113.180.204` relates to a Czech location, in particular Prague.

6.2.2 Methodology for Generating Statistics

To analyze the CESNET-TLS-Year22 dataset and generate meaningful statistics regarding the types of traffic present in the network, a structured pipeline composed

of several steps was adopted. This approach ensures reproducibility and consistency across the dataset, while minimizing the influence of noise and irrelevant traffic. The main phases of the methodology are summarized below:

1. **Dataset Import:** the dataset is imported from `cesnet-datazoo.datasets` library, where it is available alongside other CESNET3-related datasets. The CESNET-TLS-Year22 dataset is organized by week, from week 1 to week 52, and each week is stored as a CSV file containing all the traffic traces captured during that period. Furthermore, the framework allows the selection of different dataset sizes, offering flexibility depending on the computational resources available and the depth of analysis required. It is important to set the flag `return_other_fields` is set to `True` to retrieve additional metadata, such as `src_ip` and `dst_ip`, which are essential for subsequent filtering and geographical analysis.

2. **Data Filtering:** the dataset is filtered to maintain only the most relevant traffic for the analysis. Specifically, traffic is restricted to that associated with the CESNET Z.S.P.O., identified by ASN 2852. Additional filtering is applied to the TLS SNI (Server Name Indication) field to include only Czech domains, i.e., `.cz` domains, thereby focusing on the characterization of local traffic patterns.

3. **Data Cleaning:** the dataset is then cleaned by removing noisy or irrelevant entries. This includes eliminating duplicate records, filtering out anomalous traffic that does not conform to expected patterns, and handling missing or inconsistent values. This step is crucial to ensure that the subsequent statistical analysis is based on high-quality and reliable data, thereby reducing the risk of misleading conclusions.

4. **Statistical Analysis:** after filtering and cleaning, a statistical analysis is performed. First, a set of unique destination IP addresses is extracted. Subsequently, a dictionary is created in which the keys represent cities, and the values correspond to the lists of IP address ranges associated with those cities. The mapping between IP ranges and geographic locations is obtained from publicly available resources such as [22]. Once the dictionary is constructed, the statistics are computed to evaluate the distribution of traffic across different geographic locations. This provides a clear picture of which cities are the most significant endpoints within the CESNET3 network.

5. **Visualization:** the results are visualized through tables, which highlight the most frequent destination cities and, for each city, the most representative IP ranges. These visualizations are particularly useful to identify concentration points of traffic and to understand which areas are most affected. The extracted information is then integrated into the test generator to simulate realistic traffic patterns that closely reflect actual network behavior.

It is crucial to repeat the entire analysis for each of the 52 weeks of the dataset in order to obtain a comprehensive view of traffic dynamics throughout the year. This approach makes it possible to capture temporal variations, seasonal effects,

and long-term trends in network activity, thereby providing a more detailed and robust understanding of the CESNET network's behavior. Moreover, not all weeks contain traffic data to be analyzed, as some were affected by the restrictions imposed during the COVID-19 pandemic, which significantly reduced network activity and led to the closure of many academic institutions [20].

The following lines of codes, illustrates how to set this loop for reproducibility and consistency of the analysis:

Listing 6.1: Analysis of the dataset through all the 52 weeks

```
from cesnet_datazoo.datasets import CESNET_TLS_Year22
from cesnet_datazoo.config import DatasetConfig, AppSelection

# Loop through all weeks of 2022
for i in range(0, 52):
    print("Week", i)

    # Configure dataset for the current week
    dataset_config = DatasetConfig(
        dataset=dataset,
        apps_selection=AppSelection.ALL_KNOWN,
        train_period_name=f"W-2022-{i}",
        return_other_fields=True,
    )

    # Initialize dataset with the configuration
    dataset.set_dataset_config_and_initialize(dataset_config)
    train_dataframe = dataset.get_train_df()

    # Filter dataset for a specific ASN, TLS_SNI ending with
        '.cz', and remove duplicate connections
    cesnet_dataset = train_dataframe[
        (train_dataframe['DST_ASN'].isin([2852])) &
        (train_dataframe['TLS_SNI'].str.endswith('.cz'))
    ].drop_duplicates(
        subset=["SRC_IP", "DST_IP", "TLS_SNI"],
        inplace=False
    )

    # Extract unique destination IPs
    dst_ip = cesnet_dataset['DST_IP'].unique().tolist()

    # Append new destination IPs to the list
    for el in dst_ip:
        if el not in dst_ips:
            dst_ips.append(el)
```

The results of this process form the foundation for the design and implementation of the CESNET Test Generator, which aims to replicate realistic traffic

patterns derived from real-world observations. A summary of the analysis is reported in Table 6.1, which lists the main CESNET3 network nodes alongside the number of unique destination IP addresses associated with them.

Table 6.1: Distribution of IP addresses per city

City	**# IPs**
Prague	80
Ústí nad Labem	21
Brno	6
Ostrava	5
Pardubice	3
Plzeň	3
Liberec	2
České Budějovice	1
Olomouc	1
Hradec Králové	1
Jihlava	1

From Table 6.1, it can be observed that the city with the highest number of distinct destination IP addresses is Prague, with a total of 80 IPs, representing approximately 65% of the addresses in the dataset. This dominance is not surprising, considering Prague's central role as both the capital city and the main technological hub of the Czech Republic, which naturally concentrates a significant portion of the national and academic network infrastructure. In contrast, other cities such as Brno, Ostrava, and Ústí nad Labem, although still relevant, display a much lower number of unique IPs, highlighting the hierarchical structure of network traffic distribution across the country.

6.3 CESNET Test Generator Architecture and Design

The CESNET Test Generator is a framework specifically designed to transform the real-world characteristics of the CESNET3 network into test cases for Verefoo. Its primary purpose is to bridge the gap between raw traffic traces and relevant security policies, ensuring that the evaluation of Verefoo reflects realistic operational conditions. By leveraging both the dataset analysis and the architectural information of the network, the generator automatically produces a set of requirements and network policies, which are then translated into concrete test cases. These test cases are subsequently passed to Verefoo to verify whether the tool can satisfy them under the defined constraints, thereby providing a realistic and reliable validation environment.

The internal logic of the generator follows a structured process. First, it selects nodes in a proportional manner, based on the statistics retrieved from the dataset analysis. This proportionality ensures that the generated network representation

reflects the actual distribution of resources and services within the CESNET infrastructure, avoiding biased or unrealistic scenarios. Next, the generator produces security policies, which can be of different types, such as **reachability** policies, ensuring the communication between two endpoints, or **isolation** policies, ensuring that specific traffic flows are blocked. For each policy, additional constraints are added, reflecting the specific characteristics of the observed traffic and the actual topology of the CESNET3 network. Finally, the generator composes these policies into test inputs for Verefoo, which can then be executed to check the feasibility, consistency, and correctness of the requirements.

In this way, the test generator provides an automated, reproducible, and scalable means of testing Verefoo against highly realistic network conditions. Rather than relying on synthetic or manually defined scenarios, the generator ensures that test cases are derived directly from the operational environment of CESNET3. This guarantees that the resulting experiments are both representative and credible, increasing the robustness of the validation process. Additionally, the automatic generation of diverse and complex policy sets makes it possible to test Verefoo under stress conditions, evaluating not only its correctness but also its scalability when dealing with large-scale infrastructures.

Moreover, this methodology lays the groundwork for extending the same approach to GreenShield. In this context, the pipeline can be employed not only to assess and enforce security policies, but also to address additional objectives related to energy efficiency, such as monitoring and optimizing power consumption and allocating resources in a way that minimizes unnecessary energy usage. By applying the same methodological principles, the test generator can capture these dynamics and reflect them in realistic test scenarios, enabling the joint evaluation of security guarantees and energy efficiency. This flexibility demonstrates the generality of the proposed design and highlights its potential to be adapted to GreenShield's security and sustainability objectives.

