Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (35^{th}cycle)

Network Security Automation

Abstract

By

Daniele Bringhenti

******

Supervisor(s):
Prof. Riccardo Sisto, Supervisor
Dr. Fulvio Valenza, Co-Supervisor

Politecnico di Torino
2022

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Daniele Bringhenti
2022

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

I would like to thank my supervisors, Prof. Riccardo Sisto and Dr. Fulvio Valenza, for their constructive guidance, for their continuous support during my research work, for their availability in assisting me, for the opportunities they gave me, and their constant help to improve my research and successfully finalize this dissertation. I would also like to thank Prof. Guido Marchetto, who contributed to supervise my research work and provided me with useful suggestions to improve it. Finally, I would like to thank my mother for her daily support.

Abstract

In the latest years, softwarization paradigms such as Network Functions Virtualization and Software-Defined Networking shook the traditional vision of networking. The recent evolution of computer networks reflects the main driving force of these paradigms, based on high flexibility and dynamism. Network sizes and complexity are constantly increasing, because enriching a networked service with a new function only requires the deployment of a virtualized or containerized software function. To this regard, the advent of the Internet of Things paradigm led to a pervasive presence of computer and network communications in everyday activities, thus also increasing heterogeneity of applications that are connected to the network. Specifically considering cyber security, the characteristics of modern virtualized networks are no longer compliant with the traditional ways to enforce security. Traditionally, network security management was performed manually by human operators, with trial-and-error approaches where the security status was updated whenever a cyber attack was able to overcome the defenses provided by the previous status. However, such an approach can work only with small-sized and almost static networks, where everything is under the direct control of a human user. Continuing with a manual approach in modern virtualized networks would likely lead human operators to introduce vulnerabilities, which could be exploited to breach into the network, and sub-optimizations, which would reduce network efficiency.

A possible solution to limit this issue is to leverage automation in the approaches pursued for network security management. First of all, automation can contribute to minimizing the number of human interventions, as automatic approaches commonly require only input specifications and human assistance during their independent work. Moreover, automation favors the introduction of two important features, i.e., formal verification and optimization. On the one hand, formal verification can provide correctness assurance of the automatically computed management decisions, thus increasing human confidence in automatic approaches, where many tasks are not

under human control. On the other hand, optimization can improve the quality of management decisions. For example, for the task of configuring a network security function such as a firewall, if the configuration rule set is minimized, usually the function requires less time to process packets.

Unfortunately, despite all such benefits that automation could bring over to network security management, currently in the literature it has been successfully applied only to solve networking issues, and rarely to manage security ones.

Therefore, this dissertation aims to fill this literature gap by proposing novel approaches to improve the state of the art about network security automation. The main contributions of this dissertation are related to two main research areas: automatic security configuration and automatic security orchestration. For what concerns automatic security configuration, this dissertation proposes the VEREFOO approach, which solves the configuration problem of network security functions (e.g., firewalls and VPN gateways), by combining automation, formal verification, and optimization. VEREFOO follows the policy-based management paradigm: starting from a user-provided network topology with related security policies, VEREFOO computes the allocation of security functions in the network topology and their configuration rules. This result is computed in a fully automated way, it is formally guaranteed to satisfy all security policies, and it is optimized, including the minimum numbers of allocated functions and configuration rules. This result is achieved by formulating the configuration problem as a Maximum Satisfiability Modulo Theories problem, which enables pursuing correctness by construction and optimization. An extensive experimental evaluation shows not only that the proposed approach is feasible, but also that it scales to networks of significant size and that it provides better optimization than traditional configuration strategies. For what concerns automatic security orchestration, this dissertation addresses some of the open problems that should be solved in order to make this kind of orchestration reliable and efficient. First, an automatic methodology to optimize distributed firewall reconfiguration transients is proposed, in order to limit, as much as possible, the traversal of insecure transient states when new firewall allocation and configuration rules have to be deployed. Second, a novel approach is proposed for network security function selection in the orchestration workflow, based on a novel security function abstraction, which enables more optimized choices. Finally, the integration of the VEREFOO approach within state-of-the-art network orchestrators is discussed, also showing concrete integration examples. Experimental evaluation shows that these proposals can cooperate in close

synergy with orchestrators oriented to solve networking problems, thus representing an important step ahead towards full autonomy in network security.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer networks have been undergoing an incessant evolution since the beginning of the last decade. Network softwarization, declined in technologies such as *Software-Defined Networking* (SDN) [1] and *Network Functions Virtualization* (NFV) [2], simplified the network management operations, by decoupling control and data planes of the networked architecture, and by instantiating virtual functions instead of the manual installation of physical middleboxes. Agility and dynamism soon became the key terms of these revolutionary network management paradigms. As a consequence, the size of modern computer networks is constantly increasing, because of the tendency to virtualize every activity and give it access to a network. Besides, the employed functions are becoming more heterogenous among them, and they are usually more complex than the old corresponding physical middleboxes, as the software that embeds their operations is often parallelized. All these trends are confirmed by the characteristics of modern industrial networks, and also by the success of the emerging *Internet of Things* (IoT) paradigm [3], based on the idea that all the devices, despite their differences in terms of functionalities and objectives, can and should be connected to the network. Another environment where these trends are evident is the Time Sensitive SDN, where strict requirements in terms of latency and bandwidth must be respected for the scheduling of end-to-end communications [4].

Nevertheless, the introduction of new advantages is typically accompanied by the presence of inevitable drawbacks. As computer networks are becoming bigger and more complex, new opportunities have arisen for cyber attackers to intrude on them,

and the number of breaches dramatically increased. Unfortunately, the aforementioned agility and dynamism could not directly benefit network security management. The main reason is that security management is an activity that traditionally used to be performed manually with a trial-and-error approach. Administrators used to configure *Network Security Functions* (NSFs) such as firewalls or intrusion detection systems according to their initial expectation of possible attacks. If later a cyber attack had occurred, they would have simply modified the behavior of the function that could not block it, so as to avoid a possible repetition. However, such an approach could work only with small-sized networks, where everything was almost static and under the direct control of a human user, and where all the network accesses could be easily known. Instead, softwarized networks have opposite characteristics, i.e., big size, heterogeneity, dynamicity, and complexity [5].

For these reasons, automation has been proposed as a possible solution to this urgent pending problem. The idea is that, if human users cannot cope with the security management of the whole network by themselves, they can be assisted by automated tools or frameworks, in charge of replacing the traditional manual security operations. The methodologies that were proposed for the development of these automated tools were often based on the *Policy-Based Management* (PBM) paradigm [6]. According to the original definition proposed by D. Clark and D. Wilson, "a security policy specifies the security goals that the system must meet and the threats it must resist. For example, the high-level security goals most often specify that the system should prevent unauthorized disclosure or theft of information, should prevent unauthorized modification of information, and should prevent denial of service" [7]. In a PBM-based approach, administrators should simply define the security requirements by means of a set of business-level statements, the security policies, which are commonly expressed in natural languages, so as to guarantee high usability and user-friendliness. Then, the policies could be automatically transformed into low-level security management operations (e.g., the decision of which NSFs should be allocated in the network, and where, and the generation o their low-level configuration) through an operation named policy refinement [8].

Even if automation brings over great opportunities for network security, the state-of-the-art approaches that have been proposed in literature still have several shortcomings. In particular, two features that could enhance security automation but that are rarely included are formal verification and optimization. On the one hand, providing formal assurance that the results computed by an automated tool are really

correct may be essential for the security of safety-critical systems. On the other hand, optimizing those same results may improve network security efficiency (e.g., if a firewall has only the minimum number of rules that are really required to enforce all the security policies specified by the security manager, then its filtering operations take less time because fewer comparisons between rule conditions and packet fields need to be performed). Despite the relevance of these two features, several state-of-the-art approaches do not exploit them, because their introduction is considered too challenging. Most notably, the performance and scalability of automated tools is severely reduced by a naive introduction of formal verification and optimization mechanisms, making them impractical unless smart ways of introducing them are found.

In light of these motivations, this dissertation faces the challenge of investigating and formulating automated, fast, and provably correct techniques for the security configuration and orchestration of NSFs, with the final aim of improving the dependability and resilience of next-generation computer networks to cyber attacks. In fact, a central objective of this dissertation is to propose the first security management approach in literature to combine full automation, formal verification and optimization. In the definition of such an approach, a first challenge has been to define formal models of modern virtualized networks, such that they capture all the required information for automated security orchestration, without impacting on performance excessively. Another challenge has been pursuing "security by construction" by means of lightweight correctness-by-construction approaches, where automated solvers can find a solution to the security orchestration problem that does not require a traditional a-posteriori formal verification step, and, at the same time, fulfilling optimality criteria (e.g., to improve the efficiency of the security operations and to minimize resource consumption).

1.1 Dissertation structure

The reminder of this doctoral dissertation is divided into three main parts:

1. *Motivation and State of the Art*

   The first part of this dissertation describes the motivation of this study and the related state of the art, whose knowledge is required to understand how the

proposed automation methodologies can solve existing problems for network security management, and how they fill gaps that are still present in the network security automation literature. This first part is divided into two chapters.

Chapter 2 describes the limitations of the traditional manual network security management operations, explaining why they cannot be applied anymore to modern virtualized computer networks. It also discusses the introduction of automation for network security, and its consequent benefits.

Chapter 3 describes the state-of-the-art approaches for network security automation, with an emphasis on those that are applied to solve two common security management tasks, i.e., security service composition and NSF configuration. In this discussion, the limitations of these approaches are underlined, so as to motivate why further research work is required in the automation of these security management tasks.

2. *Automatic Network Security Configuration*

   The second part of this dissertation proposes an automated approach for the allocation and configuration of NSFs. It shows that the proposed approach is flexible enough to be applied to different types of NSFs. Moreover, the proposed approach is compared to the state of the art approaches to show it is the first approach in literature that combines the three features of full automation, formal verification, and optimization. This second part is divided into three chapters.

   Chapter 4 describes the approach that has been followed for the definition of the proposed methodology for NSF configuration automation, named *VErified REFinement and Optimized Orchestration* (VEREFOO). This approach formulates the auto-configuration problem as a *Maximum Satisfiability Modulo Theories* (MaxSMT) problem through constraint programming, with the aim to provide simultaneously formal correctness by construction and optimization. This chapter also defines the inputs to be specified by human users, and the expected outputs.

   Chapter 5 discusses the application of the VEREFOO approach to the most commonly used NSF for the enforcement of connectivity security policies, i.e., the packet filtering firewall. It also provides the complete formalization of the network components (e.g., topology, network functions, and traffic flows) and

of the security policies, as they represent the basis for the formulation of the constraints composing the MaxSMT problem.

Chapter 6 discusses the application of the VEREFOO approach to VPN gateways, i.e., NSFs that are commonly used for enforcing security properties such as authentication, integrity and confidentiality in some portions of the network. It illustrates the main differences, in terms of formal models and MaxSMT problem constraints, with respect to the problem formulation for firewall configuration.

This second part of the dissertation is partly based on the content of the following papers:

- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Serena Spinoso, Fulvio Valenza, and Jalolliddin Yusupov. *Improving the formal verification of reachability policies in virtualized networks*. In: IEEE Transactions on Network and Service Management, March 2021, vol. 213, issue 1, pp. 713-728. doi: 10.1109/ TNSM.2020.3045781 [9].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. *Automated optimal firewall orchestration and configuration in virtualized networks*. In: IEEE/IFIP Network Operations and Management Symposium (NOMS) 2020, Budapest, Hungary, April 20-24, 2020, pp. 1-7. doi: 10.1109/NOMS47738.2020.9110402 [10].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. *Automated firewall configuration in virtual networks*. In: IEEE Transactions on Dependable and Secure Computing, in press. doi: 10.1109/ TDSC.2022.3160293 [11].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza. *Short Paper: Automatic Configuration for an Optimal Channel Protection in Virtualized Networks*. In: Proc. of the 2nd Workshop on Cyber-Security Arms Race, co-located with ACM CCS 2020, November 9, 2020, pp. 25-30. doi: https://doi.org/10.1145/3411505.3418439 [12].

3. *Automatic Network Security Orchestration*

The third part of this dissertation discusses automatic approaches for solving problems related to network security orchestration. Differently from the allocation and configuration problem, security orchestration is a more general

problem that also encompasses tasks such as the selection of the NSFs that must be used to enforce the required security policies, the interaction with a network orchestrator to instantiate the virtual functions, or the management of transients that occur when the security status of the network is updated after the detection of an attack. This third part is divided into three chapters.

- **Chapter 7** presents an automated methodology named *FirewAll Transients Optimizer* (FATO), which is in charge of establishing the optimal scheduling of the reconfiguration changes for a distributed packet filtering firewall that needs to be updated. This methodology applies formal methods to solve this orchestration problem, so as to ensure that the identified scheduling really minimizes the number of transient states where the system is still vulnerable.
- **Chapter 8** presents a novel NSF abstraction, named projection abstraction, which aims to represent the NSFs in a way that is independent from the differences that are only related to the vendor-dependent implementation choices. Such abstraction can improve security orchestration, by allowing NSF selection to be postponed and done jointly with the deployment operation, after the configuration of the virtual security service.
- **Chapter 9** describes the integration of the VEREFOO approach with state-of-the-art orchestrators for network virtualization, such as Docker Compose and Kubernetes, and a demonstrator related to an IoT-aware smart city network. This integration enables a full orchestration of a security service, and it has been carried out in the context of projects funded by the European Union, such as ASTRID[1] and CyberSec4Europe[2].

This third part of the dissertation is partly based on the content of the following papers:

- Daniele Bringhenti and Fulvio Valenza. *Optimizing distributed firewall reconfiguration transients*. In: Computer Networks, October 2022, 109183. doi: 10.1016/ j.comnet.2022.109183 [13].
- Daniele Bringhenti, Jalolliddin Yusupov, Alejandro Molina Zarca, Fulvio Valenza, Riccardo Sisto, Jorge Bernal Bernabé, and Antonio F. Skarmeta.

[1]Link: https://www.astrid-project.eu/. Last accessed: October 18th, 2022.
[2]Link: https://cybersec4europe.eu/. Last accessed: October 18th, 2022.

Automatic, verifiable and optimized policy-based security enforcement for SDN-aware IoT networks. In: Computer Networks, August 2022, vol. 213, pp. 109-123. doi: 10.1016/j.comnet.2022.109123 [14].

- Daniele Bringhenti, Fulvio Valenza, and Cataldo Basile. *Toward Cybersecurity Personalization in Smart Homes*. In: IEEE Security & Privacy, January-February 2022, vol. 20, pp. 45-53. doi: 10.1109/MSEC.2021.3117471 [15].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza. *A novel approach for security function graph configuration and deployment*. In: IEEE 7th International Conference on Network Softwarization (NetSoft) 2021, June 28 - July 2, 2021, pp. 457-463. doi: 10.1109/NetSoft51509.2021.9492654 [16].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. *Introducing programmability and automation in the synthesis of virtual firewall ruleset*. In: IEEE 6th International Conference on Network Softwarization (NetSoft) 2020, June 29 - July 3, 2020, pp. 473-478. doi: 10.1109/NetSoft48620.2020.9165434 [17].
- Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. *Towards a fully automated and optimized network security functions orchestration*. In: IEEE International Conference on Computing Communication and Security (ICCCS 2019), October 10-12, 2020, doi: 10.1109/CCCS.2019.8888130 [18].

Finally, **Chapter 10** concludes the dissertation, and it illustrates possible future work for each research line that has been presented in this document.

Motivation and State of the Art

Chapter 2

Motivation and Problem Statement

Managing the security of a virtual computer network encompasses multiple operations. The two most common and well-known ones are security configuration and orchestration. On the one hand, configuring the security involves synthesizing the structure of the security service at the logical level, so as to be compliant with the topology of the computer network, and establishing their behavior (e.g., determining the filtering rules to be installed on a firewall, or the Security Associations of a VPN). On the other hand, orchestrating the security involves operations such as the selection of the NSFs that must be later configured, the interaction with a network orchestrator (e.g., Open Source MANO) for their deployment, and the application of changes to an already defined security service (e.g., during a reconfiguration transient). Section 2.1 describes the limitations of these operations, if they are performed manually by a human user. Section 2.2 discusses the advantages that, instead, would be bought over by introducing automation. Then, Section 2.3 presents the common workflow that is followed to automate the security management of a computer network.

2.1 Limitations of manual network security management

For most computer networks, the security manager is commonly the person in charge of collecting the security requirements formulated by network users, and using them to manually allocate and configure all the NSFs that are necessary to enforce them.

For example, if there is the requirement to block all the traffic directed to a specific website, the security manager must include the corresponding filtering rule in a firewall. Nevertheless, such a manual operation has always been more complex than the configuration of non-security-related network functions, also because any issue occurring in this task may have serious consequences for the users of the networks, who may be put under attack.

The consequence of a manual configuration that is not performed es expected is the accidental generation of anomalies. In the literature, an anomaly is defined as an incorrect specification that the security manager might introduce in the configuration of an NSF. Anomalies can be classified into three different categories [19]: conflicts (e.g., two filtering rules of a firewall have the same condition set, but they enforce contradictory actions), errors (e.g., the Security Association of a VPN gateway is configured so as to enforce a specific cryptographic algorithm for the encryption of the packet payload, but the VPN gateway on which it is installed does not support that algorithm among its cipher suites), or sub-optimizations (e.g., some rules of a firewall are never triggered, if other rules with higher priority have a condition set that includes theirs). The analysis of anomalies that can afflict security configuration has been extensively discussed in the literature. For instances, the classification proposed in [20, 21] for the anomalies related to filtering NSFs such as firewalls envisions five intra-firewall anomaly types (shadowing, correlation, generalization, redundancy, irrelevance) and four inter-firewall anomaly types (shadowing, spuriousness, redundancy, correlation). Similarly, for NSFs that are employed for enforcing security properties as confidentiality and integrity, such as VPN gateways, a complex taxonomy composed of five macro-categories (insecure communications, unfeasible communications, potential errors, suboptimal implementations, suboptimal walks) has been detailed in [22].

The exacerbation of the problem of anomalies in the configuration of NSFs has been also highlighted by the Verizon company in its annual Data Breach Investigations Report. In particular, misconfiguration is included inside the miscellaneous error macro-category for security breaches. Analyzing the Verizon reports from 2013 to 2022[1], inside that category, the percentage covered by misconfiguration has constantly increased, till reaching 46%, as depicted in Fig. 2.1. This pattern is particularly marked between the years 2016 and 2017 and the years 2018 and

[1]The reports are available at the following link: https://enterprise.verizon.com/resources/reports/dbir/. In the chart, the data related to a specific year are extracted from the report dated the following year.

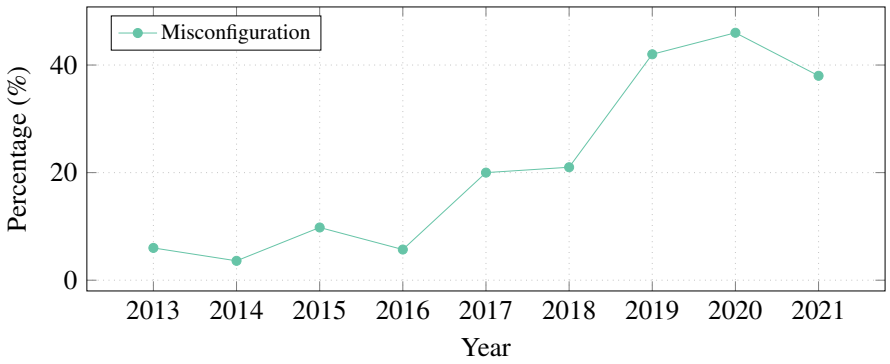


Fig. 2.1 Percentage of misconfiguration errors in the miscellaneous error category

2019, where the relative increments of breaches related to misconfiguration are larger than 50%. In 2021, Verizon noted a decrease in the percentage covered by misconfiguration, but it is not yet possible to make any assumption about a potential decreasing trend, as at the moment the 2023 Data Breach Investigations Report is not yet available. Anyhow, it is also possible to infer how heavy the misconfiguration problem is from other two observations by Verizon. In the 2020 Report, they state that misconfiguration has become the most critical cause of breaches for cyber security attacks, overtaking two other widespread causes that had always been ahead of it, i.e., misdelivery and publishing errors. Then, in the 2022 Report, they also claim that the human element represents a central factor driving breaches, as 82% of breaches involved the human element in 2021. Fig. 2.2 shows the impact of this statement with a graphic representation, where each glyph represents 25 breaches.

Not only configuration, but also security orchestration is afflicted by problems related to manual operations. For example, when a transient unfolds from a security status of the network to another one, the changes should be applied to the network as fast as possible, and they should be orchestrated in a way that minimizes the number of intermediate states where security may be undermined. However, human users cannot be as fast as necessary to manage a security transient, and they are likely to make some mistakes, including oversights due to the time constraints under which they must work. Similarly, if a user has to manually introduce the established security configuration into a network orchestrator, some wrong settings may be inadvertently introduced even if the configuration is theoretically correct.

Here is a detailed list of all the main reasons why manual security configuration and orchestration has become unbearable in modern computer networks:

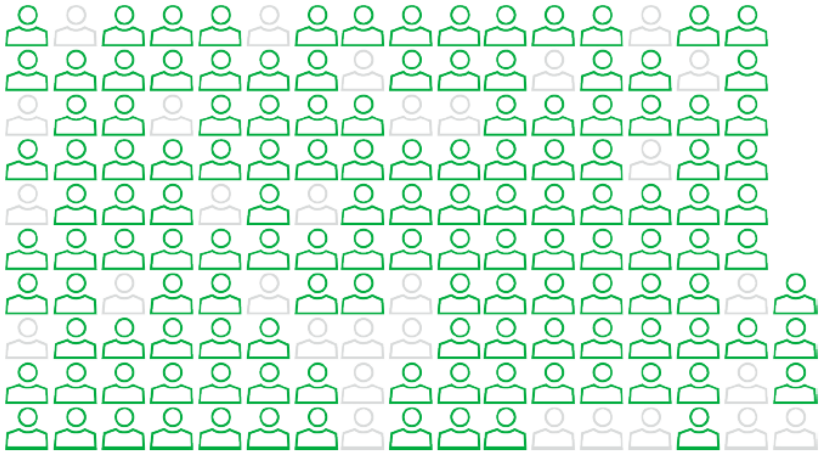


Fig. 2.2 The impact of the human element in breaches (figure derived from the 2022 Verizon Data Breach Investigations Report)

- **Role separation and lack of communication.** In most companies, network administrator and security manager are separate roles, even if both of them pertain to the same working area (i.e., the computer network where the former must ensure the correct behavior of the communications, whereas the latter must ensure the security of them). Consequently, it is not uncommon that the two people who fulfill those roles do not often communicate with each other. At that point, as each one of them does not have full knowledge about the expertise area of the other one, some trivial mistakes, but with potentially serious consequences for network security, may be made [23]. For instance, if the security manager is not properly informed by the network administrator about the current structure and setting of the network, the former may make incorrect assumptions when starting to design the security architecture.

- **Low frequency of updates for security managers**. Security managers are often not updated about new cyber security attacks or protection solutions. This trend has started to change only in the last years: the percentage of business companies that never update their managers has fallen from 26% to 17%, according to a study of the Ipsos MORI institute, in collaboration with the University of Portsmouth [24]. However, this percentage value is not low enough to state that the presence of non updated managers has no impact on the happened breaches.

- **Increasing network size**. Following the IoT trend, the number of devices and services that are interconnected to each other is constantly increasing. Besides, also more traditional services such as e-mails, in-app communications, video streaming or phone calls are nowadays more frequent than ever, as more and more people are using the Internet. The size of modern computer networks had to adapt to these numbers, and it is accordingly becoming bigger. According to a study performed by Oracle Communications [25], 76% of the interviewed companies state that the breadth of their network is expanding, with the prevision that in the future this trend will not be reversed. However, the presence of more communication channels brings a higher possibility of vulnerabilities that an attacker can exploit.

- **Increasing network complexity**. According to the KISS[2] rule, originally introduced by the U.S. Navy in 1960, complexity is the worst enemy of security. The application of this design principle in the context of network security would be in fact fundamental, since it would ease network security management by making it more intuitive. Currently, however, new kinds of attacks are emerging, and the consequence is that the complexity of the security functions is growing as a reaction. For example, firewalls are now often able to filter packets at different levels of the ISO/OSI stack, while in the past they mostly worked at levels 3 and 4. Moreover, elaborated artificial intelligence algorithms are being introduced in several devices, as the original detection algorithms are deemed not enough to face the current attacks. Guaranteeing that the result of manual security management operations is correct is quite a difficult objective to achieve under these circumstances. In fact, even if these complex functions and algorithms solve some problems, their complexity hides some new inevitable vulnerabilities that other attackers may exploit.

- **Increasing network heterogeneity**. In modern computer networks, functions of the same type may be present with implementations made by different vendors, by following the multi-vendor design principle. This heterogeneity has been even more common after the advent of network softwarization, as creating a new function implementation means writing the corresponding code and instantiating it as a Virtual Machine or Docker, instead of designing and producing the corresponding hardware middlebox. However, heterogeneous

[2]The acronym KISS stands for "Keep it simple, stupid".

networks are more prone to attacks, because each implementation may have different vulnerabilities [26]. Besides, configuration errors are more common. For example, if firewalls produced by different companies are installed in a network, their configuration would require the human user to specify the filtering rules with different languages according to their vendor-dependent implementation.

- **Trial-and-error configuration approaches**. Manual security management is typically based on a trial-and-error approach. Every time an attack is detected, the configuration of some network security function is altered in order to block it. If this approach lets security managers save time in the short term, in the long term it leads to longer and longer (and more complex) configuration files, also increasing the possibility of introducing errors.

- **Economic impact of breaches**. In addition to the technical problems that have been presented, economic concerns must be taken into account when evaluating the impact of a manual configuration of any network security service. This aspect can be quantified by esteeming the cost of each single breach. In this regard, the Ponemon Institute conducted a study [27], sponsored by IBM Security, according to which the average total cost of a data breach is equal to $3.92M, with a cost per lost record of $150 and the average time span needed to identify and contain a breach equal to 279 days (more than half a year). These numbers are clearly impressive and cannot be accepted by a company, notwithstanding the consequent reputation damage, which could be even more problematic in some sectors.

2.2 Advantages of automatic network security management

According to the definition proposed in [28], automation 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". A central objective of introducing automation for the management of a system is to minimize the number of operations manually performed by human users. A system whose behavior is automated would

just require the specifications of some external inputs from humans or from other systems, and then it would be able to continue to work without other external assistance. Automating systems requires careful planning during their design. However, if this phase is carefully managed, then both the productivity and quality of the work produced by the system can be drastically improved. On the one hand, human users are not required anymore to spend most of their time to manually perform all the operations, but they can delegate them to an automated system and just provide it with assistance or maintenance. In this way, they can also manage multiple systems at the same time. On the other hand, automated systems commonly complete their work faster, ensuring better accuracy, as they are not affected by the common human oversights.

Nowadays, automation has been successfully enforced in several engineering fields. For example, robotic process automation is a central aspect of business process automation and it is enriched by machine learning algorithms which allow the machines to adapt to new emerging problems without requiring human help.

In the wake of this trend, automation has been starting to be introduced in network security as well, so as to reduce the number of human operations related to network security management, and consequently to reduce the vulnerabilities that are exposed to attackers. The fact that automation has already been introduced or planned in security management and that it helped in facing cyber attacks can be derived from the analysis of some studies, based on interviews conducted among security technicians.

- In the survey carried out by the Deloitte company [29], security orchestration and automation is classified as the top-ranked cyber defense priority and investment area, being chosen by 20% among total participants.
- Among the organizations interviewed by Marsh for a study sponsored by Microsoft [30], 59% believe that automation is already used or it is planned in the range of novel technologies which will be soon introduced. Additionally, only 8% feel that the degree of risks that could be posed by the introduction of automation is very high.
- According to the research insights report by Enterprise Strategy Group [31], 84% of the interviewed people agree that automation will help to minimize security misconfigurations caused by manual inputs, such as security rule

contention. Moreover, 86% think that automation will enable the operations team to do more with existing resources, a critical need since 51% of the organizations report a problematic shortage of cyber security skills [32].

- Another Ponemon Institute's research [33] shows that 72% of the respondents state that automation, after being introduced in their organizations, improved cyber resilience and the ability to prevent, detect, contain or respond to a cyber attack. Moreover, their companies were less prone to cybersecurity incidents: only 49% among the organizations that invested in automation had more than one data breach.

From the analyses that have been reported beforehand, a clear indication is that the network security field is shifting towards automation, after that it has been successfully enforced in other areas. Even though automation is currently still not exploited by all the companies (according to [27], only 52% of the analyzed companies already have security automation partially or fully deployed), nevertheless a consistent percentage is evaluating this feature for the immediate future.

In fact, automation can overcome most of the limitations which have been dissected in Section 2.1.

First of all, a high level of expertise or experience in network security is not required, if human users are assisted by automated tool for security configuration and orchestration. Traditionally, many companies have a limited number of employees that have a deep security background, because of the high salaries that their experience is worth of. As a consequence, they use these experts to supervise other ones, who mostly have a networking background and who are delegated to manually perform all operations, also the ones that are related to security management. In such a context, miscommunication between these two groups or the lack of security expertise of the networking experts commonly lead to errors. Instead, if security management operations are automated, human users should just monitor the tools that perform the automatic operations, and this task is evidently much less complex than manually performing the operations themselves. Moreover, the monitoring activity would be less error-prone and time expensive than the manual management of all the security functions that are present in a computer network, so that work time can be exploited more efficaciously.

Then, automation allows managing the big size and heterogeneity of modern computer networks (e.g., virtual or IoT-based network) in a more efficient way than what a human user may do manually. On the one hand, an automated tool for security management can take decisions that are coherent for all the components of a network, because it has a complete overview of the whole network architecture. A potential problem may be the time which could be required in this case, but it is always inferior to the time needed in a manual security management operation. On the other hand, the heterogeneity of the different implementations of the same security function can be abstracted, so that the automated tool initially considers all of them as the same functionality, and only later it adapts the produced result (e.g., their configuration) to the correct vendor-dependent commands to set up the specific device. This final translation would hardly be performed manually because the human user in charge of it should have a deep knowledge of the language of each different function implementation. Instead, an automated process could be built with all the required information, so as to perform this operation faster and with a higher assurance of its correctness.

Furthermore, optimization can be more easily applied to security management, if it is paired with automation. In the networking field, an optimization problem that has been extensively investigated in literature is the minimization of the resource consumption related to the deployment of virtual functions on the general-purpose servers composing the physical network in an NFV architecture. However, optimizing security could be essential as well. Maximizing the defense effect of the protections which are set up against the potential cyber attacks would heavily decrease the number of breaches. Achieving this result manually would be extremely difficult, since correctness itself is already a hard achievement in manual operations.

One may argue that, even if automation carries over several advantages to network security management, it also brings some possible drawbacks. However, it is reasonably possible to claim that most of these disadvantages are only apparent and mostly depend on human prejudices. According to the information provided in the survey carried out by EGS [31], 31% of the interviewed security managers strongly think that automation will increase their organizational risk. Furthermore, 26% also think that automation will reduce the level of control over network security. However, the main problem is not technical, but related to the psychological field. When humans do not have full control over or knowledge of what they are using, they start to fear it, as they think it may lead to even bigger problems. As a consequence,

automation has often been deemed as potentially dangerous in history. However, this prejudice can be easily demystified. On the one hand, as beforehand stated, automation can be paired with formal verification techniques, so as to provide correctness guarantee for the produced results. On the other hand, each automated tool should have a complete documentation explaining how it should be used. Finally, if there is a problem with automation, it is not the "over-automation", but oversights or mistakes made by humans in the design of the automated tools, or in the supervisor of their work [34]. Also in this case, the problems that are introduced are indirectly related to an activity carried out by people.

Summing up, all the motivations that have been discussed underline the importance of automating network security, and the reported studies also highlight how this process has already started.

2.3 Common workflow of automated network security management

Multiple alternatives exist to automate the security management of a virtual computer network. However, a common solution is to follow the *Policy-Based Management* (PBM) approach. In such approach, security managers are not required to manually manage each function in the network service, but they can specify their requested behavior with policies, which are later automatically refined into the function concrete configuration [8].

The process to cast this approach into the context of virtual network security management can be organized in different ways. However, from an investigation of the literature, which will be further discussed in Chapter 3, it is possible to identify some common phases. A possible workflow that can be defined to include all these phases is the one represented in Fig. 2.3. It is composed of multiple tasks, most of which can be grouped into the two main security operations (i.e., network security configuration and orchestration) and which will be described in the remainder of this chapter.

Policy specification. A human user specifies the security requirements that must be enforced in the computer network by means of policies, i..e, sentences expressed with a user-friendly high-level language, that does not require high expertise to

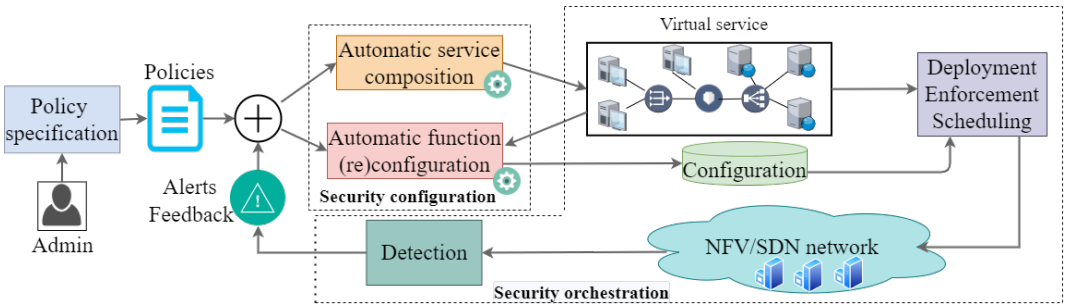


Fig. 2.3 A workflow for automated network security management

be learned. These policies must convey all the information that is required by the automated tool to establish the security of the network, such as the description of the topology, the current configuration of the network functions, and the characteristics of the traffic flows or communications that must be allowed, blocked or protected. In the ideal case, this phase is the only one that requires a direct human intervention, whereas simple assistance is enough for the other ones. During policy specification, human users may employ tools that are specifically designed for policy analysis, i.e., to identify anomalies introduced in the policy set. Examples are [20] for firewalls, and [22] for VPN gateways. Policy analysis may thus help to identify possible errors or sub-optimizations in the definition of policies.

Automatic service composition and function configuration. The user-specified policies are refined into the security configuration of the network. This process is composed of two operations: automatic service composition and automatic function configuration.

- *Automatic service composition* consists in designing the virtual service, which must operate on the logical topology representing the interconnection of the network functions (e.g., NATs, load balancers, switches) , commonly called *Service Function Graph* (SFG), or more simply *Service Graph* (SG). This operation is prone to manual errors or sub-optimizations, because a SG is characterized by a high number of access points from which communications can start, and these access points themselves can change over time. Besides, each traffic flow can potentially follow multiple paths. By leveraging automation, these issues need not to be directly addressed by human users, and all decisions can be taken automatically, so as to optimize security parameters.

- *Automatic function configuration* consists in generating the concrete configuration of each security function composing the security service built on the top of the SG. Each rule composing their configuration must be established by taking into account the policies specified by the human user, and also the positioning of the other network functions in the SG (e.g., if it has been decided that the firewall that blocks a certain traffic is located after a NAT which modified that traffic, then this transformation must be considered). This policy refinement task can be organized in multiple stages, e.g., initially the policies are refined into configuration rules expressed with a medium-level language, without any reference to a concrete function implementation, and then a final translation (into the enforcement stage of the workflow) may generate the low-level vendor-specific configuration. This task is also the most critical one in the whole workflow, because most of the breaches are made possible by NSFs configuration errors. It is the most complex operation as well, because it requires careful optimization (e.g., minimizing the number of filtering rules of a firewall may drastically improve its efficiency).

These two steps may be executed sequentially, or simultaneously in a more optimized way. The result is a virtual service, with the corresponding configuration for each function that composes it.

Deployment, enforcement and scheduling. Next, the virtual service must be embedded onto the physical network infrastructure, commonly composed of general-purpose commodity servers. This problem is known as *Virtual Network Embedding* (VNE), and it has been exhaustively researched in literature [35]. This operation is usually performed jointly with other two tasks, i.e., enforcement and scheduling. On the one hand, the configurations that have been generated at the previous stage of the workflow are enforced onto the corresponding NSFs, with a simple change of format, so as to make them adapted to the vendor-specific implementation. On the other hand, the execution of the embedded virtual functions should be scheduled so as to respect all dependencies. The three tasks composing this stage of the workflow almost exclusively deal with network optimization (e.g., the minimization of resource consumption), instead of security. An aspect that concerns security is related to the interaction or integration of the tool in charge of automatic service composition and function configuration with the orchestrator of the network embedding, as their communication should convey all the information computed in the previous stage.

Detection. Even if the security service is already active after its embedding, complete protection against all attacks is never possible. On the one hand, it may occur that the user-specified policies were not complete enough, and that they did not take into account some corner cases. On the other hand, new attacks are developed every day, and they cannot be easily foreseen. Consequently, it is necessary to install some intrusion detection systems in the network, so as to find out unstopped attacks. The literature about intrusion detection is wide [36]. Some systems have a list of attack signatures, which they use to identify malicious traffic, whereas other ones adopt machine learning strategies, which offer a better response against unknown attack types.

Mitigation and reconfiguration. However, identifying an ongoing attack is not enough. It is also essential to stop it as soon as possible. The information derived from the attack identification performed in the detection stage is used for the formulation of new policies. At that point, the whole workflow must be repeated, so as to reconfigure the security service. The way this task is carried out has a deep impact on security. The configuration of multiple functions may be subject to modifications, and the order in which these changes are orchestrated should minimize the number of transitory states in which the network is still insecure. At the same time, also identifying the minimum number of rules which should be changed in each function may allow improving the performance of this operation, and thus enforcing the required security faster. Unfortunately, orchestrating the reaction to an attack is often more challenging than simply configuring the service from scratch.

In this workflow, it is possible to identify which tasks are covered by the two main operations related to network security automation. On the one hand, network security configuration automation encompasses the automatic security service composition and function configuration. On the other hand, network security orchestration automation encompasses tasks such as the interaction or integration between the automatic configurator and the network orchestrator, the reconfiguration transient management after an attack identification, and the selection of the virtual functions that must be embedded to enforce the security configuration. All these tasks are addressed in the remainder of this dissertation.

Chapter 3

State of the Art

This chapter presents how the problems of automatic network security configuration and orchestration have been addressed so far in the literature. While discussing the approaches that have already been proposed, the open challenges that should still be faced in this research area are dissected. Section 3.1 discusses how the literature review has been performed. Then, Section 3.2 presents the literature about automatic network security configuration, whereas Section 3.3 provides an overview of the literature related to some central tasks of automatic network security orchestration.

3.1 Literature review method

The search process of conference proceedings and journal papers was carried out in the following databases: SCOPUS, Science@Direct, Wiley InterScience, IEEE Digital Library, ACM Digital Library, SPRINGER, ISI Web of Knowledge.

Five search strings have been used in the search engine of the previously listed databases, each one related to a specific problem to be investigated.