The main idea behind the test generator is to preserve a realistic representation of the network environment throughout the entire testing process. By ensuring that the generated test cases accurately reflect the traces and communications reported in the CESNET-TLS-Year22 dataset, the tool achieves a high degree of fidelity to real-world conditions. This realism allows to evaluate not only the functional correctness of Verefoo, but also its performance and scalability in practical scenarios. Consequently, the test generator does not merely serve as a validation tool, but also as a framework for experimentation, enabling the study of how policy generation and enforcement scale when applied to complex and heterogeneous network environments.

6.3.1 Implementation and Design

From an architectural perspective, the test generator is composed of two main Java classes, which cooperate to transform the dataset analysis and the configuration parameters into executable test cases for Verefoo. These classes are `TestCesnet` and `TestCaseGeneratorCesnet`:

- `TestCesnet`: this class contains the `main` method and serves as the entry point of the test generation process. Its primary role is to manage the configuration phase and then delegates the actual generation of test cases to the `TestCaseGeneratorCesnet` class. It defines four main parameters, which can be tuned to control the characteristics of the generated test set:
  - `policyNumber`: an `int` variable representing the total number of policies to be generated. This parameter implicitly controls not only the size of the test case but also the number of source-destination pairs required for the policy creation process. Reachability policies are always bidirectional, meaning that for each pair of nodes two symmetric policies are generated. Isolation policies, on the other hand, are non-bidirectional by default, but by setting the `isolationBidirectional` flag to `true`, they can also be generated as bidirectional. This parameter therefore plays a central role in shaping both the scale and the complexity of the generated test cases.
  - `reachabilityPerc`: a `Double` variable representing the proportion of reachability policies over the total number of generated policies. For instance, if set to 0.7, then 70% of the generated policies will be reachability and the remaining 30% isolation. This parameter allows users to control the balance between the two categories of policies, making it possible to stress-test Verefoo under different distributions and to evaluate how the tool scales when the percentage changes and to analyze the behavior of Verefoo in various scenarios.
  - `usePorts`: a `Boolean` variable indicating whether port constraints should be included in the generated policies. When set to `true`, the generator enriches the policies with specific port numbers, thereby increasing the complexity of the test cases. The inclusion of ports significantly increases the space of possible policy combinations, and thus increases the computational workload for Verefoo, which must verify policies at a finer granularity.
  - `isolationBidirectional`: a `Boolean` variable that specifies whether isolation policies should be bidirectional. When enabled, the generator creates two symmetric isolation policies, analogous to the behavior of reachability policies. This option is useful to evaluate scenarios where isolation requirements must be enforced symmetrically across the network.

  Moreover, the class accepts another parameter, `fileName`, which specifies the output filename and path where the generated test case will be saved. This enables the reproducibility of experiments and the archiving of test inputs for future reference.

  After generating the test case, the class calls the `TestCoarse` method, which is responsible for passing the generated input to Verefoo and measuring its performance. Execution time is computed in milliseconds: a timestamp is taken immediately before invoking Verefoo, and another one right after execution. If the result returned is `SAT`, i.e., Verefoo successfully finds a solution

to the constraints, the difference between the two timestamps is calculated and printed. This mechanism allows researchers to monitor the efficiency of Verefoo under different workloads and policy distributions.

Internally, the `TestCoarse` method serves as a lightweight wrapper around the `VerefooSerializer` class. It takes the `NFV` object produced by the `TestCaseGeneratorCesnet` class, which represents the network topology and the associated security requirements, and directly feeds it into the Verefoo engine. The `VerefooSerializer` is then responsible for parsing the model, validating the security policies, and invoking the underlying constraint solver in order to verify whether a feasible configuration exists. At the end of this process, `TestCoarse` retrieves the resulting NFV object: if a solution is found (`SAT`), this object contains the instantiated configuration that satisfies all the constraints: if no solution exists (`UNSAT`), it still provides useful feedback for analyzing infeasible scenarios.

Beyond its role in correctness verification, the `TestCoarse` function also provides an essential experimental tool. By systematically varying the parameters of the test generation, such as the number of policies, the percentage of reachability versus isolation policies, or the inclusion of port constraints, it becomes possible to evaluate how Verefoo behaves under increasingly complex workloads. The reported execution times allow researchers to assess the scalability of the tool and to identify potential performance bottlenecks in different scenarios. In this sense, `TestCoarse` closes the loop of the testing pipeline: starting from the generated NFV object, it executes the full verification cycle, collects performance metrics, and returns a structured output that can be further marshalled into XML for documentation and reproducibility.

- `TestCaseGeneratorCesnet`: this class performs the actual test case generation. It takes as input the configuration parameters defined in `TestCesnet` and uses them to build the CESNET3 network topology and generate the corresponding security policies. The topology is defined by explicitly creating nodes and links: each node is assigned a name corresponding to an IP address, a type (either `FORWARDER` or `WEBCLIENT`), and a list of neighbors. A `FORWARDER` node may serve either as an endpoint or as an intermediate node along a traffic path, whereas a `WEBCLIENT` node always represents an endpoint. After building the topology, the class selects source–destination pairs in a pseudo-random way and generates security policies according to the chosen pairs and the specified parameters, ensuring variability and realism in the test cases.

To simplify the assignment of IP addresses to the network nodes, each node is associated with a single IP address. Whenever possible, the assigned IP address is one of the destination IPs extracted from the dataset analysis, thus grounding the generated topology to real communication endpoints observed in CESNET. In this way, each node directly corresponds to an actual service or resource in the network, and the generated policies reflect authentic traffic patterns. For nodes that do not appear in the dataset, the generator assigns a conventional IP address from the reserved range `20.0.X.X`, where `X` is a number between 0 and 255. This hybrid

approach guarantees that all nodes in the generated CESNET network are assigned unique and valid IP addresses, preserving both the realism and the completeness of the generated scenarios.

The final IP address allocation is summarized in Tables 6.2 and 6.3. These tables provide a clear mapping between nodes and IP addresses, making it easier to trace back the generated test inputs to the dataset analysis and ensuring that the resulting experiments are both interpretable and reproducible.

Table 6.2: CESNET main nodes IP assignment

Node	IP Address
Prague	146.102.200.120
Ústí nad Labem	195.113.198.50
Brno	147.251.100.25
Ostrava	158.196.100.101
Pardubice	195.113.165.33
Plzeň	147.228.2.7
Liberec	147.230.17.200
České Budějovice	195.113.145.101
Olomouc	195.113.161.188
Hradec Králové	195.113.106.122
Jihlava	195.113.227.169

Table 6.3: CESNET other points IP assignment

Node	IP Address
Řež	193.84.160.28
Jenštejn	20.0.0.8
Lednice	20.0.1.99
Zlín	20.0.1.200
Opava	20.0.2.4
Cheb	20.0.4.10
Most	20.0.6.155
Děčín	20.0.7.87
Letohrad	20.0.8.100
Jindřichův Hradec	20.0.12.28

All the nodes listed in Tables 6.2 and 6.3 are configured as `FORWARDER` nodes, whereas the remaining ones are configured as `WEBCLIENT` nodes. For the sake of readability, the `WEBCLIENT` nodes are not included in the tables, since their number is significantly higher and their IP assignment follows a conventional scheme.

6.3.2 Node Selection

The node selection process is a crucial step in the CESNET Test Generator, as it determines which nodes will be included in the generated test cases. The selection is based on the statistical analysis of the CESNET-TLS-Year22 dataset, which

provides insights into the distribution of traffic across different nodes and services. The goal is to ensure that the selected nodes accurately reflect the real-world characteristics of the CESNET network.