- The search string related to automatic security service composition is:

  computer AND (network OR networking) AND security AND (automation OR automatic OR automated OR programmability OR programmable) AND (synthesis OR synthesize OR composition OR compose)

- The search string related to automatic security function configuration is:

  computer AND (network OR networking) AND security AND (automation OR automatic OR automated OR programmability OR programmable) AND (configuration OR configure)

- The search string related to the orchestration of network security reconfiguration transients is:

  computer AND (network OR networking) AND security AND (automation OR automatic OR automated OR programmability OR programmable) AND (reconfiguration OR reconfigure) AND (transient OR transitory)

- The search string related to the NSF abstraction for security orchestration is:

  computer AND (network OR networking) AND security AND (automation OR automatic OR automated OR programmability OR programmable) AND (orchestration OR orchestrate) AND abstraction AND (implementation OR vendor)

- The search string related to the integration of security and network orchestrators:

  computer AND (network OR networking) AND security AND (automation OR automatic OR automated OR programmability OR programmable) AND (orchestration platform OR orchestrator) AND (NFV OR SDN)

The results have been enriched with the snowballing technique, i.e., for each study, its references and the papers citing it have been analyzed. Then, all enriched search results have been merged by fulfilling the following criteria.

1. *Impurity and duplicates removal*: Duplicate results were removed.

2. *Inclusion criteria*: Papers were considered if they respected all the following criteria: (1) Papers describing methodologies which can be used for network

security management automation; (2) Papers published between 1995 and 2022; (3) Papers subject to peer review (e.g., journal papers, papers published as part of conference proceedings were considered, whereas white papers were discarded); (4) Papers written in English and available in full-text.

3. *Exclusion criteria*: Papers were excluded if they fulfilled at least one of the following criteria: (1) Papers describing methodologies only for network management automation, without any reference to network security; (2) Papers limited to present a formal theory for networking, without any substantial possible application to computer networks; (3) Secondary studies (e.g., systematic literature reviews, surveys); (4) Studies in the form of tutorial papers, poster papers, editorials, because they do not provide enough information due to page limitation.

4. *Combination*: If there are multiple papers related to the same study, a single record is kept for all of them. This action is necessary for ensuring completeness and traceability of results. For example: if a primary study is published in more than one paper (a conference paper, then extended to a journal version), only one instance is counted as a primary study. Generally, the journal version is preferred, since more complete.

3.2 Automatic network security configuration

This section is divided into two subsections that discuss the state of the art related to the two main tasks of automatic network security configuration, i.e., automatic security service composition and automatic security function configuration.

3.2.1 Automatic security service composition

Most of the papers challenging the problem of service composition address networking services, rather than security ones. Here, only the papers dealing with security services are considered. A comprehensive list of such papers is reported in Table 3.2. This table also provides, in its columns, a fast overview of the main features of the approaches presented in these papers. The meaning of such features is concisely described in Table 3.1.

Reference: The reference to the paper where the automated methodology is illustrated in detail.
Target: The network type for which the methodology is designed and validated (i.e., traditional, virtual or both).
Fixing: True (✓) if the methodology can automatically fix a security service or NSF configuration, false (X) otherwise.
Scratch: True (✓) if the methodology can automatically create a service or an NSF configuration from scratch.
Correctness: True (✓) if the methodology uses formal verification techniques or a formal correctness-by-construction approach.
Optimality: True (✓) if the methodology can find the optimal solution according to some optimality criteria
Knowledge base: The origin of the input information exploited by the methodology to automatically compute the solution.
Technology: The adopted virtualization paradigm, i.e., SDN or NFV (only for papers about service composition).
Supported NSFs: The NSFs that are supported by the methodology (only for papers about NSFs configuration).
Scalability: A concise indication of the scalability achieved by the methodology (i.e., number of NSFs, requirements or rules).

Table 3.1 Features analyzed for state-of-the-art papers

Reference	Target	Fixing	Scratch	Correctness	Optimality	Knowledge base	Technology	Scalability
[37]	Virtual	X	✓	X	X	U	SDN	4000 rules
[38]	Virtual	X	✓	X	✓(ILP)	U	SDN	~250 functions
[39]	Virtual	X	✓	X	✓(ILP)	U	SDN	250 switches
[40]	Virtual	X	✓	✓	X	U	SDN	No information
[41]	Virtual	X	✓	X	X	U	SDN	No information
[42]	Virtual	✓	X	X	X	U, S	SDN	No information
[43]	Virtual	✓	X	X	X	S	SDN	~70000 rules
[44]	Virtual	X	✓	X	X	U	NFV	5 functions
[45]	Virtual	X	✓	X	X	U	NFV	16 functions
[46]	Virtual	X	✓	X	X	U	NFV	15 functions
[47]	Both	✓	✓	X	X	U, S	NFV	100 functions
[48]	Virtual	✓	X	X	X	U, S	NFV	No information
[49]	Virtual	X	✓	X	✓(heuristic)	U	NFV	79 nodes
[50]	Virtual	X	✓	X	✓(heuristic)	U	NFV	10 functions
[51]	Virtual	X	✓	X	✓(heuristic)	U	NFV	8 functions
[52]	Virtual	X	✓	X	✓(heuristic)	U	NFV	No information
[53]	Virtual	X	✓	X	✓(ILP)	U	NFV	7 functions
[54]	Virtual	X	✓	X	✓(ILP)	U	NFV	No information
[55]	Virtual	X	✓	X	✓(ILP)	U	NFV	~20 functions
[56]	Virtual	X	✓	X	✓(ILP)	U	NFV	~10 functions
[57]	Both	✓	✓	X	✓(heuristic)	U, S	NFV	60 firewalls
[58]	Both	X	✓	✓	✓(iterative SMT)	U	NFV	20 firewalls

U = User-specified policies, S = Security chain

Table 3.2 Comparison among solutions for automatic network security service composition

Analysis of the state of the art

In this research area, there is a first group of studies that deal with the investigation of how a security service can be designed automatically in an SDN-based network. In this context, the security functionalities that compose the designed service may be executed by SDN switches, or by other virtual entities that can communicate with a distributed architecture of switches. The objective is commonly to generate a security service that satisfy a set of user-specified traffic steering policies. For example, the policies may specify that certain destinations must not be reached by certain traffic flows, or that some SDN switches must not be crossed by them.

A first milestone of this group was represented by the FRESCO approach [37], which uses elements named modules as basic components for security service composition. Each module can enforce a specific security functionality (e.g., intrusion detection, blacklist scan, filtering), so that they can be composed to create a complete service. The high modularity of this approach enables the creation of multiple flexible services, which may also use the same module instances. Besides, the activation of these modules does not require an external human intervention, as it can be triggered by events such as the identification of specific kinds of network traffic. As a first milestone, however, FRESCO is less elaborated than the next studies, and it does not yet integrate features such as optimization and formal verification.

Other approaches that are proposed in literature to face the automatic security service composition problem for SDN-based networks are [41], [42] and [43]. On the one hand, [41] proposes an intent-based multi-layer orchestrator, which establishes how the interconnection among the functions must be defined so as to enforce confidentiality requirements and to provide a solution against eavesdropping. On the other hand, [42] illustrates an intent-refinement process that uses machine learning and feedback from the user to translate user-specified intents (e.g., if the user requests a new firewall with some rules, this process adds it to the security service by satisfying constraints related to latency and bandwidth). Nevertheless, like [37], both these studies overlook optimization and formal verification. Then, [43] proposes an algorithm to simplify and merge security services into a single one, by employing learning finite automata named Markov models, and while keeping an acceptable level of detection accuracy. Such approach can only refactor existing services instead of generating one from scratch, and it lacks the same features as the other discussed techniques.

The optimization feature has been first proposed in [38] and [39], where the service composition problem is formulated as an *Integer Linear Programming* (ILP) problem. However, in both cases, the optimization objectives are still related to networking, rather than to security. For example, they aim to balance the traffic load among the switches of the network. Instead, the formal verification feature has been introduced in [40]. Specifically, [40] proposes a rule-based system for the automatic composition of security services structured as chains, and it formally checks if the automatically synthesized service is compliant with the required security properties.

There is also a larger group of studies that consider the possibility, enabled by the NFV technology, of instantiating softwarized functions instead of hardware middleboxes. In this scenario, optimizing the design of a security service results easier, thanks to the dynamism of the interconnections among functions and of the creation of the paths that can be crossed by traffic flows. Despite this, some studies [44–48] do not embed optimization objectives in the the problem formulation. Of course, they focus on other features. The initial proposal of [44] is to automatically generate a security service graph by selecting the functions whose capabilities match with the fields of the user-specified intents. Then, the same authors expand this idea in [45], where they propose to use the k-means clustering algorithm to select the VNFs that are required to create the service, according to the level of security they can provide. Instead, a function composition algorithm based on Trie tree is discussed in [46]. The peculiarity of this strategy is that it can automatically manage changes of the IP addresses of the virtual functions, so that it can be employed also in cloud environments, where IP addresses often change over time. Besides, [47] and [48] propose approaches that are only able to automatically fix an already deployed security service, so as to make it compliant with new security policies.

Aside from these methodologies, a more consistent number of studies [49–52, 54, 55, 53] aims to fulfill optimization objectives in the composition of a network security design. They follow various different approaches to reach this goal. First, some heuristic approaches are explored in [49–52]. Briefly, the heuristic algorithm presented in [49] automatically generates a sequential ordering of functions that satisfies all the specified policies, so that the same function instance may be present in more flow paths, thus avoiding redundancy. [50] uses a greedy iterative approach, which, at each step of the service design process, analyzes a possible function combination and gives priority to security services where the composing functions have the maximum total throughput. [51] employs a breadth-first search algorithm operating in two steps: after the required functions are selected, they are composed by constructing a breadth-first search tree. Then, [52] proposes partitioning heuristics, according to which the security service is divided into partitions and the composition problem is solved for each one independently. For all these approaches, the main optimization objective is to minimize resource consumption, related to hardware and power resource usage. This objective is coherent with the characteristics of NFV networks, where the activation and work of each virtual function require some physical resources.

Even if heuristic algorithms may be fast, they may not reach the optimal solution. Therefore, some researchers explored the use of non-heuristic optimization, by formulating the security service composition problem as an *Integer Linear Programming* (ILP) problem [53–56]. In greater detail, [53] considers all possible solutions to the problem, given a set of security policies, by generating an augmented graph, i.e., a graph with the maximum number of interconnections between the selected functions. Then, the automated resolution of the ILP problem built on the augmented graph allows identifying and choosing the interconnections that minimize the number of employed functions. [54] introduces additional security-oriented objective functions in the ILP problem formulation, such as user rating, experts trustworthiness expectations, and security evaluation. Their approach is further extended in [55] with the support of dynamic adaptation to network changes, so that the user-specified policies are still enforced even when a function is subject to an attack and fails. Instead, [56] abstracts the security policies by associating a numerical value to each one of them, and then it aims to maximize the combined security level, while minimizing CPU usage and utilization time.

The security service composition problem has also a variant, where a network graph is already existing, but it is devoid of security controls and it should be enriched with the allocation scheme of the NSFs. This problem may arise if the network administrator does not want to apply changes to the structure of the network topology, when the required security policies must be enforced on it. In this case, the NSFs should be just allocated between a pair of existing network functions. Two studies dealing with these problems are [57] and [58]. On the one hand, [57] proposes a heuristic algorithm that establishes how firewalls should be allocated in a computer network, while minimizing the cardinality of the rule sets that would be needed to satisfy the user-specified policies. On the other hand, [58] addresses the problem for a larger number of functions (firewalls, VPN gateways, and intrusion detection systems), aims to minimize the number of allocated NSFs, and pursues a formal approach by modeling the problem as an iterative *Satisfiability Modulo Theories* (SMT) problem. At each iteration of the proposed algorithm, the definition of the NSF allocation scheme is tuned until all the user-specified policies are correctly enforced.

Final considerations and open issues

From the analysis of the studies related to automatic security service composition, it is already possible to draw some considerations about limitations that should be overcome.

First, the few studies that introduce formal verification in the automatic security service composition, i.e., [40] and [58], have limitations that should still be overcome. Specifically, the technique described in [40] can only generate security chains, while modern computer networks commonly have ramified topologies. Besides, it is specifically designed to work with security services for Android applications connected to SDN.-based networks, and its possible applicability to other virtual environments is not discussed. Instead, the approach proposed in [58] can allocate security functions in a network composed of only end points and routers. Therefore, it cannot manage the presence of more complex middleboxes, such as network address translators and load balancers.

Second, even if optimization criteria related to networking are important in this research area and they are embedded in some of the state-of-the-art approaches for service composition, however they are rarely paired with security-oriented criteria. For example, [38] optimizes the load balancing of the traffic load among the switches of the network, [39, 50] aim to reach the optimum throughput of the traffic that must be routed in an SDN-based network, [49, 51, 54, 55] optimize the efficiency of resource consumption, and [53] achieves minimal bandwidth demand on its links. Two studies that embed security-oriented optimization objectives are [52] and [56]. However, the effective optimization criteria that are enforced in these techniques are not explicitly clear. [52] aims to achieve the maximum compliance with well-known security related best practices and [56] the maximum security level of the service, given by the sum of the security levels of each composing security function. Both criteria (i.e., the well-known security practices and the security levels) are not sufficiently detailed and put in relation with real security problems (e.g., the optimization of the security operations of a firewall, so that it can takes decisions in the least required time to block possible attacks).

Third, the variant problem addressed by [57] and [58], i.e., enriching an already existing network topology with security services, is more complex than the standard one, because the behavior of the middleboxes that are present in the already existing

network must be analyzed and considered in the automatic generation of the NSF allocation scheme. It is also a considerably interesting problem, because introducing security after an initial design that is only related to networking is quite common. Nevertheless, at the moment, the literature about this problem is still limited to the two aforementioned studies. Besides, these studies limit their proposed approaches to just define the allocation scheme, without providing the user with the concrete configuration of the allocated functions.

Fourth, most of the analyzed approaches, with the exception of [40, 54, 55], do not address the automatic configuration problem of the NSFs composing the automatically generated service of the allocation scheme, even though this problem is strictly dependent on the decisions taken in the service composition task. After all, these three studies do not solve the combined problem of security service composition and configuration in an adequate way. All three of them can only address it for service function chains, even though the topology of modern virtualized computer network is commonly a ramified graph. Moreover, as previously mentioned, the generality of the approach described in [40], specifically designed for Android applications connected to SDN-based networks, is not validated. Instead, [54, 55] propose optimization techniques that overlook security-oriented objectives, and that do not provide formal correctness assurance for the computed solutions.

3.2.2 Automatic security function configuration

Automatic security function configuration has been investigated more extensively than automatic security service composition, as it is a problem that was deemed interesting already for traditional networks. Virtualization clearly determined a strong comeback of this topic in the literature, and in the last years the state of the art has been enriched with more approaches. A concise overview of the contributions to this topic is shown in Table 3.3. For the description of the table legend, Table 3.1 is again the reference. The contributions will be analyzed and discussed separately for each kind of NSF, followed by a summary of the open issues.

Firewalls and access control devices

A first group of NSFs that have been object of investigation is composed of firewalls and access control functions, as they provide protection enough for an end-to-end

Reference	Target	Fixing	Scratch	Correctness	Optimality	Knowledge base	Supported NSFs	Scalability
[59, 60]	Traditional	X	✓	X	X	U	Firewall	Single firewall
[61]	Traditional	X	✓	X	X	U	Access control devices	~10 devices
[62]	Traditional	X	✓	X	X	U	Firewall	Distributed firewall
[63]	Traditional	X	✓	X	X	U	Firewall	No information
[64]	Traditional	X	✓	✓	X	U	Firewall	No information
[65]	Traditional	X	✓	✓	X	U	Access control devices	~1000 nodes
[66]	Traditional	X	✓	✓	X	U	Access control devices	No information
[67]	Traditional	X	✓	✓	X	U	Firewall	Single firewall
[68]	Virtual	X	✓	✓	X	U	Firewall	Single firewall
[69]	Both	X	✓	✓	X	U	Firewall	~5 firewalls
[70]	Both	X	✓	X	X	U	Access control devices	~50 devices
[71]	Both	X	✓	X	X	U	Access control devices	~200 devices
[72]	Both	X	✓	✓	X	U	Firewall	No information
[73]	Both	✓	✓	✓	X	U	Access control devices	~10 devices
[74]	Virtual	X	✓	X	X	U	Firewall	~1700 firewalls
[75]	Both	X	✓	X	X	U	Access control devices	~1000 policies
[76]	Both	X	✓	X	X	U	Access control devices	~60 policies
[77]	Both	X	✓	✓	X	U	Firewall	3 firewalls
[78]	Both	X	✓	✓	X	U	Access control devices	~100 devices
[79]	Traditional	✓	X	X	X	A	Access control devices	No information
[80]	Traditional	✓	X	X	X	F	Firewall	No information
[81]	Traditional	✓	X	X	X	F	Firewall	60 rules
[82]	Traditional	✓	X	X	X	F	Firewall	No information
[83]	Traditional	✓	X	X	X	F	Firewall	No information
[84]	Traditional	✓	X	✓	X	F	Firewall	Single firewall
[85]	Traditional	✓	X	✓	X	F	Firewall	Single firewall
[86]	Traditional	✓	X	✓	✓(MaxSMT)	A	Access control devices	~400 devices
[87]	Traditional	✓	X	✓	X	A	Access control devices	~5 devices
[88, 89]	Virtual	X	✓	X	X	U	SDN switch	~10 switches
[90]	Virtual	✓	✓	X	X	U	SDN switch	No information
[91]	Virtual	X	✓	X	X	U	SDN switch	~100 switches
[92]	Virtual	X	✓	X	X	U	SDN switch	~15 switches
[93]	Virtual	✓	X	X	✓(ILP)	U	SDN switch	~35 switches
[94]	Virtual	X	✓	X	X	T	SDN switch	~15 switches
[95]	Virtual	✓	X	X	X	U	SDN switch	~10 switches
[96, 97]	Traditional	X	✓	X	X	U	VPN gateway	~35 gateways
[98]	Traditional	X	✓	X	X	U	VPN gateway	~85 requirements
[99]	Traditional	X	✓	X	X	U	VPN gateway	~50 gateways
[100]	Traditional	X	✓	X	X	U	VPN gateway	60 requirements
[101]	Traditional	✓	X	X	X	V	VPN gateway	~1000 gateways
[102]	Traditional	✓	X	X	X	V	VPN gateway	500 rules
[103]	Virtual	X	✓	X	X	U	VPN gateway	No information
[104–107]	Traditional	X	✓	X	X	U	embedded systems	No information
[108]	Traditional	X	✓	X	X	U	embedded systems	~10 devices
[109]	Both	X	✓	✓	X	U	embedded systems	~100000 devices
[110]	Traditional	X	✓	✓	X	U	embedded systems	~20000 devices
[111]	Both	X	✓	✓	X	U	embedded systems	~ 10 devices
[112, 113]	Virtual	X	✓	X	X	U	embedded systems	No information
[114]	Virtual	X	✓	✓	X	U	embedded systems	No information
[115]	Virtual	X	✓	X	X	U	embedded systems	~50 devices
[116]	Traditional	✓	X	X	X	F, V	Firewall and VPN gateway	No information
[117]	Traditional	X	✓	✓	✓(logic programming)	F, I	Firewall, NIDS	No information
[118]	Traditional	X	✓	X	✓(ILP)	U	Firewall, IDS, VPN	No information
[54]	Virtual	X	✓	X	✓(ILP)	U	All NSFs	No information
[55]	Virtual	X	✓	X	✓(ILP)	U	All NSFs	~20 functions
[119]	Virtual	X	✓	X	X	U	All NSFs	No information
[40]	Virtual	X	✓	✓	X	U	All NSFs	No information

U = User-specified policies, A = Access control configuration, F = Firewall configuration, V = VPN configuration, I = IDS configuration, N = Network addresses, T = Network traffic

Table 3.3 Comparison among solutions for automatic network security function configuration

service in a large number of situations without requiring a too complex configuration. After an initial basic proposal by [61], which was based on a very high level abstraction of the filters for the routers of a distributed access control system, the real milestone is represented by Firmato [59], a management toolkit which allows automatic generation of a firewall configuration expressed by means of an abstract, vendor-independent language, close enough to real firewall configuration files. The extension module described in [120] enables a query-and-answer session for the communication between the toolkit and a human user. Throughout this session, users easily identify which kinds of traffic are allowed or blocked in the networks, and then they can use Firmato to address any identified vulnerability (e.g., to block a

traffic that is potentially malicious but still allowed in the network). Despite the relevance of these studies, the non-applicability to a distributed architecture represents a serious limitation. Next studies, i.e., [60, 62, 63], tried to overcome this limitation, proposing approaches that can work on distributed filtering architectures, even if only [62] effectively shows a real validation for a distributed system. Nonetheless, all these studies overlook the two features of formal verification and optimization. Formal techniques to provide assurance for the correctness of the automatically generated configuration have been investigated in [64], [65], [66], and [67], which respectively employ formal logic programming methods, a Boolean satisfiability problem formulation, formal methods applied to abstract machine notation, formal argumentation and preference reasoning. However, as it can also be inferred from their publication date, all the studies discussed so far are designed and validated for traditional networks, and do not take into account the complexity of virtual networks.

The advent of network softwarization increased the importance of this problem, and a new group of approaches [68–74] addressed it. Most of these methodologies can be applied to traditional networks as well, with the exception of [68, 74], as [68] only works with Netfilter, while [74] with iptables. Just to mention the main novelties introduced by these approaches, [69] proposes an abstraction named arbiter, which represents the network box where each user-specified policy can be enforced and which is based on algebraic requirements. [70] formulates the auto-configuration problem as a *Satisfiability Modulo Theories* (SMT) problem, by running a stratified Datalog program based on a declarative logic programming language. [71] improves the previous study, by defining a domain-specific heuristics, such as partial evaluation, so as to reduce the solution space of the SMT problem. [72] attacks the problem of automatic firewall configuration decompilation, by proposing another language that is independent of the specific low-level settings of the firewalls. [73] pursues a comprehensive approach for access control policy refinement and formal verification, where it is possible to both refine user-specified policies and fix an existing configuration.

As proof of the relevance of the auto-configuration problem for firewalls and access control functions, a new group of studies ([75–78]) enriched the related literature at the beginning of this new decade. On the one hand, [75, 76] try to improve the scalability of the automatic configuration methodologies, by respectively applying machine learning on the operator-provided feedback and priority-based domain type enforcement. On the other hand, [77, 78] employ formal methods to

ensure the correctness of the firewall rules through preference-based argumentation reasoning and metagraph algebra.

All the studies discussed so far can automatically compute firewall configurations in a situation where the functions are already allocated in the network, but devoid of filtering rules. Instead, another group of papers ([79–87]) propose approaches to refactor and fix existing configurations that are not compliant with the user-specified security policies. Computing the rules from scratch commonly requires more time, but the reconfiguration of the NSFs should be as fast as possible to stop some ongoing attacks. Besides, these methodologies can also be enforced for the scenario where a human user wants to introduce a limited number of new rules in a firewall configuration to directly enforce new policies.

SDN switches

For what concerns the automatic configuration of SDN switches, a first problem that was addressed is the definition of user-friendly languages that can be used by humans to specify the security policies in a way that is independent from the vendor-specific implementations. This problem has been relevant since the first proposals related to SDN networks, even before network security automation started to be investigated for virtualized computer networks. The main studies related to this topic are Ethane [121], Frenetic [122] and PolicyCop [123]. Ethane [121] is a language that allows writing access control policies with a flow-based security language that can be easily understood by human users. Frenetic [122] is instead specifically designed to be compliant with OpenFlow switches and it provides the syntax for the specification of events against which the configuration should be modified. Then, PolicyCop [123] can be used for the specification of service-level agreements within Openflow. A further discussion of high-level languages for SDN policy specification is out of scope for this dissertation. A complete discussion about this topic can be found in [124].

While the research focused on the definition of these policy specification languages, in the meantime OpenFlow became the most commonly used SDN protocol. Therefore, in the literature, the milestone studies about the automatic configuration of SDN switches used that protocol as a foundation [88, 90, 91]. Even if all these three approaches can automatically refine policies expressed in natural language into the concrete configuration of a distributed SDN switch architecture, each one

has some peculiarities and differs from the others. In particular, CloudWatcher [88] is enhanced with the feature of identifying the shortest route for each traffic flow to reach a detection point, so as to allow that all the traffic crossing the network is inspected at least once. Procera [90] introduces the possibility of specifying reactive policies that capture all the information needed to enforce security constraints after a specific event occurs in the network. OpenSec [91] has a native reaction mechanism and is characterized by higher user-friendliness of the language employed for policy specification. On the trend initiated by these three approaches, other studies [92–94, 89, 95] continued the investigation of the auto-configuration problem for SDN switches, trying to overcome some of their limitations. From this point of view, the approach discussed in [92] can also be applied to inter-domain environments, and its extension presented in [94] introduces an enforcement mechanism directly embedded in each switch, so as to enable their proactive prevention of attacks. Instead, [93] aims to minimize the number of rounds through which traffic flows can reach a specific waypoint towards their destination, [89] analyzes policies related to agility specifications (i.e., security actions that must be executed in reaction to mutation events), and [95] employs multi-attributed graphs for policy specification in order to delete conflicts that are present among the SDN switch rules.

VPN gateways

The configuration of VPN gateways poses additional challenges with respect to access control functions, firewalls and SDN switches. First of all, tunnels are often created by encapsulating packets. Such an operation must be considered for the definition of automated configuration methodologies, and correctly modeled if they use formal verification techniques. Besides, many alternative technologies exist for the creation of VPNs, from the protocols that can be used, to the operational modes for the enforcement of security properties such as integrity and confidentiality.

In this research area, the milestone study is represented by [96]. In this paper, the policies that a human user may specify for having a secure communication with a VPN are classified into four main categories: 1) access control policies, which are defined to restrict access to the network only for trusted traffic; 2) security coverage policies, that are defined to specify which algorithms (e.g., for encryption) should be applied to the traffic; 3) content access policies, that are defined to decide which network elements can analyze plain traffic, without encryption; 4) security

association policies, that are defined to specify how *Security Associations* (SAs) can share security attributes between the end points of the VPN. The presence of multiple policy categories is another proof of the higher complexity of the auto-configuration problem for VPN gateways. On the basis of these categories, the methodology proposed in [96] is composed of three alternative approaches: 1) a direct approach, where for each request a separate tunnel is created; 2) a bundle approach, where the traffic flows interested by the policies are grouped and, for each set, a single tunnel is generated, providing completeness at the expense of speed; 3) a combined approach, as a trade-off of the previous two. As these three approaches always lead to VPN rule sets that have a larger cardinality than the required one to enforce all the user-specified policies, the same authors propose a fourth approach in [97], called ordered-split approach. In this technique, the optimal solution is identified so as to minimize the number of required VPN tunnels, through the application of the traditional “task scheduling" algorithm. Nevertheless, these initial studies do not solve the VPN auto-configuration problem for inter-domain environments. Their extension presented in [98] aims to overcome this limitation, by proposing a negotiation protocol through which gateways of different domains, commonly known as Autonomous Systems, can negotiate the automatic generation of the VPN tunnels.

Other relevant approaches have been presented in the next years by [99–102]. In greater detail, [99] aims to tune the user-specified policies so as to remove any possible conflicts before refining them into the concrete VPN configuration, through an iterative approach such that, at each step, the policies are ordered by decreasing tunnel length and possible conflicts are removed starting from the longest tunnel. [100] uses recessive binary trees as supporting data structure for the policy refinement operation, with the advantage that these trees allow reusing already generated rules for the satisfaction of other policies by simply changing some of their conditions or selectors. [101] focuses on providing robustness regarding potential failures and high scalability, as the proposed approach is meant to be applied to nested networks and mobile environments. [102] proposes a tuning strategy to solve conflicts in existing VPN configurations.

All the studies analyzed so far about the automatic configuration of VPN gateways solve this problem only for traditional networks. The only approach that tries to go beyond this limitation is the one described in [103], where an SDN-based

architecture is proposed to automatically configure VPN tunnels among network devices belonging to different providers.

Embedded devices

The heterogeneity, pervasiveness and distributed nature of embedded devices make their manual configuration much more error-prone than what it already is for traditional NSFs. For example, in IoT environments, not only access control policies, but also privacy and data protection ones must be considered for an automatic configuration process. The process itself should have a high degree of flexibility in compliance with the adaptation and self-healing features of IoT infrastructures. Therefore, specifying and refining policies to manage the security of distributed embedded systems is a complex task.

Some early research efforts related to this research topic are [104–106]. On the one hand, [104] proposes a security toolkit that, while communicating with multiple distributed systems through the MQTT protocol, works as an MQTT broker and has the capability of refining authorization and obligation policies. On the other hand, [105] describes a security-aware policy enforcement framework that can provide access control and service provisioning so as to be compliant with security and quality constraints. This strategy is later adapted in [106] for networked embedded systems of the smart health scenario, where additional requirements such as the anonymity of personal information must be considered. Nevertheless, these first attempts cannot provide formal assurance of the configuration correctness. This problem has been then addressed by other studies [109–111]. Specifically, [109] proposes a formal vendor-independent graph-based policy specification mechanism to work in multi-administrative IoT environments. [110] formulates the configuration problem as an SMT problem to refine security requirements, operational integrity invariants, and robustness constraints related to smart meters.[111] pursues a tree search-based algorithm for policy refinement, paired with a verification method to check if all the threats specified in policies requested by the user are successfully enforced in the generated configuration.

Among all the possible security policies related to embedded devices, the so-called sticky policies, originally defined in [125], allow attaching security and privacy requirements to owners' data in order to drive access control decisions and policy enforcement. In the literature, two studies [107, 108] propose automated approaches

to manage this type of policies. On the one hand, [107] allows each user to specify their own policies on the data they own, as long as they satisfy constraints defined by a trust authority. On the other hand, [108] enables IoT users to attach their personal privacy preferences to smart objects as meta-data, also used in the policy enforcement mechanism.

Given the relevance of researching new methods to automate IoT security configuration, a recent EU H2020 research project, named ANASTACIA [112], had the main objective to dynamically refine user-specified security preferences into the configuration of cyber physical systems and IoT architectures. In the frame of this project, multiple studies have been performed [113–115]. For example, [113] investigates multiple abstraction levels for IoT policies, with the aim to of enabling a technology-agnostic policy refinement mechanism. Instead, [114] exploits a logic formalism based on rule reasoning and the Semantic Web technology, so as to infer new knowledge from events occurring in the networks and to formally verify the resulting automated configuration process. Finally, [115] investigates the automatic configuration of honeynets composed of highly interactive honeypots, by refining proactive security policies, specified by the users to configure monitoring agents.

Heterogeneous security services

All the approaches analyzed so far can be applied for the configuration of a single type of NSF (firewall, access control function, SDN switch, VPN gateway, embedded device). The assumption under which these approaches work is that all the other NSFs that are possibly present in the network are already configured. Even if this assumption is often acceptable, especially when policies of the same type are specified, there are circumstances where multiple NSFs must be automatically configured. However, the literature about the automatic configuration of heterogeneous services is limited, as the only studies that face this problem are [116–118, 54, 55, 119].

The first study that addressed the auto-configuration problem for a heterogeneous security service is [116]. The approach proposed there uses formal methods to check possible violations of user-specified policies and recommend which NSFs should be reconfigured to eliminate the identified issues. However, it has a strong limitation: it cannot automatically generate the NSF configuration from scratch, but it can only be applied to tune an existing one. Next, [117] overcame this limitation by proposing a refinement technique, based on the formulation of the auto-configuration

problem with a logic programming language, for both firewall and IDSs. It also pairs automation with optimization, as the resulting solution of the automated process aims to place each configuration rule on existing NSFs so as to minimize bandwidth usage and packet drop rate. Nevertheless, firewalls and NIDSs are configured at different stages of the process. Therefore, only locally optimal solutions are actually computed by this proposal.

A simultaneous configuration of multiple types of NSFs has been achieved by [118, 54, 55, 119]. First, MIRAGE [118] proposes a top-down refinement process of global security policies into configurations of three different NSF types: firewalls, VPN gateways, and IDSs. The problem is formulated as a linear programming problem, whose objective function is to minimize the number of used function instances for the placement of the rules, and it includes a bottom-up analysis of a possibly already deployed network security configuration to guarantee its consistency with the user-specified policies. Second, [54] and its extension [55] describe a two-step policy refinement algorithm, where initially the NSFs rules are derived from the policies and expressed with an abstract and vendor-independent representation, and then a translator adapts these rules to the syntax required by the specific NSF implementation. The NSFs that are supported by this algorithm are stateless packet filtering firewalls, L7 filters, basic content inspection, but not VPN gateways, proxies and IDSs. Third, [119] employs an automatic data model mapper to automate the refinement of high-level user-specified policies, which are expressed as a tree graph through a YANG data model. In this way, the minimum tree edit distance is used to refine the user-specified policies into the low-level configuration of the NSFs.

Final considerations and open issues

On the basis of this extensive analysis, state-of-the-art studies about automatic NSF configuration have several limitations that should still be addressed.

First, even if formal verification is paired with automation more often than for security service composition methodologies, optimization is instead usually neglected. However, optimizing the configuration rule sets of NSFs may improve security efficiency and network performance. For example, a firewall with a minimum number of filtering rules takes less time to identify a malicious traffic to block. Even in the studies that embed optimization, the objectives are commonly related to networking issues. In particular, [86] computes the minimal number of fixes to be applied to an

existing security configuration, [93] minimizes the number of rounds through which traffic flows can reach a specific waypoint towards their destination, [117] minimizes bandwidth usage and packet drop rate, and [118, 54, 55] optimizes the efficiency of resource consumption.

Second, most studies explicitly focus on a single NSF type, and they cannot be applied to configure a heterogeneous security service. The studies which address the automatic configuration problem for heterogeneous services, i.e., [116–118, 54, 55, 119], still have limitations that should be overcome. [116–118] propose automatic approaches that are designed for traditional networks and do not consider all the characteristics deriving from the complexity of virtual network functions. [54, 55] present methodologies that can work only on simple service function chains, and lack the formal verification feature. [119] focuses on performing a mapping operation of the user-specified policies onto the low-level function configuration, but overlooking the behavior of other network functions that may be already present and configured in the same network.

Third, as it can be seen in the "Scalability" column of Table 3.3, when a validation of the approach is proposed in the related paper, it shows that the methodology can rarely scale to networks composed of a hundred of NSFs. So as to be able to manage virtual networks of medium-big size, automatic approaches for security configuration should be able to refine around 100 policies into the configuration of around 100 NSFs. Besides, this task should be expected to per performed in some minutes (e.g., five minutes), as that is a time that is compatible with the total time required to set up a virtual service.

Fourth, with the exception of [40], [54] and its extension [55], no state-of-the-art approach can also solve the security service composition problem, even though it is strictly dependent on the decisions taken during the NSF configuration task. Besides, as already mentioned, these three studies do not solve the combined problem of security service composition and configuration in an adequate way. All three of them can only address it for service function chains, even though the topology of modern virtualized computer network is commonly a ramified graph. Moreover, the generality of the approach described in [40], specifically designed for Android applications connected to SDN-based networks, is not validated. Instead, [54] and [55] proposes optimization techniques that overlook security-oriented objectives, and that do not provide formal correctness assurance for the computed solutions.

Finally, none of the three can address the problem of automatically computing the NSF allocation scheme in an existing network, which is usually more complex than the service generation from scratch.

3.3 Automatic network security orchestration

This dissertation focuses on three relevant problems related to automatic network security orchestration, whose state of the art is discussed in this section: the orchestration of network security reconfiguration transients (Subsection 3.3.1), the NSF abstraction for security orchestration (Subsection 3.3.2), and the integration of security and network orchestrators (Subsection 3.3.3).

3.3.1 Orchestration of network security reconfiguration transients

State-of-the-art studies have broadly investigated the problem of guaranteeing that some network security properties are still valid during a network reconfiguration, as discussed in an exhaustive review about this topic [126]. In particular, during a transient unfolding when a network configuration is modified, there are three types of security properties that may be violated: 1) connectivity consistency, i.e., the capability of the network to keep delivering packets to their respective destinations; 2) policy (or path) consistency, i.e., the capability of the network to keep delivering packets through a specific path of middleboxes for the whole update; 3) capacity consistency, i.e., the networks' capability to manage availability and limits of resources as bandwidth and latency.

In the literature, two main functions that have been object of study for the orchestration of network security reconfiguration transients are packet filtering firewalls and SDN switches.

Firewall reconfiguration transients

A security function for which guaranteeing security properties during a reconfiguration transient is important is the firewall. Specifically, managing a firewall recon-

figuration transient is a problem that is related to the consistency of connectivity policies during that transient. Connectivity policies can be divided into reachability and isolation policies. The former aim at ensuring that some network nodes can mutually reach each other, with the objective of avoiding service disruptions. The latter aim at blocking specific kinds of traffic flows, with the objective of avoiding access control violations and privilege escalation. e.g., after an undetected intrusion [127]. However, the literature about firewall reconfiguration transient is limited from many points of view.

First, the problem of optimizing reconfiguration transients for packet filtering firewalls has been studied in literature only for intra-firewall reconfiguration. In fact, the studies that address this problem [128–132] exclusively work on centralized firewalls. Therefore, the transient problem is reduced to a simpler intra-firewall policy deployment, where the firewall configuration is meant to be only its rule set. The only objective is to identify a safe scheduling of the update operations for the filtering rules of a single firewall. The operations that are considered by these studies are rule appending or deleting for firewalls of Type I [128–130], and additionally rule moving for firewalls of Type II [128, 131, 132]. This lack of studies represents a gap that needs to be filled, because the complexity of transient management for inter-firewall reconfiguration is much higher. Inter-firewall reconfiguration takes longer, so the number of transient states is higher and passing through unsecure states is more dangerous. This complexity derives from a larger number of reconfiguration operations that must be considered (e.g., deployment of new instances, removal of useless instances). Besides, the intra-firewall problem becomes even less important in virtualized networks, as it is easier and faster to launch a new virtual firewall with the new filtering rule set to replace the old one, instead of modifying single rules.

Second, these state-of-the-art approaches lack formal verification. They are simply based on greedy algorithms, which cannot provide correctness of the computed scheduling, and their result might not be the most efficient by their admission [128].

Third, as previously mentioned, the firewall reconfiguration transient is a problem that should deal with guaranteeing the consistency of connectivity policies. However, all state-of-the-art studies about firewall reconfiguration transients only guarantees an internal coherence of firewall configuration states, i.e., any packet that is permitted (or denied) by both the initial and target firewall configurations is always permitted (or denied) in the intermediate configuration state of the reconfiguration transient. Such

characterization of this problem has become less interesting in modern networks composed of distributed functions, where multiple communications might happen among the connected devices.

Fourth, these papers mainly targeted traditional firewalls. Situations deriving from network softwarization, such as the usage of containers (e.g., Dockers) that require the deployment of a new process if the configuration of the previous one must be changed, are totally overlooked.