This process is composed of three main steps, which are:

- **Probabilities Computation:** this step involves the computation of the probabilities behind the selection of the nodes. There are some nodes, also called main nodes, which have the higher probabilities, i.e., the 90%, while all the other nodes are considered minor nodes and all together have a probability of 10%. In addition, between the main nodes, Prague is the node with the higher probability, i.e., 80%, while the remaining percentage is assigned to Ústí nad Labem, Ostrava, and Brno, respectively the 10%, 5% and 5%. The idea behind this process is to have a pseudo-random node generation, that reflects the proportions retrieved from the dataset analysis, in order to have a realistic node selection.

- **Node Selection:** this step is performed by the `selectRandomNodes()` function, which takes as arguments the Graph object, the number of policies to be generated, and the Constraints object. This function plays a central role, since it not only selects nodes according to the probabilities computed in the previous step, but also ensures that the selection is consistent and suitable for the generation of valid security policies. More specifically, it generates the same number of sources and destinations, equal to the number of policies requested, and carefully checks that the couples are unique in both directions. For instance, if a pair `(Prague, Brno)` has already been selected, the reverse pair `(Brno, Prague)` will not be generated again.

  Moreover, the function is designed to handle the presence of `FORWARDER` nodes in a realistic way. Since `FORWARDER` nodes can act as sources or destinations, but often represent intermediate devices in the traffic path, the function guarantees that they are always paired with a valid `WEBCLIENT` node to complete the communication. To achieve this, when a `FORWARDER` is selected, the function automatically creates a corresponding `WEBCLIENT`, which inherits the same IP address of the original node, while the `FORWARDER` itself is reassigned to a new conventional IP address obtained by incrementing the previous one. This mechanism prevents inconsistencies in the couples and ensures that each policy reflects a feasible communication scenario.

  In practice, the `selectRandomNodes()` function implements the logic that connects the statistical distribution of the dataset with the technical requirements of the test generation, transforming abstract probabilities into concrete node pairs that can be used to generate realistic policies.

- **Allocation Constraints Introduction:** this step is fundamental to avoid the placement of firewalls by Verefoo within the link between the `FORWARDER` node and the corresponding `WEBCLIENT` just created. This is done using the `AllocationConstraints` object and checking if, for each couple, the endpoints are one of these pseudo nodes. If they are, the function ensures that an `AllocationConstraint` is declared between the `WEBCLIENT` and the corresponding `FORWARDER`, specifying that no firewall can be placed on the link

between them. This is important to specify because in the real topology, the `WEBCLIENT` node does not exist, so a placement of the firewall in that link would result in a wrong solution.

6.3.3 Policy Generation

Policy generation is a critical component of the CESNET Test Generator, as it defines the security requirements that Verefoo will evaluate. The generator creates two types of policies: **reachability** and **isolation** policies. Reachability policies specify that communication must be allowed between two nodes, while isolation policies specify that communication must be blocked between two nodes. These policies are generated according to the parameters defined in the `TestCesnet` class, particularly the total number of policies to generate, the percentage of reachability policies, whether isolation policies should be bidirectional, and whether port constraints should be included.

This process is carried out by the `createPolicy()` function, which plays a central role in transforming the selected nodes and their connections into concrete security requirements. The function takes as input the selected nodes, organized in a `HashMap` where the keys represent source nodes and each key maps to an `ArrayList` of corresponding destination nodes. Additional inputs include the total number of policies to generate, the flag indicating if isolation policies must be bidirectional, the reachability percentage, and the flag for including port constraints. By combining all these inputs, the function ensures that the generated policies are consistent, realistic, and aligned with the parameters defined by the user.

The process can be divided into two main steps:

- **Couples Composition:** the first step involves creating all source-destination pairs from the selected nodes. The function iterates through the `HashMap` and generates unique pairs for each source node with its associated destination nodes. This ensures that each couple represents a valid potential communication path in the network and avoids duplicate or conflicting pairs. The function also preserves the directionality of the pairs, which is particularly important for isolation policies that can be unidirectional or bidirectional depending on the configuration.

- **Policy Generation:** in this step, the actual security policies are generated based on the previously composed couples. The function calculates the number of reachability policies as the product of the total number of policies and the reachability percentage, while the remaining couples are used for isolation policies. For example, if `policyNumber` is 10 and `reachabilityPerc` is 0.7, there will be 10 couples and 10 security policies to generate. Reachability policies are always bidirectional, meaning that for each reachability policy created, a corresponding symmetric policy is automatically added. Isolation policies are unidirectional by default, but if the `isolationBidirectional` flag is set to `true`, they are also mirrored, effectively doubling the number of isolation policies. In the previous example, this would result in a total of 20 policies (14 reachability and 6 isolation).

Furthermore, if the `usePorts` flag is enabled, the function assigns specific port numbers to the policies to more accurately simulate realistic traffic. Since the dataset contains TLS traffic, the source port is set to 2852 and the destination port to 443, reflecting typical network configurations. In the case of bidirectional policies, the ports are inverted in the mirrored policy to maintain correct correspondence between the nodes and the expected communication ports. This step ensures that the generated policies not only respect the network topology and statistical distribution of nodes but also incorporate realistic protocol-specific constraints, thereby enhancing the fidelity of the test cases.

Overall, the `createPolicy()` function translates abstract node selections and parameter settings into concrete, executable security policies that can be directly processed by Verefoo. It ensures that the generated policies are valid, non-conflicting, and representative of real-world network behavior, providing a robust foundation for subsequent verification and performance evaluation.

Chapter 7

Validation and Results

This chapter presents the validation process of the CESNET Test Generator, a critical step in ensuring that the generated test scenarios accurately reflect real-world conditions and effectively evaluate the performance and correctness of Verefoo. Validation is essential to confirm that the tool not only produces syntactically correct test cases, but also generates scenarios that are representative of the CESNET3 network's operational environment, encompassing realistic traffic patterns, network topologies, and security requirements.

The validation process serves multiple purposes. First, it evaluates the functional correctness of Verefoo when processing the generated test cases, ensuring that reachability and isolation policies are properly enforced and that the tool can handle the full range of complexity introduced by realistic network conditions. Finally, the validation explores the scalability and performance of the test generator and Verefoo, testing how both components behave under increasing numbers of policies, nodes, and constraints.

This chapter begins by presenting the criteria and methodology used to validate the CESNET Test Generator, emphasizing reproducibility, coverage, and realism. It then presents a series of validation experiments, detailing the setup, metrics, and results. It includes the comparison of generated scenarios using different configurations and parameters, analyzing how these variations impact the effectiveness of Verefoo in verifying network policies. The chapter concludes by discussing the implications of the validation results for the broader application of the test generator, highlighting its robustness and reliability as a tool for verifying network security policies in complex and heterogeneous environments.

Through this structured validation process, the chapter establishes a high level of confidence in the CESNET Test Generator as a practical framework for testing Verefoo under conditions that closely mirror those of real-world networks. Moreover, the analysis of the results provides valuable insights into the system's scalability, Verefoo's behavior under complex workloads, and the robustness of the implemented security policies. This methodical approach not only guarantees the reproducibility and consistency of the tests but also lays the groundwork for future extensions. In particular, applying the same framework to GreenShield is envisioned as future work, which would allow the evaluation of security policies, network behavior, and the impact on energy consumption in real-world scenarios such as the

CESNET3 network, thereby further confirming the potential and flexibility of the test generator as a tool for advanced validation and experimentation.

7.1 Validation Methodology

The validation methodology for the test generator is designed to ensure that the generated test cases are both realistic and effective in evaluating the performance of Verefoo. The methodology encompasses several key components:

- **Reproducibility:** each test scenario is generated using a fixed topology and a set of parameters derived from the CESNET-TLS-Year22 dataset. This ensures that the scenarios can be consistently reproduced for validation purposes.
- **Coverage:** the generated test cases are designed to cover a wide range of network configurations, including varying numbers of nodes, which are selected in a pseudo-random manner to ensure diversity. The scenarios also include different types of policies, such as reachability and isolation, to test various aspects of Verefoo's functionality.
- **Realism:** the scenarios are constructed to reflect realistic network conditions, including the distribution of IP addresses, traffic patterns, and the presence of both internal and external nodes. This is achieved by analyzing the CESNET-TLS-Year22 dataset to inform the generation process.
- **Metrics:** the validation process includes the collection of various metrics, such as the number of nodes, policies, and constraints in each scenario, as well as the performance of Verefoo in processing these scenarios. These metrics are used to assess the effectiveness of the test generator and the correctness of Verefoo's outputs.

The validation experiments are structured to evaluate the test generator across different configurations. Each experiment involves the following parameters:

- **Policy Number:** the total number of policies to be generated in the test. This parameter is crucial for assessing how well Verefoo can handle varying levels of complexity in network policies and to evaluate its scalability and performance under different workloads.
- **Reachability Percentage:** the percentage of reachability policies among the total number of policies. This parameter is important for testing Verefoo's ability to correctly enforce reachability constraints, which are fundamental to network security and functionality. A lower percentage indicates a higher number of isolation policies, which can increase the complexity of the network configuration.

- **Ports Usage:** a boolean flag indicating whether to include port constraints in the generated policies. This parameter is significant for evaluating Verefoo's capability to handle more granular and specific network policies that involve the usage of ports, adding complexity to the policies, making it a critical aspect of the validation process.
- **Isolation Bidirectional:** a boolean flag indicating whether to include bidirectional isolation policies in the generated test scenarios. This parameter is important for assessing Verefoo's ability to enforce isolation constraints that apply in both directions between nodes. The inclusion of bidirectional isolation policies can significantly impact the complexity and behavior of the generated network configurations. In the whole validation process, this value is set to `true`, meaning that all isolation policies are bidirectional.

The experiments are conducted by systematically varying these parameters to generate a comprehensive set of test cases. Each case is composed of a fixed percentage of reachability policies and a set value for the ports, while the number of policies is incremented until a certain limit. This approach allows for a thorough evaluation of the CESNET Test Generator's ability to produce diverse and realistic test cases, as well as the performance and correctness of Verefoo when processing these scenarios. The resulting execution times are taken as the average of 30 runs for each configuration, ensuring statistical significance and reliability in the results. This is done both for the generation of the scenarios, which is pseudo-random, and for the execution of Verefoo, which can be influenced by various factors such as system load and memory usage.

In the context of Verefoo, the two main algorithms used for validation are **Allocation Places** (AP) and **Maximal Flows** (MF). The AP algorithm deals with the management of possible firewall allocation points in the allocation graph. Each network link can potentially host a firewall, by adding an Allocation Place for each link in the graph, and then explores these options while respecting the constraints defined by the service designer, e.g., mandatory or forbidden positions. The use of AP therefore reduces the problem to a well-defined set of candidate nodes, limiting the search space and making the MaxSMT problem formulation more efficient.

The MF algorithm, on the other hand, is based on the concept of maximal flows, which represent sets of packets traversing the same sequence of nodes and undergoing the same transformations. Compared to packet-level modeling, the MF approach drastically reduces the number of distinct cases to be considered, lowering both the number of free variables and constraints in the MaxSMT problem. This results in significant performance improvements, as the computation of maximal flows is fast and only marginally impacts the overall execution time, while still ensuring correctness and completeness of the model [3].

For the validation process, the same tests are repeated with both algorithms, in order to compare their performance and assess the impact of different modeling strategies. This dual evaluation allows for verifying not only the correctness of the results, but also the differences in terms of efficiency and scalability.

Finally, the MaxSMT solver used in these experiments is Z3, a high-performance theorem prover developed by Microsoft Research. Z3 is widely recognized for its

efficiency and effectiveness in solving complex logical formulas, making it a suitable choice for the validation of Verefoo. The solver's capabilities are leveraged to handle the constraints and variables generated by both the AP and MF algorithms, ensuring that the validation process is robust and reliable [3].

7.2 Validation Experiments

The experiments are designed to systematically explore the impact of these parameters on the performance of Verefoo, as well as to assess the realism and coverage of the generated scenarios. These are conducted on a machine with a Ryzen 7 7000 series CPU, 16 GB of RAM, and running Windows 11 Home. The results are collected and analyzed to provide insights into the effectiveness of the CESNET Test Generator and the performance of Verefoo under various conditions.

The experiments are divided into four main scenarios:

- **AP and MF using a MaxSMT approach with Z3 v4.8.8**: this scenario evaluates the performance of Verefoo using both the AP and MF algorithms, with the MaxSMT solver Z3 version 4.8.8. This version is the default one used in the original implementation of Verefoo [3].
- **AP and MF using a MaxSMT approach with Z3 v4.14.1**: this scenario evaluates the performance of Verefoo using both the AP and MF algorithms, with the MaxSMT solver Z3 version 4.14.1. This version is chosen to evaluate the performance changes in this newer release of the solver.
- **AP and MF using a heuristic approach**: this scenario tests the performance of Verefoo with a heuristic-based approach for both the AP and MF algorithms, aiming to provide a comparison with the MaxSMT-based methods.
- **Extended AP and MF using a heuristic approach:** this scenario extends the previous heuristic approach by incorporating an additional type of policies, called `CompleteReachability`, which requires that all nodes in the network can reach each other. This extension is designed to evaluate how the inclusion of more complex policies affects the performance of Verefoo.

Each experiment involves generating test scenarios with varying numbers of policies, reachability percentages, and port usage configurations. In particular, the experiments are conducted with a fixed reachability percentage and port usage setting, while incrementally increasing the number of policies by 20 for each test. This systematic variation allows for a comprehensive assessment of how these parameters influence the performance and scalability of Verefoo.

In general, it is crucial to note that the each experiments is repeated 30 times for each configuration, and the execution times are averaged to ensure statistical significance and reliability in the results. This approach helps to mitigate the effects of the pseudo-random selection of the nodes and the variability in system performance, providing a more accurate representation of Verefoo's capabilities

under different conditions. This helps also to smooth out any anomalies or outliers that may arise due to transient system states or other external factors.

7.2.1 MaxSMT Approach With Z3 v4.8.8

The first set of experiments focuses on evaluating the performance of Verefoo using both the Allocation Places (AP) and Maximal Flows (MF) algorithms, with the MaxSMT solver Z3 version 4.8.8. The experiments are conducted with varying configurations, which are composed of:

- **Reachability Percentage:** this parameter is fixed in the experiment, and can take different values, i.e., 10%, 30%, 50%, 70%, and 90%. This allows for assessing how the proportion of reachability policies affects the performance of Verefoo.
- **Policy Number:** this parameter is incremented by 20 for each test, starting from a base value. This systematic increase enables the evaluation of Verefoo's scalability and performance under different levels of policy complexity. This value is varied from 20 to 200, and it is incremented after 30 iterations of the same test configuration.
- **Use Ports:** this boolean parameter is first set to `false`, meaning that no port constraints are included in the generated policies, and after the whole experiments, this is repeated with the value set to `true`, meaning that port constraints are included.

7.2.2 MaxSMT Approach With Z3 v4.14.1

The second set of experiments evaluates the performance of Verefoo using both the Allocation Places (AP) and Maximal Flows (MF) algorithms, with the MaxSMT solver Z3 version 4.14.1. This version is chosen to assess the performance changes in this newer release of the solver. The configurations for these experiments are the same as those used in the previous set, varying the reachability percentage, policy number, and port usage settings. This allows for a direct comparison of the performance of Verefoo with different versions of the Z3 solver, providing insights into how updates to the solver may impact the efficiency and scalability of the tool.

7.2.3 Heuristic Approach and Extension

The last set of experiments evaluates the performance of Verefoo using a heuristic-based approach for both the Allocation Places (AP) and Maximal Flows (MF) algorithms. This approach is designed to provide a comparison with the MaxSMT-based methods, focusing on the efficiency and scalability of the heuristic techniques. The experiments are conducted with the same configurations as the previous set, varying the reachability percentage, policy number, and port usage settings.