SDN switch reconfiguration transients

A security function for which guaranteeing security properties during a reconfiguration transient is important is the SDN switch. In the literature, this problem has been investigated also for distributed architectures composed of multiple SDN switches [126]. The advent of SDN has revived interest in the transient management problem, casting it into the context of programmable networks [133]. Automation has become a critical factor in reducing operational times, so reconfiguration of network and security functions can easily occur with a higher frequency than with hardware-based middleboxes. This possibility has allowed reaching better scalability in network management [134], and improving elasticity control of network functions [135].

However, the studies dealing with SDN switch reconfiguration transients are limited only to two specific types of policy consistency. The most common type, addressed by the majority of related papers ([136], [137], [138] [139], [140], [141], [142]), is the *Per-Packet Consistency* (PPC) policy. In this case, only two paths must be crossed by the packets: one related to the original configuration of the SDN switches, the other one to the final configuration after the changes. Therefore, this kind of policy requires that every packet travels either on its initial or final path, never on intermediate ones. However, PPC might result too restrictive in some instances (e.g., sometimes it is sufficient to guarantee that the traffic passes through a firewall, independently of the other crossed functions). As such, in some papers ([143], [144]) a mitigation of PPC is represented by the *Way-Point Enforcement* (WPE) policy, for which it is just required that the packets can always cross a set of specific waypoints during the transient.

There also exist studies ([145], [146]) where policies are more complex (e.g., it may be required that, in the network of a large Internet Service Provider, specific

traffic flows follow certain sub-paths). However, in all the mentioned cases, the specification of connectivity policies is not addressed. This shortcoming translates into a limitation for the proposed approaches, because connectivity policies represent the most general type of security policies through which it is possible to check the reachability or isolation of traffic flow (e.g., checking if a reachability policy is satisfied in a transient state means to analyze all the possible paths where the traffic might flow, and consider the behavior and configuration of all the possibly crossed functions).

Final considerations and open issues

From the analysis of the state of the art about the orchestration of network security reconfiguration transients, it emerges that the only distributed function for which this problem has been studies is the SDN switch. However, such studies have several limitations.

On the one hand, studies about SDN switches address security issues concerning the violation of connectivity policies only partially, because the configuration of SDN switches is mainly defined to address networking issues with respect to firewalls. On the other hand, they overlook the analysis of the impact that the behavior of other networks or security functions, which are present in the network, may cause to the reconfiguration transient. Moreover, optimization criteria (e.g., maximization of the secure transient states depending on the importance of each connectivity policy) should be enforced as well. Instead, in most of the studies analyzed so far only heuristics and greedy approaches are pursued, except for [143] and [144], where exact algorithms based on a *Mixed-Integer Program* (MIP) formulation are defined for computing an update scheme requiring the minimal number of intermediate states.

Therefore, focusing exclusively on SDN switches is a limitation that should be overcome by addressing the transient management problem for more general distributed packet filtering firewalls. Integrating formal verification and optimization would also enhance the automatic techniques that may be proposed to address this problem.

3.3.2 NSF Abstraction for Security Orchestration

Analysis of the state of the art

The advent of network virtualization led to the development of a high number of NSF implementations as software program that can be executed on Virtual Machines or containers. NSF implementation that are developed to perform the same operations often differ only for vendor-dependent characteristics related to their specific configuration language and set up. However, in those case, the security functionalities that they perform (e.g., firewalling, encryption, detection) are the same. Therefore, a possible idea would be to abstract NSF implementations from the vendor-dependent characteristics, and to consider only their security functionalities in the orchestration of network security management.

A main advantage of such an abstraction would be the possibility to reorganize and simplify the security orchestration workflow. The NSF implementations are commonly selected at the beginning of the approach that is usually pursued for enforcing security in a virtual network [147], i.e., before the generation of their configuration and of the deployment in the physical infrastructure. Nevertheless, this ordering of the operations leads to overlook network information (e.g., the topology of the virtual service, or the behavior of service functions like load balancers and network address translators)in the selection of the NSF implementations. If instead the vendor-independent abstractions of those implementations are selected and configured at the beginning, then the effective selection of the required implementations could be postponed to be performed jointly with their deployment, thus providing higher optimization for the security orchestration..

However, models for NSF abstraction have barely been investigated in the literature. A series of IETF RFC drafts, of which [148] is the most recent one, proposes the *Capability Information Model* (CapIM) to describe the security properties that an NSF can enforce in a vendor-neutral manner. With such a description, it is not required to refer to a specific technology or vendor-dependent function when defining a security service. However, these ideas have not been completely formalized and exploited for researching novel ways to perform security configuration and deployment. Among the research studies based on the ideas of these drafts, [149] uses a capability model for abstracting NSFs, but that work is restricted to access control and forwarding virtual functions. The work presented in [55] broadens the research

to other types of security functions. However, as in [149], the capability model is not used to innovate the security configuration workflow, and the vNSF selection for the security enforcement is performed before the allocation and configuration stage as usual, losing all the benefits deriving from a possible postponement. Then, [150] enhances the architecture based on these models to make it compliant with the SDN technology. [151] uses a tailored version of the model for the dynamic management of authentication, authorization, and accounting.

Final considerations and open issues

At the moment, CapIM, together with its extensions and customizations, represents the first and last effort in the literature to provide a higher abstraction of security-related operations that can be performed by VNFs. It also has several limitations that should be overcome. First, the possible capabilities that can be associated with NSF implementations are fixed and represent all the possible operations that the NSF might do. Instead, a capability should be a flexible representation, because it should express the security-related operation that an NSF should perform to enforce a specific policy. Besides, CapIM is never used to innovate the security configuration workflow. The study presented in [55] proposes a simple algorithm for the automatic configuration of network security functions, with an intent-based technique that refines user-specified policies identifying the required capabilities to enforce them. However, in that work, NSF selection is carried out so that NSFs are employed for the synthesis of the virtual security graph as usual, thus losing all the benefits deriving from a possible postponement of that selection.

In conclusion, in order to improve the conventional ways for security orchestration, there is the need of an additional level of NSF abstraction, so as to answer the high productivity of software development in network security.

3.3.3 Integration of Security and Network Orchestrators

Analysis of the state of the art

In virtualized environments, the orchestration of network security functions has been often paired with controllers which follow an *Event-Condition-Action* (ECA) paradigm for the configuration of the data plane [152]. In this context, security

orchestration is delivered through a comprehensive platform that is expected to deploy and manage the lifetime of multiple network services.

In a narrow sense, typical tasks of orchestration are life-cycle management, balancing of traffic floods, and assurance for the availability of applications and services [153]. Inside an orchestrating platform, the controller is the control plane element which focuses on the configuration of data plane elements, reacting to and processing events that might come from external users of the data plane itself. A typical example is an SDN controller, which incorporates the logic that is needed to decide, in a reactive manner, the filtering rules to be installed on each SDN switch, possibly taking into account countermeasures to security attacks [154].

Instead, a full orchestration of a network security service requires two essential contributions: a detection and mitigation strategy for the identification of cyber attacks, and a reconfiguration mechanism for addressing the identified threats and reinforce network security. However, in the literature, most of the efforts were devoted to single areas, and the proposed strategies were seldom optimized for synchronization and cooperation. On the one hand, detection and mitigation strategies always represented a flourishing research area in literature, and the advent of network softwarization simply gave them another boost. A consistent number of surveys have already classified and described the most well-known algorithms that might be implemented in intrusion detection and prevention systems for virtualized networks. In particular, [155] surveys algorithms that can be employed in the cloud, [156] focuses on SDN-based networks, [157] broadens the horizons of these surveys to include NFV. Recently, machine learning and artificial intelligent have been extensively used to improve the accuracy of attack detection, at the expense of more pressing requirements of computational resources [158], [159]. On the other hand, research on re-configuration mechanisms has been more limited in the past, until SDN and NFV provided the reactivity and programmability that are essential ingredients for these mechanisms.

In light of these considerations, only a limited number of orchestrating platforms with ECA-based controllers have been proposed in the literature as possible integration of existing network orchestrators. There is a first group of them [160, 161, 94] focusing on the orchestration of an SDN-based network. Deriving their prominent features from past policy-based management frameworks which leveraged SDN for network orchestration [123, 38, 91], these papers focus on ECA-based strategies for

the mitigation of cyber attacks. A second group of papers [149, 162] addresses the problem for NFV-based networks. In more detail, [149] describes the design of a policy-based NFV orchestrator, which can handle the life-cycle of virtual network functions and dynamically instantiate business applications as service chains. The behavior of this orchestrator is governed by user-defined ECA policies, which establish the management actions that are needed upon specific events. Instead, the other paper [162] proposes an Intent-Based Cloud Services orchestrator, based on an ECA policy model that is used to express the network intents.

Final considerations and open issues

State-of-the-art security orchestrators for virtual computer networks still have limitations that should be addressed.

On the one hand, since these orchestrators belonging to the first analyzed group [160, 161, 94] are designed to work exclusively on software-defined networks, the ideas behind their mitigation strategies cannot be brought over to NFV environments. In fact, the behavior of SDN switches is simply governed by the installed filtering rules, while the heterogeneity of virtual network security functions translates into a huge variety of different configurations to be managed during the mitigation of an attack. On the other hand, the orchestrators belonging to the second analyzed group [149, 162] mostly focus on automating the modular design of a service chain and on interfacing the virtual functions for a correct and safe communication, instead of integrating reconfiguration mechanisms within the orchestration workflow.

Therefore, the problem of integrating configuration and orchestration mechanisms should still be investigated in literature. Besides, an interesting research path is to investigate how security management can be automated with orchestrators working with containers, such as Docker Compose and Kubernetes.

Automatic Network Security Configuration

Chapter 4

The VEREFOO Approach

A main proposal of this dissertation is a novel methodology, named *VErified REFinement and Optimized Orchestration* (VEREFOO), which is the first approach in literature to combine automation, formal verification and optimization to simultaneously solve the allocation and configuration problems for NSFs in virtual computer networks.

The VEREFOO approach works as illustrated in Fig. 4.1. By embedding the Policy-Based Management paradigm, it requires the specification of network security policies describing the expected security behavior of the virtual network (e.g., describing which traffic flows should be blocked because potentially malicious, and which other ones should be encrypted to provide confidentiality of the communication). The user-specified policies are automatically refined into the NSF allocation scheme and configuration. The former describes how the NSFs required for the policy enforcement should be positioned in the logical topology of the virtual networks. Instead, the latter represents the rules that should be installed on the allocated NSFs to establish their security behavior.

The VEREFOO approach is designed to be a general method, that can be applied to any NSF type. However, in this dissertation, it has been fully developed only for two NSF types, i.e., packet filtering firewalls and VPN gateways, which are the most commonly used functions to enforce respectively connectivity policies and communication protection policies. However, the behavior of the VEREFOO approach can be easily adapted to work with other NSF types.

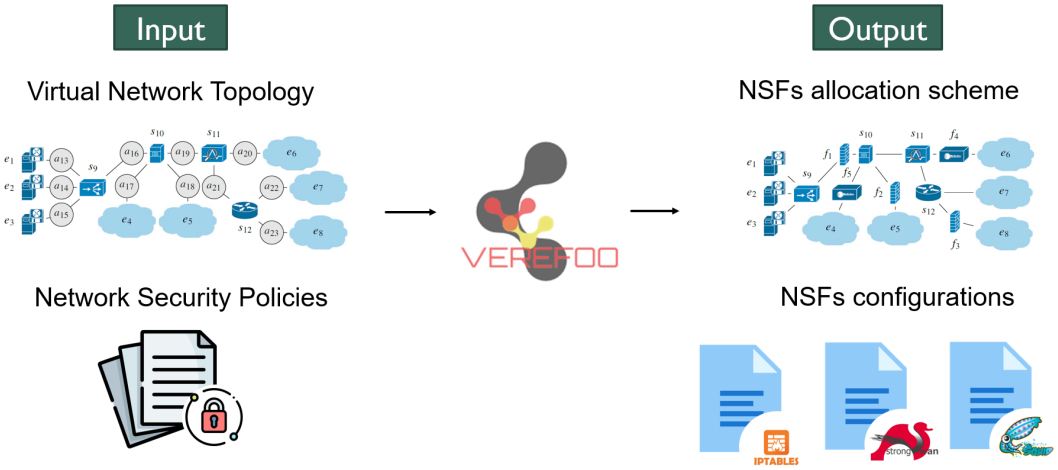


Fig. 4.1 The VEREFOO approach

The VEREFOO approach is characterized by full automation: the human user must only specify the initial inputs of the refinement and optimized orchestration algorithm, which later works by itself without needing any other external intervention. Formal correctness assurance of the NSF allocation scheme and configuration produced by VEREFOO is provided by following a "correctness-by-construction" approach. The adoption of this approach avoids the need of a-posteriori formal verification techniques (e.g., model checking), which would increase the overall computation time. Optimization is pursued by aiming to minimize the number of NSFs that compose the automatically established allocation scheme in the logical topology of the virtual network, and the cardinality of the configuration rule set for each allocated NSF. Minimizing the number of NSFs implies minimizing resource consumption in the network where the NSFs must be later deployed. Instead, minimizing rule set cardinality ensures a two-fold achievement. On the one hand, the minimum amount of memory is used to store the rules. On the other hand, the efficiency of the decisional operations is optimized, as fewer comparisons between the rule conditions and the packet fields are required.

In the reminder of this chapter, Section 4.1 provides more detail about the input specification and output generation, while Section 4.1.3 discusses the design choices that have been taken to model the allocation and configuration problem so as to combine automation, formal verification and optimization.

4.1 Inputs and Outputs of the VEREFOO Approach

4.1.1 Input: Service Graph and Allocation Graph

A *Service Graph* (SG) is the logical topology of a virtual network, i.e., an interconnection of service functions and network nodes providing a complete end-to-end network service. It represents a generalization of a Service Function Chain because the functions do not need to be positioned in a linear combination, but they can be organized within a complex architecture where the traffic can flow through alternative paths, not in a single route. An SG is defined by a network service designer, without involving security considerations. The only purpose is to provide a networking service to the users, whose points of access are represented by the end points of the SG (e.g. clients, servers, subnetworks). The functions that the service designer can exploit for the creation of an SG are *Network Functions* (NFs) implementing various functionalities, such as web caching and load balancing. These functionalities are characterized by different behaviors. Some of them can be neglected when solving the NSF allocation and configuration problem while others must be taken into account:

- Functionalities such as traffic monitoring are able to update internal counters and to send alerts according to the results of an inspection made on the received packets. However, then, they send these packets to the out-ports without modifying them. Therefore, the presence of a traffic monitor does not influence the configuration of NSFs.
- Functionalities such as load balancing forward the traffic flow according to some internal logic, e.g., a load balancer decides the server of the cluster which must receive a packet according to parameters like the current server load. However, in this case too, a load balancer does not influence how the NSF rules are defined, because the rules deal with the end points of the network and it is not possible to say a priori to which server a load balancer will forward a packet. Consequently, it is as though the load balancer had to send a packet to every server of the cluster to check the satisfaction of the security policies.
- Functionalities such as NAT are able to modify some header fields, like replacing private IP addresses with public ones. Because of these actions, the

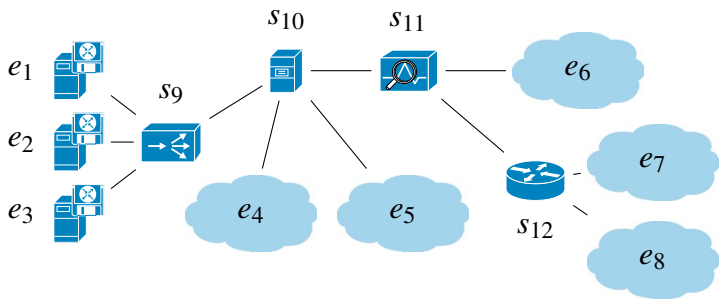


Fig. 4.2 Input Service Graph example

Identifier	IP address	Function type / role
e_1	130.10.0.1	HTTP web server
e_2	130.10.0.2	HTTP web server
e_3	130.10.0.3	HTTP web server
e_4	40.40.41.∗	IT office of Company A
e_5	40.40.42.∗	Business office of Company A
e_6	88.80.84.∗	Company B
e_7	192.168.1.∗	IT office of Company C
e_8	192.168.2.∗	Business office of Company C
s_9	130.10.0.4	Load balancer
s_{10}	33.33.33.2	Web cache
s_{11}	33.33.33.3	Traffic monitor
s_{12}	220.124.30.1	NAT

Table 4.1 IP addresses and function types

rules that should be configured in an NSF could depend on how NATs behave, according to the position in the Service Graph where the NSF is allocated.

- Functionalities such as web caching decide if a packet must be forwarded to some out-ports on the basis of fields that do not involve the IP quintuple, e.g., a web cache forwards a packet received from a web client to a web server only if it does not have a cached copy associated with the requested URL. Their presence may have an impact for the configuration of web-application firewalls, but not for the most common NSFs, such as packet filtering firewalls and VPN gateways.

Low-level functions such as switches and routers, which exclusively forward the incoming packets to the out-ports selected by means of a forwarding or routing table, are not included explicitly in the SG. This statement does not imply that these functions are not present in the real network, but the SG provides a more abstract

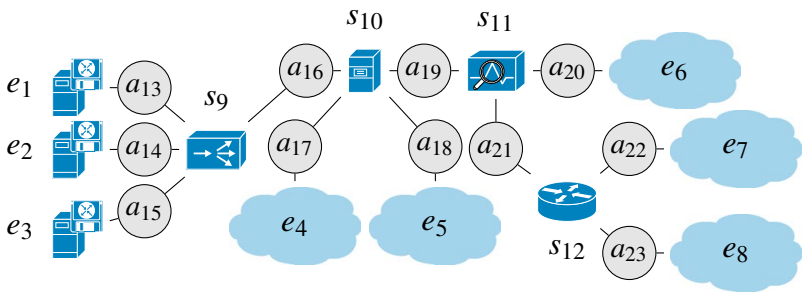


Fig. 4.3 Allocation Graph with Allocation Places

view of the possible paths that packets can follow. This abstraction focuses on the more complex service functions, under the assumption that the low-level ones correctly implement the SG connections [163].

An example of input SG that could be defined by the user of the VEREFOO approach is represented in Fig. 4.2. Alongside with this description of the service topology, the user should also provide more information about the configuration of the functionalities included in the SG, such as their IP addresses. For the example SG, this information is reported in Table 4.1.

The SG provided by the user is automatically processed to create an internal representation called *Allocation Graph* (AG). Without further specifications from the user, for each link between any pair of network nodes or functions, a placeholder element, called *Allocation Place* (AP), is generated. In this position, the approach can decide to put an NSF in order to reach the optimal allocation scheme. However, a security skilled service designer can either force the allocation of an NSF in a specific AP, without allowing it to be removed by the automatic allocation procedure, or prohibit that specific APs are considered as valid NSF positions. This capability enriches the flexibility of the proposed methodology, and at the same time it decreases computation time, by reducing the solution space the algorithm must search to solve the problem. Moreover, it is useful in mixed scenarios, where NSFs are implemented not only by VNFs, but also by existing hardware NSFs, which can, in fact, be modeled as NSFs that cannot be removed. Despite these benefits, on the other hand, it is evident how this manual contribution to the configuration of an AG can lead to the impossibility to find a solution or to an unoptimized solution, because some acceptable – and potentially optimal – solutions can be pruned based on the user input.

An example of AG that is derived from the SG of Fig. 4.2 is represented in Fig. 4.3. The only difference is the presence of an AP in-between each pair of network nodes.

4.1.2 Input: Network Security Policies

The *Network Security Policies* (NSPs) describe the security requirements that must be enforced in the network. The user of the VEREFOO approach can specify them with a medium-level language, which abstracts from the vendor-dependent characteristics of the NSF implementations that must be used to enforce the NSPs. This language does not require a high expertise from the user in terms of network security, so that the VEREFOO approach can also be pursued by a network administrator with limited security skills.

Multiple NSP types can be specified, depending on the security properties that must be guaranteed in the network. As mentioned before, in this dissertation the VEREFOO approach has been designed to work with two NSF types: packet filtering firewalls and VPN gateways. The NSPs that these functions can enforce are respectively connectivity policies and communication protection policies.

Connectivity Policies

Connectivity policies describe isolation and reachability properties for end-to-end communications, i.e., they specify which traffic flows must be blocked by packet filters before reaching their destination, and which ones must be able to reach it.

The VEREFOO approach supports four different profiles for the specification of connectivity policies. Each one is characterized by a *default behavior*, describing how the traffic flows for which no specific policies are formulated must be managed, and a set of specific NSPs, exclusively referred to certain traffic types.

In the first profile (*whitelisting*), the default behavior is set to block traffic flows and the user can only additionally specify *reachability* policies, so that all traffic flows must be blocked except for those that the user explicitly allows. In the second profile (*blacklisting*), vice versa, the default behavior is set to allow traffic flows and the user can only additionally specify *isolation* policies; in this case, all traffic flows must be allowed with the exception of those that the user specifically requests to deny.

Action	IPSrc	IPDst	pSrc	pDst	tProto
Allow	192.168.1.∗	192.168.2.∗	∗	∗	∗
Allow	192.168.2.∗	192.168.1.∗	∗	∗	∗
Allow	192.168.1.∗	130.10.0.∗	∗	80	TCP
Deny	192.168.1.∗	130.10.0.∗	∗	≠80	TCP
Deny	192.168.1.∗	130.10.0.∗	∗	∗	UDP
Deny	192.168.2.∗	130.10.0.∗	∗	∗	∗
Allow	130.10.0.∗	192.168.1.∗	∗	∗	∗
Allow	40.40.41.∗	130.10.0.∗	∗	80	TCP
Deny	40.40.41.∗	130.10.0.∗	∗	≠80	TCP
Deny	40.40.41.∗	130.10.0.∗	∗	∗	UDP
Deny	40.40.42.∗	130.10.0.∗	∗	∗	∗
Allow	130.10.0.∗	40.40.41.∗	∗	∗	∗
Allow	40.40.42.∗	40.40.41.∗	∗	∗	∗
Deny	40.40.41.∗	40.40.42.∗	∗	∗	∗
Allow	88.80.84.∗	40.40.∗.∗	∗	∗	∗
Deny	88.80.84.∗	130.10.0.∗	∗	∗	∗

Table 4.2 Example set of connectivity policies

The other two profiles, called *rule-oriented specific* and *security-oriented specific*, let the user explicitly formulate both specific isolation and reachability policies, but without manually setting a default behavior. The way the VEREFOO approach manages all the other cases, which the user is not interested in, is automatically decided, in order to achieve some other goals. In the *rule-oriented specific* approach, the goal is only to minimize the number of rules. In the *security-oriented specific* profile, the system allows only the communications that are strictly necessary in order to satisfy all user requirements, according to the least-privilege principle.

In all the four profiles, the specific NSPs are characterized by an action (deny or allow) and a set of conditions, expressed on the fields of the IP packet and identifying the traffic flows on which the action requested in the NSP must be enforced. As we are considering the allocation and configuring of packet filters, these conditions can be expressed on the IP 5-tuple. An example set of specific connectivity policies, which may be specified for a *rule-oriented specific* or *security-oriented specific* profile, is shown in Table 4.2. The main difference related to which profile is actually chosen concerns the outcome of the VEREFOO approach. If the *rule-oriented specific* profile is selected for this set of connectivity policies, the rules of the allocated firewalls are automatically computed so as to be as few as possible. Instead, if the *security-oriented specific* profile is selected, the number of allowed traffic flows is minimized in the following way: (i) if an allocated firewall has a

whitelisting configuration, the allowing rules must allow only the required traffic flows; (ii) if the firewall has a blacklisting configuration, the denying rules should block the largest number of flows.

Communication Protection Policies

Communication protection policies describe which security properties related to the generation of VPNs (i.e., confidentiality, integrity) must be enforced on the traffic flows crossing a network.

The VEREFOO approach supports two profiles for the specification of communication protection policies. The *min-allocation* profile aims to allocate the least number of VPN gateways. This achievement would be useful in both traditional and virtualized networks. In the former, less middleboxes would be bought and manually installed. In the latter, less resources of the general-purpose servers would be used for deploying virtual functions. This objective can be achieved by promoting the generation of end-to-end VPNs where end points make their communications secure by themselves, instead of site-to-site VPNs. The counterbalance of this criterion is that in end-to-end VPNs bandwidth consumption is higher, as the traffic flows that are identified by the security policies are protected in their whole paths, even when it is not strictly necessary. The *min-bandwidth* profile, instead, aims to improve the performance of network communications in terms of bandwidth. This objective consists in generating an allocation scheme where VPN gateways are preferred over end points for covering the role of VPN gateway. This preference causes protection to be enforced for as short as possible paths, therefore saving bandwidth. The counterbalance of this criterion is that resource consumption may result higher, as it may be necessary to install a number of middleboxes for VPN generation higher than necessary.

For each profile, a set of specific communication protection policies must be specified. In particular, each communication protection policy is characterized by the following elements:

- the condition set, representing the packet features that enable the identification of the traffic flows to protect;

Policy conditions	Algorithms for confidentiality	Algorithms for integrity	Enforcement modes	Trustworthiness and Inspection
IPSrc = 192.174.1.∗, IPDst = 144.14.2.∗, pSrc = ∗, pDst = 80, tProto = ∗	{AES-GCM-256, 3DES-CBC}	∗	(true, true, false)	N^U = {125.22.2.2}, N^I = {128.28.8.5}
IPSrc = 122.33.33.3, IPDst = 12.67.84.2, pSrc = 110, pDst = ∗, tProto = ∗	{NULL}	{HMAC-SHA-512}	(false, true, d.c.)	L^U = {link between 55.44.33.22 and 55.44.33.27}

Table 4.3 Examples of Communication Protection Policies

- the security properties to be applied on the matching flows (e.g., confidentiality, integrity), together with the algorithms to be applied for their enforcement (e.g., AES-128-CBC, HMAC-SHA-512);
- information about the VPN enforcement modes, i.e., whether confidentiality must be enforced on the header and payload of the original packet (or only payload if there is encapsulation), whether integrity must be enforced on the header and payload of the original packet (or only payload if there is encapsulation), and whether integrity must be enforced on the header of the encapsulating packet (if there is encapsulation).
- information about the trustworthiness of the network, i.e., the set of untrusted nodes and links, where no trust assumption can be made, and the set of inspector nodes, where the crossing traffic is required to be plain, so that it can be analyzed by them.

A pair of examples of communication protection policies are shown in Table 4.3.

4.1.3 Expected outcome

After receiving the AG and the NSPs, the security allocation and configuration problem is automatically solved.

In case of positive outcome, the provided result is composed of (i) the allocation scheme of the NSFs in the input SG; (ii) the configuration of each allocated NSF. The NSF allocation scheme specifies the APs where each NSF has to be allocated. The configuration of each allocated NSF specifies its configuration rules (e.g., the filtering rules for a firewalls, or the communication protection rules for a VPN gateway). The allocation scheme contains the minimum number of NSFs required to enforce all NSPs, so minimizing resource consumption, while the configuration of each

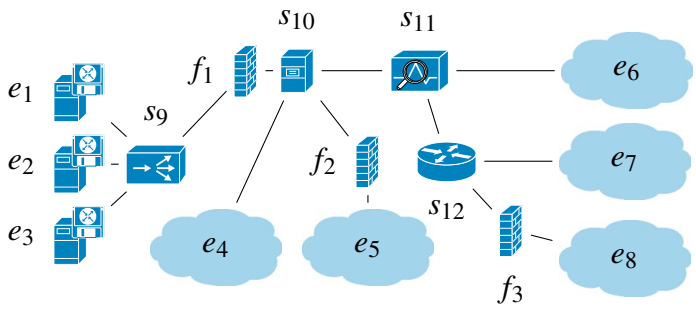


Fig. 4.4 Final Service Graph with allocated firewalls

allocated NSF contains the minimum number of configured rules, thus minimizing the amount of memory needed to store them and maximizing the NSF performance.

The allocation scheme is only generated at the logical abstraction level represented by the SG. It must not be confused with the embedding scheme in the physical network, which is defined at a later stage by solving a classical *Virtual Network Embedding* (VNE) problem [35] (i.e., defining how the virtual functions are placed in the physical hosts). Hence, the output solution can be deployed automatically into the virtual network by means of existing technologies. After deployment, in case a new security configuration becomes necessary, e.g., to react to an attack that has just been detected, new NSPs have to be defined by the administrator and provided to the tool, which will then automatically compute the new configuration to be deployed.

Instead, if no solution to the problem can be found, a non-enforceability report is generated for the user, who can try to guess why it has not been possible to enforce the NSPs. A possible reason for a negative outcome can be that the SG defined by the service designer does not provide adequate APs for the NSFs because of some user-defined constraints set about their generation. Hence, one possible strategy to get a working solution is to run the procedure again, releasing some of the user-defined constraints.

An example of outcome that is produced by the VEREFOO approach to enforce the connectivity policies listed in Table 4.2 with a *security-oriented specific* profile in the AG of Fig. 4.3 is shown in Fig. 4.4 (allocation) and in Table 4.4 (configuration). Even in a network of this size, a manual approach is prone to both human errors and sub-optimizations. For example, human beings may fail to understand that allocating a firewall in a_{16} would avoid the need of having firewalls in several other APs (e.g., a_{13}, a_{14}, a_{15}, and a_{21}). They may also fail in setting up correct rules that take into

Firewall fw_1

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	220.124.30.1	130.10.0.4	∗	80	TCP
2	Allow	40.40.41.∗	130.10.0.4	∗	80	TCP
3	Allow	130.10.0.4	∗.∗.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall fw_2

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	40.40.42.∗	40.40.41.∗	∗	∗	∗
2	Allow	88.80.84.∗	40.40.42.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall fw_3

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	∗.∗.∗.∗	192.168.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Table 4.4 Filtering Policy rules for allocated firewalls

account all the possible changes the NAT s_{12} or the load balancer s_9 may apply to the traffic. For example, let us assume the human administrator has decided to allocate a firewall in AP a_{16} with default action deny. In order to satisfy the connectivity policy that allows all TCP traffic from 192.168.1.∗ to 130.10.0.∗ port 80, it would be wrong to install the rule (Allow, 192.168.1.∗, 130.10.0.∗, ∗ , 80, TCP) in this firewall, because the NAT s_{12} changes the source address. The right solution, instead, is rule # 1 for fw_1, as shown in Table 4.4. These issues increase dramatically when the network topology size and the number of policies increase.

4.2 MaxSMT problem formulation

The formally correct and optimal solution for the automatic NSF allocation and configuration problem is obtained by formulating and solving a partial weighted MaxSMT problem, which is a generalization of the traditional SMT problem.

4.2.1 The SMT problem

The SAT problem, also called propositional satisfiability problem, is a traditional problem in computer science, whose goal is to determine, given a propositional

formula containing a set of propositional boolean variables (i.e., variables that can assume as possible values only true or false), if a combination of values for these variables exists to satisfy the formula, i.e., to make it true. An SAT solver, for this reason, needs just to find a single solution, among all the possible ones that may exist, in order to state that the input formula is satisfiable.

The language which is exploited by SAT solvers is the traditional boolean logic. A generalization is represented by the *Satisfiability Modulo Theories* (SMT) problem, where the language is instead the first-order logic, which includes the boolean operations as a specific case, but it can use several other theories, such as theories of real numbers, integers, lists, arrays, bit vectors and many other data structures. Basically, an SMT problem is composed of a set of predicates, where each predicate is a binary function defined over non-binary variables. Consequently, the SMT language is much richer than the SAT language, it allows to express more complex models and it represents a formalized approach to constraint programming for constraint satisfaction problems.

To solve SMT problems, it is possible to use efficient state-of-the-art solvers such as Z3 [164], which integrate search pruning methods, and also heuristic combinations of algorithmic proof methods called tactics. As such, they are characterized by several parameters whose correct tuning allows achieving good performance.

4.2.2 The MaxSMT problem

The *Maximum Satisfiability Modulo Theories* (MaxSMT) problem is an extension of the SMT problem in the optimization context, where given a set of clauses (i.e., predicates) containing predicate variables, the goal is to find an assignment of these variables that maximizes the number of satisfied clauses in the set. The main difference with respect to an SMT problem is that it is not sufficient to find a solution that satisfies the predicate clauses, but, among all the possible solutions, one that maximizes the number of satisfied clauses must be found.

Like the SAT and SMT problems, the MaxSMT problem is NP-*complete* in terms of worst-case computational complexity [165]. However, the NP-completeness only implies exponential time for the worst case, but the actual time for solving a MaxSMT instance is often less than the worst case time, also depending on which theories are used in the formulas. In particular, many instances of this problem can be

solved in polynomial time with respect to the problem dimension [166]. Therefore, modeling real problems such as the security configuration problem and automatically solving them can be a viable path to face them.

The MaxSMT problem formulation can have different variants. The most common ones are:

- the *weighted MaxSMT*, where a different weight can be assigned to each clause, and, consequently, the search for the best solution prioritizes the satisfiability of the most valued clauses;
- the *partial MaxSMT*, where some constraints are not relaxable because they must be satisfied, while other clauses do not require to be necessarily satisfied for the achievement of a valid solution;
- the *weighted partial MaxSMT*, which means weighted and partial.

Considering a weighted partial MaxSMT problem characterized by a set S of relaxable clauses s_i, also called *soft constraints*, and a set H of non-relaxable clauses h_j, called *hard constraints*, a formal definition of the problem can be given as:

$$max \sum_{i=1}^{s} w_i * s_i$$

$$\text{subject to } h_j, \forall j \in [1, H]$$

4.2.3 Advantages of the partial weighted MaxSMT formulation

The partial weighted MaxSMT formulation is key to achieve all the three main objectives of the VEREFOO approach: full automation, optimization, and formal correctness. Full automation is achieved because a MaxSMT problem can be solved without human intervention, except for the input specification. Optimization can be achieved by expressing the optimization objectives by means of soft constraints, and formal correctness can be achieved by expressing the formal correctness requirements as hard constraints. Adopting this formal correctness-by-construction approach is beneficial not only because it improves the assurance and confidence that the computed solution is correct, but also because it avoids performing a-posteriori formal verification. Indeed, the solution can already be considered formally correct

as far as all problem components are correctly modeled, being fundamental that such models capture all the information that may influence the correctness of the solution. Specifically, such models must capture both the security requirements and the forwarding behavior of the network where they must be enforced. At the same time, the number and complexity of constraints in the MaxSMT problem must be kept limited, in order to make the approach scalable. For all the above reasons, the modeling of the problem components, when formulating the MaxSMT problem, represents a big challenge.

In light of these considerations, it is clear that the MaxSMT problem formulation is intrinsically tied to the modeling of network components and security policies. For this reason, the next chapters of the dissertation provide the full definition of the MaxSMT problem starting from the definition of the models, respectively to solve the auto-configuration problem for firewalls in Chapter 5 and for VPNs in Chapter 6.

Chapter 5

Automatic Firewall Configuration

This chapter describes the application of the VEREFOO approach to the automatic allocation and configuration of packet filtering firewalls. TABLE 5.1 includes the main formal notations (symbols, functions, predicates, operators) used in this chapter.

5.1 Network and Policy Models

This section defines the formal models of the main components of the auto-configuration problem (i.e., SG with its network functions, NSPs, traffic flows). These models are later used to formulate the hard and soft constraints of the MaxSMT problem.

5.1.1 Service and Allocation Graph models

An SG is modeled as a directed graph $G_S = (N_S, L_S)$ where N_S is the set of vertices, representing the network nodes of the SG, while L_S is the set of edges, representing directed connections between nodes. N_S is the union of two disjoint sets, i.e., $N_S = E_S \cup S_S$, where E_S is the set of the end points (i.e., single hosts or edge subnetworks), while S_S is the set of intermediate service functions.

Each element of N_S is uniquely identified by a non-negative integer index k, and n_k denotes the element of N_S identified by k, so that each element of L_S is uniquely identified by a pair of non-negative integers, i.e., $l_{i,j} \in L_S$, with $i \neq j$, is the edge from n_i to n_j. The function $index(n_k) = k$ is then defined. Moreover, each $n_k \in N_S$

Symbol/Function/Predicate/Operator	Definition
$\mathbb{B} = \{\text{true, false}\}$	boolean set
$G_S = (N_S, L_S)$, $G_A = (N_A, L_A)$	directed Service Graph (SG) and Allocation Graph (AG)
E_S, E_A, S_S, S_A	end points, middleboxes
$n_k \in N_S$	the element of N_S identified by k
$n_s, n_d \in N_A$	source/destination endpoint
$l_{i,j} \in L_S$	the edge from n_i to n_j
A_A	the set of the APs where FWs can be potentially placed
t, t^0	a class of packets and empty set of packets
$t_{i,j}$	the traffic transmitted from n_i to n_j
$\mathscr{I}_i^d$, $\mathscr{I}_i^a$	packets that are dropped/allowed in n_i
$\mathscr{D}_{i,j}$	traffic that is transformed by $\mathscr{T}_{i,j}$
$f \in F$	a flow, i.e., class of packets generated by n_s
$r \in R_s$	Network Security Requirement element
$r = (C, a)$	C is a condition set , a is the action (i.e, Allow/Deny)
$A_T = \{\text{DENY}, \text{ALLOW}\}$	set indicating isolation/reachability requirements
P_h	set of all the placeholder rules
U_h	set of effectively configured rules
F_r^M	flows that are not subflows of any other flow in F_r
α: $N_S \to 2^I$	maps $n \in N_S$ to its set of IP addresses
$\mathscr{T}_i$: $T \to T$	maps an input traffic to the corresponding output traffic
$\mathscr{T}_{i,j}$: $T \to T$	maps part of $\mathscr{D}_{i,j}$ to the corresponding output traffic
π: $F \to (N_A)^*$	maps a flow to the ordered list of nodes that are crossed by that flow
τ: $F \times N_A \to T$	maps a flow and a node to the ingress traffic
ν: $N_A \times F \to N_A + \{n_0\}$	maps a network node n and a traffic flow f to the next node crossed by f after n
$\sigma : A_A \to \mathbb{Z}$	model the sign of the weights
allocated: $N_A \to \mathbb{B}$	true $\Leftrightarrow$ a FW is allocated in a_h
forbidden: $L_S \to \mathbb{B}$	true $\Leftrightarrow$ the creation of an AP on $l_{i,j}$ prohibited
forced: $L_S \to \mathbb{B}$	true $\Leftrightarrow$ allocation of a firewall on $l_{i,j}$ is required
deny$_i$: $T \to \mathbb{B}$	true $\Leftrightarrow$ n_i drops all the packets
wlst: $A_A \to \mathbb{B}$	true $\Leftrightarrow$ the def. act. of FW allocated in the AP is DENY
enforces: $A_T \times R \to \mathbb{B}$	true $\Leftrightarrow$ the FW def. act. enforces requirement r
configured: $P_h \to \mathbb{B}$	true $\Leftrightarrow$ the placeholder rules included in U_h
matchAll: $P_h \times Q \to \mathbb{B}$	true $\Leftrightarrow$ the rule conditions completely match the 5-tuple
matchNone: $P_h \times Q \to \mathbb{B}$	true $\Leftrightarrow$ the rule conditions do not match any 5-tuple fields
$t_1 \subseteq t_2 \in T$	t_1 is a sub-traffic of t_2
$\wedge$, $\vee$, $\neg$	used for conjunction, disjunction, negation
.	used to denote a specific tuple element (e.g., given a tuple $t = (a,b,c)$, $t.a$ identifies element a of tuple t)

Table 5.1 Notation

is characterized by a single IP address, an IP address range or, more generically, a set of IP addresses. Let us denote I the set of all IP addresses and α: $N_S \rightarrow 2^I$ the function that maps each element $n \in N_S$ to its set of IP addresses.

An AG is modeled similarly to an SG, as a directed graph $G_A = (N_A, L_A)$, with the same indexing scheme for vertices and edges already used for the SG. In this case, however, N_A is the union of 3 disjoint subsets: $N_A = E_A \cup S_A \cup A_A$, where E_A and S_A represent, respectively, the end points and the middleboxes, while A_A is the set of the APs where the firewalls can be potentially placed.

When the AG is automatically generated from the SG, taking into account the set of additional requirements about the allocation of firewall instances provided by the service designer, end points and service functions are not modified, i.e., $E_A = E_S$ and $S_A = S_S$. Similarly, as the assignment of IP addresses does not change from SG to AG, α is simply lifted to be a partial function on the N_A domain.

Let $\mathbb{B} = \{\text{true, false}\}$ denote the Boolean set, and *allocated*: $N_A \rightarrow \mathbb{B}$ be a predicate that formalizes allocation decisions, by specifying if each network node is actually allocated in the AG. For each $n_k \in E_A \cup S_A$, $\mathit{allocated}(n_k)$ is true by definition, whereas for each $a_h \in A_A$, the automatic procedure decides whether $\mathit{allocated}(a_h)$ has to be true or not (i.e., whether a firewall has to be allocated in a_h or not).

For each $l_{i,j} \in L_S$, $i \neq j$, the requirements about the possible allocation of firewalls coming from the service designer are formally represented by two predicates: *forbidden*: $L_S \rightarrow \mathbb{B}$ and *forced*: $L_S \rightarrow \mathbb{B}$. For each $l_{i,j} \in L_S$, $\mathit{forbidden}(l_{i,j})$ is true if and only if the creation of an AP on $l_{i,j}$ has been prohibited, while $\mathit{forced}(l_{i,j})$ is true if and only if the allocation of a firewall on $l_{i,j}$ has been required. The constraint (5.1) expresses that the two requirements cannot coexist for the same $l_{i,j}$.

$$\forall\, l_{i,j} \in L_S.\, \neg(\mathit{forbidden}(l_{i,j}) \wedge \mathit{forced}(l_{i,j})) \tag{5.1}$$

According to the requirements expressed by the *forbidden* predicate, A_A and L_A are computed as the smallest sets that satisfy the conditions (5.2) and (5.3).

$$\forall\, l_{i,j} \in L_S.\, (\neg \mathit{forbidden}(l_{i,j}) \implies a_h \in A_A \wedge l_{i,h} \in L_A \wedge l_{h,j} \in L_A) \tag{5.2}$$

$$\forall\, l_{i,j} \in L_S.\, (\mathit{forbidden}(l_{i,j}) \implies l_{i,j} \in L_A) \tag{5.3}$$

According to (5.2), for each SG edge $l_{i,j}$, if the creation of an AP on it is not prohibited, i.e., if $\mathit{forbidden}(l_{i,j}) = \text{false}$, an AP a_h is added to the AG between nodes n_i and n_j, i.e., it is included in A_A, and it is connected by the edges $l_{i,h} \in L_A$ and

$l_{h,j} \in L_A$, replacing edge $l_{i,j} \in L_S$. Instead, according to (5.3), no AP is created on $l_{i,j}$ if it is prohibited, i.e., if $forbidden(l_{i,j}) = \text{true}$. In this case, $l_{i,j}$ is simply included in L_A. Note that, if both $forbidden(l_{i,j}) = \text{true}$ and $forbidden(l_{j,i}) = \text{true}$, a single AP a_h is created for them .

Finally, (5.4) is used to force the allocation of a firewall in the AP a_h created on link $l_{i,j}$ when the user requests it.

$$\forall l_{i,j} \in L_S.\ (forced(l_{i,j}) \implies allocated(a_h)) \tag{5.4}$$

5.1.2 Traffic and Network Functions models

A class of packets, also called traffic, t, is modeled as a predicate defined over the values of the TCP/IP 5-tuple packet fields. More precisely, t is modeled as a disjunction of predicates $q_{t,1} \vee q_{t,2} \vee ... \vee q_{t,n}$, where each $q_{t,i}$ is defined over the 5-tuple fields. A packet belongs to class t if and only if its 5-tuple satisfies at least one $q_{t,i}$. In order to keep the model simple but at the same time fairly general, it is assumed that each $q_{t,i}$ is the conjunction of five predicates, one for each field of the 5-tuple. For simplicity, each $q_{t,i}$ is written as

$$q_{t,i} = (IPSrc, IPDst, pSrc, pDst, tPrt) \tag{5.5}$$

where *IPSrc*, *IPDst*, *pSrc*, *pDst* and *tPrt* are the 5 predicates.

Considering IPv4 addresses, it is assumed that *IPSrc* and *IPDst* are conjunctions of four predicates, one for each byte of the IP address. Each one of these four predicates can identify either a single integer value or a range of values, not exceeding the range 0 to 255. The predicates that make up *IPSrc* or *IPDst* are concisely written by means of the dotted-decimal notation $ip_1.ip_2.ip_3.ip_4$, where ip_i is a single decimal number or a range of values, written $[ip_{i,l}, ip_{i,h}]$. The range [0, 255] is concisely represented by the wildcard $*$. If ip_i is a range, the predicates on its right must be $*$. For example, $IPSrc = 130.192.5.*$ stands for the predicate $x_1 = 130 \wedge x_2 = 192 \wedge x_3 = 5$, where x_i is the variable representing the i-th byte of the source IP address packet field, and this predicate identifies all the IP addresses matching 130.192.5.0/24.

The predicates about source and destination ports *sPort* and *dPort* can identify either a single integer number or a range of values, not exceeding the range 0 to 65535, and the same notation used for each byte of an IP address is also used for

the port number, with the range [0, 65535] symbolized by the wildcard $*$. For example, 80 stands for the predicate $x = 80$ and $[80, 100]$ stands for the predicate $x <= 100 \wedge x >= 80$ where x is the variable that represents the port field.

The predicate about the transport-level protocol *tPrt* can identify a single value or a subset of values among a finite set of possible values (e.g., a set including the "TCP" and "UDP" values). The set of all the possible values in this set is concisely symbolized by the wildcard $*$.

Finally, the special symbol t^0 identifies the empty set of packets, i.e., $t^0 = \text{false}$, which means absence of traffic.

Let us denote Q the set of all the predicates $q_{t,i}$ that can be specified with the above notation, and T the set of all the disjunctions of such predicates, i.e., the set of all packet classes t that can be represented by this model. It can be proved that T is closed under conjunction, disjunction and negation. Given two traffic predicates $t_1, t_2 \in T$, t_1 is said to be a sub-traffic of t_2, written $t_1 \subseteq t_2$, if t_1 represents a subset of the packets represented by t_2, i.e., if $t_1 \Rightarrow t_2$.

Each VNF in the SG acts on its input traffic and generates a corresponding output traffic. Its behavior, which depends on its code and configuration, is modeled abstractly by means of two functions, capturing respectively the forwarding behavior (i.e., which input packets are dropped by the VNF) and the transformation behavior (i.e., which packets may be output by the VNF for each class of input packets).

The function that models the forwarding behavior of the VNF in node $n_i \in N_A$ is the predicate $deny_i$: $T \rightarrow \mathbb{B}$ which is true for ingress traffic $t \in T$, if and only if n_i drops all the packets represented by t. Moreover, for node n_i, the traffic $\mathscr{I}_i^d$ specifies the packets that are dropped by the function, i.e., $deny_i(t)$ is true if and only if $t \subseteq \mathscr{I}_i^d$. Instead, the traffic $\mathscr{I}_i^a$ is the complement of $\mathscr{I}_i^d$, since it specifies the packets that are allowed (i.e., not dropped) by the function. Clearly, $\mathscr{I}_i^d \vee \mathscr{I}_i^a = \text{true}$ and $\mathscr{I}_i^d \wedge \mathscr{I}_i^a = \text{false}$.

The transformation behavior of the VNF in node $n_i \in N_A$ is instead modeled by the function $\mathscr{T}_i$: $T \rightarrow T$, called transformer, which maps an input traffic to the corresponding output traffic.

One may argue that function $\mathscr{T}_i$ alone would be enough, e.g., by setting $\mathscr{T}_i(t) = t^0$ for all t such that $deny_i(t)$ is true. However, keeping these two functions distinct and independent brings some advantages to the proposed approach. Note that a

transformer describes traffic transformations independently of whether packets are dropped or not. For example, a firewall can be characterized simply as a VNF having $\mathscr{T}_i(t) = t$ (i.e., $\mathscr{T}_i$ is the identity function, because the firewall does not modify the forwarded traffic), and a $deny_i(t)$ predicate that is true if each packet represented by t is dropped according to the firewall rules. With this separation of duties, it is possible to first compute how traffic is transformed when crossing the network, and then reason about firewall configurations by using the $deny_i$ predicates only.

For many VNFs, $\mathscr{T}_i$ is the identity function, and the following constraint is applied to the *deny* predicate:

$$deny_i(t) = \text{false} \tag{5.6}$$

This simple model applies, for example, to all traffic monitoring functions, because they just inspect packets and forward them without modification. However, the same model also applies to load balancers, because they forward each packet without modification to a destination decided each time based on some internal logic. As it is impossible to know the outcome of this decision beforehand, and all decisions are possible, a load balancer can be modeled as a function that can forward each packet to each destination.

An example of a VNF with a non-identity transformer is a Network Address Translator (NAT)[1]. A NAT performs one of two different transformations selected according to the features of the input packet: if the source address belongs to the set of shadowed addresses, while the destination address does not, the source address is translated into one of the public addresses of the NAT (shadowing). If instead the source address does not belong to the set of shadowed addresses and the destination address is one of the public addresses of the NAT, the destination address is translated into one of the shadowed addresses (reconversion). In all other cases, the packet is not modified. When, as in this case, the function operates different transformations for different packet classes, the transformer can be expressed as $\mathscr{T}_i(t) = \vee_j(\mathscr{T}_{i,j}(\mathscr{D}_{i,j} \wedge t))$, where $\mathscr{T}_{i,j}: T \to T$ is the transformer applied for the packet class defined by predicate $\mathscr{D}_{i,j}$. In the case of NAT, we have $\mathscr{T}_i(t) = \mathscr{T}_{i,1}(\mathscr{D}_{i,1} \wedge t) \vee \mathscr{T}_{i,2}(\mathscr{D}_{i,2} \wedge t) \vee \mathscr{T}_{i,3}(\mathscr{D}_{i,3} \wedge t)$, where $\mathscr{T}_{i,1}$ is the shadowing transformer, $\mathscr{T}_{i,2}$ is the reconverting transformer and $\mathscr{T}_{i,3}$ is the identity transformer that is applied in all other cases. Let us denote $p_1, ..., p_m$ the predicates representing the shadowed IP addresses and $a_1, ..., a_l$ the predicates representing the public IP addresses of the NAT. Then,

[1] In this example, a NAT which can perform only a simple address translation is considered, without the feature of port translation.

considering a generic traffic $t = \vee_{k=1}^{h}(q_{t,k})$, the predicates $\mathscr{D}_{i,j}$ and the transformers $\mathscr{T}_{i,j}$ can be defined as

$$\mathscr{D}_{i,1} = \vee_{x=1}^{m}(p_x, \neg(\vee_{z=1}^{m}(p_z)), *, *, *) \tag{5.7}$$

$$\mathscr{T}_{i,1}(t) = \vee_{y=1}^{l} \vee_{k=1}^{h} (a_y, q_{t,k}.IPDst, q_{t,k}.pSrc, q_{t,k}.pDst, q_{t,k}.tPrt) \tag{5.8}$$

$$\mathscr{D}_{i,2} = \vee_{y=1}^{l}(\neg(\vee_{x=1}^{m}(p_x)), a_y, *, *, *) \tag{5.9}$$

$$\mathscr{T}_{i,2}(t) = \vee_{x=1}^{m} \vee_{k=1}^{h} (q_{t,k}.IPSrc, p_x, q_{t,k}.pSrc, q_{t,k}.pDst, q_{t,k}.tPrt) \tag{5.10}$$

$$\mathscr{D}_{i,3} = \neg(\mathscr{D}_{i,1}) \wedge \neg(\mathscr{D}_{i,2}) \tag{5.11}$$

$$\mathscr{T}_{i,3}(t) = t \tag{5.12}$$

5.1.3 Traffic flows model

The transformation behavior of an entire AG is described by means of its set of traffic flows F. Specifically, each flow $f \in F$ represents a class of packets that are generated by a source endpoint $n_s \in N_A$, directed to a destination endpoint $n_d \in N_A$, and steered to pass through an ordered list of intermediate nodes $n_a, n_b, ... \in N_A$ that may forward them at each hop, possibly changing them (e.g., an intermediate NAT can change packet addresses), or drop them. Accordingly, a flow is formally modeled as a list $[n_s, t_{s,a}, n_a, t_{a,b}, n_b, ..., n_k, t_{k,d}, n_d]$, where $t_{i,j}$ represents the traffic (i.e., class of packets) transmitted from n_i to n_j, and each $t_{i,j}$ is the result of the transformation of the previous traffic in the flow by node n_i, i.e., $\forall t_{i,j} \neq t_{s,a}.\ t_{i,j} = \mathscr{T}_i(t_{k,i})$, where n_k is the node that precedes n_i in the flow. Moreover, each $t_{i,j}$ is homogeneous for node n_j. This means that all the packets it represents are handled in the same way by n_j, i.e., either all of them or none of them are dropped and, if n_j applies different transformations to different classes of packets (e.g., n_j is a NAT), they belong all to the same class.

Alongside with this definition, three auxiliary functions are introduced to characterize the flows of an AG:

- π: $F \rightarrow (N_A)^*$, which maps a flow to the ordered list of network nodes that are crossed by that flow, including the destination, but not the source;
- τ: $F \times N_A \rightarrow T$, which maps a flow and a node to the ingress traffic of that node belonging to that flow. In case flow f does not cross node n, we have $\tau(f, n) = t^0$;

- ν: $N_A \times F \rightarrow N_A + \{n_0\}$, which maps a network node n and a traffic flow f to the next node crossed by f after n. In case n is not in f or is the last node, this function returns n_0, a symbol representing no node ($n_0 \notin N_A$).

In order to better clarify the traffic flow concept, let us use the example of Fig. 4.3. A flow from network e_7, shadowed by NAT s_{12}, to TCP port 80 of web server e_1, located behind load balancer s_9, can be represented as f_1 = [e_7, $t_{7,22}$, a_{22}, $t_{22,12}$, s_{12}, $t_{12,21}$, a_{21}, $t_{21,11}$, s_{11}, $t_{11,19}$, a_{19}, $t_{19,10}$, s_{10}, $t_{10,16}$, a_{16}, $t_{16,9}$, s_9, $t_{9,13}$, a_{13}, $t_{13,1}$, e_1], and each traffic is characterized by a single 5-tuple:

$$t_{7,22} = t_{22,12} = (192.168.1.*, 130.10.0.4, *, 80, \text{TCP})$$

$$t_{12,21} = t_{21,11} = t_{11,19} = t_{19,10} = t_{10,16} = t_{16,9} = (220.124.30.1, 130.10.0.4, *, 80, \text{TCP})$$

$$t_{9,13} = t_{13,1} = (220.124.30.1, 130.10.0.1, *, 80, \text{TCP})$$

Note that the source IP address is modified after f_1 crosses the NAT s_{12}, while the destination IP address is changed by the load balancer s_9. For flow f_1 we have $\pi(f_1)$=[a_{22}, s_{12}, a_{21}, s_{11}, a_{19}, s_{10}, a_{16}, s_9, a_{13}, e_1]. Moreover, we have, for example, $\tau(f_1, a_{19}) = \tau(f_1, s_9)$, $\tau(f_1, a_{18}) = t^0$, $\nu(a_{22}, f_1) = s_{12}$, $\nu(s_{11}, f_1) = a_{19}$, and $\nu(a_{13}, f_1) = e_1$.

Given two flows f_1,f_2, f_1 is said a subflow of f_2, written $f_1 \subseteq f_2$, if f_1 and f_2 pass through the same list of nodes and for each one of such nodes the ingress traffic of f_1 is a subset of the ingress traffic of f_2, i.e., $\pi(f_1) = \pi(f_2)$ and $\forall n \in \pi(f_1).\tau(f_1, n) \subseteq \tau(f_2, n)$.

5.1.4 Network Security Policy model

The NSPs relevant for firewall allocation and configuration are those expressing isolation and reachability properties. Such policies are modeled by a default behavior D, which is an element of the set {blacklisting, whitelisting, rule-oriented-specific, security-oriented-specific}, and a set of specific Network Security Requirements (NSRs), R_s. Each NSR $r \in R_s$ is expressed in medium-level language as a pair $r = (C, a)$, where C is a condition and a is the action that must be performed on the flows that satisfy C.

Before defining what are the flows that satisfy a condition, let us refine the definition of C. Each condition C that occurs in a NSR is a predicate similar to the

ones defined for modeling packet classes, i.e., $C = (IPSrc, IPDst, pSrc, pDst, tPrt)$. The predicates *IPSrc* and *pSrc* specify the traffic sources the NSR refers to. Instead, the predicates *IPDst*, *pDst*, and *tPrt* specify the traffic destinations and the protocol the NSR refers to.

A flow $f = [e_s, t_{s,a}, ..., t_{k,d}, e_d]$ satisfies C if the following three conditions are satisfied:

1. its source and destination endpoints e_s, e_d have IP addresses matching *IPSrc* and *IPDst* respectively, i.e., $\alpha(e_s) \subseteq C.IPSrc$ and $\alpha(e_d) \subseteq C.IPDst$;
2. its source traffic satisfies *IPSrc* and *pSrc*, i.e., $t_{sa} \subseteq (C.IPSrc, *, C.pSrc, *, *)$;
3. its destination traffic satisfies *IPDst*, *pDst*, and *tPrt*, i.e.,

   $t_{kd} \subseteq (*, C.IPDst, *, C.pDst, C.tPrt)$.

Let then $F_r \subseteq F$ denote the set of flows that satisfy $r.C$. From these definitions, it follows that all the subflows of a flow that is in F_r are in F_r too.

Each action a is one of the two elements of the set $A_T = \{\text{DENY}, \text{ALLOW}\}$. If $r.a = \text{DENY}$, r is an isolation requirement, meaning that all flows that satisfy $r.C$ must be blocked, i.e., their destination must be reached by no packet of the flow, otherwise r is a reachability requirement, meaning that at least one flow that satisfies $r.C$ is blocked (i.e., at least one class of its packets is allowed to reach its destination).

Formally, an isolation requirement r can be translated into the following logical formula:

$$\forall f \in F_r. \exists i. (n_i \in \pi(f) \wedge allocated(n_i) \wedge deny_i(\tau(f, n_i))) \tag{5.13}$$

while a reachability requirement r can be translated into

$$\exists f \in F_r. \forall i. (n_i \in \pi(f) \wedge allocated(n_i) \implies \neg deny_i(\tau(f, n_i))) \tag{5.14}$$

It is evident that the truth of the above formulas implies the corresponding requirements are satisfied.

R_s is assumed to be conflict-free. This is not a restriction because conflicts can be eliminated by means of well-known anomaly analysis techniques (e.g., [20, 22]).

Let us define R_D as a set of requirements that represent, in an explicit way, the default behavior when D is blacklisting or whitelisting. R_D is made of requirements

taking the same form as those in R_S. More precisely, for each valid combination of 5-tuple elements for which there is no requirement in R_s whose condition matches it, there is an element of R_D, r, such that $r.C$ matches it, and $r.a =$ ALLOW if $D =$ blacklisting and $r.a =$ DENY if $D =$ whitelisting. By construction, each element of R_D does not conflict with any NSR in R_s. If D is rule-oriented-specific or security-oriented-specific, instead, we define $R_D = \emptyset$.

Finally, we also define $R = R_s \cup R_D$.

5.2 Maximal Flows Computation

The conditions expressed by (5.13) and (5.14), associated with a NSR r, depend on the set of flows F_r. However, in order to improve the efficiency of the proposed methodology, it is possible to consider only a subset of F_r, which is smaller than F_r but equally representative: the set of maximal flows that satisfy $r.C$. This set, denoted F_r^M, is defined as the subset of F_r that contains only the flows that are not subflows of any other flow in F_r, i.e. $F_r^M = \{f_r^M \in F_r | \nexists f \in F_r.(f \neq f_r^M \wedge f_r^M \subseteq f)\}$. From the definition of flow, it descends that the predicates $n_i \in \pi(f)$ and $\text{deny}_i(\tau(f,n_i))$ are true for a flow f if and only if they are true for all the subflows of f. Therefore, the following formulas are equivalent to (5.13) and (5.14) respectively:

$$\forall f \in F_r^M. \exists i. (n_i \in \pi(f) \wedge allocated(n_i) \wedge deny_i(\tau(f,n_i))) \tag{5.15}$$

$$\exists f \in F_r^M. \forall i. (n_i \in \pi(f) \wedge allocated(n_i) \implies \neg deny_i(\tau(f,n_i))) \tag{5.16}$$

In fact, all the flows of F_r that are not in F_r^M are subflows of flows that are in F_r^M.

Through this definition, multiple flows that behave in the same way (i.e., that cross the same node sequence and are subject to the same changes) are grouped into a single maximal flow, becoming their subflows. Computing the maximal flows reduces the number of different cases to be considered and, hence, also the number of constraints composing the models, to the minimum one. In fact, the number of flows to be considered is minimized. Another advantage of maximal flows is that their generation occurs before the formulation of the MaxSMT problem, so that the variables composing the flow model are not free when included in the MaxSMT problem formulation, but they are already assigned specific values. In this way, the number of free variables is kept low, limiting the solution space to be searched in the MaxSMT problem, and improving performance.

Algorithm 1 computation of F_r^M

Input: a requirement r, and an AG G_A, **Output:** F_r^M

1: $F_r^M = \emptyset$
2: **for each** $p = [n_0, n_1, ..., n_{m+1}] \in paths(r, G_A)$ **do**
3: $F \leftarrow \{[n_0, t^r_{0,1}, n_1, \text{true}, n_2, ..., \text{true}, n_{m+1}]\}$
4: **for** $i = 1, 2, ..., m$ **do**
5: $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' \mid l + [b_i, n_i] + l' \in F, b'_i \in \{\mathscr{I}_i^a, \mathscr{I}_i^d\}\}$
6: $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' \mid l + [b_i, n_i] + l' \in F, b'_i \in \{\mathscr{D}_{ij}\}\}$
7: $F \leftarrow \{l + [b_i, n_i, b_{i+1} \wedge \mathscr{T}_i(b_i), n_{i+1}] + l' \mid l + [b_i, n_i, b_{i+1}, n_{i+1}] + l' \in F\}$
8: **end for**
9: $F' \leftarrow \{l + [t^r_{m,m+1} \wedge b_{m+1}, n_{m+1}] \mid l + [b_{m+1}, n_{m+1}] \in F\}$
10: **for** $i = m, m-1, ..., 1$ **do**
11: $F' \leftarrow \{l + [b_i \wedge \mathscr{T}_i^{-1}(b_{i+1}), n_i, b_{i+1}] + l' \mid l + [b_i, n_i, b_{i+1}] + l' \in F'\}$
12: **end for**
13: **if** $F \neq F'$ **then**
14: $F \leftarrow F'$
15: **goto** line 4
16: **end if**
17: $F_r^M \leftarrow F_r^M \cup F$
18: **end for**
19: **return** F_r^M

For each NSR r, F_r^M can be computed on the basis of the transformation behavior of NFs, by means of Algorithm 1.

Initially, the set $paths(r, G_A)$ containing the paths of G_A that satisfy $r.C$ is computed. These are all the paths of G_A with end points e_s, $e_d \in E_A$ such that $\alpha(e_s) \wedge r.C.IPSrc \neq$ false and $\alpha(e_d) \wedge r.C.IPDst \neq$ false. Each path is represented by a list of nodes $p = [n_0, \cdots, n_{m+1}]$, where $n_0 = e_s$ and $n_{m+1} = e_d$.

For each path p, the elements $f \in F_r^M$ such that $\pi(f) = p$ are computed and added to the result set. This computation is performed iteratively. At each iteration, two sets of lists F and F' of alternating nodes and packet classes are computed. The first set F initially contains only the list $[n_0, t^r_{0,1}, n_1, \text{true}, n_2, ..., \text{true}, n_{m+1}]$ (line 3). In this list, $t^r_{0,1} = (\alpha(n_0) \wedge r.C.IPSrc, *, r.C.pSrc, *, *)$ is the largest traffic that satisfies the source components of $r.C$, while the other packet classes are set to true (i.e., the class of all packets). Then, at each iteration, a forward traversal and a backward traversal of the path p are performed. In the forward traversal (lines 4-8), each list in

F is progressively updated to take into account the way the traffic is transformed by each NF, and it is split into sub-lists that satisfy the homogeneity property of flows: for each node n_i of the path, the predicate b_i, which represents the ingress traffic of n_i in the current list, is split into the largest homogeneous subclasses of packets for node n_i by intersecting it with the classes of packets $\mathscr{I}_i^a$, $\mathscr{I}_i^d$, and D_{ij}, i.e., the classes that can be distinguished by the $deny_i$ predicate and by the $\mathscr{T}_i$ transformer respectively (lines 5-6). In these formulas, the operator + means list concatenation. Note that, for the packet filters in the APs, $\mathscr{I}_i^a$ and $\mathscr{I}_i^d$ are unknown, because their configuration is not yet decided. For these nodes, $\mathscr{I}_i^a$ and $\mathscr{I}_i^d$ are set respectively to true and false, i.e., no splitting occurs. The lists resulting from this split are then restricted through the conjunction of the predicate b_{i+1}, which represents the egress traffic of n_i, and $\mathscr{T}_i(b_i)$, i.e., the result of the transformation of node n_i on the traffic b_i (line 7). Then, the flows computed in the forward traversal have to be restricted in order to select only those that satisfy the destination components of *r.C*. This is done by the backward traversal, which computes a new set of lists, F', starting from the set F computed in the forward traversal. F' is initialized to contain each element of F, with its last traffic restricted to the largest traffic that satisfies the destination components of *r.C* (line 9). Here, $t^r_{m,m+1}$=($*$, $\alpha(n_{m+1}) \wedge$ *r.C.IPDst*, $*$, *r.C.pDst*, *r.C.tProto*). In the backward traversal (lines 10-12), the predicates representing the ingress traffic of each node are restricted by propagating the restricted versions backwards. The procedure stops when, after the last iteration, the flows in F and in F' are the same. If not, a new iteration starts with F initially containing the flows present in F' at the end of the previous iteration.

5.3 Firewall Allocation and Configuration

This section defines the abstract model that is proposed for the packet filtering firewall behavior. Then, it discusses the hard and soft constraints that are defined to express the allocation and configuration decisions, and it summarizes how the MaxSMT problem is formulated by including the clauses related to all the problem components.

5.3.1 Firewall configuration model

The *Filtering Configuration* (FC) for each possibly allocated firewall makes up its behavior, and it is modeled within a user-friendly abstract language, which is independent of specific firewall configuration languages, but fairly general.

In greater detail, the FC of a firewall that may be placed in $a_h \in A_A$ is characterized by a default action d_h and a set of specific rules U_h. The default action d_h has less priority than the rules in U_h, because it is applied only if there is not any rule in U_h that matches the packet fields. Besides, the action of each rule is the opposite of the default action. The rules $u \in U_h$ are the typical packet filtering rules, internally represented as $u = (C, a)$, where $u.C = (IPSrc, IPDst, pSrc, pDst, tProto)$ and $u.a \in A_T = \{\text{DENY, ALLOW}\}$. Although this representation is similar to the NSRs formalization, this does not mean that for each NSR a single corresponding firewall rule is needed or is enough in each firewall. A single firewall rule can be sufficient to enforce multiple NSRs, while multiple rules in different firewalls could be needed to enforce a single NSR, depending on the AG.

Besides, for the VEREFOO approach, given the potentially high number of FC rules that can be configured for each firewall instance, limiting their number is crucial for achieving good performance of the method. This limitation demands for a trade-off between accuracy and scalability.

For this reason, each default action d_h is determined before solving the MaxSMT problem, choosing the value that would minimize the number of rules needed to satisfy all the NSRs. In order to determine this value for d_h, the only NSRs r that are relevant are those for which the set of maximal flows F_r^M contains at least a traffic flow that passes through the AP a_h (i.e., such that (5.17) holds).

$$\exists f \in F_r^M.\ a_h \in \pi(f) \tag{5.17}$$

Let z_r be the total number of 5-tuples composing the packet classes $\tau(f, a_h)$ such that $f \in F_r^M$ and r is a reachability requirement for which (5.17) holds. Let instead z_r^i be the number of 5-tuples composing the packet classes $\tau(f, a_h)$ such that $f \in F_r^M$ and r is an isolation requirement for which (5.17) holds. If $z_r > z_r^i$, then d_h is set to ALLOW, otherwise to DENY. This decision is consistent with the aim of reaching an optimal solution. If the default action of each firewall is determined in this way, specific policy rules are not needed for the NSRs with the same action and the cardinality of the possible rule set is consequently minimized. The result of this

decision is modeled by the *wlst*: $A_A \rightarrow \mathbb{B}$ predicate, which is true for an AP a_h if the default action of the firewall allocated on a_h is DENY. Another predicate related to the default action is *enforces*: $A_T \times R \rightarrow \mathbb{B}$. $enforces(d_h, r)$ is true if the firewall default action d_h enforces requirement r, i.e., if $d_h = r.a$.

Then, given for granted this first criterion, it is necessary to determine, for each $a_h \in A_A$, the set of all the rules that potentially can be useful for a firewall that may be allocated in a_h, i.e., the candidates for being included in U_h. These rules are called placeholder rules. P_h is the set of all the placeholder rules which are defined for a firewall in a_h, and each $p_i \in P_h$ is represented as $p_i = (C, a)$, as for the elements of U_h. After having determined these rules, a soft clause is introduced for each one of them, which is true if the rule is not included in U_h, thus getting the minimization of the cardinality of U_h.

Placeholder rules are determined by selecting each NSR r that can influence a specific $a_h \in A_A$, and by considering the possible ingress traffics of a_h in the flows that belong to F_r^M. In particular, for each $a_h \in A_A$, let us define Q_h as the set of 5-tuples for which a rule might be needed in a firewall placed in a_h. For each requirement $r \in R$, and each flow $f \in f_r^M$, the 5-tuples composing the packet class $\tau(f, a_h)$ are in Q_h if the following two conditions are satisfied: 1) the flow f passes through a_h, i.e., $a_h \in \pi(f)$; 2) the default action d_h assigned to the firewall allocated in a_h would not enforce r, i.e., enforces(d_h, r) is false. The set Q_h is thus computed as the smallest set of the 5-tuples, such that (5.18) holds.

$$\begin{aligned} \forall r \in R. \forall f \in F_r^M. (a_h \in \pi(f) \wedge \neg\, enforces(d_h, r) \\ \implies (\forall q \in \tau(f, a_h).\ q \in Q_h)) \end{aligned} \tag{5.18}$$

For each $q \in Q_h$, a placeholder rule is defined in P_h. Consequently, the cardinality of P_h is the same as the cardinality of Q_h. The action of each placeholder rule $p_i \in P_h$ is the opposite of the default action d_h (i.e., $p_i.a$ = ALLOW if $wlst(a_h)$ is true, $p_i.a$ = DENY otherwise). Instead, the conditions of each p_i are defined over free variables. The values of these variables are not predetermined, but they are automatically computed by the solver. The computation of their values is bounded to the hard constraints that are introduced in the MaxSMT problem.

This abstract model is inspired by the one proposed in the Firmato methodology [59], which pursued a similar approach to provide a firewall configuration that is independent from the specific different implementations. Besides, this model has been also mapped to multiple real firewalls (i.e., iptables, ipfirewall, eBPF-based

firewall, Open vSwitch), as the generation of their settings simply involves a syntax change with respect to the abstract model. The feasibility of this mapping also proves that the proposed model is general enough to support multiple types of packet filtering firewalls.

5.3.2 Firewall allocation and configuration constraints

Soft constraints are used to find the optimal firewall allocation and configuration choices, which in turn are limited by the presence of hard constraints. Besides, the soft constraints include free variables representing the choices that the solver can make.

Soft clauses are defined so as to reach the two main optimization goals: 1) to minimize the number of allocated firewalls; 2) to minimize the number of rules for each allocated firewall. As free variables are shared among all clauses, the optimal result is reached for both goals at the same time. However, weigths are assigned so as to give priority to goal 1).

For goal 1), as a firewall can be allocated in any $a_h \in A_A$, a set of soft clauses is introduced to state that it is preferable that in each $a_h \in A_A$ no firewall is allocated. This is expressed by (5.19), where $\text{Soft}(f, c)$ stands for a soft clause with formula f and weight c.

$$\forall a_h \in A_A.\, \text{Soft}(\mathit{allocated}(a_h) = \text{false},\, c_h) \tag{5.19}$$

An indication about the value assigned to c_h will be provided later on, after the other soft constraints have been presented.

For goal 2), each placeholder rule is actually included in U_h only if the optimizer engine establishes that it is really necessary in order to reach the optimal solution. The *configured*: $P_h \rightarrow \mathbb{B}$ predicate is introduced to represent this decision. It is true for the placeholder rules that are included in U_h. To achieve the optimization goal, the soft constraint (5.20) is thus exploited to represent the ideal condition in which no firewall rule needs to be configured.

$$\forall p_i \in P_h.\, \text{Soft}(\neg \mathit{configured}(p_i),\, c_{h,i}) \tag{5.20}$$

If at least a rule belonging to P_h is configured, then a firewall instance must be allocated in a_h, because this rule is needed to satisfy a security requirement. This

condition is expressed by the following hard constraint:

$$(\exists p_i \in P_h . \mathit{configured}(p_i)) \implies \mathit{allocated}(a_h) \tag{5.21}$$

If the consequent of this hard constraint is true, the soft clause defined by (5.19) cannot be satisfied for that packet filter, which needs to be allocated in the topology.

Since the priority of goal 2) is less than the priority of goal 1), the weight of (5.19) must be higher than the sum of the weights of the soft clauses (5.20) related to all the firewall placeholder rules:

$$\sum_{i:p_i \in P_h} (c_{h,i}) < c_h \tag{5.22}$$

An additional set of soft constraints must be introduced, in case of a *security-oriented specific* approach, to minimize the number of allowed traffic flows. This objective is modeled in the following way: (i) if the firewall has a whitelisting configuration, the fields of the ALLOW rules should not be configured with the wildcards feature, allowing only the required traffic flows; (ii) if the firewall has a blacklisting configuration, the fields of the DENY rules should exploit the wildcards feature, so as to block the largest number of flows. This objective is represented by soft clause (5.23) for the source IP address, and similar clauses are defined for the other 5-tuple fields. In these clauses, the $\sigma : A_A \rightarrow \mathbb{Z}$ function, defined in (5.24), is exploited to model the sign of the weights.

$$\forall p_i \in P_h . \forall j \in \{1,2,3,4\} . \mathrm{Soft}(p_i.\mathit{IPSrc}_j = *, \sigma(a_h) \cdot c_{h,i,j}) \tag{5.23}$$

$$\sigma(a_h) = \begin{cases} -1 & \text{if } \mathit{wlst}(a_h) \\ +1 & \text{if } \neg \mathit{wlst}(a_h) \end{cases} \tag{5.24}$$

As this is not one of the two main optimization goals, its priority is lower than the one for goals 1) and 2). Consequently, the sum of the weights $c_{h,i,j}$ of all these clauses must be less than the weight $c_{h,i}$ assigned to clause (5.20) for rule minimization.

In addition to the clauses presented so far, some other hard clauses are necessary, in order to finalize the configuration of the FC of each firewall consistently with the allocation and configuration choices. To achieve this purpose, it is necessary to consider, for each AP a_h, the set of possible input traffics $T_h = \{\tau(f, a_h) \mid f \in F_r^M \text{for some } r \in R\}$. For each traffic $t \in T_h$, two hard constraints are needed: one, to define how the policy of the firewall in a_h must be configured in case the firewall must drop t, and another one to define how the same policy must be configured in case the firewall must not drop t .

In order to formulate these clauses, the *matchAll*: $P_h \times Q \rightarrow \mathbb{B}$ and *matchNone*: $P_h \times Q \rightarrow \mathbb{B}$ predicates are introduced. Given a placeholder rule $p_i \in P_h$ and a 5-tuple $q \in Q$, the *matchAll* predicate is true if the rule conditions completely include the values of the 5-tuple fields, as expressed by (5.25). Instead the *matchNone* predicate is true if the packet classes expressed by the rule conditions and by the 5-tuple fields are disjunct, as expressed by (5.26).

$$matchAll(p_i, q) \Leftrightarrow q \subseteq p_i.C \tag{5.25}$$

$$matchNone(p_i, q) \Leftrightarrow \neg(q \wedge p_i.C) \tag{5.26}$$

Having defined *matchAll* and *matchNone*, for each $t \in T_h$, the two hard clauses in (5.27) and (5.28) are introduced to finalize the configuration of the FC for the firewall in a_h. The *allocated* predicate in the antecedent of both the implications is motivated by the fact that, if the firewall is not effectively allocated, then the configuration of its policy is meaningless.

$$\begin{gathered} allocated(a_h) \wedge deny_h(t) \implies \text{(a)} \vee \text{(b)} \\ \text{(a)} = wlst(a_h) \wedge \forall q \in t.(\forall p_i \in P_h.(\neg configured(p_i) \vee matchNone(p_i, q))) \\ \text{(b)} = \neg wlst(a_h) \wedge \forall q \in t.(\exists p_i \in P_h.(configured(p_i) \wedge matchAll(p_i, q))) \end{gathered} \tag{5.27}$$

$$\begin{gathered} allocated(a_h) \wedge \neg deny_h(t) \implies \text{(a)} \vee \text{(b)} \\ \text{(a)} = wlst(a_h) \wedge \forall q \in t.(\exists p_i \in P_h.(configured(p_i) \wedge matchAll(p_i, q))) \\ \text{(b)} = \neg wlst(a_h) \wedge \forall q \in t.(\forall p_i \in P_h.(\neg configured(p_i) \vee matchNone(p_i, q))) \end{gathered} \tag{5.28}$$

The truth of the antecedents directly depends on the consequences of the implications represented by (5.15) and (5.16). If the firewall is the function identified by (5.15) to block a specific traffic flow, then it must be in whitelisting without any rule that allows the 5-tuples of that traffic, or in blacklisting with specific rules that block them. Instead, if the firewall should allow a traffic because of (5.16), then it should be in whitelisting with a specific allowing rule that blocks each 5-tuple of the traffic, or in blacklisting with no rule that would block them.

5.3.3 Summary of MaxSMT problem formulation

Here a summary is presented to describe how the problem modeling constraints described in this chapter are used in the formulation of the MaxSMT problem.