In addition to the standard heuristic approach, an extended version is also tested, which incorporates an new type of policy called `CompleteReachability`. The extended heuristic approach aims to evaluate how the inclusion of more complex policies affects the performance of Verefoo. The configurations for this extended approach are similar to the previous experiments, but the experiments are conducted only with a reachability percentage of 50%, where 50% are isolation policies, 30% reachability policies, and 20% complete reachability policies. Moreover, the experiments are conducted with a number of policies that starts from 100 and is incremented by 100 until reaching 1000 policies, always conducting the experiments with both ports usage set to `true` and `false`.

7.3 Results and Analysis

The results of the validation phase are presented in a series of charts that show the performance of Verefoo under different configurations. Each chart corresponds to a specific set of parameters, including the reachability percentage, policy number, and port-usage settings. Execution times are reported in milliseconds (ms) and the average value is taken over 30 runs for each configuration. In particular, each configuration is composed of a fixed reachability percentage and a set value for the ports, while the number of policies is incremented until a certain limit.

Overall, this phase illustrates the performance of Verefoo across different situations, highlighting differences between the AP and MF algorithms, the impact of different Z3 solver versions, and also the contrast between MaxSMT and heuristic approaches. The results provide insights into the scalability and efficiency of Verefoo, as well as the effectiveness of the CESNET Test Generator in producing realistic and diverse test scenarios.

The analysis compares the different configurations and examines trends and patterns observed in the execution times. It is organized to evaluate the performance and scalability of Verefoo across the various scenarios, highlighting the strengths and weaknesses of each approach. Moreover, the results are discussed in the context of the validation objectives, with particular attention to the realism and coverage of the generated test cases. Comparisons are reported only for those configurations that show significant differences and where the analysis is more interesting.

The first case concerns the comparison between two versions of the MaxSMT solver Z3, i.e., v4.8.8 and v4.14.1, using both the AP and MF algorithms. The analysis focuses on the execution times and scalability of Verefoo with the two solver versions, in order to evaluate performance variations between them. A first comparison can be made by analyzing the configuration with a reachability percentage of 50% and no port usage, as shown in Figure 7.1 and Figure 7.2 for the AP algorithm.

In the charts, it is shown that using the newer version of Z3, execution times are significantly lower and the scalability is improved, since Verefoo reaches 120 policies while in the older version it stops at 100 policies. In particular, for 100 policies, the execution time with Z3 v4.14.1 decreases of about the 23% compared to Z3 v4.8.8, and this trend is consistent across different policy numbers.

Figure 7.1: 50%, no ports, AP with Z3 v4.8.8

Figure 7.2: 50%, no ports, AP with Z3 v4.14.1

Another interesting comparison can be made by analyzing the configuration with a reachability percentage of 90% and port usage both disabled and enabled, again using the AP algorithm. In the first case, the execution time with the newer version of Z3 is significantly lower than the older version, showing a decrease of about 21%, as reported in Figure 7.3 and Figure 7.4, thus confirming the trend observed in the previous comparison. In the second case, with the port usage enabled, the performance differs from the previous one, since, in general, Verefoo does not scale well with either version of Z3, as shown in Figure 7.5 and Figure 7.6. In particular, execution times are considerably higher, and scalability is limited, since Verefoo stops at 100 policies.

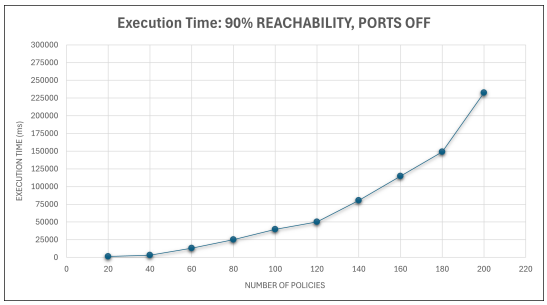

Figure 7.3: 90%, no ports, AP with Z3 v4.8.8

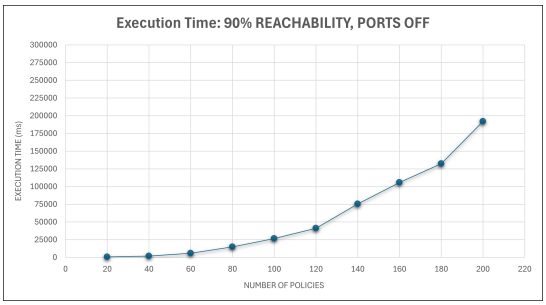

Figure 7.4: 90%, no ports, AP with Z3 v4.14.1

In the second case, the reduction in execution time is about 20% for 100 policies, confirming the trend observed in the previous comparisons. However, it is important to note that the performance degradation with port usage enabled is significant, indicating that this configuration introduces additional complexity that impacts the efficiency of Verefoo.

In general, the decreasing trend observed in these comparisons is confirmed across all the tested configurations, showing that the newer version of Z3 provides significant performance improvements for Verefoo when using the AP algorithm. On the other hand, using MF algorithm scalability does not improve significantly, as for both Z3 versions, Verefoo stops at 20 policies for a reachability percentage of 90%, both with and without port usage, while for the other configurations the tool is unable to find a solution. In addition, the execution times are much higher than those observed with the AP algorithm, indicating that the MF approach may not be as effective in handling the complexity of the generated scenarios. The results

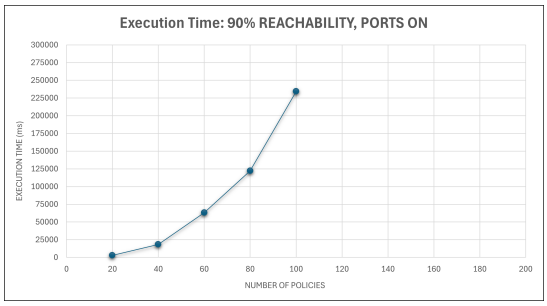

Figure 7.5: 90%, ports enabled, AP with Z3 v4.8.8

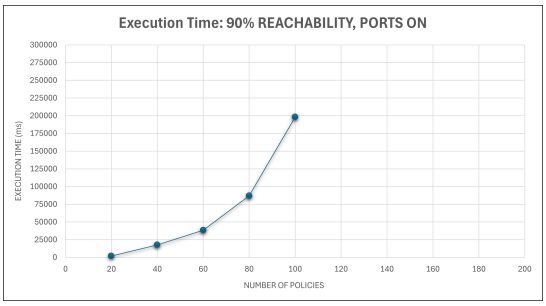

Figure 7.6: 90%, ports enabled, AP with Z3 v4.14.1

are summarized in the Table 7.1:

Table 7.1: MF Algorithm Comparison Between Z3 v4.8.8 and Z3 v4.14.1

Z3 Version	**Ports Usage**	**Execution Time (ms)**
v4.8.8	False	469178
v4.14.1	False	456928
v4.8.8	True	572176
v4.14.1	True	468957

From this table, it can be observed that execution times, when port usage is disabled, are quite similar, with only a slight improvement in the newer version of Z3, i.e., a decreasing of about 2.6%. However, when port usage is enabled, the performance difference becomes more significant, with the newer version of Z3 showing a significant improvement of about 18%.

The second case focuses on the MaxSMT approach using Z3 v4.8.8 and the heuristic approach, both with the AP and MF algorithms. The analysis evaluates the execution times and scalability of Verefoo using the two approaches, to analyze the performance variations between the MaxSMT-based methods and the heuristic techniques.