For what concerns hard constraints, on one side, for each requirement r, hard constraint (5.15) (for isolation) or (5.16) (for reachability) is introduced to state that

r must be satisfied in the AG enriched with the allocated and configured firewalls. On the other side, for each network function in the AG, a set of hard constraints is introduced to constrain how it can forward flows. Different function types require different constraints, as it has been discussed in Subsection 5.1.2. For example, for each function in node n_i that cannot drop flows, (5.6) is introduced, while for each allocation place a_h where a firewall can be allocated, (5.27) and (5.28) are introduced to define the forwarding behavior of this firewall.

For what concerns soft constraints, for each allocation place a_h, (5.19) is used to minimize the number of allocated firewalls, while constraints (5.20) are used to minimize the number of rules in each allocated firewall, and constraints (5.23) to prefer aggregate rules. The weights of soft constraints are decided so as to satisfy constraints (5.22) and (5.24). For each allocation place a_h, the additional hard constraint (5.21) is introduced in order to require that a firewall is allocated in a_h only if the solution includes at least one configured rule in it, along with the hard constraint (5.4), which forces the allocation of firewalls in the positions where the administrator necessarily requests the presence of a firewall.

For all the hard and soft constraints that have quantifiers in their definition, the quantifiers are removed by using appropriate semantics-preserving transformations, so as to make the solution of the MaxSMT problem more efficient.

The MaxSMT solver is fed with all these hard and soft constraints, and it computes the optimal solution if a correct one that satisfies all the hard constraints can be found. In particular, the values of the *allocated* and *configured* predicates respectively provide information about the APs where firewalls have been allocated and the rules that have been configured in their rule sets. Additionally, the values that the solver has assigned to the free variables composing the configured rules indicate how the 5-tuple-based conditions and the actions of the rules must be set up in each allocated firewall.

The optimality of the problem solution can be proved under the assumption that the MaxSMT solver is correct:

Theorem 5.3.1. *If a solution s of the MaxSMT problem exists, and s allocates n firewalls, then, the problem does not admit another solution s' that allocates $n' > n$ firewalls.*

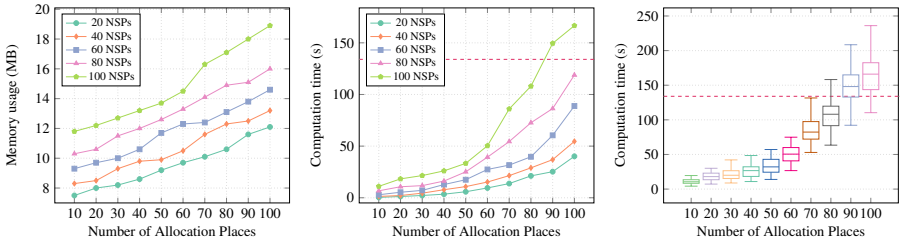
Proof. By contradiction, let us assume that s' exists. The sum of weights of s's true soft constraints is $c_h(|A_A| - n) + \delta$, where $c_h(|A_A| - n)$ is the contribution from soft clauses (5.19) and $\delta < \sum_{i:p_i \in P_h}(c_{h,i})$ the one from soft clauses (5.20). Similarly, the same sum for s' is $c_h(|A_A| - n') + \delta'$, with $\delta' < \sum_{i:p_i \in P_h}(c_{h,i})$. As both s and s' are solutions of the problem, their sum of weights of satisfied soft clauses must be the same, i.e., $c_h(|A_A| - n) + \delta = c_h(|A_A| - n') + \delta'$ which implies $c_h(n' - n) = \delta' - \delta$. From $\delta < \sum_{i:p_i \in P_h}(c_{h,i})$ and $\delta' < \sum_{i:p_i \in P_h}(c_{h,i})$, we have also $(\delta - \delta') < \sum_{i:p_i \in P_h}(c_{h,i})$. Then, $c_h(n' - n) < \sum_{i:p_i \in P_h}(c_{h,i})$ and, since $n' > n$, we have $c_h \leq c_h(n' - n) < \sum_{i:p_i \in P_h}(c_{h,i})$, which contradicts (5.22). ■

A similar theorem can be proved for the minimization of the number of rules.

From what concerns correctness, only the intuition of the formal proof is presented, as the proof would be too complex. The solver correctness assumption implies that, if a solution is found, it is formally guaranteed to satisfy all the hard constraints of the problem. Of course, it is possible to prove that the solution is correct, as long as it is possible to prove or assume that the hard constraints of the problem presented in this chapter, which are first order logic formulas, imply correctness. Specifically, this holds provided that the hard constraints modeling the possible forwarding behavior of network functions and the possible traffic flows are a correct representation of reality. About this point, it is worth mentioning that in literature there are approaches, such as [167], that can extract a formal model of the forwarding behavior of a virtual function automatically from a behavioral representation expressed in a high-level programming language like Java. Using these approaches, it is possible to get high confidence about adherence of the models to the real function behavior.

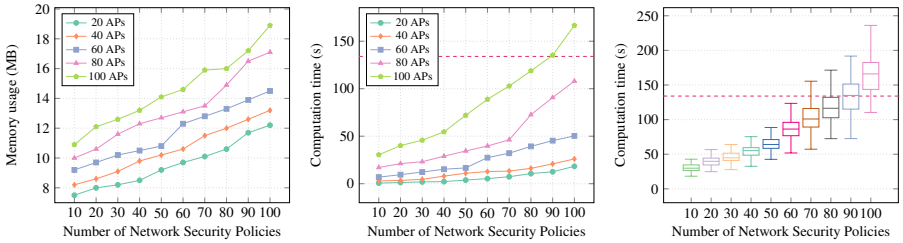
5.4 Implementation and Validation

The application of the VEREFOO approach to firewall configuration has been implemented by means of a Java framework, which exploits the APIs offered by the z3 solver [164] to formulate and solve the MaxSMT problem. The code is publicly available at the following link: https://github.com/netgroup-polito/verefoo/tree/Budapest. The framework is accessible through its REST APIs, so that it can be exploited by



(a) Memory usage chart (b) Computation time chart (c) Whisker plot

Fig. 5.1 Scalability for increasing number of Allocation Places



(a) Memory usage chart (b) Computation time chart (c) Whisker plot

Fig. 5.2 Scalability for increasing number of Network Security Policies

external tools as a component of a more complex architecture, or through its GUI for human users.

The framework has been validated in different ways: on synthetically generated networks, to prove the scalability of the approach (Subsection 5.4.1), and on topologies inspired by production networks, to verify the correctness and optimization properties in realistic and complex environments (Subsection 5.4.2). It has also been compared to the state-of-the-art approaches (Subsection 5.4.3). Finally, the optimization provided by this approach has been evaluated (Subsection 5.4.4). All the MaxSMT instances have been solved on a machine with an Intel i7-6700 CPU at 3.40 GHz, 32GB of RAM, and z3 version 4.8.5. The z3 solver offers a parallel execution mode for selected theories, but this mode has not been employed because it is still experimental and not available for all the theories.

The decision of using this machine with medium computing power for these tests and all the other ones that are discussed in this dissertation is motivated by the intention of evaluating the performance level of the proposed approaches when executed on medium-level, cheap hardware. If a more powerful machine was used, some improvement could be expected, even if probably not so significant.

5.4.1 Scalability evaluation

The scalability of the approach has been evaluated varying the follwing factors: the number of APs where firewall instances can be allocated, the number of NSRs that must be fulfilled, and the enforceability of the MaxSMT problem. The results have been analyzed also in relation to the number of constraints of the MaxSMT problem and the number of maximal flows to be computed.

Scalability versus number of APs and NSRs

The charts in Fig. 5.1 and Fig. 5.2 present the results of a series of tests performed to evaluate scalability versus number of APs and NSRs. For each test case with a given number of APs and NSRs, 100 runs have been executed. For the same test case, runs are differentiated only by the IP addresses that are assigned to the network nodes, while all the other parameters (e.g., topology of the AG, types of NSRs) are kept the same. This choice is motivated by the way z3 manages the integer theory; the results are, in fact, different in terms of computation time, according to the integer numbers that are introduced in the clauses of the MaxSMT problem.

The AGs that have been exploited for scalability validation have been synthetically generated as extensions of the AG illustrated in Fig. 4.3. The number of end points and middleboxes is properly adapted to the number of APs and NSRs that characterize each test case. However, the middleboxes are functions that do not modify the received packets, so that the focus of the validation is kept only on the efficiency of the z3 formulas related to allocation and automatic configuration of firewalls on one side, and on security requirements, on the other side. Moreover, the NSRs considered for the tests are defined using the *security-oriented specific* approach, which is the worst case because it introduces the biggest number of soft constraints in the MaxSMT problem. For each test scenario, half of the requirements are isolation properties, and half are reachability properties. All tests refer to cases for which all NSRs can be enforced. This choice is motivated by the consideration, confirmed by experiments (see section 5.4.1), that this is the worst case.

Fig. 5.1a and Fig. 5.2a show the peak memory usage that is required to build the variables and constraints of the MaxSMT problem by using the z3 Java APIs. This may be a critical parameter for the solver, which is highly memory-demanding. However, even in the worst case considered, the amount of memory that is required

is inferior to 19 MB. Consequently, this result shows that the framework can work without any worrisome limitation due to memory.

Fig. 5.1b and Fig. 5.2b show the average computation time of each test case. From these two charts, the most important result is that, even though the MaxSMT problem belongs to the NP-complete class in terms of computational complexity, the computation time does not increase exponentially. This achievement is valid for the scalability both versus the number of APs and versus the number of NSRs. According to such results, the framework can manage AGs with up to 100 APs and 100 NSRs in less than 200 seconds (for each test case, the total number of nodes is two times the number of the APs, but higher nodes to APs ratios do not require significantly greater resources). This result can be motivated by three reasons. First, and most importantly, NP-completeness only implies exponential time for the worst case, but the actual time for solving a MaxSMT instance is often less than the worst case time, also depending on which theories are used in the formulas [166]. Second, formal models have been defined so as to capture all the required aspects, but avoiding excessive complexity in the actual SMT problem to be solved (e.g., avoiding redundancy in variables and constraints, avoiding quantifiers, and solving maximal flow computations separately). Leveraging this trade-off between expressiveness and complexity was a key factor that enabled the achievement of such scalability results. Third, state-of-the-art solvers like Z3 employ internal strategies that are quite efficient in exploring the solution space [164].

Fig. 5.1c and Fig. 5.2c show the value distribution of the computation time measures by means of whisker plots; the number of NSRs is fixed to 100 in Fig. 5.1c, and the number of APs is fixed to 100 in 5.2c. Analyzing the whisker plots, it is possible to state that the distribution of the values is mostly gathered between the first and the third quartiles. The number of values that are outside this interval is really low. Another consideration is that the average value is almost identical to the median value: this also proves that the number of outliers, which would make the average much bigger than the median, is almost null.

In Fig. 5.1b, 5.2b, 5.1c, and 5.2c, a baseline (red dotted horizontal line) is introduced, in order to have a reference: it is the Deployment Process Delay (DPD) introduced by a well known orchestrator (Open Source MANO) for deployment. DPD is the time the orchestrator takes to deploy and instantiate a VNF within an already booted VM and setup an operational network service. According to [168],

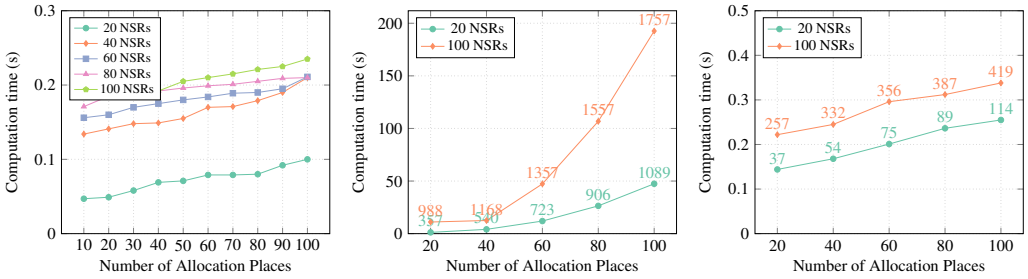
APs	20	40	60	80	100	20	40	60	80	100
NSRs	20	20	20	20	20	100	100	100	100	100
Hard	240	376	494	621	748	673	734	795	923	1037
Soft	117	164	229	285	341	315	434	562	634	720
Total	357	540	723	906	1089	988	1168	1357	1557	1757

Table 5.2 Number of hard and soft constraints

this time is 134ms. The figures of these experiments show that the time taken by the framework to automatically allocate and configure firewalls in SGs with up to 100 APs and with up to 80 NSRs does not exceed the DPD, so being acceptable even in highly dynamic situations.

Scalability versus enforceability

All the MaxSMT problems that were employed for the previous scalability validation could be solved, i.e., there always existed a solution that could satisfy all their hard constraints. Here some cases of non-enforceability have been analyzed, i.e., when some NSRs cannot be successfully refined into the firewall allocation scheme and configuration. For example, there may not be enough APs in the logical topology represented by the AG. New tests have been run to assess how memory usage and computation time change for these cases. These tests were carried out under the same conditions that have been previously explained for the tests related to Fig. 5.1, with the difference that the topology is built so that no solution can be found by the MaxSMT solver. The memory usage is the same as the one shown in Fig. 5.1a, because it is determined by the set of variables and clauses, depending on the graph size, which is the same. Instead, the computation time required to manage non-enforceability cases is way less, as shown in Fig. 5.3a. More precisely, managing non-enforceability cases requires a computation time that is two magnitude orders less than the other cases, under the same network size and the same NSRs set. This outcome was expected, as state-of-the-art MaxSMT solvers like z3 usually can efficiently find out if no solution exists to satisfy the constraints of a certain problem [164]. This feature is beneficial to the VEREFOO approach, as the user of the framework can know in really fast times if a solution of the problem can be found, or if the specified NSRs cannot be refined into a correct firewall configuration.



(a) Non-enforceability cases (b) Impact of constraints (c) Impact of maximal flows

Fig. 5.3 Other tests related to the scalability validation

Scalability versus number of MaxSMT constraints

A central factor in assessing the scalability of the approach is the number of constraints composing the MaxSMT problem. Their number is strictly dependent not only on the numbers of APs and NSRs, but also on the way the formal models are used for the definition of the MaxSMT problem. Therefore, validating the approach with respect to this factor contributes to validating the impact of formalization on performance. After all, defining formal models with a trade-off between expressiveness and performance has been a main motivation behind the formulation proposed for the VEREFOO approach.

The results of the scalability tests related to the number of constraints are presented in Fig. 5.3b. This chart shows the average computation time, on 100 iterations, for ten different combinations of numbers of APs and NSRs. The number of constraints composing the MaxSMT problem corresponding to each combination is reported on top of each symbol in the plot lines. The information about the number of constraints is enriched by TABLE 5.2, where the exact division between hard and soft constraints is included.

From the results depicted in Fig. 5.3b, it can be observed that the number of soft and hard clauses required for generating the use case with 20 NSRs and 100 APS can be solved by z3 in around 50 seconds. Instead, it requires almost 200 seconds to obtain the result for the use case with 100 APs and NSRs. The former case is characterized by 748 hard and 341 soft constraints, while the latter by 1037 hard and 720 soft constraints. From the relative increases of the two clause categories, it derives that soft constraints have a bigger impact then hard constraints on the performance of the methodology. This result was expected, because soft constraints

are relaxable and therefore the MaxSMT solver has the faculty of deciding whether to satisfy each one of them, thus increasing the size of the overall solution space. These clauses are also the ones that involve the highest number of free variables. Instead, all hard constraints must be satisfied, so the solver is simply in charge of checking if the variable assignments are compatible with those clauses.

Impact of maximal flows computation time

All the tests that have been carried out so far envision both the application of the maximal flows algorithm, discussed in Section 5.2, and the resolution of the MaxSMT problem to output the firewall allocation scheme and configuration. However, interesting considerations can be taken by evaluating the performance of the former separately from the latter, so as to understand the scalability of the code that implements the flow algorithm.

The results of the scalability tests related to the maximum flows algorithm are presented in Fig. 5.3c. This chart shows the computation time, averaged on 100 iterations, for ten different combinations of numbers of APs and NSRs. The number of maximal flows corresponding to each combination is reported on top of each symbol in the plot lines. From these results, it is clear how the time required for the computation of the maximal flows is negligible with respect to the time needed for solving the MaxSMT problem. For example, when the network is composed of 100 APs and 100 NSRs must be enforced, the total time required by the framework is 166s, but the computation of the maximal flows only takes a minimum fraction of that time, i.e., 0.338s. Therefore, the impact of this algorithm on performance is minimum. At the same time, it allows bringing over all the advantages showed in Section 5.2.

The maximal flow algorithm may be parallelized, as the flows can be computed separately for each user-specified NSR. However, as this algorithm has a negligible impact on performance, the parallelization has not been implemented.

	GÉANT AG	Internet2 AG
Number of vertices	49	53
Number of end points	19	18
Number of APs	11	17
Number of directed links	86	80
Number of NSRs	50	50
Number of flows	106	201
Average path length (vertices)	22	31
Average computation time (s)	32.77	144.27

Table 5.6 Test results for GÉANT and Internet2 AGs

5.4.2 Correctness and optimization verification

The correctness and optimization verification has been performed in two AGs inspired by the production networks GÉANT and Internet2 [2]. In comparison with the synthetic topology exploited in the scalability tests, the two graphs representing these topologies have a much more complex and ramified structure. This characteristic has some critical consequences: not only the number of middleboxes that each traffic flow has to cross to reach the destination is higher, but also the number of traffic flows that satisfy the conditions of the same NSR is higher.

With the aim to verify that the NSRs are correctly enforced by the computed solution, we have used the Mininet emulator to instantiate the two use cases in a controlled environment. The tests made on the Mininet emulation confirmed that all NSRs are satisfied, as expected. Therefore, these tests also allowed checking the correctness of the proposed approach. Then, with regards to optimization, the solution computed by the framework has been checked to be the one that minimizes the number of allocated firewalls and configured rules, by comparing it with all the possible solutions to the same problem instances.

TABLE 5.6 reports the main characteristics of the two AGs and the average time which is required, out of 100 runs, to compute the firewall allocation scheme and configuration, when the number of NSRs is set to 50 (this is a reasonable number with respect to the size of the end point set). The number of APs is low with respect to the total number of vertices and links. A first reason for it is that these topologies have been built directly as AGs, with the APs placed only in specific positions

[2]Links: https://geant3plus.archive.geant.net/, https://www.internet2.edu/. Last accessed: October 18th, 2022.

Approach	Network	Allocation	Configuration	Formal	Optimization	Scalability
[59]	Traditional	X	✓	✓(with [120])	X	No Info
[63]	Traditional	X	✓	X	X	No Info
[61]	Traditional	X	✓	X	X	10 devices - No info
[58]	Both	✓(SG)	X	✓	✓(**allocation**: minimize deployment cost, and maximize security and usability)	20FW - 80s
[55]	Virtual	✓(SFC)	✓	X	X	20 devices - 4s
[64]	Traditional	X	✓	✓	X	No Info
[69]	Both	X	✓	✓	X	5FW - 50s
[40]	Both	✓(SFC)	✓	✓	X	No Info
[57]	Both	✓(SG)	X	X	✓(**allocation**: minimize rule set cardinality)	60FW - No Info
VEREFOO	Both	✓(SG)	✓	✓	✓(**allocation & configuration**: minimize number of FWs and of rules)	100FW - 90s

Table 5.7 Comparison with most related approaches (features versus scalability)

inspired from the GÉANT and Internet2 features. A second reason is that in these AGs links are bidirectional, so each bidirectional link counts as two directed links.

The computation time required for the Internet2 AG is higher than that taken for the GÉANT AG, because in the former case the number of possible paths between any pair of end points and, consequently, also the number of traffic flows, is higher. However, in both cases, the achieved result is satisfactory, considering that it is much less than what we could expect for a manual configuration and that the automated approach reduces the possibility of human errors.

5.4.3 Comparison with state-of-the-art approaches

TABLE 5.7 shows a comparison of the VEREFOO approach with the most related state-of-the-art approaches available in the literature. The table compares the following main features of each approach: “Network” specifies if the approach can work only on traditional physical networks, only on virtual networks, or on both types; “Allocation” specifies if the approach has the capability of computing the firewall allocation scheme (on a SG or on a SFC); “Configuration” specifies if the approach has the capability of computing the firewall configuration rules; 4) “Formal” specifies if the approach is formal; 5) “Optimization” specifies if the approach finds an optimized solution and with which criteria; 6) “Scalability” reports the maximum size of the problem (in terms of number of firewalls) the approach has been tested on, and the computation time required by it for computing a solution in a network of that size, if they are provided by the related paper.

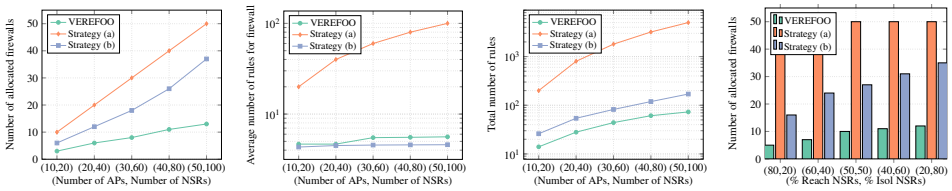
The table confirms that no other existing approach jointly computes the firewall allocation scheme and the configuration starting from a provided SG, as the VEREFOO approach does. Also, no prior work achieves all the three features of full

automation, optimization, and formal correctness, with the exception of [58], which, however, supports only the automatic generation of the firewall allocation scheme, without computing the configuration of each allocated instance, and it pursues different optimization goals. Finally, even no combination of existing approaches can be used to obtain automatically the same results that the VEREFOO approach can obtain.

Nevertheless, it may be interesting to make a rough comparison between the scalability data reported by the other approaches and the data retrieved by the validation of the VEREFOO approach (the "Scalability" column of TABLE 5.7). The VEREFOO framework proves to be competitive with respect to the other relevant works in terms of scalability, especially considering the added value of the results achieved. Many existing approaches, such as [59], [63], and [64], do not provide any information about the size of the networks on which the approach has been tested or the computation time. Other approaches, such as [61], [58], [55], and [69], can scale up to small sized networks (between 5 and 20 firewalls). An approach that has been tested on bigger networks is [57]. However, even though it was tested on a distributed firewall having up to 10^5 rules, the authors do not report the time taken, so that the actual scalability remains unknown. In addition, the huge number of rules reported in the paper may be due to the fact that this approach does not support wildcards ($*$) in rules. The VEREFOO approach, instead, using wildcards, defines more powerful rules, each one capable of representing many wildcard-free rules. Finally, it is important to recall that the approach in [57] solves a problem that is much simpler than the proposed one, because it cannot start from a given SG, but it generates, from scratch, a network of firewalls that interconnect the given end points and it assumes a simple pre-defined strategy to populate firewalls with rules.

5.4.4 Optimization evaluation

The optimization that the proposed methodology can achieve in terms of number of allocated firewalls and number of rules has been evaluated varying the size of the problem. Because, as already discussed, there are no other existing automated approaches that can obtain the same results, as a reference some traditional configuration strategies are considered: (a) the worst-cost strategy that allocates a firewall in each AP, with *allow* default action, and installs one *deny* rule for each isolation NSR; (b) a more optimized strategy that, for each isolation NSR, allocates a firewall with



(a) Optimization related to allocated firewalls

(b) Optimization related to average number of firewall rules

(c) Optimization related to total number of firewall rules

(d) Optimization related to allocated firewalls

Fig. 5.4 Optimization tests, in comparison with configuration strategies

allow default action in each AP that is closest to the source specified by the NSR, with a rule that enforces the NSR. .

Fig. 5.4a, Fig. 5.4b and Fig. 5.4c report, respectively, the total number of allocated firewalls, the average number of generated rules for each allocated firewall, and the total number of generated rules for varying numbers of NSRs and APs, considering a number of NSRs that is twice as big as the number of APs. In this evaluation, all approaches, including VEREFOO, adopt a blacklisting target. Fig. 5.4a shows that the VEREFOO approach achieves a relevant gain in terms of allocated firewalls, which increases progressively with bigger topologies (and higher numbers of NSRs), not only with respect to the worst-case strategy (a), but also with respect to strategy (b). Looking at Fig. 5.4b and Fig. 5.4c, the VEREFOO approach achieves a good gain in terms of total number of rules, while the average number of rules per allocated firewall is lower for strategy (b). This is easily explained by the fact that strategy (b) allocates many more firewalls. Note that, because of the big difference between the results of strategy (a) and the other ones, the y-axis in Fig. 5.4b and Fig. 5.4c is in logarithmic scale, to make this difference observable.

For each case analyzed for the tests whose results are depicted in the previous figures, 50% of the NSRs are isolation NSRs, the other ones are reachability NSRs. Additional tests have been carried out to evaluate the impact of the NSR types on optimization. Fig. 5.4d reports the optimization in terms of total number of allocated firewalls, while considering the worst case of the previous tests, i.e., keeping the numbers of APs and NSRs respectively fixed to 50 and 100, but varying the percentages of isolation versus reachability NSRs. In these tests, the VEREFOO approach worked with the security-oriented profile, which is more complex than the blacklisting one. Instead, the other reference strategies remain as previously described, as they cannot support the security-oriented mode. Therefore, the number of reachability NSRs

does not influence their results. From Fig. 5.4d, it is evident how both strategy (b) and the VEREFOO method generate bigger allocation schemes and rule sets when the number of isolation NSRs is higher, but the increase for VEREFOO is much more mitigated than the one for approach (b). Finally, the behavior of the proposed method, when used in the more complex security-oriented specific approach, has been proved to be similar to the one characterizing the blacklisting approach. This result was expected, because the minimization of the firewall allocation scheme is always the primary optimization objective in VEREFOO, whatever approach is employed.

Chapter 6

Automatic VPN Configuration

This chapter describes the application of the VEREFOO approach to the automatic allocation and configuration of *Communication Protection Systems* (CPSs) such as VPN gateways. TABLE 6.1 includes the main formal notations (symbols, functions, predicates, operators) used in this chapter.

6.1 Network Model

This section defines the formal models of the main components of the network where the auto-configuration problem must be solved (i.e., SG with its network functions, traffic flows). These models are later used to formulate the hard and soft constraints of the MaxSMT problem.

6.1.1 Service and Allocation Graph models

Similarly as for the application of the VEREFOO approach for the automatic firewall allocation and configuration, an SG is modeled as a directed graph $G_S = (N_S, L_S)$, whereas the corresponding AG is modeled as another directed graph $G_A = (N_A, L_A)$. The only difference consists in how the N_A set is defined. In particular, it is the union of four subsets: $N_A = N^E \cup N^V \cup N^A \cup N^F$. N^E is the subset of all the network end points (clients, servers, subnetworks), N^V is the subset of the CPSs, such as VPN gateways, that have been already allocated in the network (i.e., they were present also in the input SG), N^A is the subset of the APs where other CPSs could

Symbol/Function/Predicate/Operator	Definition
$\mathbb{B} = \{\text{true, false}\}$	boolean set
$G_S = (N_S, L_S)$, $G_A = (N_A, L_A)$	directed Service Graph (SG) and Allocation Graph (AG)
$N_A = N^E \cup N^V \cup N^A \cup N^F$	nodes of the AG
N^E	end points
N^V	Communication Protection Systems
N^A	Allocation Places
N^F	other middleboxes
$n_k \in N_A$	the element of N_A identified by k
$n_s, n_d \in N_A$	source/destination endpoint
$l_{i,j} \in L_A$	the edge from n_i to n_j
$t = (h^i, h^a) \in T$	a class of packets
$h^i \in H$	original header
$h^a \in H$	additional header
$t_{i,j}$	the traffic transmitted from n_i to n_j
$f \in F$	a flow, i.e., class of packets generated by n_s
$p = (C, A^c, A^i, S, W) \in P$	Communication Protection Policy
C	condition set of p
A^c, A^i	sets of cipher algorithms of p
S	communication protection properties of p
$W = (N^U, N^I, L^U)$	trustworthiness or inspection information of p
N^U	untrustworthy nodes
N^I	inspector nodes
L^U	untrustworthy links
$F^P \subseteq F$	flows that satisfy the conditions of at least one $p \in P$
$r \in R_n$	placeholder rule for $n \in N_A$
$\eta: T \to H$	maps a traffic t to its most external header h
$\nu: N_A \times F \to N_A + \{n_0\}$	maps a network node n and a traffic flow f to the next node crossed by f after n
$\pi: F \to (N_A)^*$	maps a flow to the ordered list of nodes that are crossed by that flow
$\rho: F^P \to P$	maps a traffic flow f to the policy p for which it has been computed
$\tau: F \times N_A \to T$	maps a flow and a node to the ingress traffic
$\phi: P \to \mathscr{P}(F^P)$	maps a policy p to the set of traffic flows computed for p
$\mathscr{T}_i: T \to T$	maps an input traffic to the corresponding output traffic
$allocated: N_A \to \mathbb{B}$	true $\Leftrightarrow$ a CPS is allocated in a_h
$configured: R_n \to \mathbb{B}$	true $\Leftrightarrow$ the placeholder rule is actually used
$delimiters: N_A \times N_A \times F \to \mathbb{B}$	true $\Leftrightarrow$ two nodes n_i and n_j delimit a VPN for a flow f
$deny_i: T \to \mathbb{B}$	true $\Leftrightarrow$ n_i drops all the packets
$protect^c: N_A \times F \to \mathbb{B}$	true $\Leftrightarrow$ the CPS on n adds confidentiality to flow f
$protect^i: N_A \times F \to \mathbb{B}$	true $\Leftrightarrow$ the CPS on n adds integrity to flow f
$supported: N_A \times \mathbb{A}^c \times \mathbb{A}^i \to \mathbb{B}$	true $\Leftrightarrow$ the two cypher algorithms are supported by $n \in N_A$
$tunneled: T \to \mathbb{B}$	true $\Leftrightarrow$ a traffic t has an external header
$unprotect^c: N_A \times F \to \mathbb{B}$	true $\Leftrightarrow$ the CPS on n removes confidentiality from flow f
$unprotect^i: N_A \times F \to \mathbb{B}$	true $\Leftrightarrow$ the CPS on n removes integrity from flow f
$n_i \subseteq n_j \in N_A$	n_i precedes n_j in the node list of a flow
$t_1 \subseteq t_2 \in T$	t_1 is a sub-traffic of t_2
$\wedge, \vee, \neg$	used for conjunction, disjunction, negation
.	used to denote a specific tuple element (e.g., given a tuple $t = (a, b, c)$, $t.a$ identifies element a of tuple t)

Table 6.1 Notation

be allocated, and N^F is the subset of the other network functions that do not have security capabilities for communication protection.

On the elements of the N_A set, two functions and a predicate can be applied:

- the $index_A^N$: $N_A \rightarrow \mathbb{N}_0$ function maps each node to a unique non-negative integer number;
- the *address*: $N_A \rightarrow 2^I$ function maps each node to its IP addresses.
- the *allocated*: $N_A \rightarrow \mathbb{B}$ predicate maps a node n to true if n has a CPS capability (therefore, for each $n \in N^V$, *allocated(n)* = true).

6.1.2 Traffic flows model

A traffic t represents a packet class. The model of t captures the information that is required to express possible tunneling of packets for enforcing communication protection properties. As such, it is defined as a tuple $t = (h^i, h^a)$. In this definition, h^i models the original header of the packet that must be protected, whereas h^a models the additional header that may be added by a VPN to that original packet. Each h^x, with $x = \{i, a\}$, is a conjunction of five predicates, one for each field of the IP 5-tuple, i.e., $h^x = (IPSrc, IPDst, pSrc, pDst, tProto)$. The definition of each one of these five predicates is the same as for the ones defined in Section 5.1 for firewall auto-configuration. Then, the set of all the possible traffics is denoted as T, while the set of all the possible headers as H.

Given a traffic t, it is not necessarily characterized by two headers. Even if communication protection properties are enforced, encapsulation is not always required (e.g., when the Authentication Header is used in IPSec). This possibility is modeled with the η: $T \rightarrow H$ function, which maps a traffic t to its most external header h, among the headers that exist in its definition. This function maps t to $t.h^i$ if the external header does not exist, to $t.h^a$ otherwise.

$$\eta(t) = \begin{cases} t.h^a & \text{if } \mathit{tunneled}(t) = \text{true} \\ t.h^i & \text{otherwise} \end{cases} \tag{6.1}$$

In this definition, the *tunneled*: $T \rightarrow \mathbb{B}$ predicate maps a traffic t to true if t has an external header.

The traffic flow model is defined on the top of the traffic model, in the same way as defined in Section 5.1. Briefly, given the set of all flows F, $f \in F$ represents how a specific packet class would be transformed when crossing a list of nodes and it is formally represented as a list $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_j, t_{jk}, n_k, ..., n_p, t_{pd}, n_d]$, with alternating node and traffic elements. Three utility functions can be defined over F:

- π: $F \rightarrow (N)^*$ maps a flow f to the ordered list of nodes that are crossed by f.

$$\pi([n_s, t_{sa}, n_a, t_{ab}, ..., n_k, t_{kd}, n_d]) = [n_s, n_a, ..., n_k, n_d] \tag{6.2}$$

  For two nodes n_i and n_j belonging to the same node list, $n_i \prec n_j$ means that n_i precedes n_j in that list. Similarly, $n_i \preceq n_j$ means that n_i precedes n_j, or it is n_j itself.

- τ: $F \times N_A \rightarrow T$ maps a flow f and a node n to the traffic that precedes n in the definition of f.

$$\tau([n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d], v_b) = t_{ab} \tag{6.3}$$

- ν: $F \times N_A \rightarrow N_A$ maps a flow f and a node n to the node that follows n in $\pi(f)$. If no node that follows n exists, then $\nu(f, n) = n_0$.

$$\nu([n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d], n_a) = n_b \tag{6.4}$$

6.1.3 Network functions model

The network functions models is directly mutated from the one illustrated in Section 5.1. Again, the behavior of a network functions, which depends on its code and configuration, is modeled abstractly by means of two functions, capturing respectively the forwarding behavior (i.e., which input packets are dropped by the VNF) and the transformation behavior (i.e., which packets may be output by the VNF for each class of input packets).

The function that models the forwarding behavior of the VNF in node $n_i \in N_A$ is the predicate $deny_i$: $T \rightarrow \mathbb{B}$ which is true for ingress traffic $t \in T$, if and only if n_i drops all the packets represented by t. Instead, the transformation behavior of the VNF in node $n_i \in N_A$ is instead modeled by the function $\mathscr{T}_i$: $T \rightarrow T$, called transformer, which maps an input traffic to the corresponding output traffic.

6.2 Communication Protection Model

This section defines the formal models of the main components necessary for the channel protection auto-configuration problem (i.e., the security policies and the behavior of the CPSs). These models are later used to formulate the hard and soft constraints of the MaxSMT problem.

6.2.1 Communication Protection Policies model

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

C is the condition set, identifying the traffic portion the CPP must be applied to. C has the same formalization that has been introduced for modeling traffic headers, i.e., $C = (IPSrc, IPDst, pSrc, pDst, tProto)$. Specifically, the predicates $C.IPSrc$ and $C.pSrc$ refer to the traffic generated by the traffic sources, whereas $C.IPDst$, $C.pDst$ and $C.tProto$ refer to the traffic received by the destinations.

A^c and A^i are the sets of cipher algorithms that must be used to enforce, respectively, confidentiality and integrity (with authentication) on the traffic, while $\mathbb{A}^c$ and $\mathbb{A}^i$ are the sets of all the possible algorithms for confidentiality and integrity (i.e., $A^c \subseteq \mathbb{A}^c$ and $A^i \subseteq \mathbb{A}^c$).

S conveys the information about how the communication protection properties must be applied on the traffic. S is modeled as a tuple $S = (s_{ci}, s_{ii}, s_{ia})$, where each component can be a ternary value: true, false, or d.c. (which stands for "don't care"). Given a traffic t, s_{ci} is true if confidentiality must be enforced on the original header $t.h^i$, s_{ii} is true if integrity must be enforced on $t.h^i$ and s_{ia} is trues if integrity must be enforced on the additional header $t.h^a$ (if any).

W conveys the information about the trustworthiness or inspection status of network nodes and links. W is a tuple $W = (N^U, N^I, L^U)$, where $N^U \subseteq N_A$ is the untrustworthy nodes set, $N^I \subseteq N_A$ is the inspector nodes set, $L^U \subseteq L$ is the untrustworthy links set.

From the sets of the CPPs P and the set of all traffic flows F, it is possible to identify the subset $F^P \subseteq F$ of the flows that satisfy the conditions of at least one

$p \in P$, i.e., a flow $f = [n_s, t_{sa}, n_a, t_{ab}, ..., n_k, t_{kd}, n_d]$ belongs to F^P if $\exists p \in P$ such that the following two conditions are both true:

- t_{sa} satisfies predicates $p.C.IPSrc$ and $p.C.pSrc$, i.e.,

$$\eta(t_{sa}) \implies (p.C.IPSrc, *, p.C.pSrc, *, *) \tag{6.5}$$

- t_{kd} satisfies predicates $p.C.IPDst$, $p.C.pDst$ and $p.C.tProto$, i.e.,

$$\eta(t_{kd}) \implies (*, p.C.IPDst, *, p.C.pDst, p.C.tProto) \tag{6.6}$$

As the definition of F^P is analogous to the one of the F_r^M set introduced for the auto-configuration of firewalls, the subset $F^P \subseteq F$ can be computed with the maximal flow algorithm, which has been already discussed and illustrated in Section 5.2. After the computation of these flows, two utility functions can be employed to relate policies and flows:

- ϕ: $P \to \mathscr{P}(F^P)$ maps a policy p to the set of traffic flows computed for p;
- ρ: $F^P \to P$ maps a traffic flow f to the policy p for which it has been computed.

6.2.2 Communication Protection Systems model

The behavior of each CPS that is or may be allocated in the network depends on a set of rules, which report the actions that must be performed and the conditions identifying the traffic elements subject to those actions. This behavior must be modeled with free variables, as the solver that will be employed for solving the MaxSMT problem can have the freedom to choose correct and optimal values to establish the CPS behavior. In light of these considerations, for each $n \in N_A$ where a CPS may be allocated, a set of free variables is created and is composed of rules named "placeholder rules" because, at the moment of their creation, it is not yet known whether these rules will get concrete values or not in the final solution.

For a node $n \in N_A$, the placeholder rules set is denoted as R_n. Each rule $r \in R_n$ is modeled as a tuple $r = (C, a^c, a^i, S, m, act)$, where:

- $C = (IPSrc, IPDst, pSrc, pDst, tProto)$ identifies the traffic portion on which the rule actions must be enforced;

- a^c is the algorithm that is applied on the traffic to enforce confidentiality (it can be NULL);
- a^i is the algorithm that is applied on the traffic to enforce integrity (it can be NULL);
- S expresses the enforcement modes, and it is modeled with a tuple of three Boolean elements, similarly as the S element of a $p \in P$;
- m is a Boolean value, which is true if the traffic must be encapsulated through a tunnel-based VPN, false otherwise;
- act specifies if the security properties must be applied to t (i.e., when $act = protect$), or must be removed from it (i.e., when $act = unprotect$).

Each R_n contains one placeholder rule $r_{p,f}$ for each policy $p \in P$ and flow $f \in F^P$ such that n is crossed by f, as expressed in (6.7).

$$\forall p \in P. \forall f \in F^P. \, (n \in \pi(f) \implies r_{p,f} \in R_n) \tag{6.7}$$

Four main predicates, named $protect^c$, $protect^i$, $unprotect^c$ and $unprotect^i$, are introduced for expressing the security properties enforced by a CPS, where:

- $protect^x : N_A \times F \to \mathbb{B}$, with $x \in \{c, i\}$, maps a node n and a flow f to true if a CPS allocated on n adds the corresponding security property (confidentiality when $x = c$, integrity when $x = i$) to $\tau(f, n)$, to false if it does not add it;
- $unprotect^x : N_A \times F \to \mathbb{B}$, with $x \in \{c, i\}$, maps a node n and a flow f to true if a CPS allocated on n removes the corresponding security property from $\tau(f, n)$, to false if it does not remove it.

The values taken by such predicates are left free as well in the MaxSMT problem formulation.

Additionally, the predicate $supported : N_A \times \mathbb{A}^c \times \mathbb{A}^i \to \mathbb{B}$ maps a node in N_A and a pair of algorithms in $\mathbb{A}^c$ and $\mathbb{A}^i$ to true if the pair composed of those cipher algorithms is supported by that node, i.e., it has the capabilities required to apply them to network packets. For example, if an end point $n \in N^E$ or a network function $n \in N^O$ can create an IPSec channel and enforce the communication protection

properties with a cipher suite composed of AES-128-CBC and HMAC-SHA-256, then $supported(n, \text{AES-128-CBC, HMAC-SHA-256}) = \text{true}$.

Finally, the $delimiters: N_A \times N_A \times F \rightarrow \mathbb{B}$ predicate is defined to express when two nodes n_i and n_j delimit a VPN for a flow f in order to enforce policy $\rho(f)$. As shown in (6.8), n_i and n_j delimit a VPN for a flow f and policy $\rho(f)$ if they belong to the path $\pi(f)$ crossed by f, n_i precedes n_j, n_i enforces the protection required by $\rho(f)$ while n_j removes it, and in-between them there exists no any other node that enforces or removes protection on f for satisfying $\rho(f)$.

$$\begin{aligned} delimiters(n_i, n_j, f) = n_i \in \pi(f) \wedge n_j \in \pi(f) \wedge n_i \prec n_j \wedge \\ protect^x(n_i, f) \wedge unprotect^x(n_j, f) \wedge (\forall n_k \in \pi(f) | n_i \prec n_k \prec n_j. \\ \neg(protect^x(n_k, f) \vee unprotect^x(n_k, f))) \end{aligned} \tag{6.8}$$

As for the model of the packet filtering firewall behavior, also the abstract model proposed for CPSs is vendor-independent, but at the same time representative of the characteristics of real systems. The generation of settings for different implementations simply involves a syntax change with respect to the abstract model. To this end, the proposed model has been also mapped to an open-source VPN gateway,(i.e., strongSwan), to prove that a translation operation is enough to establish the low-level configuration of a CPS from the abstract model.

6.3 MaxSMT Problem Formulation

This section presents the hard and soft constraints defined for the formulation of the MaxSMT problem.

6.3.1 Constraints on CPPs enforcement

The enforcement of the CPPs is expressed with hard constraints, all of which must be satisfied to achieve a correct solution.

First, three classes of hard constraints express the trustworthiness information specified by each CPP $p \in P$.

- If a node n_i crossed by a flow f satisfying $p.C$ is defined as untrustworthy, at least a node n_j preceding n_i in $\pi(f)$ must enforce the required protection on

$\tau(f, n_j)$, and this protection must not be removed by any node n_k in-between n_j and n_i.

$$\begin{aligned} &\forall f \in \phi(p).\ \forall n_i \in \pi(f) |\ n_i \in p.W.N^U.\\ &\exists n_j \in \pi(f) |\ n_j \prec n_i.(protect^x(n_j, f) \wedge \\ &(\forall n_k \in \pi(f) |\ n_j \prec n_k \prec n_i.\neg unprotect^x(n_k, f))) \end{aligned} \tag{6.9}$$

- If a link l_{ab} such that nodes n_a and n_b are crossed by a flow f satisfying $p.C$ is defined as untrustworthy, at least a node n_j preceding n_b in $\pi(f)$ must enforce the required protection on $\tau(f, n_j)$, and this protection must not be removed by any node n_k in-between n_j and n_b.

$$\begin{aligned} &\forall f \in \phi(p).\ \forall l_{ab} \in p.W.L^U |\ n_a, n_b \in \pi(f).\\ &\exists n_j \in \pi(f) |\ n_j \prec n_b.(protect^x(n_j, f) \wedge \\ &(\forall n_k \in \pi(f) |\ n_j \prec n_k \prec n_b.\neg unprotect^x(n_k, f))) \end{aligned} \tag{6.10}$$

- If a node n_i crossed by a flow f satisfying $p.C$ is defined as inspector, f must not be enriched with the confidentiality property when crossing n_i. If a node n_j preceding n_i in $\pi(f)$ enforces confidentiality on $\tau(f, n_j)$, this protection must be removed by a node n_k in-between n_j and n_i.

$$\begin{aligned} &\forall f \in \phi(p).\ \forall n_i \in \pi(f) |\ n_i \in p.W.N^I.\\ &\forall n_j \in \pi(f) |\ n_j \prec n_i.\ (protect^c(n_j, f) \implies \\ &(\exists n_k \in \pi(f) |\ n_j \prec n_k \prec n_i.\ unprotect^c(n_k, f))) \end{aligned} \tag{6.11}$$

Then, two classes of hard constraints express the guarantee that the protected traffic can reach its destination, and that it is plain when it actually reaches the destination.

- If a node n_i adds protection to a flow f, this protection must be removed by a node n_j following n_i in $\pi(f)$, where node n_j can also be the destination node.

$$\begin{aligned} &\forall f \in \phi(p).\ (\exists n_i \in \pi(f).\ protect^x(n_i, f)) \implies \\ &(\exists n_j \in \pi(f) | n_i \prec n_j.\ unprotect^x(n_j, f)) \end{aligned} \tag{6.12}$$

- A flow f satisfying $p.C$ can reach its destination if no node in $\pi(f)$ blocks it.

$$\forall f \in \phi(p).\ \forall n_i \in \pi(f).\ \neg deny(n_i, \tau(f, n)) \tag{6.13}$$

Finally, two classes of hard constraints express the conditions under which a traffic is tunneled by a CPS.

- If the two nodes n_i and n_j delimiting a VPN for a flow f are end points, the traffic is not tunneled.

$$\begin{aligned} &delimiters(n_i, n_j, f) \wedge n_i \in N^E \wedge n_j \in N^E \implies \\ &\forall n_k \in \pi(f) | n_i \prec n_k \preceq n_j . \neg tunneled(\tau(f, n_k)) \end{aligned} \tag{6.14}$$

- If at least one of the two nodes n_i and n_j delimiting a VPN for a flow f is not an end point, the traffic is tunneled.

$$\begin{aligned} &delimiters(n_i, n_j, f) \wedge \neg(n_i \in N^E \wedge n_j \in N^E) \implies \\ &(\forall n_k \in \pi(f) | n_i \prec n_k \preceq n_j . \, tunneled(\tau(f, n_k))) \wedge \\ &\tau(f, \nu(f, n_i)).h^a.IPSrc = address(n_i) \wedge \tau(f, n_j).h^a.IPDst = address(n_j) \end{aligned} \tag{6.15}$$

The presence of all these classes of hard constraints may constrain some values of the *protect*, *unprotect* and *deny* predicates to be assigned with specific values, depending on the decisions taken by the solver of the MaxSMT problem.

6.3.2 Constraints on network functions behavior

The hard constraint defined in (6.13) imposes that the protected traffic finally reaches its destination. In order to provide formal guarantee of this event, it is also necessary to build some hard constraints upon the $deny_i$ predicate, for each node crossed by the flow the protected traffic belongs to. These hard constraints strictly depend on the service function type and on its configuration settings.

The definition of these hard constraints is the same as for the firewall-auto configuration problem. Specifically, Section 5.1 reports two examples (traffic monitoring functions and network address translating functions) which are still valid for the VPN auto-configuration problem.

6.3.3 Constraints on CPSs allocation and configuration

Other hard constraints are also necessary to express the CPSs allocation and configuration decisions for each node $n \in N_A$.

The allocation decision for node n is modeled by the $allocated(n)$ predicate, which is left free in the problem formulation. The only exception is represented by nodes of the N^V set, because they already have a CPS capability. This is expressed

by the hard constraints shown in (6.16).

$$\forall n \in N^V \,.\, allocated(n) \tag{6.16}$$

The configuration decision is modeled by the $configured : R_n \to \mathbb{B}$ predicate which takes value true for a placeholder rule $r \in R_n$ if r is actually used, false otherwise. If $configured(r)$ = true, the actual configuration rule is determined by the values assigned to the free variables modeling r, which are constrained by the enforcement of the CPPs. Those constraints are defined over the *protect* and *unprotect* predicates. However, these same predicates are bound to the parameters of the placeholder rules by the additional hard constraints shown in (6.17) and (6.18).

$$\begin{aligned} protect^x(n,f) \implies \exists r \in R_n.(configured(r) \land \tau(f,n) \subseteq r.C_r \land \\ r.act = protect \land r.k \in \rho(f).K \land r.a^x \in \rho(f).A^x \land r.S = \rho(f).S \land \\ r.m = tunneled(\tau(f,v(f,n))) \land supported(n, r.a^c, r.a^i)) \end{aligned} \tag{6.17}$$

$$\begin{aligned} unprotect^x(n,f) \implies \exists r \in R_n.(configured(r) \land \tau(f,n) \subseteq r.C_r \land \\ r.act = unprotect \land r.k \in \rho(f).K \land r.a^x \in \rho(f).A^x \land r.S = \rho(f).S \land \\ r.m = tunneled(\tau(f,n))) \land supported(n, r.a^C, r.a^I)) \end{aligned} \tag{6.18}$$

In particular, (6.17) requires that if a CPS n must enforce a communication protection property x (with $x = c$ for confidentiality, $x = i$ for integrity) on flow f due to the CPP $\rho(f)$, then it must have a configured communication protection rule with a "protect" action, that can enforce the specified protection on $\tau(f,n)$, and it must also support the configured technology and algorithm. Similarly, (6.18) requires that if a CPS n must remove a communication protection property x on the traffic flow f due to the request of CPP $\rho(f)$, then it must have a configured communication protection rule with an "unprotect" action, that can remove the specified protection from $\tau(f,n)$, and it must also support the configured technology and algorithm, as shown in (6.18).

Finally, if a node has at least a communication protection rule configured (i.e., it applies communication protection properties to a packet class), this implies that in that node a CPS has to be allocated, which is expressed by the additional hard constraint (6.19).

$$(\exists r \in R_n.\, configured(r)) \implies allocated(n) \tag{6.19}$$

6.3.4 Constraints on the optimization profiles

The optimization objectives represented by the input optimization profile are translated into soft constraints.

The *min-allocation* profile requires that the allocation of CPS on end points is preferred over other network nodes. Two classes of soft constraints (6.20) and (6.21) are required to enforce this optimization objective. Each soft constraint of the first class is satisfied if no CPS is allocated in the corresponding end point. Each soft constraint of the second class is satisfied if no CPS is allocated in the corresponding intermediate node. Then, (6.22) sets the weight for (6.21) greater than the sum of the weights for (6.20), so that it expresses the preference to allocate CPSs on the end points.

$$\forall e \in N^E.\ Soft(\neg allocated(e), w_e) \tag{6.20}$$

$$\forall n \in N_A \backslash N^E.\ Soft(\neg allocated(n), w_n) \tag{6.21}$$

$$\forall n \in N_A \backslash N^E. \left(\sum_{e \in N^E} w_e \right) < w_n \tag{6.22}$$

Instead, the *min-bandwidth* profile prefers allocating CPS functionalities in intermediate nodes rather than in end points. In this case, (6.23) imposes a different relationship between the weights of soft constraints (6.20) and (6.21), so as that it is preferable to allocated CPS functionalities in all intermediate nodes rather than in a single end point.

$$\forall e \in N^E. \left(\sum_{n \in N_A \backslash N^E} w_n \right) < w_e \tag{6.23}$$

6.3.5 Solution computation

All these hard and soft constraints are grouped to compose a MaxSMT problem, representing the allocation and configuration problem for CPSs. Then, a MaxSMT solver is employed to search for the optimal solution that satisfies all hard constraints. If a solution is found, then it is expressed by the values the solver assigns to the free variables and predicates. In particular, the two outputs, i.e., allocation scheme and protection rules for the CPSs, can be easily retrieved by those values. For each $n \in N_A$, *allocated(n)* states if a CPS has been allocated by the solver in that network position or not. Therefore, the allocation scheme is defined by all the output values computed for this predicate. Instead, for each $n \in N_A$ such that *allocated(n)* $=$ true,

the configuration of the allocated CPS is made up of the rules $r \in R_n$ such that $configured(r)$ = true. The solver also assigns, for each one of these rules r, a specific value for each free variable that models it, e.g., the rule conditions are specified by $r.C$, and the information about the algorithms to be applied by $r.a^c$ and $r.a^i$.

6.4 Implementation and Validation

As for firewalls, also the application of the VEREFOO approach to VPN configuration has been implemented by means of a Java framework, which exploits the APIs offered by the z3 solver [164] to formulate and solve the MaxSMT problem. The framework has been validated in different ways: on specific use cases to prove the optimization and correctness of the approach (Subsection 6.4.1), and on synthetically generated networks, to prove the scalability of the approach (Subsection 6.4.2). All the MaxSMT instances have been solved on a machine with an Intel i7-6700 CPU at 3.40 GHz, 32GB of RAM, and z3 version 4.8.5.

6.4.1 Correctness and optimization verification

The correctness and optimization of the VEREFOO approach for CPS configuration has been checked on a specific use case. The AG of this use case is illustrated in Fig. 6.1. In this scenario, the human administrator specifies multiple CPPs. For example, the traffic between each subnetwork (i.e., e_4, e_5, e_6, e_7 and e_8) and the servers (i.e., e_1, e_2, e_3), must be encrypted. Each subnetwork also requires a different encryption algorithm (e.g., AES-GCM-128 for e_4, 3DES-CBC for e_5). Besides, the traffic between each pair of subnetworks must be protected by ensuring its integrity, with the exception of the pair of subnetworks e_7 and e_8. For all these CPPs, f_{10} and f_{11} are untrustworthy nodes, because the users of these devices are not sufficiently trusted, so the traffic should be encrypted when crossing them. Instead, f_{12} is an inspector node, because it is an intrusion detection system that must check all the traffic.

The developed framework has been run on these inputs and has produced the solution shown in Fig. 6.2, for which both optimization and correctness have been checked.

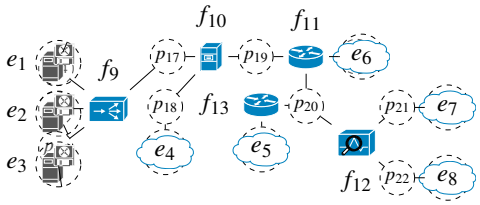


Fig. 6.1 Allocation Graph of the use case

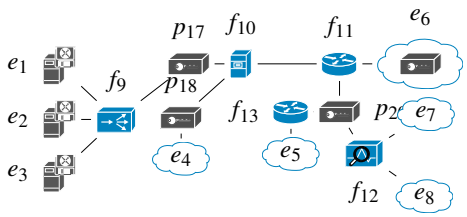


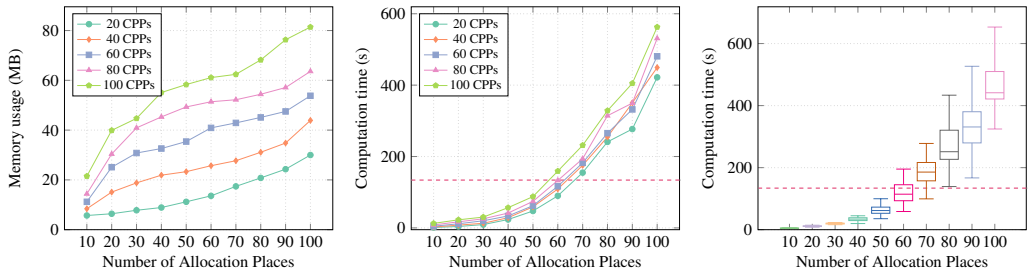
Fig. 6.2 CPSs allocation scheme of the use case

With regards to optimization, the solution computed by the framework has been checked to be the one that minimizes the number of allocated CPSs and configured rules, by comparing it with all the possible solutions to the same problem instance. For example, a single CPS is installed in p_{17} on the right of load balancer f_9, instead of having three separate CPSs on its left. Similarly, a CPS is allocated in p_{20}, so that it can manage traffic flows involving three subnetworks (e_5, e_7, e_8) instead of a single one.

With regards to correctness, the result has been checked with Mininet, an emulator which can be used to test virtualized networks. Each allocated CPS is emulated as a Strongswan VPN gateway, and its configuration is derived from the rules computed by the tool, after having translated them into the Strongswan VPN configuration file language. This translation step consists in a simple syntax translation. After configuring each element of the network simulated in Mininet, it has been verified that each traffic has the requested communication protection properties by analyzing the traffic on the different nodes. Similar experiments have been done with variations of this use case.

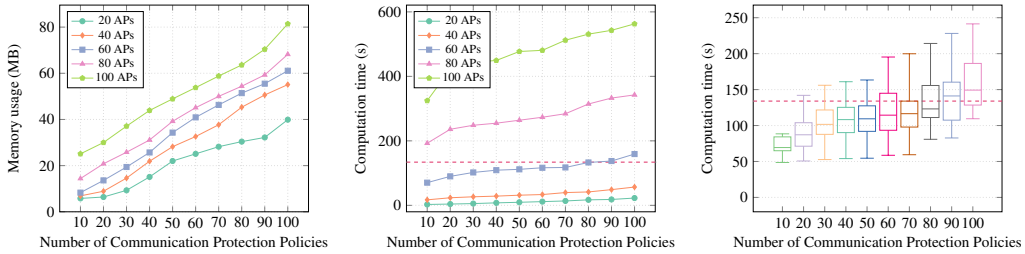
6.4.2 Scalability evaluation

The scalability, in terms of computation time and memory usage, of the approach has been tested for two main parameters, i.e., the number of APs and the number



(a) Memory usage chart (b) Computation time chart (c) Whisker plot

Fig. 6.3 Scalability for increasing number of Allocation Places



(a) Memory usage chart (b) Computation time chart (c) Whisker plot

Fig. 6.4 Scalability for increasing number of Communication Protection Policies

of CPPs that must be enforced in the networks. The network topologies of the AGs where the tests have been carried out are artificially synthesized as extensions of the network illustrated in Fig. 6.1. Fig. 6.3 and Fig. 6.4 report all the results that have been obtained for the scalability validation of the framework.

Fig. 6.3a and Fig. 6.4a show the peak memory usage of the framework, for increasing values of the two analyzed parameters. As it can be seen from the two plots, the memory requirements for the usage of the tool are in line with similar software programs. In fact, even in the worst case that has been analyzed (i.e., a scenario composed of 100 APs and 100 CPPs), the peak memory usage that has been measured is only 81.4 MB.

Fig. 6.3b and Fig. 6.4b show the computation time of the framework. In these plots, each value is the average computed over 50 runs of the framework on the same problem instance, i.e., with the same topology and the same CPPs, but with different IP addresses. As it was already explained in Section 5.4, when a programming constraint problem is built upon the integer theory, the performance of the Z3 solver can significantly vary depending on the actual integer numbers that are used. As each IP address is modeled as the conjunction of four predicates defined over integer

variables, their values have a non-negligible impact to the overall performance of the framework. This consideration is further confirmed by the value distribution the whisker plots depicted in Fig. 6.3c and Fig. 6.4c, for which the number of policies is fixed to 60 in Fig. 6.3c, and the number of APs is fixed to 60 in Fig. 6.4c. The parameter that most impacts the performance of the framework is the number of APs, with respect to the number of CPPs. The reason is that that for each possible CPS a certain number of placeholder rules must be defined, depending on the number of traffic flows that cross it. Each rule is composed of free variables, and for each one of them the solver must decide the optimal value. Therefore, the number of possible solutions becomes quite high. Nevertheless, considering a topology composed of 100 nodes, the combined allocation and configuration problem of the CPSs would be an impractical task if performed manually, it would take hours and probably would end up with errors or sub-optimizations.

Additionally, as already discussed in Chapter 3, in the literature there are no other approaches that jointly solve the allocation and configuration problem for CPSs, and that combine the three features of automation, optimization, and formal correctness into a single methodology. Besides, the papers describing related methodologies do not provide an explicit validation for them. The only exceptions are the approaches described in [96], [55] and [58]. On the one hand, the methodologies proposed in [96] and [55] can only automatically compute the communication protection rules for CPSs in fixed positions in a network chain. The scalability of these techniques is also quite limited. On the other hand, the approach in [58] only establishes the CPS allocation scheme without computing their protection rules. For a small network with around 20 positions where CPSs may be allocated, the required computation time is in the same magnitude order as the VEREFOO approach (i.e., tens of seconds). In summary, despite the related work addresses simpler or partial problems, the performance of our methodology, which provides superior results, is still in line with the previous state of the art.

Finally, as also done in Section 5.4, in Fig. 6.3b, 6.4b, 6.3c, and 6.4c, a baseline (red dotted horizontal line) is introduced, in order to have a reference. It is again the Deployment Process Delay (DPD) introduced by a well-known orchestrator (Open Source MANO) for deployment. DPD is the time the orchestrator takes to deploy and instantiate a VNF within an already booted VM and set up an operational network service. According to [168], this time is 134ms, which is in the same magnitude

order as the computation time of the VEREFOO approach to solve the automatic allocation and configuration problem for CPSs.

Automatic Network Security Orchestration

Chapter 7

Orchestration of Firewall Reconfiguration Transients

This chapter presents the ***F**irewA**l**l **T**ransients **O**ptimizer* (FATO) approach, which has been designed to automatically compute the optimal scheduling of the reconfiguration changes for a distributed firewall, so as to minimize the number of intermediate transient states where the network security policies are violated.

7.1 Problem Statement

This section firstly characterizes the transient problem for a distributed firewall reconfiguration. Then, it explains the issues that may derive from an incorrect or unoptimized transient management under real-world constraints. A motivating example, based on a realistic network, is used to underline further the problems occurring when connectivity policies are not adequately respected during a reconfiguration transient. Finally, a solution based on automation, formal methods and optimization strategies is proposed to overcome the issues.

7.1.1 Characterization of a firewall reconfiguration transient

As already discussed in Chapter 5, the configuration of a distributed packet filtering firewall involves two management aspects at the same time: the establishment of the

allocation scheme and the definition of the filtering rules. Therefore, it is clearly a complex task, which requires a high level of expertise if manually managed by a human being. Nevertheless, multiple approaches have been presented in the literature, where this task is performed automatically without the need of manual interventions ([69], [10] [11]). These contributions have eased the work of security managers and, at the same time, have strengthened the security guaranteed by such a kind of network functions. In light of this, nowadays it is easier and more common to establish a new firewall configuration, when the original configuration is not valid anymore. Specifically, the invalidity of the original configuration may be due to network topology changes (e.g., when a server is scaled in such a way to have new multiple instances, then all the connectivity policies concerning that server should be enforced for each new instance as well), or due to indications provided by the security manager (e.g., an intrusion detection system might have informed the security manager about an incoming attack and therefore there is the need that a network host must be isolated from the others).

When a new firewall configuration is computed, it differs from the initial configuration for at least one of the two management aspects: the allocation scheme might have been changed (e.g., a new firewall instance has been introduced, or an existing one has been removed), or the firewall rules might have been adjusted to be compliant with new connectivity policies. Therefore, the security service must be updated accordingly, by applying a series of operations of different types: deployment of a new virtual firewall, removal of an existing firewall, update of the filtering rules of a firewall, deviation of a traffic flow so that it can reach a new firewall that was not part of the initial distributed firewall. The firewall reconfiguration transient consists of a specific ordering of these operations, so that the global configuration is changed from the initial state to the target one. The number of intermediate states corresponding to this transient is thus equal to the number of changes that are applied to the firewall configuration, i.e., the number of newly deployed instances, the number of removed instances, the number of modified filtering rule sets.

7.1.2 Issues of a firewall reconfiguration transient

Under real-world constraints, the security preservation of the connectivity policies during reconfiguration transients becomes an important matter when the time length of these transients is not negligible. If a transient lasts only a few seconds, the

problem is less felt because, in such a short span of time, an attacker could not easily perform access control violations, privilege escalations, or other attack types that undermine the connectivity policies. However, this is not the typical case of virtual networks.

From the analysis of the studies discussed in Section 3.3, the typical transients that have been evaluated are composed of a few tens of states. Each state transition consists of the deployment/removal of a virtual function (e.g., a softwarized SDN switch, a Virtual Machine) or the update of the rules of a function (e.g., all the rules of a virtual firewall). The time required for these operations may not be negligible. On the one hand, Openstack requires more than 5 seconds to deploy a single machine [169], and it has an update rate of 250 rules per second [170], if all the rules pertain to the same machine. On the other hand, a well-known NFV orchestrator, i.e., Open Source MANO, requires a Deployment Process Delay (DPD) of 134s [168], where DPD is the time the orchestrator takes to deploy and instantiate a VNF within an already booted VM and setup an operational network service. Therefore, supposing that a transient is composed of 20 states and each of them consists of the deployment of a virtual machine, the transient may require around 100 seconds in an environment based on Openstack, more than 10 minutes with Open Source MANO. Parallelization may improve these worst-case times, but not drastically. Transients requiring some minutes are common in big virtual networks, and these long times are perfect chances for intruders to exploit intermediate states where services are not protected.

A reconfiguration transient is triggered to update a firewall configuration so as to comply with a different set of connectivity policies (i.e., reachability and isolation policies), which must be enforced at least in the final reconfiguration transient state. In this dissertation, these policies will be referred to as target connectivity policies. As anticipated, they might be satisfied in each intermediate state, even though preserving their satisfaction as much as possible would be required to ensure a high security level during configuration. In particular, two well-known issues that may occur due to this problem are discussed below.

The first issue is *service disruption*. For example, let us suppose that a service is linked to clients external to its subnetwork through two paths, each guarded by a firewall, but only one of them has a rule allowing communication between the server and the clients. If the firewall reconfiguration establishes that the allowing rule must be removed from the original firewall node and added to the other one, if the latter

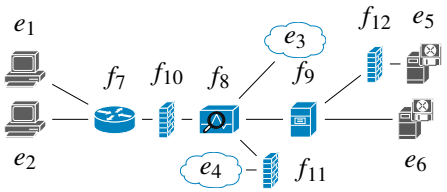


Fig. 7.1 Network topology example

operation is executed some seconds or minutes after the former, the service becomes unaccessible for the clients during the transient.

The second issue is the possible opening to cyber attacks related to *undetected intrusions*. For example, let us suppose that a firewall must be removed, another one must be set up in a different position of the graph, and the first operation happens before the second one. In that case, there might exist a period (i.e., some intermediate states of the transient) where some kinds of traffic, which were previously blocked by the removed firewall and would be later blocked by the firewall to be deployed, can pass through those positions, thus violating some isolation policies.

7.1.3 Motivating example

These problems can also be explained with the aid of a motivating example. Fig. 7.1 represents a snapshot of the configuration of a distributed firewall, with three instances, in a network whose topology is a simplified version of a real one, i.e., the network of our university department. At a certain time, a cyber-attacker manages to take over node e_2, hosting a *mysql* service. This event demands to block any communication towards e_2, and at the same time to make the *mysql* service hosted by node e_5 available for any other network component, as it is a mirror of that in e_2. The security manager is thus required to perform multiple changes to the firewall configuration: (i) a new instance must be deployed between e_2 and f_7 to make the former inaccessible; (ii) f_{12} must be removed, as e_5 must become the replacement of e_2; (iii) the rule sets of f_{10} and f_{11} must be updated, so as to allow traffic respectively between e_1 and e_4 on one side, and the *mysql* service listening to port 3306 on the other side.

Deciding the ordering of these operations is not trivial and it can impact network security or service availability during the transient. If the new firewall instance

blocking e_2 is deployed before removing f_{12} or updating f_{10} and f_{11}, any secondary effect of the cyber-attack is immediately stopped, but the *mysql* service remains unavailable for a longer time. Alternatively, if first f_{12} is removed, that service can be accessed by some network nodes (but not all of them, until f_{10} and f_{11} are not updated). However, the attacker that has taken control over of e_2 would still have some time to propagate the attack to other parts of the company. Therefore, the decision depends on the priority that is assigned to the connectivity policies.

Additionally, this decision should be taken in quite strict times, as demanded by ever-changing virtual environments. Due to these circumstances, human beings are under more pressure and more prone to make mistakes. So, not only the scheduling of configuration changes may be sub-optimal (i.e., the connectivity policies are not enforced in as many states as possible), but it may even be incorrect (e.g., a policy that must not be violated in any transient state is violated in at least one of them).

7.1.4 Solutions to improve transient management

In light of all these considerations, when a reconfiguration transient is triggered, its management may be optimized if the reconfiguration changes for the distributed firewall are ordered so as to maximize security for the intermediate states of the transient. This optimization objective translates into maximizing the number of target connectivity policies that are already satisfied in each transient state. Besides, in almost all the cases in which this objective is fulfilled, the security policies are also satisfied as early as possible, because the enunciate objective aims at their enforcement since the first states of the transient. For example, when the position of a firewall must be changed to block a specific traffic flow, the reconfiguration transient involves two operations as it is common practice in virtual networks, i.e., removing the current instance and deploying the new one in the required position. Only performing the latter operation before the former satisfies the optimization objective previously expressed, because in this way the isolation policy related to that traffic flows is already satisfied, even before removing the old instance.

However, establishing an optimal scheduling of a distributed firewall reconfiguration changes, compliant with the definition of this objective, becomes manually unmanageable as the complexity of the distributed firewall and the number and complexity of policies increase. Therefore, this task should be automated so that the

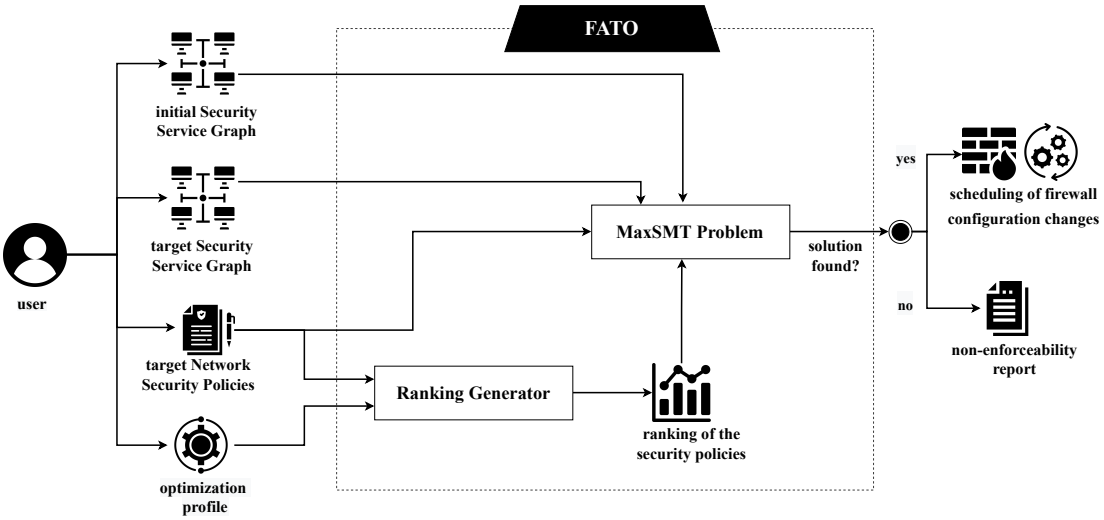


Fig. 7.2 Workflow of the approach

scheduling is automatically computed based on information related to the network topology and the connectivity policies. Additionally, by automatizing this task, it is possible to enrich the process with two other important factors, i.e., optimization and formal verification. On one side, commonly some policies are more important than others. This aspect should thus be considered when computing the scheduling of the operation changes, but at the same time represents another factor that makes a manual operation impractical or deeply unoptimized. On the other side, providing higher assurance that the computed scheduling is correct would be an important feature for environments where safety-critical or mission-critical systems are present.

To summarize, automation, paired with formal methods and optimization, comes in handy for overcoming these limitations, and reaching quickly a solution characterized by high confidence in its correctness.

7.2 The Proposed Approach

This section describes the approach pursued in the definition of the ***F**irew**A**ll **T**ransients **O**ptimizer* (FATO) methodology, which aims to solve the stated problem automatically. Fig. 7.2 depicts the full workflow of the proposed approach, showing the inputs specified by the user (Subsection 7.2.1) and the interaction among the different components of the FATO methodology (Subsection 7.2.2).

7.2.1 Inputs for FATO

FATO requires the following inputs:

- the initial virtual network topology, together with the initial configuration of the distributed firewall (i.e., the start state of the reconfiguration transient);
- the target virtual network topology, together with the target configuration of the distributed firewall (i.e., the final state of the reconfiguration transient);
- a set of target connectivity policies that must be satisfied by the target configuration. Optionally, a subset of these policies may be specified as a special class, called *persistent* policies: when a policy belongs to this class, it must be satisfied throughout the whole transient, not only in the final state;
- an optimization profile, providing FATO with useful information to establish the relative priority of the connectivity policies. Some optimization profiles additionally require the specification of a partial or total order relationship for the policies.

The first inputs, i.e., the initial and target network topologies, enriched with the initial and target firewall configurations, are referred to as *initial Security Service Graph* (G_I) and *target Security Service Graph* (G_T). For each one of them, the firewall configuration is composed of the allocation scheme of the firewall instances and the filtering rules of each instance. The allocation scheme represents how the firewalls have been positioned in the network topology, which may also be composed of other types of network and security functions. Instead, the filtering rules for each instance of the packet filtering firewall are composed of a set of well-known IP 5-tuple-based rules and a default action applied whenever no rule matches a received packet. Specifically, each firewall rule defines the conditions to be matched by the values of the five elements of the IP 5-tuple (i.e., source and destination IP addresses, source and destination ports, transport-level protocol) and the corresponding action to be enforced when the conditions are satisfied. The firewall rules belonging to the initial and target configuration are respectively called *initial Firewall Rules* (R_I) and *target Firewall Rules* (R_T).

The third input, i.e., the connectivity policies that must be satisfied by the target configuration, are called *target Network Security Policies* (P_T). Connectivity policies

specify which packet flows, identified by the values of the IP 5-tuple fields, must reach their destination, and which ones must instead be blocked. Therefore, they can be respectively divided into reachability and isolation policies. The P_T set includes both persistent and non-persistent policies.

The fourth input, i.e., an optimization profile, is a compact indication for FATO about the relative priority of the connectivity policies. The following profiles have been defined, but more can be defined:

- *security-max*: the objective is that the isolation policies must have higher priority than the reachability policies;
- *service-max*: the objective is that the reachability policies must have higher priority than the isolation policies;
- *policy-max*: the objective is to maximize the number of policies that are satisfied in each intermediate state;
- *state-max*: the objective is to maximize the number of intermediate states where each policy is satisfied depending on an order relationship specified by the user.

The first two profiles (i.e., *security-max* and *service-max*) allow the user to additionally specify a partial order relationship for each group of policies (i.e., for the group of isolation policies and the group of reachability policies). For each group, the user can define some policies with higher priority and other policies with lower priority. The *policy-max* profile does not allow the user to specify any relationship between policies. Instead, the *state-max* profile always requires a partial or total order relationship defined by the user for the policies. Note that relationships cannot be defined for persistent policies, because they must be enforced in any intermediate transient state a-priori.

7.2.2 FATO Methodology

The FATO methodology works as follows.

Firstly, from the specification of the optimization profile and, optionally, the partial or total order relationship for the policies, FATO defines a ranking for the input

policies (with the exclusion of the persistent policies, because they must be enforced in any intermediate state), as it comes handy for the definition of the optimization problem. The same rank can be assigned to multiple policies, if they have the same relative priority. The user could have personally defined this ranking. Still, such an operation would not have been suitable to be manually performed by a human being. Possible reasons may be that the number of policies may be high, or the person in charge of this task may want to specify only a partial order relationship between the policies instead of a complete ranking. Additional information about the ranking generation is provided in Section 7.4.

Then, the initial and target security graphs with the firewall configurations, the target policies and their ranking are used by FATO to formulate a MaxSMT problem. Throughout this formulation, the aim of the methodology is to maximize the number of intermediate states in the transient from G_I to G_T where the policies of the P_T set are enforced. After solving this optimization problem, FATO identifies the optimal order of configuration changes, in such a way that the optimization fulfills the criteria derived from the ranking (i.e., from the optimization profile and the order relationship for the policies, specified by the user). This scheduling can be followed by a human who manages the virtual network, or a state-of-the-art orchestrator can exploit it to perform the required actions. Besides, according to the correctness-by-construction principle enabled by MaxSMT, the computed solution is correct, as long as the formal models defined for the problem (i.e., the set of first-order logic formulas that express them) correctly model the real problem. A more detailed discussion about the conditions under which the solution correctness is guaranteed has been already presented in Section 5.3, and it applies also to this application of the MaxSMT formulation.

7.3 Formal Models

In order to implement the approach outlined above, a formalization of the network components and of the security policies is required. This section deals with the illustration of such formal models. TABLE 7.1 includes the main formal notations (symbols, functions, predicates, operators) used in this chapter.