The first comparison considered involves a reachability percentage of 50% with port usage both disabled and enabled. In the case of AP algorithm and port usage disabled, the most evident result is that the heuristic approach scales much better than the MaxSMT approach, reaching 200 policies, while the MaxSMT approach stops at 100 policies. Moreover, the execution times are significantly lower for the heuristic approach, i.e., 6135 ms for 200 policies, compared to 262482 ms for 100 policies with MaxSMT. This trend also holds when port usage is enabled, since the scalability remains higher for the heuristic approach, reaching 200 policies, while the MaxSMT approach stops at 40, and execution times are significantly lower, as shown in Table 7.2. On the other hand, with the port usage enabled, the execution times are significantly higher for both the approaches, i.e., 41481 ms for 200 policies in the heuristic case, but the difference between the two is still substantial, with the heuristic approach being much more efficient and scalable (Table 7.2).

At the same time, other two complementary configurations can be analyzed, i.e., a reachability percentage of 10% and 90%, both with port usage disabled

Table 7.2: Comparison Between MaxSMT and Heuristic Approaches Using AP Algorithm with 50% and Ports Disabled/Enabled

Policies	**MaxSMT (Ports Disabled)**	**Heuristic (Ports Disabled)**	**MaxSMT (Ports Enabled)**	**Heuristic (Ports Enabled)**
20	1832	147	50525	579
40	5582	447	330262	2651
60	28258	998	UNSAT	6766
80	62374	1719	UNSAT	11691
100	262482	2327	UNSAT	15340
120	UNSAT	2991	UNSAT	19882
140	UNSAT	3867	UNSAT	25156
160	UNSAT	4741	UNSAT	28688
180	UNSAT	5275	UNSAT	33553
200	UNSAT	6135	UNSAT	41481

and enabled. In the first case, the scalability is generally much better for both approaches, with the MaxSMT approach reaching 180 policies, while the heuristic approach reaches 200 policies, and the proportions between the execution times are similar to those observed in the previous comparison, as shown in Figure 7.7 and Figure 7.8. Moreover, also in the case with port usage enabled, the difference between the two approaches is still significant, with the heuristic approach being much more efficient and scalable, as shown in Figure 7.9 and Figure 7.10.

Figure 7.7: 10%, ports disabled, MaxSMT approach

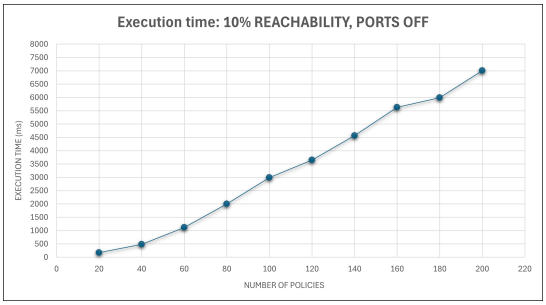

Figure 7.8: 10%, ports disabled, Heuristic approach

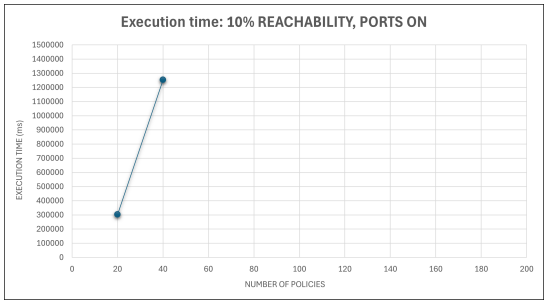

Figure 7.9: 10%, ports enabled, MaxSMT approach

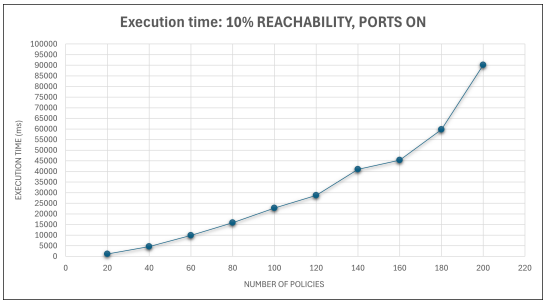

Figure 7.10: 10%, ports enabled, Heuristic approach

Finally, to complete the analysis, the configuration with a reachability percentage of 90% is considered. This is particularly interesting because it is complementary to the previous one, and it allows for assessing how the proportion of reachability policies affects the performance of Verefoo. In the first case, with 90% reachability and port usage disabled, the scalability is generally better for both approaches since they reach 200 policies, but the execution times are still significantly different, with the heuristic approach being much more efficient, as shown in Table 7.3. On the other hand, with port usage enabled, the scalability is significantly limited in the case of the MaxSMT approach, while the heuristic approach reaches 200 policies and the execution times are still significantly different, with the heuristic approach being much more efficient, as shown in Table 7.4.

Table 7.3: Comparison Between MaxSMT and Heuristic Approaches Using AP Algorithm with 90% and Ports Disabled

Number of Policies	**MaxSMT**	**Heuristic**
20	1244	109
40	3129	388
60	12983	865
80	25063	1548
100	39538	2106
120	50254	2529
140	80425	3494
160	114867	4155
180	149140	4838
200	232534	5443

Table 7.4: Comparison Between MaxSMT and Heuristic Approaches Using AP Algorithm with 90% and Ports Enabled

Number of Policies	**MaxSMT**	**Heuristic**
20	2844	403
40	18251	1884
60	63267	5439
80	122225	8054
100	234273	11521
120	UNSAT	12819
140	UNSAT	16090
160	UNSAT	21223
180	UNSAT	23499
200	UNSAT	25054

A third case concerns the heuristic approach, using both the AP and MF algorithms. The experiments are conducted with varying configurations to analyze the performance differences between the two algorithms.

Using MF, it is immediately evident that execution times are quite similar for both the port usage disabled and enabled. At the same time, the execution times

are significantly lower than those observed with the AP algorithm. The worst case, using the MF algorithm, is with a reachability percentage of 10%, with both port usage disabled and enabled. In this case, the execution time for 200 policies is 1538 ms with port usage disabled, and 2197 ms with port usage enabled, resulting as the higher values obtained running the tests with MF algorithm.

On the other hand, using AP algorithm, the execution times are significantly higher than those observed with the MF algorithm, both with port usage disabled and enabled. In addition, the execution times have a significant difference between the two cases, since the port usage introduces additional complexity that impacts the efficiency of Verefoo, in particular when using AP algorithm. The results are shown in Figure 7.11, Figure 7.12, where is reported the execution times using AP algorithm to compare the obtained results when using the port usage disabled and enabled, with a reachability percentage of 90%.

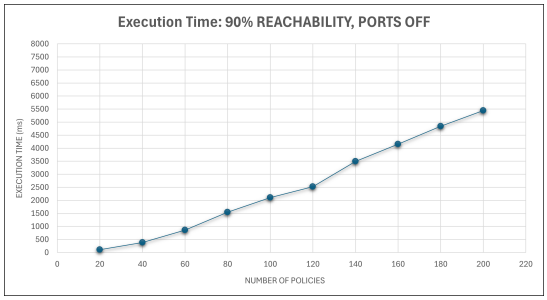

Figure 7.11: 90%, ports disabled, Heuristic approach with AP

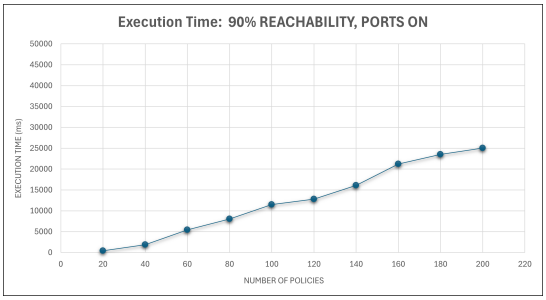

Figure 7.12: 90%, ports enabled, Heuristic approach with AP

An additional comparison can be done showing the obtained results using AP, with 30% reachability. In this case, the execution times are significantly higher with respect to the previous case, because the number of isolation policies is higher, and this increases the complexity of the problem. In addition, the difference between the execution times with port usage disabled and enabled is still significant, as shown in Figure 7.13 and Figure 7.14.

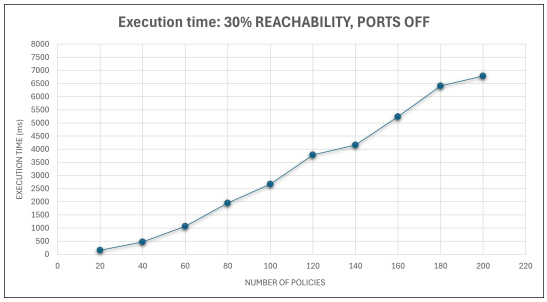

Figure 7.13: 30%, ports disabled, Heuristic approach with AP

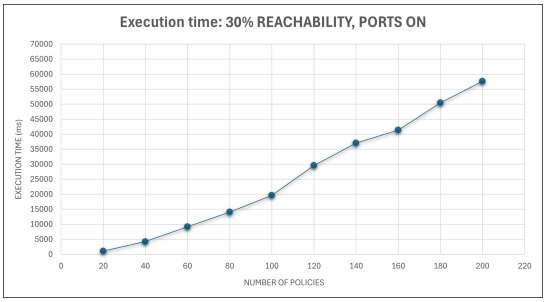

Figure 7.14: 30%, ports enabled, Heuristic approach with AP

Finally, an analysis can be done on the extended experiments using heuristic approach, which incorporates the `CompleteReachability` policies. This analysis focuses on the execution times and scalability of Verefoo using the extended heuristic approach, to analyze the performance variations introduced by the inclusion of

more complex policies. The experiments are conducted with a reachability percentage of 50%, both port usage disabled and enabled, and with a number of policies that starts from 100 and is incremented by 100 until reaching 1000 policies.

In the case with port usage disabled, using both AP and MF algorithms, Verefoo reaches 1000 policies. Using the MF algorithm, the execution times are significantly lower than those observed with the AP algorithm, as shown in Figure 7.15 and Figure 7.16. In particular, using the AP algorithm, the execution time is about 4.5 times higher than using the MF algorithm, for 1000 policies.

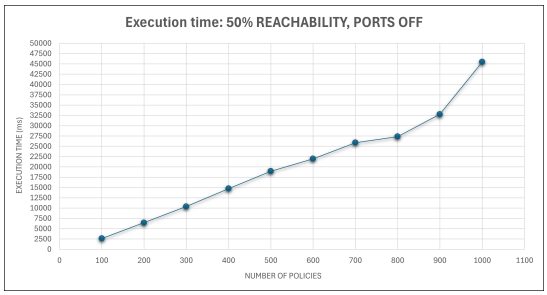

Figure 7.15: 50%, ports disabled, Extended Heuristic approach with AP

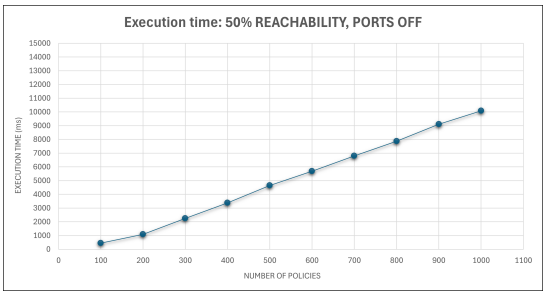

Figure 7.16: 50%, ports disabled, Extended Heuristic approach with MF

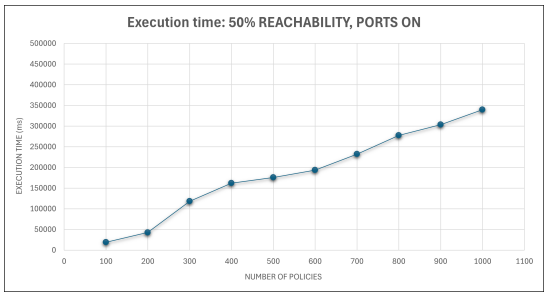

Figure 7.17: 50%, ports enabled, Extended Heuristic approach with AP

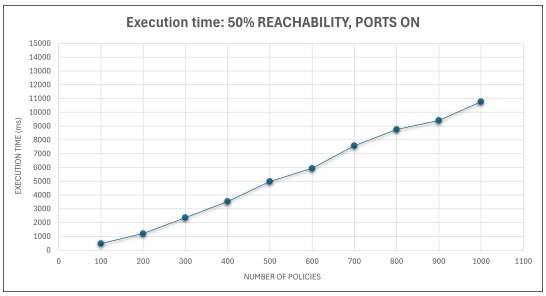

Figure 7.18: 50%, ports enabled, Extended Heuristic approach with MF

On the other hand, in the case with port usage enabled, using both AP and MF algorithms, Verefoo reaches 1000 policies. Moreover, it is important to notice that using an heuristic approach with MF algorithm, the difference between the port usage enabled and disabled is not so significant, while using the AP algorithm, the execution times are significantly higher when port usage is enabled, as shown in Figure 7.17 and Figure 7.18. In particular, using the AP algorithm, the execution time is about 8.5 times higher than using the MF algorithm, for 1000 policies. In addition, also the difference between the execution times with port usage enabled and disabled, using AP, is very high, since the execution time is about 7.5 times higher when port usage is enabled, for 1000 policies.

This final analysis shows that the inclusion of more complex policies, such as `CompleteReachability`, can impact the performance of Verefoo, but using a heuristic approach, the scalability is not affected significantly, and the execution times remain manageable, especially when using the MF algorithm.

Overall, the results of the validation phase demonstrate the effectiveness of the CESNET Test Generator in producing realistic and diverse test scenarios, as

well as the performance and scalability of Verefoo under various configurations. The analysis presented here provides insights into the strengths and weaknesses of different approaches, highlighting the importance of choosing the right algorithm and solver for specific use cases. In particular, it highlights the differences between the MaxSMT and heuristic approaches, as well as the impact of different Z3 solver versions, and also the difference in using the AP and MF algorithms, providing a comprehensive understanding of Verefoo's capabilities and limitations.

Chapter 8

Conclusions and Future Works

This final chapter analyzes the results obtained throughout the thesis work, summarizing the main contributions and findings. It presents the key aspects of the work, emphasizing their significance and implications, and ultimately outlines potential future research directions that could build upon the work presented.

The first contribution of this thesis is the design and integration of GreenShield in the Edge-to-Cloud (E2C) Continuum. This required adapting the original architecture of GreenShield to meet the specific requirements and constraints of the E2C scenario. The integration process included the introduction of new components, both at the node and firewall levels, through the inclusion of virtual nodes and the adaptation of the virtual firewall instances, in order to ensure multi-tenancy and scalability of the solution. The integration also required a careful analysis of the virtualization technologies power consumptions, to understand which technology would be more suitable for the E2C scenario, between containers and virtual machines. This analysis considered not only energy consumption but also the amount of resources required by each technology, as well as the trade-offs between the two aspects. The findings suggest that containers are generally more energy-efficient than virtual machines, due to their lower overhead and resource requirements, while virtual machines provide better isolation and security guarantees.

Furthermore, the integration process included the proposal of new hard and soft constraints to be included in the implementation of GreenShield, ensuring that firewall placement is appropriate for the E2C context. These constraints take into account factors such as the power consumption of single nodes, the existance of optimized solutions for the virtualization of the firewall instances, and the need to manage the balance between energy consumption and performance.

Finally, the thesis presents a use case to demonstrate the effectiveness of the proposed integration in a real-world scenario, specifically the application of the E2C to a smart city environment. It includes the deployment of various endpoints, which generate a significant amount of data that needs to be processed and analyzed in real-time. These endpoints are connected to edge nodes, which are responsible for processing and filtering the data before sending it to the cloud. At this stage, the GreenShield solution is applied to ensure that the network security policies are enforced in an energy-efficient manner. In addition, cloud containers are used to store and analyze the data, providing insights and information that can be used to

improve the services provided to the citizens. This use case illustrates how a smart city can integrate the E2C paradigm, including all its instances, from the edge nodes to the cloud data centers, and how they can help to improve the efficiency and sustainability of the city's operations.