Symbol/Function/Predicate/Operator	Definition
$\mathbb{B} = \{\text{true, false}\}$	Boolean set
$G_I = (N_I, L_I)$, $G_T = (N_T, L_T)$	initial and target Security Service Graphs
$G_U = G_I \cup G_T = (N_U, L_U)$	union Security Service Graph
$n_k \in N_U$	the element of N_S identified by k
$c_1 = \lvert N_U \backslash N_I \rvert$	number of times when a firewall becomes active
$c_2 = \lvert N_U \backslash N_T \rvert$	number of times when a firewall is removed
$c = c_1 + c_2$	number of transient states
$S = [s_0, s_1, .., s_{c-1}, s_c]$	state sequence
T	set of all the packet classes
$t = (IPSrc, IPDst, pSrc, pDst, tProto)$	single traffic
t, t^0	a class of packets and empty set of packets
F	set of all traffic flows
$f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d]$	traffic flow
d_n	default action of firewall n
R_n	filtering rules of firewall n
$r = (c_r, a_r)$	single rule of firewall n
$c_r = (IPSrc, IPDst, pSrc, pDst, tProto)$	rule condition of firewall n
a_r	rule action of firewall n
$P_T = P_T^P \cup P_T^N$	set of Network Security Policies
P_T^P	persistent Network Security Policies
P_T^N	non-persistent Network Security Policies
$p = (C, a)$	single Network Security Policy
$p.C = (IPSrc, IPDst, pSrc, pDst, tProto)$	policy condition
$p.a$	policy action
M_D	dominance matrix
π: $F \rightarrow (N_U)^*$	maps a flow to the ordered list of
τ: $F \times N_U \rightarrow T$	maps a flow and a node to the ingress traffic
transform: $N_U \times T \rightarrow T$	maps a node and a traffic to the transformed traffic
bti: $\mathbb{B} \times \{0,1\} \rightarrow T$	maps a Boolean value to the corresponding integer
active: $N_U \times S \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the node is active in the state s
configUpdate: $N_U \times N_U \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the two nodes share the same AP
deny: $N_U \times T \rightarrow \mathbb{B}$	true $\Leftrightarrow$ n drops all the packets of traffic t
match: $R_n \times T \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the rule conditions match the traffic
satisfied: $P_T \times S \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the policy is satisfied in the state s
$t_1 \subseteq t_2 \in T$	t_1 is a sub-traffic of t_2
$p \succ p' \in P_T^N$	p dominates p'
$p \parallel p' \in P_T^N$	p and p' are independent
$\wedge$, $\vee$, $\neg$	used for conjunction, disjunction, negation
.	used to denote a specific tuple element (e.g., given a tuple $t = (a,b,c)$, $t.a$ identifies element a of tuple t)

Table 7.1 Notation

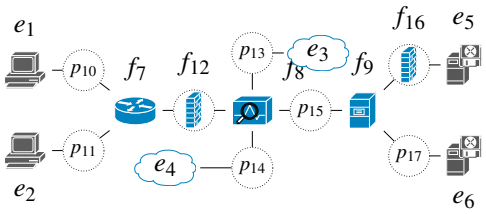


Fig. 7.3 Security Service Graph with firewall allocation scheme

7.3.1 Security Service Graphs model

The initial and target Security Service Graph are modeled as directed graphs, i.e., respectively $G_I = (N_I, L_I)$ and $G_T = (N_T, L_T)$, by reusing the same notation that was applied for the Allocation Graph model (i.e., G_A) of the VEREFOO approach in Section 5.1 and Section 6.1. As for G_A, also the vertex sets of these two graph models include *Allocation Places* (APs), i.e., the the logical positions where a firewall instance may be positioned. For example, supposing that Fig. 7.3 depicts an initial Security Service Graph, in this example two of the APs are effectively filled with firewalls (i.e., f_{12} and f_{16}). The others are empty, but some instances may be deployed there in a Security Service Graph representing a different state of the transient (e.g., the final one). Each element of N_I and N_T is uniquely identified by a non-negative integer index k, i.e., $index(n_k) = k$. Analogously, each element of L_I is uniquely identified by a pair of non-negative integers, i.e., $l_{ij} \in L_S$, with $i \neq j$, is the edge from n_i to n_j. It is possible that an element of N_T has the same index as an element of N_I, and it is possible that an element of L_T is denoted by the same pair of non-negative integers as an element of L_I. This simply means that the elements are the same for both the graphs.

Then, the concept of *union Security Service Graph* (G_U) is derived from these two graph models. It is a directed graph modeled as $G_U = (N_U, L_U)$, where $N_U = N_I \cup N_T$ and $L_U = L_I \cup L_T$. It basically represents the union of the initial and target graphs, where a single instance of the nodes with the same index which appear in both is kept, and the same is applied to the links. It is evident that, in this graph, nodes which are not active at the same time might be present (e.g., a firewall which was present in G_I, but has been removed in G_T). Nevertheless, this representation is useful to consider all the possible paths in any situation, independently of the moment when a node might be active.

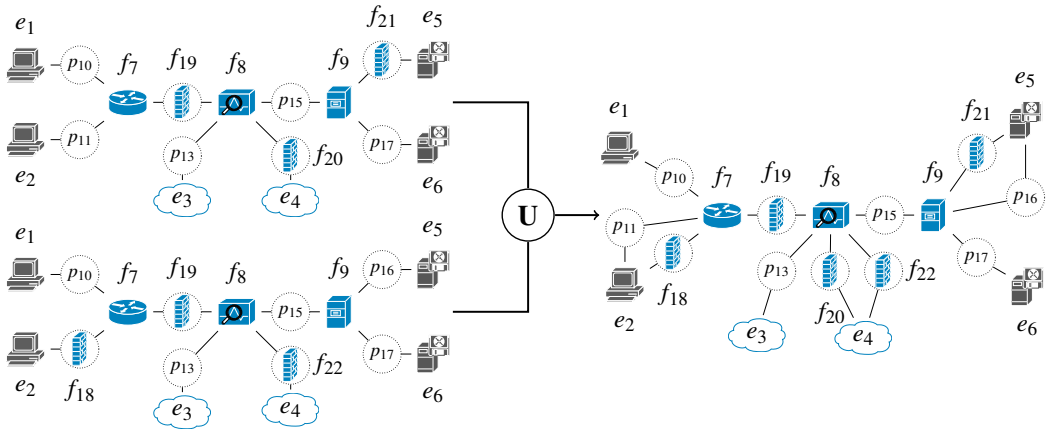


Fig. 7.4 Generation of the union Security Service Graph

A clarifying example of how G_U is computed from the input G_I and G_T is represented in Fig. 7.4. In their models, the APs that have not been filled with firewall instances are still present, but they have the simple role of forwarders, i.e., it is as if they forward each received packet to the next hop. Their presence eases the formalization of the state models and the hard constraints of the MaxSMT problem. In the automatically derived G_U, it is possible that two firewalls, with different indexes, refer to the same logical position (as for nodes f_{20} and f_{22}). This means that, even though they share the same position in the allocation scheme, they have different rules, so they are distinct entities in the model.

To this regard, in some types of virtualized networks (e.g., where firewalls are implemented as Virtual Machines, and not as containers), the configuration of a single instance is allowed, without the need to instantiate a new process replacing it. This possibility is modeled with the $configUpdate : N_U \times N_U \rightarrow \mathbb{B}$ predicate, which is true if the two input nodes of N_U share the same AP and they represent two different configurations of the same virtual firewall instance. Referring to the previous example of Fig. 7.4, if G_U derives from the representation of an NFV-based network, and f_{20} and f_{22} represent two different configurations for the same instance, then $configUpdate(f_{20}, f_{22}) = true$.

7.3.2 State Sequence model

In the transformation of G_I into G_T, four elementary configuration changes have been considered: (1) deployment of a new firewall instance, (2) removal of an existing

firewall instance, (3) update of the filtering rules for a firewall instance, (4) deviation of the traffic that was sent to a removed firewall instance to another instance that has been deployed. In modeling the sequence of states characterizing the transient due to the distributed firewall reconfiguration, however, it is enough to consider the number of changes of type (1) and (2), i.e., deployment and removal of single instances. The reasons why the other change types are not explicitly represented for the state sequence models are explained below.
Regarding the update of the filtering rules for a firewall instance, this event cannot be always performed instantaneously. If a firewall implemented as a virtual machine in an NFV-based network could be updated very quickly, the same does not apply to a container, where there is instead the necessity to launch a new firewalling process, with the newly required rule set. Therefore, the rule update can be represented as a combination of firewall deployment and removal operations, and for the NFV-based example it is enough to introduce a specific hard constraint in the MaxSMT problem so that during the rule update other configuration changes are not performed (more details are presented later, in Subsection 7.5.2).
Regarding the deviation of a traffic flow so that it can reach a different firewall located in the same AP where a removed one was previously present, this operation can be represented as well through a hard constraint. This constraint can impose that the new instance becomes active (i.e., it can receive and forward packets) only after the previous one has been deactivated (i.e., it cannot receive traffic anymore).

Combinations of these elemental operations are also enough to represent more complex types of configuration changes. For example, a firewall policy migration can be modeled as the removal of the node of the N_U set representing the old configuration, and the introduction of the node representing the new one, also in a different position of the topology. Instead, a firewall policy combination can be modeled as the removal of two nodes of N_U, and the introduction of a new one.

In light of these considerations, the total number of states for the reconfiguration transient is computed as $c = c_1 + c_2$. Specifically, $c_1 = |N_U \backslash N_I|$ and $c_2 = |N_U \backslash N_T|$, where, given two sets of nodes N_x and N_y, $N_x \backslash N_y$ represents the set of nodes which are present in N_x, but not in N_y. In this definition, c_1 represents the number of times a firewall that was not present in the service is deployed and becomes active, while c_2 represents the number of times a firewall that was previously active is removed.

Once computed c, a state sequence S is created, defined as a list $S = [s_0, s_1, ..., s_{c-1}, s_c]$. Each $s \in S$ is a state of the transient, and in-between two consecutive states a single action (i.e., firewall deployment or removal) is performed. Given two states s_i and s_j in the same list, $s_i \prec s_j$ means that s_i precedes s_j in that sequence, $s_i \preceq s_j$ means that s_i precedes s_j or it is s_j itself, $s_i \succ s_j$ means that s_i follows s_j, and $s_i \succeq s_j$ means that s_i follows s_j or it is s_j itself.

Having thus formalized the concept of state sequence, the $active: N_U \times S \rightarrow \mathbb{B}$ predicate can be now introduced. This predicate is applied to a node $n \in N_U$ and a state $s \in S$, and it returns true if the node is active (i.e., already deployed and capable of receiving traffic) at the transient state identified by s. The presence of this predicate in the proposed formal model for the state sequence allows capturing the changes in the firewall configuration over the time for the optimization problem. Even if G_U includes all the firewall instances and rule sets that were present in both G_I and G_T, the *active* predicate can discriminate if in a state s a certain firewall instance or rule set is in use or not. For example, if $active(n,s) =$ false for each $s \prec s_c$, and $active(n,s') =$ true for each $s' \succeq s_c$, this means that firewall n instance was not present in the time interval before the state s', then it gets deployed in s' and from then it filters the packet it received according to its rule set. Consequently, several hard constraints of the MaxSMT problem will be imposed upon this predicate, as it represents the key element for the computation of the scheduling of the reconfiguration changes.

7.3.3 Traffic and Network Functions model

The class of packets (or traffic) model, the traffic flow model and the network functions model are the same as those presented for the application of the VEREFOO approach to packet filtering firewalls, already illustrated in Section 5.1.

A traffic $t \in T$, where T is the set of all the possible packet classes, is again modeled as a conjunction of five predicates that are defined over the IP 5-tuple packet fields, i.e., $t = (IPSrc, IPDst, pSrc, pDst, tProto)$. Instead, given the set of all the possible traffic flows F, a flow $f \in F$ represents a class of packets that are generated by a source endpoint $n_s \in N_U$, directed to a destination endpoint $n_d \in N_U$, and steered to pass through an ordered list of intermediate nodes $n_a, n_b, ... \in N_A$, i.e., $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d]$. Two auxiliary functions based on the traffic flows models and required for the definition of some hard constraints in the MaxSMT

problem are the π: $F \rightarrow (N_U)^*$ and τ: $F \times N_U \rightarrow T$, that again were already defined in Section 5.1.

The network functions model covers two main aspects: their configuration and their behavior. The configuration model varies on the basis of the different kinds of network function. As an example, the configuration of a firewall instance $n \in N_U$ is modeled as a pair (d_n, R_n), as it has been already done in Section 5.3. The network functions behavior is then modeled by means of two functions, which correspond to the forwarding behavior and the transformation behavior. The former is related to establishing which types of traffic are allowed or dropped by the network function, the latter expresses how the packets might be modified by the function. The function that models the forwarding behavior of the VNF in node $n \in N_U$ is the predicate *deny*: $N_U \times T \rightarrow \mathbb{B}$ which is true for node $n \in N_U$ and ingress traffic $t \in T$, if and only if n drops all the packets represented by t. Instead, the transformation behavior of the VNF in node $n \in N_U$ is modeled by the function *transform*: $N_U \times T \rightarrow T$, which maps an input traffic to the corresponding output traffic.

7.3.4 Network Security Policies model

Also the NSP model is similar to the one described in Section 5.1 for the VEREFOO approach. In FATO, each policy p of the P_T set (i.e., the set of all the target Network Security Policies) is formally defined as a pair $p = (C, a)$, where $C = (IPSrc, IPDst, pSrc, pDst, tProto)$ represents the policy condition, whereas a is the action that is applied on all the traffic flows satisfying the condition and is one of the two elements of the set $A = \{allow, deny\}$. When $p.a = allow$, then p is defined a reachability policy: a policy of this type is satisfied in an intermediate state of the transient if, in that state, there exists at least a flow satisfying the policy condition that reaches its destination. Instead, when $p.a = deny$, p is defined an isolation policy: in this case, the policy is satisfied in a state if all the flows satisfying its conditions cannot reach their destination.

An example P_T set is shown in Table 7.2.

The P_T set is split into two subsets, i.e., $P_T = P_T^P \cup P_T^N$. The elements of P_T^P are the persistent policies, for which the MaxSMT problem requires the formulation of hard constraints, because these policies must be satisfied in all the intermediate states of the reconfiguration transient. Instead, the element of P_T^N are all the other

Action	IPSrc	IPDst	pSrc	pDst	tProto
allow	192.168.1.∗	192.168.2.∗	∗	∗	∗
allow	192.168.2.∗	192.168.1.∗	∗	∗	∗
allow	192.168.1.∗	130.10.0.∗	∗	80	TCP
deny	192.168.1.∗	130.10.0.∗	∗	≠80	TCP
deny	192.168.1.∗	130.10.0.∗	∗	∗	UDP
deny	192.168.2.∗	130.10.0.∗	∗	∗	∗
allow	130.10.0.∗	192.168.1.∗	∗	∗	∗
allow	40.40.41.∗	130.10.0.∗	∗	110	TCP
deny	40.40.41.∗	130.10.0.∗	∗	≠110	TCP
deny	40.40.41.∗	130.10.0.∗	∗	∗	UDP

Table 7.2 Target Network Security Policies

non-persistent policies. They must be organized in a ranking and their satisfaction is not strictly required for any intermediate state; therefore, their presence is reflected to a set of soft constraints.

Finally, the *satisfied*: $P_T \times S \rightarrow \mathbb{B}$ predicate is introduced. This predicate returns true if the input policy is satisfied in the input state of the transient. In the MaxSMT problem, some constraints will be imposed upon the *active* and *satisfied* predicates, to enforce or require that they assume specific values in some conditions, while in other cases their values will be established by the solver as output.

7.4 Ranking Generation

The FATO methodology requires that a ranking is defined for all the policies of the P_T^N set, before the formulation of the optimization problem. The generation of this ranking is performed by FATO as long as the user provides the methodology with an *optimization profile*, i.e., a working mode for the ranking computation.

The computation of the ranking is then performed on the basis of this information provided by the user throughout two steps: 1) computation of a matrix, called dominance matrix, which captures the information about the relationships between policies in P_T^N (Subsection 7.4.1); 2) generation of the ranking on the basis of the dominance matrix previously computed (Subsection 7.4.2). Alternatively, the ranking might be directly defined by the user, if she has the required skills (e.g., she has total control on the network, and a high level of security expertise). In this particular case,

the usage profile and the relationship set for the policies are not required by FATO, which directly receives the ranking from the user.

7.4.1 Dominance Matrix Computation

The relationship between two policies $p, p' \in P_T^N$ can be of two different types:

- a dominance relationship, written $p \succ p'$, if p dominates p' (complimentary, $p' \prec p$ and p' is dominated by p), i.e., if p has priority higher than p' for satisfaction in the transient ;
- an independence relationship, written $p \parallel p'$, if p and p' are independent, i.e., there is not a strict imposition that one policy dominates the other.

The dominance operators $\succ$ and $\prec$ are characterized by the transitivity property. This property does not characterize, instead, the $\parallel$ operator.

The relationships among the elements of P_T^N are established depending on the working profile selected by the user for the Ranking Generation algorithm. As explained in Subsection 7.2.2, there are four available profiles: *security-max*, *service-max*, *policy-max* and *state-max*.

Security-max profile. This profile determines two types of constraints. On the one hand, each isolation policy dominates any reachability policy. This constraints is represented in (7.1), according to which, given a policy p whose action $p.a$ is *deny* and a policy p' whose action $p'.a$ is *allow*, the former must have higher priority than the latter. On the other hand, two policies of the same type are independent, unless the user specifically forces a dominance relationship between them, as it is in their faculty. This constraints is represented in (7.2), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other.

$$\forall p \in P_T^N \mid p.a = deny.\ \forall p' \in P_T^N \mid p.a = allow.\ (r \succ r') \tag{7.1}$$

$$\forall p, p' \in P_T^N \mid p.a = p'.a.\ (p \parallel p') \tag{7.2}$$

Service-max profile. This profile determines two types of constraints. On the one hand, each reachability policy dominates any isolation policy. This constraint is represented in (7.3), according to which, given a policy p whose action $p.a$ is

allow and a policy p' whose action $p'.a$ is *deny*, the former must have higher priority than the latter. On the other hand, two policies of the same type are independent, unless the user specifically forces a dominance relationship between them, as it is in their faculty. This constraints is represented in (7.4), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other.

$$\forall p \in P_T^N |\ p.a = allow.\ \forall p' \in P_T^N |\ p.a = deny.\ (p \succ p') \tag{7.3}$$

$$\forall p, p' \in P_T^N |\ p.a = p'.a.\ (p \parallel p') \tag{7.4}$$

Policy-max profile. The user has the faculty to specify some dominance relationships between pairs of policies. The policies in each pair for which a dominance relationship is not enforced are independent.

State-max profile. Each pair of policies, independently of their types, is characterized by an independence relationship. This constraints is represented in (7.5), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other. This guarantees that, when trying to enforce their satisfiability in each transient state, no policy has a higher priority than another. Therefore, the global objective of the FATO methodology, i.e., maximizing the number of states where each policy is satisfied, translates into maximizing the number of policies satisfied in each intermediate state.

$$\forall p, p' \in P_T^N.\ (p \parallel p') \tag{7.5}$$

These profiles have been identified after an analysis of the current needs in the management of reconfiguration transients. However, if in the future new needs will arise, FATO is flexible enough to be extended to support other profiles representing these new requirements. It would be enough to establish how the dominance and independence relationships are specified for the policies in P_T^N for each new profile, in a similar way as it has been explained for the four ones that are currently supported.

Then, the computation of a matrix M_D, called dominance matrix is performed. If the number of policies in P_T^N is n, and defining $N = \{0,1\}$, then $M_D \in N^{n \times n}$ summarizes the information about all the relationships between elements of P_T^N in a compact way. In particular, for each pair $p, p' \in P_T^N$, $M_D[p, p']$ expresses their relationship, and this value is computed following two simple rules:

- if $p \succ p'$, then $M_D[p, p'] = 1$;

M_D	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
p_1	0	0	0	0	1	1	1	1
p_2	0	0	0	0	1	1	1	1
p_3	0	0	0	0	1	1	1	1
p_4	0	0	0	0	1	1	1	1
p_5	0	0	0	0	0	0	0	0
p_6	0	0	0	0	0	0	0	0
p_7	0	0	0	0	0	0	0	0
p_8	0	0	0	0	0	0	0	0

(a) First matrix example

M_D	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
p_1	0	1	1	1	1	1	1	1
p_2	0	0	0	0	1	1	1	1
p_3	0	0	0	0	1	1	1	1
p_4	0	0	0	0	1	1	1	1
p_5	0	0	0	0	0	0	0	0
p_6	0	0	0	0	1	0	0	0
p_7	0	0	0	0	1	0	0	0
p_8	0	0	0	0	1	0	0	0

(b) Second matrix example

M_D	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
p_1	0	0	0	0	0	0	0	0
p_2	0	0	0	0	0	0	0	0
p_3	0	0	0	0	0	0	0	0
p_4	0	0	0	0	0	0	0	0
p_5	0	0	0	0	0	0	0	0
p_6	0	0	0	0	0	0	0	0
p_7	0	0	0	0	0	0	0	0
p_8	0	0	0	0	0	0	0	0

(c) Third matrix example

Fig. 7.5 Matrix examples

- in all the other cases, $M_D[p, p'] = 0$.

A few examples of dominance matrices are now illustrated to clarify the computation mechanism. To this end, let us consider the set of policies $P_T^N = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$, where p_1, p_2, p_3, p_4, are isolation policies, while the others are reachability policies. Three different scenarios are analyzed:

- the user decides to adopt a security-oriented profile, without specifying any dominance relationship for policies of the same type (i.e., isolation or reachability). The resulting dominance matrix is shown in Fig. 7.5a;
- the user decides to adopt a security-oriented profile, and specifies that p_1 dominates all the other isolation policies, while p_5 is dominated by all the other reachability policies. The resulting dominance matrix is shown in Fig. 7.5b;
- the user decides to adopt a complete profile. The resulting dominance matrix is shown in Fig. 7.5c.

7.4.2 Ranking Generation Algorithm

Supposing that the cardinality of P_T^N is n, the associative array *rank*, composed of n elements, associates a policy $p \in P_T^N$ to the corresponding rank. The value of each element *rank*$[p]$ is computed on the basis of the dominance matrix M_D through Algorithm 2.

Initially (line 2), all the elements of the associative array *dominated*, whose indexes are the n policies of P_T^N, are initialized to 0. The value of each *dominated*$[p]$ element identifies the number of policies that p dominates. This value is computed by summing all the elements of the M_D row corresponding to p (line 4).

Algorithm 2 computation of the ranking

Input: the set of n policies P_T^N, and the relationship matrix M_D
Output: the value of rank[p] for each $p \in P_T^N$

```
 1: for each p ∈ P_T^N do
 2:     dominated[p] ← 0
 3:     for each p' ∈ P_T^N do
 4:         dominated[p] ← dominated[p] + M_D[p, p']
 5:     end for
 6: end for
 7: l_P ← descendingOrder(P_T^N, numPrevious)
 8: rankCounter ← 1
 9: for each i = 1, 2, ..., n do
10:     if i ≠ n ∧ dominated[l_P[i]] > dominated[l_P[i+1]] ∧
11:       l_P[i] ≻ l_P[i+1] then
12:         rankCounter ← rankCounter + 1
13:     end if
14:     rank[p] ← rankCounter
15: end for
16: return rank
```

Next (line 7), the policies are sorted in descending order of the number of policies each one dominates. As such, the *descendingOrder* function works on the P_T^N set and the *dominated* array, so as to compute another array, identified by l_P. If two or more policies dominates the same number of policies, their relative ordering is indifferent. This array can be accessed by means of integer indexes from 1 to n, and in particular $l_P[i]$ returns the policy in i-th position in the computed descending ordering.

The final operation is the computation of the value of *rank*[p] for each $p \in P_T^N$. To this end, after initializing the auxiliary variable *rankCounter* to 1 (line 8), the policies are analyzed one by one, according to the previously computed ordering. At the i-th iteration step, the *rankCounter* variable is incremented by one unit, if the following conditions are satisfied (lines 10-11): (1) the policy returned by *dominated*$[l_P[i]]$ dominates a larger number of policies than the policy *dominated*$[l_P[i+1]]$ (because they might have the same value of dominated policies, and in that case the i+th policy antecedes the (i+1)-th one in this ordering is casual); and (2) the i-th policy dominates the (i+1)-th one (because it might happen that they are independent, so there is not relative priority between them).

After the decision on increasing the *rankCounter* variable, its updated value is assigned to $rank[p]$ (line 14). After this operation is repeated for all the policies, the value of $rank[p]$ for each $p \in P_T^N$ has been computed. These values will come in handy for the formalization of the soft constraints of the MaxSMT problem.

7.5 MaxSMT Problem Formulation

The reconfiguration transient problem is formulated as a MaxSMT problem. In this section, the derivation of the hard and soft constraints of this formulation are illustrated.

7.5.1 Hard constraints on boundary states

The presence of a node in the N_I or N_T sets (i.e., respectively in the G_I or in the G_T graphs) determines some boundary conditions on the initial and final states of the reconfiguration transient. These conditions are formalized as three classes of hard constraints, represented by equations (7.6), (7.7) and (7.8). In these formulas, the outcome of the *active* predicate is constrained when applied to the initial state s_0 and the final state s_c, because the conditions of each node in those states are already known from the inputs of the approach. In light of these considerations, the three classes of hard constraints are formulated as:

- A node $n \in N_U$ that is present in N_I but not in N_T is active in s_0, while not active in s_c.

$$\forall n \in N_U \setminus N_T . (active(n, s_0) \wedge \neg active(n, s_c)) \tag{7.6}$$

- A node $n \in N_U$ that is present in N_T but not in N_I is not active in s_0, while active in s_c.

$$\forall n \in N_U \setminus N_I . (\neg active(n, s_0) \wedge active(n, s_c)) \tag{7.7}$$

- A node $n \in N_U$ that is present in both N_I and N_T is active in both s_0 and s_c.

$$\forall n \in N_I \cap N_T . (active(n, s_0) \wedge active(n, s_c)) \tag{7.8}$$

7.5.2 Hard constraints on intermediate states

For the majority of the pairs composed of a node $n \in N_U$ and a state $s \in S$, the outcome of the *active* predicate is established by the MaxSMT solver when the solution is computed. However, there are some cases where this outcome can be determined in advance, and they can be represented as hard constraints. For their formulation the introduction of the $bti\colon \mathbb{B} \to \{0,1\}$ ("bti" stands for "bool_to_int") function is required. This function returns 0 if the Boolean input is false, 1 otherwise.

In greater detail, three classes of hard clauses are defined for the conditions on the intermediate states of the transient, and they are represented by equations (7.9), (7.10) and (7.11).

- For each node n that is active in both s_0 and s_c, no change of state is required in the transient. Therefore, for each state s that composes the transient, the outcome of the *active* predicate is forced to be true when it is applied to node n and state s.

$$\begin{gathered}\forall\, n \in N_U.\,((active(n,s_0) \wedge active(n,s_c)) \implies \\ (\forall s \in S.\,(active(n,s))))\end{gathered} \tag{7.9}$$

- For each node such that its initial and final states are different, then only one change of state is required in the transient. Therefore, the *active* predicate, when applied to that node n, changes value from a state s_i to the following state s_{i+1} only once. This constraint is enforced by imposing that the module of the difference between $active(n,s_{i+1})$ and $active(n,s_i)$ is equal to 1 only once, i.e., that the sum of all the differences for each pair of consecutive states is still equal to 1.

$$\begin{gathered}\forall\, n \in N_U.\,((active(n,s_0) \neq active(n,s_c)) \implies \\ \left(\sum_{s_i \in (S \setminus s_c)} (|bti(active(n,s_{i+1})) - bti(active(n,s_i))|) = 1\right)\end{gathered} \tag{7.10}$$

- For each intermediate state, only for one node the state is different from the previous one, because a single operation is performed in-between two consecutive states. Therefore, it is imposed that, given two consecutive states s_i and s_{i+1}, it occurs only for a node n that the outcome of $active(n,s_i)$ id different from the one of $active(n,s_{i+1}$.

$$\forall\, s_i, s_{i+1} \in S.\,\exists! n \in N_U.\,(active(n,s_i) \neq active(n,s_{i+1})) \tag{7.11}$$

Additionally, the case where two nodes $n, n' \in N_U$ share the same AP and their update is allowed without the need of instantiating a new software process (i.e., as explained in Subsection 7.3.1, $configUpdate(n, n') = true$) is formulated with hard constraints as well. This translates into the hard constraint shown in (7.12), stating that, when the node n is not active anymore in a state s_i, then in the next state s_{i+1}, the node that has become active must be n'. Thus, even though in the modelization, the nodes are distinct, the result is the same as if they were a single one, and only the configuration was changed.

$$\begin{aligned} &\forall\, n, n' \in N_U .\, (configUpdate(n, n') \wedge active(n, s_0) \wedge \\ &\neg active(n', s_0) \implies (\exists i.\, \neg active(n, s_i) \implies active(n', s_{i+1}))) \end{aligned} \tag{7.12}$$

7.5.3 Hard constraints on the forwarding behavior

The forwarding behavior of the network functions (i.e., the decision if a middlebox must drop the input traffic or forward it) is formalized with some hard constraints impacting on the outcome of the *deny* predicate. The truth or falseness of this predicate, in turn, impacts the satisfiability of the hard constraints related to the security policies, as it will be described in Subsection 7.5.4. Each network function type is characterized by different hard constraints, in accordance with the approach that has been already described for modeling the network functions in Section 5.1. Examples of hard constraints expressing the forwarding behavior of two function types (i.e., simple functions such as normal forwarders, and packet filtering firewalls) have been already reported in that section, and they are also valid for the formulation of the MaxSMT problem in the FATO approach.

7.5.4 Hard constraints on the security policies

A first consideration related to the satisfaction of the security policies is that all the policies in the P_T set must be satisfied in the final state of the transient (i.e., in s_c). Therefore, this statement translates in hard constraints that impose the truth of the *satisfied* predicate, when applied for each policy on state s_c.

$$\forall\, p \in P_T .\, (satisfied(p, s_c)) \tag{7.13}$$

For all the intermediate states of the transient, a set of hard constraints must be introduced to map the outcome of the *satisfied* predicate to the forwarding behavior

of the functions that are present in the paths crossed by the flows satisfying the conditions of each policy. The hard constraints are different depending on the policy type (i.e., reachability or isolation). The formalization of the hard constraints for a reachability policy $p \in P_T$ is shown in (7.14). In a state s the policy p is satisfied if there exists at least a flow f satisfying $p.C$ such that all the nodes of the paths crossed by f are active in that state and do not block the incoming traffic of f.

$$\begin{aligned} \forall s \in S.(satisfied(p,s) \iff (\exists f \in F_p. \\ (\forall n \in \pi(f).(active(n,s) \wedge \neg deny(n,\tau(f,n)))))) \end{aligned} \tag{7.14}$$

Instead, the formalization of the hard constraints for an isolation policy $p \in P_T$ is shown in (7.15). In a state s the policy p is satisfied if for each flow f satisfying $p.C$ there exists at least a node of the path of f that is not active in that state (i.e., it cannot receive traffic), or it is active and blocks the incoming traffic of f.

$$\begin{aligned} \forall S \in S.(satisfied(p,s) \iff (\forall f \in F_p. \\ (\exists n \in \pi(f).(\neg active(n,s) \vee (active(n,s) \wedge deny(n,\tau(f,n))))))) \end{aligned} \tag{7.15}$$

Finally, if $p \in P_T^P$, i.e., it is a persistent policy, then it must be satisfied in each intermediate state. Therefore, the class of constraints shown in (7.16) imposes that the *satisfied* predicate must be true when applied to a persistent policy p and to any state, because that policy must be satisfied in any intermediate state of the transient.

$$\forall\, p \in P_T^P.\, (\forall s \in S.(satisfied(p,s))) \tag{7.16}$$

7.5.5 Soft constraints

The optimization objective of the MaxSMT problem is to maximize the number of transient states where each policy $p \in P_T^N$ is satisfied, while considering their relative priority expressed throughout the ranking computed as illustrated in Section 7.4. The persistent policies are excluded from this objective, because they are already managed as shown in (7.16). The achievement of this objective requires the formalization of some weighted soft constraints, so that the MaxSMT solver gives priority in the satisfaction of the clauses with the highest weight, trying to maximize the sum of the weights assigned to the satisfied soft clauses.

Each soft constraint is related to the application of the *satisfied* predicate to a pair composed of a policy p and a state s. Therefore, if the number of states is c (excluding the final state), the total number of soft constraints is $c \cdot |P_T^N|$. The

Algorithm 3 computation of the weights for the soft constraints of the MaxSMT problem

Input: the set of policies P_T^N, the associative array *rank*, the number of ranks m, and the number of transient states c
Output: the value of weight[p] for each $p \in P_T^N$

```
1:  weightSum ← 0, weightValue ← 1
2:  for each i = m, m − 1, ..., 2, 1 do
3:      for each p ∈ P_T^N do
4:          if rank[p] = i then
5:              weight[p] ← weightValue
6:              weightSum ← weightSum + (weightValue · c)
7:          end if
8:      end for
9:      weightValue ← weightSum + 1
10: end for
11: return weight
```

computation of their weights is described in Algorithm 3, and these weights are returned by means of the associative array *weight*, indexed by the elements of P_T^N. Supposing that the number of different ranks is m, the algorithm starts to assign the weights to policies having the lowest rank, i.e., the m-th rank (line 2). The weight that is assigned to all the policies within the same rank must be the same as well. Initially, the weight assigned to the policies, identified by the auxiliary variable *weightValue*, is set to a conventional number, which is 1 for simplicity (line 1). Whenever an element of the *weight* array is assigned with the proper value (line 5), a counter (*weightSum*) is incremented by the product of the current weight value (*weightValue*) and the number of states c (line 6). The reason is that this same weight will be used for that policy for c soft constraints, for each transient state. Then, when the algorithms iterates to a higher rank, the value of the variable of *weightValue* is incremented by the sum of all the previous weights (*weightSum*) plus 1 (line 9), so that the soft constraints will be characterized by a weight that is higher than the sum of all the weights assigned to soft clauses for policies of lower rank.

The formulation of the soft constraints is represented in (7.17). In this representation, the *Soft*(c, w) notation identifies a soft clause, expressing the constraint c and having weight w. Each soft constraint simply assigns *weight*[p], computed as previously explained, to the *satisfied* predicate, applied to the specific policy p and

to any transient state s.

$$\forall p \in P_T^N . \forall s \in S \backslash \{s_c\} . (Soft(satisfied(p,s), weight[p])) \tag{7.17}$$

7.5.6 Solution Computation

After the MaxSMT problem is built by combining the hard and soft constraints illustrated beforehand, a MaxSMT solver is employed to compute the optimal and correct solution. In case the problem is not satisfiable (e.g., a persistent policy cannot be satisfied in all the intermediate states of the transient), the solver cannot reach any correct solution for the problem. Instead, it assigns the most appropriate Boolean values for the *active* and *satisfied* predicates so as to achieve most of the optimization objectives. From the results of the *active* predicate, in particular, it is possible to identify the order in which the nodes are introduced or removed in G_T with respect to G_I, i.e., the optimal scheduling of the operations. In fact, when a node becomes active in a state after not being active in the previous state, it means that in-between these two states the traffic flows have been redirected to this node, which has been successfully deployed in the virtual network.

7.6 Implementation and validation

The FATO methodology has been implemented as a Java-based framework, employing a state-of-the-art theorem prover called z3 for the formulation and resolution of the MaxSMT problem. This framework offers REST APIs for the interaction with other tools, e.g., a tool for the automatic computation of firewall configurations in virtualized networks [10]. Through this RESTful interface, the framework can also interact with NFV MANO (Management and orchestration) tools, such as Open Source MANO. More specifically, the framework can retrieve information about the virtual network from the MANO. Then, after running the FATO methodology, it provides the MANO with information about the order the different operations composing the distributed firewall reconfiguration must be executed in the network.

The framework has been validated in different ways: i) the correctness and optimization of the approach have been proved with some realistic use cases, based on computer networks having varying topologies (Subsection 7.6.1); ii) the computation

time and memory usage of the approach has been evaluated with an extensive series of scalability tests (Subsection 7.6.2). All the tests have been carried out on an 8-core Intel Core i7-10700E CPU @ 2.90GHz workstation with 32 GB RAM. For these tests, the adopted z3 version is 4.8.5.

7.6.1 Correctness and optimization verification

The correctness and optimization of the approach have been checked with some use cases, based on different networks having topological structures that vary from simple to complex. Four networks topologies have been considered: i) the network of our university department; ii) a three-tier data center network; iii) the GÉANT[1] topology; iv) the Internet2[2] topology.

About optimization, all the possible solutions in solving the transient reconfiguration problem have been enumerated for the networks. Then, the framework has been run and it has been checked that the output produced by the MaxSMT solver corresponds to the optimal solution, i.e., it minimizes the number of intermediate transient states where policies are violated. About correctness, the intermediate states of the transient derived from the solution computed by the framework have been simulated by using GNS3, a software that allows real-time network simulation for pre-deployment testing without the need for network hardware[3]. Under those conditions, some communications have been established in the network to check if the specified security policies were violated or not in the transient states.

Here the way the effectiveness validation has been performed is described for a fairly complex network topology, i.e., the three-tier data center network depicted in Fig. 7.6. It is possible to suppose that the human user, i.e., a network administrator, has specified four security policies and has assigned them with the rank shown in Table 7.3. The table also reports the firewall configuration changes required for the full enforcement of the corresponding policy.

The FATO methodology establishes that the following order of the configuration changes must be scheduled: $f_{19} \rightarrow f_{33}$, $f_{21} \rightarrow f_{34}$, $f_{26} \rightarrow f_{36}$, $f_{30} \rightarrow f_{37}$, and $f_{24} \rightarrow f_{35}$. In the following, an explanation is provided about how the intermediate states

[1] https://geant3plus.archive.geant.net/
[2] https://www.internet2.edu/
[3] https://www.gns3.com/

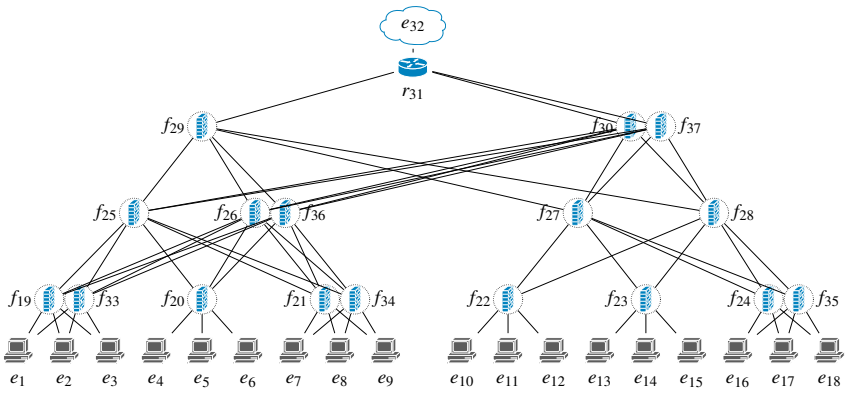


Fig. 7.6 Topology of the three-layer data center network

#	Action	Source	Destination	Required changes
1	Deny	e_{32}	e_1	$f_{19} \rightarrow f_{33}$
2	Deny	$e_{4,5,6}$	$e_{4,5,6}$	$f_{21} \rightarrow f_{34}$
3	Allow	e_1	e_{10}	$f_{19} \rightarrow f_{33}$, $f_{26} \rightarrow f_{36}$, $f_{30} \rightarrow f_{37}$
4	Allow	e_{16}	e_{18}	$f_{24} \rightarrow f_{35}$

Table 7.3 Network Security Policies

of the transient of the computed solution still maintaining security and service availability according to the requirements deriving from specified ranking.

State after $f_{19} \rightarrow f_{33}$: the network administrator required that e_1 and e_{32} must be isolated as soon as possible, with a higher priority than the isolation policy among hosts of the subnetworks e_4, e_5 and e_6. An example reason may be that an external attacker, represented by e_{32}, has found out a breach in e_1 and is currently exploiting it for privilege escalation. The consequences would be dramatic for the whole data center. So f_{19} is immediately replaced by f_{33}, which can block packets coming from e_{32}. Unfortunately, the policy inquiring isolation among e_4, e_5 and e_6 is not satisfied yet. The required security is not fully maintained in this transient state, but it shows how the most important security policy is prioritized.