In the second part of the thesis, the focus is on the implementation of a test generator, designed to create a set of test cases for evaluating the performance of Verefoo in a real-case network, such as the CESNET3 network. The test generator is designed to create a variety of test cases, based on the network topology and the specific requirements of the CESNET3. It is designed to generate a variety of test cases, with different characteristics, based on the four parameters: the number of policies, the percentage of reachability policies, the port usage, and the generation of bidirectional isolation policies. It is implemented according to a specific analysis of a dataset, i.e., the CESNET-TLS-Year22, containing real TLS traffic data from the CESNET3 network, captured during the year 2022. The analysis of this dataset provides insights into the traffic patterns and characteristics of the CESNET3 network, which are then used to design the generated test cases.

Finally, the last part of the thesis validates the generated test cases, by running them through Verefoo and analyzing the results. Validation is performed by configuring the previous parameters, and for each configuration running 30 iterations to compute the average execution time. These results are then compared to evaluate the performance of Verefoo in handling the generated test cases, in terms of execution time and scalability.

When running Verefoo with the MaxSMT approach to solve the verification problem, the results show that the execution time increases with the number of policies, and also using a lower percentage of reachability policies. This is due to the fact that reachability policies are easier to verify than isolation policies. Furthermore, the results also show that the execution time increases with the port usage, as more ports lead to a larger number of possible connections between policies, and so more combinations to verify. By contrast, running the MF algorithm, Verefoo is unable to find a solution in the majority of the test cases, except for a few cases with a low number of policies and high reachability percentage. Finally, running two different versions of the Z3 solver, i.e., v4.8.8 and v.4.14.1, the results show that the newer version of the solver performs better in terms of execution time, especially for larger test cases with a higher number of policies. In general, the newer version decreases the execution time by approximately 20% compared to the older version.

When running Verefoo with a heuristic approach to solve the verification problem, results show that the execution time is generally lower than the MaxSMT approach, and in the case of MF algorithm, it does not depends on the usage of ports. On the other hand, using the AP algorithm, the execution time increases with the port usage, but it is still lower than the MaxSMT approach. In addition, the heuristic approach shows a better scalability, being able to handle a larger number of policies without a significant increase in execution time, since Verefoo reaches 1000 policies in feasible time.

These findings suggest that the heuristic approach is more suitable for handling large-scale networks with a high number of policies, while the MaxSMT approach

may be more appropriate for smaller networks with a lower number of policies. Moreover, the results also highlight the importance of using an up-to-date solver, as it can significantly improve the performance of the verification process.

In conclusion, this thesis represents the starting point of a broader and more complex research path. As future work, it would be valuable to extend the test generator to GreenShield, in order to evaluate its performance not only in terms of execution time and scalability, but also regarding energy efficiency. Such an extension would provide a more comprehensive evaluation of GreenShield's behavior in real-world network scenarios, while also identifying opportunities for further improvement and optimization. At the same time, another interesting future direction would be to explore the analysis of the power consumptions of additional security functions, such as VPN Gateways. Extending the evaluation to multiple functions would allow GreenShield to be assessed not only in terms of firewall placement, but as a broader framework for orchestrating different security mechanisms with energy efficiency in mind.

Bibliography

[1] 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. 561–578, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3409073

[2] 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, 2023. [Online]. Available: https://doi.org/10.1145/3616401

[3] 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, Mar/Apr 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3160293

[4] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, "Autonomous Attack Mitigation Through Firewall Reconfiguration," *International Journal of Network Management*, vol. 35, no. 1, pp. 1–18, 2024. [Online]. Available: https://doi.org/10.1002/nem.2307

[5] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "A Looping Process for Cyberattack Mitigation," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024, pp. 276–281. [Online]. Available: https://doi.org/10.1109/CSR61664.2024.10679501

[6] D. Bringhenti, "Network Security Automation," Ph.D. dissertation, Graduate School of Politecnico di Torino, 2022.

[7] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "Atomizing Firewall Policies for Anomaly Analysis and Resolution," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 3, pp. 2308–2325, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3495230

[8] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Automatic and Optimized Firewall Reconfiguration," in *2024 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Seoul, South Korea: IEEE, 2024, pp. 1–9. [Online]. Available: https://doi.org/10.1109/NOMS59830.2024.10575212

[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, Dec. 2024. [Online]. Available: https://doi.org/10.1109/TNSM.2024.3452150

[10] 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, Aug 2024. [Online]. Available: https://doi.org/10.1109/TNSM.2024.3407159

[11] D. Bringhenti and F. Valenza, "A Twofold Model for VNF Embedding and Time-Sensitive Network Flow Scheduling," *IEEE Access*, vol. 10, pp. 44 384–44 398, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3169863

[12] A. Al-Dulaimy, M. Jansen, B. Johansson, A. Trivedi, A. Iosup, M. Ashjaei, A. Galletta, D. Kimovski, R. Prodan, K. Tserpes, G. Kousiouris, C. Giannakos, I. Brandic, N. Ali, A. B. Bondi, and A. V. Papadopoulos, "The computing continuum: From IoT to the cloud," *Internet of Things*, vol. 27, no. 101272, pp. 1–33, 2024. [Online]. Available: https://doi.org/10.1016/j.iot.2024.101272

[13] F. Firouzi, B. Farahani, and A. Marinšek, "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)," *Information Systems*, vol. 107, no. 101840, pp. 1–22, 2022. [Online]. Available: https://doi.org/10.1016/j.is.2021.101840

[14] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D'Andria, "A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues," *Future Internet*, vol. 15, no. 383, pp. 1–27, Nov 2023. [Online]. Available: https://doi.org/10.3390/fi15120383

[15] D. Khalyeyev, T. Bureš, and P. Hnětynka, "Towards Characterization of Edge-Cloud Continuum," *Software Architecture - ECSA 2022 Tracks and Workshops*, no. 13928, pp. 215–230, 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-36889-9_16

[16] R. Morabito, "Power Consumption of Virtualization Technologies: An Empirical Investigation," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 522–529. [Online]. Available: https://doi.org/10.1109/UCC.2015.93

[17] M. Warade, K. Lee, C. Ranaweera, and J.-G. Schneider, "Monitoring the Energy Consumption of Docker Containers," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 1703–1710. [Online]. Available: https://doi.org/10.1109/COMPSAC57700.2023.00263

[18] C. Xu, Z. Zhao, H. Wang, R. Shea, and J. Liu, "Energy Efficiency of Cloud Virtual Machines: From Traffic Pattern and CPU Affinity Perspectives," *IEEE Systems Journal*, vol. 11, no. 2, pp. 835–845, 2017. [Online]. Available: https://doi.org/10.1109/JSYST.2015.2429731

[19] R. Shea, H. Wang, and J. Liu, "Power Consumption of Virtual Machines with Network Transactions: Measurement and Improvements," in *Proceedings of IEEE INFOCOM 2014*. IEEE, 2014, pp. 1051–1059. [Online]. Available: https://doi.org/10.1109/INFOCOM.2014.6848035

[20] K. Hynek, J. Luxemburk, J. Pešek, T. Čejka, and P. Šiška, "CESNET-TLS-Year22: A year-spanning TLS network traffic dataset from backbone lines," *Scientific Data*, vol. 11, no. 1156, 2024. [Online]. Available: https://doi.org/10.1038/s41597-024-03927-4

[21] ipwhois.io, "IP Geolocation API and IP Location Lookup Tools," https://ipwhois.io/.

[22] ipinfo.io, "IP Data Intelligence for Developers & Enterprises," https://ipinfo.io/AS2852.