State after $f_{21} \rightarrow f_{34}$: the network administrator required that hosts of the subnetworks e_4, e_5 and e_6 must be isolated one from each other, e.g., because Dockers belonging to different companies have been launched and they cannot interact due to privacy. As the security level is thought inferior than the previous policy, f_{21} is replaced by f_{34} after the $f_{19} \rightarrow f_{33}$ configuration change. At this point, the intermediate state fully maintains the desired level of security, as all traffic flows that must be blocked cannot reach their destinations. However, service availability is not

Topology	# vertices	# directed links	# firewalls	Time (10 states)	Time (20 states)
University department	14	28	6	0.64 s	7.73 s
Three-tier data center	37	140	17	1.47 s	12.88 s
GÉANT	49	86	35	3.83 s	15.10 s
Internet2	53	80	40	6.61 s	23.91 s

Table 7.4 Computation times for the four network topologies

complete, because both the reachability policies (between e_1 and e_{10}, between e_{16} and e_{18}) are not satisfied.

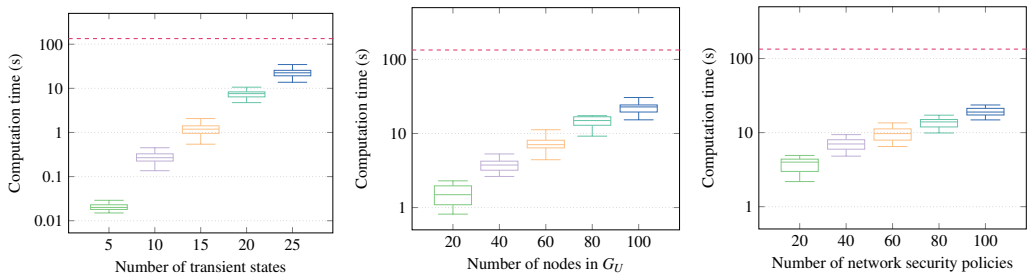
State after $f_{26} \rightarrow f_{36}$: the reachability policy between between e_1 and e_{10} must be prioritized than the policy between e_{16} and e_{18}. Consequently, more transient states will occur before the result is achieved, as multiple configuration changes must be scheduled. The $f_{19} \rightarrow f_{33}$ was already scheduled at the beginning of the solution. Now $f_{26} \rightarrow f_{36}$ is scheduled, but is still not sufficient. Therefore, this transient state does not change anything with respect to the previous, in terms of policy satisfaction.

State after $f_{30} \rightarrow f_{37}$: the reachability policy between between e_1 and e_{10} is fulfilled only after f_{30} is replaced by f_{37}. In this intermediate state, service availability improves, and the transient is almost concluded. Unfortunately, the reachability policy between e_{16} and e_{18} is not satisfied yet, even though it only requires a single change. The reason is that its rank was the lowest, so minimum priority was assigned to it.

State after $f_{24} \rightarrow f_{35}$: this last configuration change concludes the transient, and the final state achieves both maximum security and service availability.

Even though only 5 configuration changes had to be scheduled in this exemplifying use case, 120 different solutions could be produced. The scheduling computed by FATO is optimal, as the transient states maintain security and service availability compatibly with the ranking. For example, might have been scheduled before $f_{26} \rightarrow f_{36}$ and $f_{30} \rightarrow f_{37}$ to satisfy the fourth policy immediately. However, the enforcement of the third policy would have been postponed, and this was not an optimal solution as it had a higher priority.

Table 7.4 reports the time that was required for computing the optimal scheduling for the four different topologies: 1) the topology of our university department network, illustrated in Fig. 7.1; 2) the topology of the three-tier data center network, illustrated in Fig. 7.6; 3) a network topology inspired by the production network



(a) Time scalability versus number of transient states

(b) Time scalability versus number of network nodes

(c) Time scalability versus number of security policies

Fig. 7.7 Time scalability

GÉANT[4]; 4) a network topology inspired by the production network Internet2[5]. This table also reports information about the size of these networks, in terms of number of vertices, number of directed links and number of firewalls. The experiments were carried out under two different transient lengths, respectively 10 and 20 states. In both cases, the FATO methodology succeeded in computing the solution in much less time than the typical length of a reconfiguration transient, as already discussed in Section 7.1. Additionally, Additionally, FATO is executed before starting the transient, in order not to lengthen its duration. Besides, FATO can be used each time a reconfiguration transient has to be started (e.g., after the identification of an attack, or after a policy is modified by the human user). This consideration also holds for more complex and varied topologies (e.g., GÉANT and Internet2), where the number of filtering functions is much higher and the structure of their allocation scheme is not trivial.

7.6.2 Scalability evaluation

Scalability has been evaluated in terms of computation time and memory usage by means of a series of tests, with the aim to show the feasibility of the approach considering the requirements of virtualized networks. The four metrics that have been considered for scalability evaluation are: i) the number of transient states; ii) the number of nodes composing the network topology; iii) the number of network security policies that determined the reconfiguration transient; iv) the number of filtering rules in each instance of the distributed firewall.

[4]Link: https://geant3plus.archive.geant.net/. Last accessed: October 18th, 2022.
[5]Link: https://www.internet2.edu/. Last accessed: October 18th, 2022.

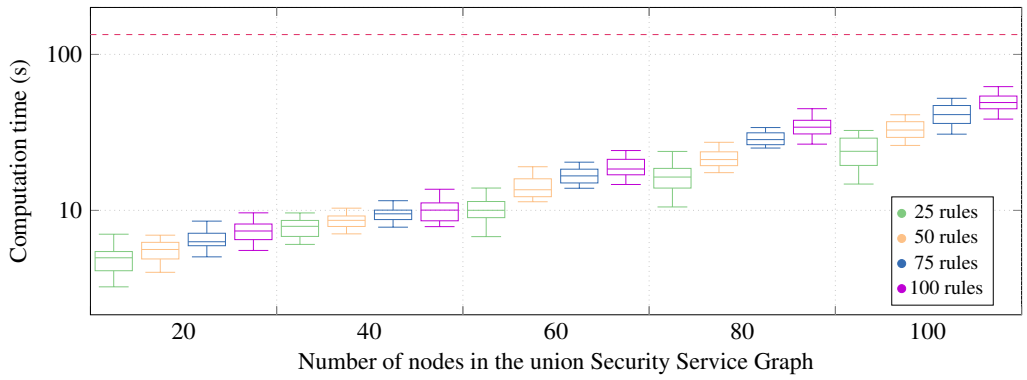


Fig. 7.8 Time scalability versus number of rules in each firewall instance

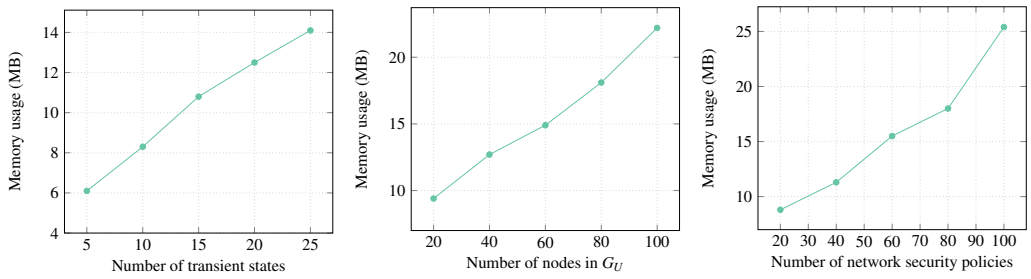
Fig. 7.7 and Fig. 7.8 report the results of time scalability tests. First, Fig. 7.7a analyzes the performance of the implementation when the number of transient states (i.e., the number of changes that occurred in the transient from G_I to G_T, as formalized in Subsection 7.3.2) progressively increases. For those tests, each scenario characterized by a certain number of transient states is based on a topology of corresponding size (e.g., when the number of states is 20, in the resulting G_U the number of firewalls subject to configuration changes is 20). The enforcement of a congruent number of network security policies is requested as well. Second, Fig. 7.7b analyzes how the framework behaves for increasing sizes of the network on which it is applied (i.e., the G_U), while keeping the number of transient states fixed to 20 and the number of policies fixed to 50. Third, Fig. 7.7c evaluates scalability versus the number of network security policies that should be enforced in the reconfiguration transient, while keeping the number of transient states fixed to 20 and the network size fixed to 50 nodes. Fourth, Fig. 7.8 depicts the experimental results for the scalability tests related to the number of filtering rules (from 25 to 100) in each firewall instance. The percentage of firewall instances is 2/3 for testing the scalability versus firewall rules, so that the framework is evaluated in situations that might be stressful for its application. The number of intermediate states of the transients is again fixed to 20.

For all these time scalability tests, the topology is an extension synthetically derived from the network represented in Fig. 7.4, whereas the policies are similar to the examples shown in Table 7.2. Besides, for each scenario, the corresponding whisker plot shows minimum, 5th percentile, median, 95th percentile and maximum values, computed over the results of 500 iterations. Many iterations have been

performed because the computation time of z3 for the resolution of optimization problems involving the integer theory may differ depending on the effective integers employed in the definition of the IP addresses, as already discussed in Section 5.4. The previously mentioned figures are semi-logarithmic plots, with a logarithmic scale for the Y-axis ("Computation time") and a linear scale for the X-axis. Through this representation, whisker plots depicted for low numbers of transient states can be better visualized, while they would be otherwise stretched to the bottom part of the plot.

As it has been discussed in the problem statement, the number of transient states is equal to the number of the distributed firewall configuration changes. In virtualized networks, the best practice is to deploy a new instance with the required configuration instead of the previous one. Therefore, the number of states is also directly proportional to the number of firewall instances that are subject to the reconfiguration. In light of these considerations, the numbers of transient states and security function instances characterizing the scenarios under which the FATO approach has been tested are in line with the experimental tests carried out for the validation of related approaches ([143][144][171]). In those studies, the maximum number of nodes composing the distributed function architecture of SDN is 20-30. Considering that not all of the instances are usually subject to changes during a reconfiguration, then this shows that the transient length considered as reference is also the most recurrent one.

From the results shown in Fig. 7.7a, the computation time of the MaxSMT solver progressively increases with a quadratic behavior when the number of states of the transients gets bigger. On the one hand, the scalability is worse when the number of states is bigger than 20. On the other hand, the framework is particularly fast for transients composed of at most 20 states: all the MaxSMT instances representing the evaluated scenario are successfully solved in less than 10 seconds. Considering that related approaches work on a similar number of transient states as mentioned before, this is a significant result for validating the methodology illustrated in this chapter. It can also be noted that in [143] some problem instances with more than 20 functions could not be solved in 600 seconds, while in [144] the time required to find a first feasible solution for the optimization problem is in the same magnitude order as the results of these validation tests.



(a) Memory scalability versus number of transient states (b) Memory scalability versus number of network nodes (c) Memory scalability versus number of security policies

Fig. 7.9 Memory scalability

Similar considerations apply to the scalability versus the number of nodes in G_U and the number of network security policies. The results plotted in Fig. 7.7b and Fig. 7.7c respectively show that the proposed approach can successfully manage fairly big networks while checking the satisfaction of a large set of policies. Scalability with respect to network size and policy set cardinality is even better than scalability with respect to the number of transient states. This is due to the fact that the increment of soft constraints in the formulation of the MaxSMT problem is lower. Additionally, Fig. 7.8 shows that, even if the number of rules in each firewall instance increases, the difference among the resulting computation times is not significantly large. As such, the methodology is also suitable for managing firewalls with a high number of rules.

Time scalability is also in line with the times that state-of-the-art approaches require for the security management of virtualized networks. The common workflow to enforce security in a virtualized networks is composed of multiple steps [147]: i) establishing the configuration of the security functions; ii) determining the embedding scheme of their virtual implementations in the physical infrastructure; iii) the instantiation of the virtual security service; iv) the effective deployment of the virtual functions. The same process should be also followed in case of a reconfiguration, as such the one that determines the transients studied in this chapter. By looking at the related literature, the computation times that are commonly needed to perform those tasks are bigger than the ones that the FATO approach can reach for the management of the reconfiguration transient. [11] reports that configuring a security service with a distributed firewall composed of 100 instances requires around 3 minutes. [172] underlines that establishing the embedding scheme of 10 virtual functions on a physical network composed of 50 nodes varies according to the adopted methodology, from

1400s for the resolution of the exact problem to around 100ms when the heuristic that is proposed there ix executed. [173] experimentally checked that the instantation time of a network security service takes more than 100 seconds, when the service is composed of around 30 virtual functions, for the Virtual Infrastructure Managers of two-well known orchestrators: Open Source MANO and Openstack. [168] states that DPD time related to the deployment of a single virtual function is 134s. If these numbers are combined, it is clear that the time introduced by the FATO framework does not represent a delay, as the scheduling of the reconfiguration changes may be easily computed by FATO while another step, such as the service instantation, is performed. This is even more evident when the DPD time is compared to the time scalability of the FATO approach. In transients composed of 20 states, more than a single virtual function must be deployed and instantiated, but just deploying one takes more time than running the FATO methodology. Therefore, the value of the DPD time has been represented for reference as a red dotted horizontal line in Fig. 7.7 and Fig. 7.8. These comparisons with state-of-the-art approaches for the security management of virtualized networks may not seem fair, as they were made with older approaches. Nevertheless, they are the only comparisons that could be done, as there are not newer approaches in literature with the same characteristics as the FATO methodology.

Finally, peak memory scalability has been evaluated by running the framework under the same conditions of Fig. 7.7. This analysis was required, because memory may be a critical parameter for the solver, which is highly memory-demanding. The results are reported in Fig. 7.9, where whisker plots are not used because memory usage is not influenced by the different IP addresses used in the MaxSMT formulation, differently from the computation time. Two considerations can be derived from these plots. On the one hand, memory usage increases linearly with respect to all the analyzed metrics. On the one hand, even the worst case that was identified (i.e., when 100 network security policies are formalized in the MaxSMT problem) is inferior to 26 MB. Therefore, all these results shows that the implementation of the FATO methodology can work without any worrisome limitation due to memory.

Chapter 8

A Functionality Model for Security Orchestration

This chapter proposes a new abstraction of virtual network security functions, i.e., the projection abstraction, which can be used to optimize the workflow of network orchestration. This abstraction captures only the security functionalities that security functions can execute, but represented independently from the differences that are only related to the vendor-dependent implementation choices.

8.1 Problem Statement

The introduction of softwarization paradigms in networking determined high flexibility for choosing the security functions that should be employed to enforce the requested security protection in a computer network. For the enforcement of the user-specified *Network Security Policies* (NSPs), security providers have access to a large pool of alternative *Virtual Network Functions* (VNFs), programs which can run on general Virtual Machines or Dockers without requiring special-purpose embedding hardware, and which may offer multiple functionalities (e.g., firewalling, VPN generation).

However, this freedom of choice has a drawback. In the approach that is commonly pursued for enforcing security in a virtual network [147], security providers select the VNFs required to enforce the NSPs before the two next stages of security

orchestration, i.e., the security enforcement in the virtual service, through the definition of allocation scheme and configuration of the selected vNSFs in the topology (this step can be automatically managed by VEREFOO, as explained in Chapter 4), and their deployment in the physical infrastructure of the network. Due to this ordering of the operations, network information (e.g., the virtual service topology, and the presence, in this topology, of network functions like load balancers and network address translators) is overlooked in the selection of the VNFs. Neglecting this information may result into sub-optimizations impacting the stages that follow VNF selection, as discussed below.

Two main sub-optimizations that may derive from selecting VNFs without taking network information into due account are those concerning deployment costs and energy efficiency. For example, if there is a requirement to block two different types of traffic (e.g., the web traffic to some domain and the mail traffic to some other domain), the administrator may end up with choosing a distinct VNF to satisfy each requirement (e.g., a web application firewall for the first type and a packet filter for the second one). However, it is possible that the network topology is such that a single firewall, capable of performing both web application filtering and packet filtering, is enough to block both traffic flows, because there is an allocation place that is on the paths of both flows. If this is the case, deploying distinct VNFs for the two security functionalities means deploying more virtual functions than strictly required, which entails consuming more server resources (e.g., memory and CPU) and more energy for their operation. Even though virtualized networks have higher flexibility and higher operational efficiency than physical networks, deployment and energy optimizations are still open problems [174].

In light of these considerations, the idea developed in this thesis to address this problem is to postpone the VNF selection phase, making the selection when the network service topology has been completely defined, and more network information can be exploited to perform the optimum selection. In order to make this possible, a new abstraction level for the VNFs, named projection abstraction, is proposed. In each VNF that could be potentially selected, a set of functionalities (i.e., function features that can enforce corresponding security properties) is selected, by projecting all the VNF functionalities onto the NSPs that must be enforced in the network. Thanks to this projection concept, the actual VNF selection may be postponed to be performed jointly with VNF deployment, after the security configuration step, which can be performed by allocating and configuring projections instead of real VNFs.

Symbol	Definition
$\mathbb{B} = \{\text{true, false}\}$	Boolean set
v	VNF
$M_v = (F_v, A_v)$	manifest of the VNF v
$F_v = (F_v^+, F_v^*)$	configuration features and fields of the VNF manifest
F_v^+	features for which the VNF can take a decision and which it can configure
$F_v^*)$	features for which a VNF can take a decision, but without configuring them
A_v	actions of the VNF manifest
$n = (C_n, S_n)$	NSP
$C_n = \{c_1, c_2, ..., c_m\}$	condition set of the NSP n
$S_n = \{s_1, s_2, ..., s_l\}$	action set of the NSP n
$S_n = [s_1, s_2, ..., s_l]$	action list of the NSP n
$s = (a_s, B_s)$	single action of the NSP n
a_s	operations of the action s
B_s	enforcement modes of the action s
$p = (C_p, S_p)$	projection
C_p	conditions of the projection p
S_p	actions of the projection p

Table 8.1 Notation

Only later, depending on the allocated projections and their computed configuration, the actual VNFs are selected and deployed, in a final single and optimized step. In the vision of this proposal, this approach is fully automated, so that all the stages of the process (i.e., projection allocation and configuration, VNF selection and placement) work on inputs provided by service providers but without requiring additional human intervention.

8.2 The Projection Abstraction

This section presents the formal models of the VNFs that can be deployed in the network, the NSPs that have been requested to enforce security protection, and the projection abstraction. TABLE 8.1 includes the main formal notations used for the definition of these models.

8.2.1 VNF Model

Each VNF is characterized by the information related to its security behavior, which is represented by the actions and the configuration fields identifying the traffic on which the actions are applied. Examples of configuration fields are the conditions

expressing the layer of the ISO/OSI stack where the VNF can work (e.g., conditions on the source and destination IP addresses, or on the web domains), and the algorithms it can execute (e.g., AES-256-CBC for encryption). In the proposed model, this information is grouped in a single representation, named *VNF manifest*.

For a VNF v, the corresponding manifest M_v is composed of two sets, i.e., $M_v = (F_v, A_v)$:

- F_v is the set of all the configuration features and fields that can be used by a VNF to take a decision or to enforce an action. This set includes packet fields (e.g., the five fields of the IP 5-tuple, web-application fields as domain or url) and other configuration elements that determine the VNF behavior (e.g., the encrypting algorithm and the key length for VPN gateways);
- A_v is the set of all the actions that can be applied by the VNF.

In greater detail, the field set F_v is composed of two subsets, i.e., $F_v = (F_v^+, F_v^*)$. This distinction allows discriminating the fields which a VNF can configure on itself from those for which it can only take decisions:

- F_v^+ is the set of all the features for which the VNF can take a decision and which it can configure (e.g., for a packet filtering firewall such as iptables, all the fields of the IP 5-tuple);
- F_v^* is the set of all the features for which a VNF can take a decision, but without configuring them, i.e., by configuring other fields which may allow reaching the same security property (e.g., if a specific web domain must be blocked, iptables might be used, however it cannot configure a "domain" field, but only a corresponding IP address).

The following four VNF manifests may help to explain this modelization. In these manifests, for the sake of conciseness, only a subset of all the fields that may be present present in the F_v^+ and F_v^* sets are shown.

$$\begin{array}{|l} \text{VNF } v_1\text{: iptables} \\ F_{v_1}^+ = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}\} \\ F_{v_1}^* = \{\text{domain}, \text{url}, \text{mailAddress}, \text{payload}, ...\} \\ A_{v_1} = \{\text{allow}, \text{deny}\} \end{array} \tag{8.1}$$

VNF v_2: Squid
$F^+_{v_2} = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}, \text{domain}, \text{url}, ...\}$
$F^*_{v_2} = \{\text{mailAddress}, \text{payload}, ...\}$
$A_{v_2} = \{\text{allow}, \text{deny}, \text{log}\}$ (8.2)

VNF v_3: MyLogger
$F^+_{v_3} = \{\text{domain}, \text{url}\}$
$F^*_{v_3} = \{\text{mailAddress}, \text{payload}, ...\}$
$A_{v_3} = \{\text{allow}, \text{log}, \text{alert}\}$ (8.3)

VNF $v_3$4: strongSwan
$F^+_{v_1} = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto},$
$\text{encrytion\_algorithms} = \{\text{AES-CBC}, \text{AES-GCM}, ...\},$
$\text{encrytion\_key\_length} = \{256 \text{ bits for AES}, 512 \text{ bits for AES}, ...\}\}$
$F^*_{v_4} = \{\text{mailAddress}, \text{payload}, ...\}$
$A_{v_4} = \{\text{allow}, \text{encrypt}, \text{decrypt}, \textit{compute a MAC}, ...\}$ (8.4)

The first example, shown in (8.1), is the manifest of a packet filtering VNF such as iptables, ipfirewall or equivalent firewall implementations. Such VNF can only work at layers 3 and 4 of the ISO/OSI stack, not at higher layers (e.g., the application layer). Therefore, it can decide if a received packet is allowed to be forwarded to the next hop or not depending on the values of the five fields composing the IP 5-tuple. However, a packet filtering firewall can make decisions for packets having fields such as web domain and url. For example, it may block packets directed to a certain web domain, filtering them according to their destination IP address and port.

The second example, shown in (8.2), is the manifest of a web application firewalling VNF such as Squid. With respect to a packet filter, this type of firewall can also configure rules based on web domains, urls, HTTP methods (e.g., POST, GET), etc. The $F^*_{v_2}$ set includes all the other fields which were present in $F^*_{v_1}$, since Squid is still a firewall, just working at higher layers of the ISO/OSI stack. For the same reason, also such VNF does not have any parameter related to encryption (e.g., encryption algorithm, encryption key length) in its manifest. Besides, Squid also supports the operation of logging the received traffic.

The third example, shown in (8.3), is the manifest of a logging VNF. The advent of virtualization in networking allows software developers to write their own

implementation of security functionalities, and run them through Virtual Machines or containers. The proposed concept and model of VPN manifests is general and flexible enough to support any kind of VNF, not only the most common ones. In the case of this particular logging VNF, which the developer has named 'MyLogger", the VNF cannot filter any packet, but it can only log the reception of specific kinds of packets and notify the human network administrator about that event. Additionally, it has been developed in such a way that the only fields that are present in the configuration rules are web domain and url. Therefore, the fields of the IP 5-tuple itself are absent from the F^{+}_{v3} set. They are not in the F^{*}_{v3} set either, because domain and url are related to ISO/OSI layers higher than 3 and 4.

The fourth example, shown in (8.4), is the manifest of a VPN gateway such as strongSwan. With respect to the previous examples, this manifest also provides information about the encryption algorithms, with the relative key lengths, that are supported by the VNF to provide confidentiality. Similar features for the ingenuity and authentication properties are included in the manifest of such VNF, but they are here omitted for sake of conciseness. All these features belong to the F^{+}_{v4}, because they are related to configuration parameters that can be set up for the VNF, e.g., it is possible to specify which algorithm the VNF should use to encrypt specific kinds of traffic.

8.2.2 NSP Model

An NSP n is modeled as $n = (C_n, S_n)$:

- C_n expresses the conditions identifying the traffic on which the policy actions must be applied;
- S_n expresses the actions that must be applied to the traffic identified by the policy conditions, and the ways these actions must be performed (e.g., the algorithms to be employed).

C_n is a set, modeled as $C_n = \{c_1, c_2, ..., c_m\}$. Each $c \in C_n$ is defined over a field f, which is represented by the $c.f$ notation. The condition can specify a single value for the field (e.g., IPSrc = 10.0.0.1), a range of values (e.g., pSrc = [80-100]) or the special symbol $*$, meaning that each possible value that can be assigned to that field is valid.

S_n can be a set $\{s_1, s_2, ..., s_l\}$ or a list without repetitions $[s_1, s_2, ..., s_l]$. The set notation is used when the order of actions is not important, while the list notation when the order is relevant. Each $s \in S_n$ is modeled as (a_s, B_s), where:

- a_s is the action that must be applied (e.g., deny, decrypt);
- B_s is a set of bindings "field – (optional) value", specifying additional information about the enforcement mode of the action (e.g., the binding "IPSrc = 20.1.2.4" may specify how the source IP address must be changed by a network address translator, whereas "algorithm = AES-128-CBC" might specify the encryption algorithm a VPN gateway must use). If no binding is specified (e.g., when the action is applied on the whole packet satisfying the conditions), $B_s = \emptyset$.

The actions in S_n can be optionally grouped into multiple subsets K_1, K_2, ..., K_r, ..., K_p, where each subset must contain at least two actions. If two or more actions belong to the same K_r, it means they must be enforced by the same VNF. This formalization is introduced to support the cases where the actions cannot be managed by different VNFs. For example, a network administrator may require that all the packets satisfying certain conditions are logged and blocked by the same VNF, because they are dangerous and must be discarded as soon as possible, avoiding any further hop.

A first example of NSP is shown in (8.5). This policy requires that, for each packet satisfying all the conditions, firstly its source and destination IP addresses are logged, and then the whole packet is blocked so that it cannot reach the destination. In this case, for the $a_s = \log$ action, the two corresponding B_s elements are bindings characterized only by the fields (i.e., IPSec and IPDst), without any value. The reason is that they do not have to be modified during the logging operation.

$$\begin{array}{|l} \text{NSP } n_1 \\ C_{n_1} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\} \\ S_{n_1} = [(\log, \{\text{IPSrc}, \text{IPDst}\}), (\text{deny}, \emptyset)] \end{array} \tag{8.5}$$

A second example of NSP is shown in (8.6). This policy requires that, for each packet satisfying all the conditions, the source and destination IP addresses must be changed to the value specified in the two bindings associated to the $a_s = \text{modify}$

action.

$$\boxed{\begin{array}{l}\text{NSP } n_2 \\ C_{n_2} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}\} \\ S_{n_2} = \{(\text{modify}, \{\text{IPSrc=145.10.6.2}, \text{IPDst=80.2.5.24}\})\}\end{array}} \tag{8.6}$$

A third example of NSP is shown in (8.7). The difference with respect to (8.6) is that, if the NSP in(8.6) simply requires that the log and deny actions are performed sequentially, instead the NSP in(8.6) requires that they are executed by the same VNF, as they are grouped in the same K subset of the S_3 set.

$$\boxed{\begin{array}{l}\text{NSP } n_3 \\ C_{n_3} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\} \\ S_{n_3} = \{K = \{(\text{log}, \{\text{IPSrc}, \text{IPDst}\}), (\text{deny}, \emptyset)\}\}\end{array}} \tag{8.7}$$

8.2.3 Projection Model

A projection p represents the security operations that a VNF v should perform to enforce an NSP n in the network. A projection p is obtained by mapping the elements composing an NSP n (i.e., the policy conditions and actions) onto what a VNF v can offer to enforce the NSP (i.e., the VNF configuration settings). In this mapping operation, all the implementation-dependent and vendor-dependent characteristics of each VNF against which the NSP is projected are neglected. Therefore, if the same NSP is projected against different VNFs that fulfill the same security objectives, the resulting projections are the same.

As a projection directly derives from an NSP, it is modeled similarly, i.e., $p = (C_p, S_p)$ where:

- C_p expresses the conditions that determine the traffic on which the corresponding actions must be applied;
- S_p expresses the actions that the VNF against which the NSP is projected can enforce to fulfill it.

For example,considering the manifests of the three VNFs v_1, v_2 and v_3 presented in (8.1), (8.2) and (8.3), the projections deriving from mapping the policy n_1

presented in (8.5) onto these manifests are:

$$\begin{array}{l}\text{Projection } p_1\text{, derived by mapping the NSP } n_1 \text{ onto the VNF } v_1 \\ C_{p_1} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}\} \\ S_{p_1} = [(\text{deny}, \emptyset)]\end{array} \tag{8.8}$$

$$\begin{array}{l}\text{Projection } p_2\text{, derived by mapping the NSP } n_1 \text{ onto the VNF } v_2 \\ C_{p_2} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\} \\ S_{p_2} = [(\text{log}, \{\text{IPSrc}, \text{IPDst}\}), (\text{deny}, \emptyset)]\end{array} \tag{8.9}$$

$$\begin{array}{l}\text{Projection } p_3\text{, derived by mapping the NSP } n_1 \text{ onto the VNF } v_3 \\ C_{p_3} = \{\text{domain} = \text{dangerousSite.com}\} \\ S_{p_3} = [(\text{log}, \{\text{IPSrc}, \text{IPDst}\})]\end{array} \tag{8.10}$$

In order to enforce an NSP, it is necessary that all the actions required by the NSP are supported by a single projection or by a combination of projections. In this example, the projection p_2 supports both the actions of logging and packet blocking. Therefore it might be sufficient to enforce the NSP. Instead, p_1 and p_3 are not enough by themselves, because the former can only block packets, while the latter can only perform logging operations. As such, a combination of p_3 followed by p_1 would be needed.

8.3 Projection IDentification (PID)

***P**rojection **ID**entification* (PID) is a two-step algorithm that is proposed for the computation of the projections of the NSPs onto the VNF manifests. Fig. 8.1 shows the high-level structure of this algorithm.

The first step, named ***P**rojection **EX**traction* (PEX), requires the specification of two inputs: the NSPs that must be enforced in the network, and the VNFs that are available to the network administrator to be possibly deployed in the physical infrastructure. The PEX step maps each input NSP onto the manifest of each input VNF. The process is optimized so as to avoid redundancy in the produced result. In particular, if multiple projections are identical (e.g., it may occur that multiple VNFs

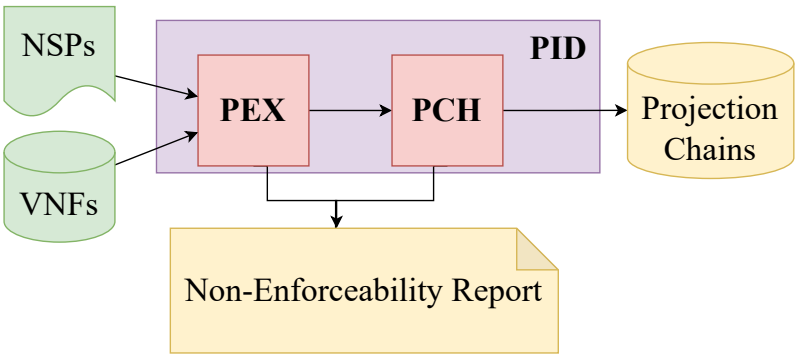


Fig. 8.1 Projection IDentification: the two stages

are implementations of the same security functionalities), a single instance is kept in the result. Instead, if no NSP projection can be derived from mapping that NSP onto a specific VNF manifest, that VNF is excluded from the rest of the algorithm, as it would be useless. This step, called ***P**rojection **EX**traction* (PEX), is detailed in Subsection 8.3.1.

The second step, named ***P**rojection **CH**aining (PCH)*, concatenates the projections computed in the PID stage, so as to create chains that can enforce all the actions requested by the input NSPs. As previously mentioned, not always a single projection can fulfill this objective, as there are cases where each projection conveys partial information about the fulfillment of an NSP. The chains that are so generated can be used later by other orchestrator tools to establish the security configuration (e.g., they can be used by the VEREFOO framework). This step is detailed in Subsection 8.3.2.

In both steps, the algorithm may halt due to some error conditions.

In case no valid chain is identified for an NSP, the global process immediately halts, and an early non-enforceability report is produced, notifying the network administrator why the projection identification failed. Otherwise, the valid combinations are passed on to the next stage of the process, for the synthesis of the virtual security graph. In this way, the security configuration will be performed in a way that is agnostic to the VNF implementation.

8.3.1 Projection EXtraction (PEX)

The *Projection EXtraction* (PEX) step projects each NSP against the manifest of each VNF. It is represented in Algorithm 4.

First, given a policy n and the manifest m_v of a VNF v, the condition set C_p of the corresponding projection $p_{v,n}$ is computed (lines 1-13). For each policy condition $c \in C_n$ based on a field f, the manifest of the VNF should include that field in the F_V^+ set or in the F_V^*. In the former case, the condition c simply becomes a condition of the projection as well (line 5), as the VNF can configure that field with specific values. In the latter, a new condition $f = *$ is created and included in the condition set of the projection, because the VNF can only take decisions regarding that field, but it cannot configure it (line 7). If the NSF manifest does not support any condition field of the policy, the resulting condition set of the projection remains empty. In this case, the algorithm immediately halts, and an early non-enforceability report is produced to inform the user about this issue (line 13).

Second, the action set of the projection is computed (lines 14-38). Differently from the condition case, it is not enough that a policy action $s \in S_n$ is included in the action set A_v of the VNF manifest (line 16). It is possible that the action s must be enforced on some fields and parameters (e.g., the packet source address must be modified). All the fields on which the action s is applied must belong to the F_V^+ set of the VNF, because the function must be able to directly operate on those fields (line 19). If one of them is not supported, then the action cannot be part of the output projection.

The creation of the action set requires an additional check with respect to the condition set. In the policy specification, the user may have requested that two or more actions must be necessarily applied by the same function (e.g., a packet satisfying certain conditions must be logged and then discarded avoiding any other hop in the network). Therefore, either all those actions are included in the action set of the projection, or none of them. After generating the A_v set, if the algorithm notices that only a subset of actions that should be applied by the same function is present in it, they are removed (lines 28-35). At that point, if the produced action set is empty, that would again trigger the generation of an early non-enforceability report to the user (lines 35-36). Otherwise, the projection is finally produced with the computed condition and action sets (line 39).

Algorithm 4 computation of p_{vn}

Input: a policy n, a VNF manifest m_v
Output: $p_{v,n}$

1: $C_p \leftarrow \emptyset$ ▷ Creation of the condition set
2: **for each** $c \in C_n$ **do**
3: **if** $\exists f \in F_v \mid f = c.f$ **then**
4: **if** $f \in F_v^+$ **then**
5: $C_p \leftarrow C_p + \{c\}$
6: **else**
7: $C_p \leftarrow C_p + \{f = *\}$
8: **end if**
9: **end if**
10: **end for**
11: **if** $C_p = \emptyset$ **then**
12: **exit**(no field is supported)
13: **end if**
14: $S_p \leftarrow \emptyset$ ▷ Creation of the action set
15: **for each** $s \in S_n$ **do**
16: **if** $s.a_s \in A_v$ **then**
17: supported$(s.a_s) \leftarrow$ true
18: **for each** $b \in s.B_s$ **do**
19: **if** $b.f \notin F_v^+$ **then**
20: supported$(s.a_s) \leftarrow$ false
21: **end if**
22: **end for**
23: **if** supported$(s.a_s) =$ true **then**
24: $S_p \leftarrow S_p + \{s\}$
25: **end if**
26: **end if**
27: **end for**
28: **for each** $s \in S_p$ **do**
29: **for each** $s' \in S_n$ **do**
30: **if** $s \neq s' \wedge s' \notin S_p \wedge (\exists K_l | s, s' \in K_l)$ **then**
31: $S_p \leftarrow S_p \setminus \{s\}$
32: **break**
33: **end if**
34: **end for**
35: **end for**
36: **if** $S_p = \emptyset$ **then**
37: **exit**(no action is supported)
38: **end if**
39: **return** (C_p, S_p)

The worst-case time complexity of Algorithm 1 can be estimated as the sum of the time complexities of three sequential code blocks. Lines 1-13 have $O(|C_p|)$ complexity, because $O(1)$ operations are performed on each element of C_p. Lines 14-24 have $O(|S_n| \cdot \max_{s \in S_n}(s.B_s))$ complexity because they represent a nested loop.

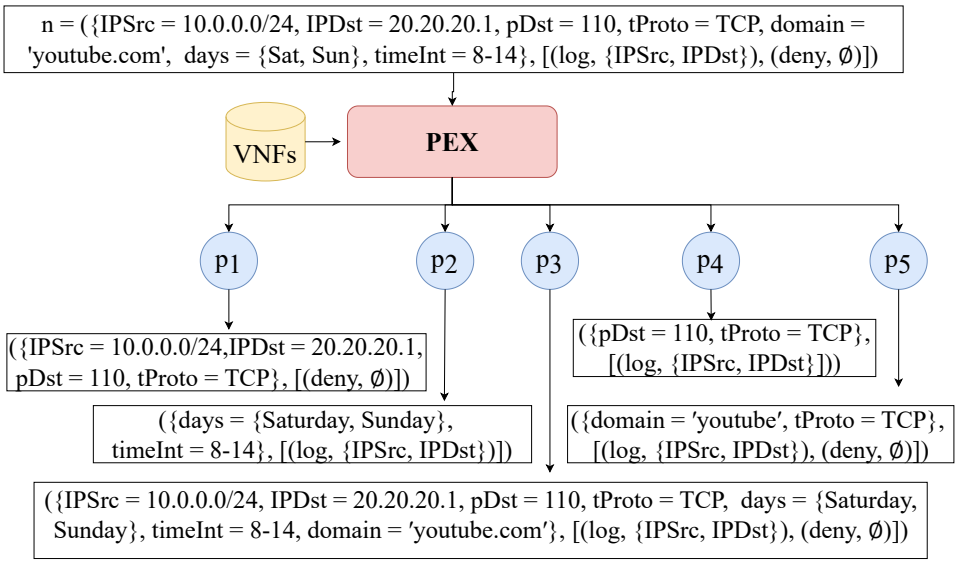


Fig. 8.2 Projection EXtraction: a visual example

The external one iterates on each element of S_n, whereas the internal one requires a number of iterations that in the worst case is equal to the cardinality of the largest $s.B_s$ set, with $s \in S_n$ Then, lines 28-39 have $O(|S_n|^2)$ complexity, because both the external and internal loops in those lines iterate on the S_n set. Summing up, the overall worst-case time complexity is $O(|C_p| + |S_n| \cdot \max_{s \in S_n}(s.B_s) + |S_n|^2)$. However, by considering the NSPs that are commonly defined in studies related to security policy refinement [8][55], the condition set size is commonly higher than the action set size (e.g., just by imposing conditions on the IP 5-tuple, five conditions are included in the C_p set). Therefore, the dominant term in the asymptotic complexity formula is $|C_p|$.

A visual example is shown in Fig. 8.2. Each projection denoted by the p symbol derives from mapping policy n against a different vNSF manifest. For instance, p_1 derives from a simple packet filter that cannot manage fields related to domain or time interval, p_3 derives from a VNF that may fully enforce the NSP, and p_4 from a VNF that can only log the IP 5-tuple of the received traffic.

Given an NSP, the algorithm is repeated for each VNF that is available. The resulting projections may not contain all the information of the original NSP, e.g., they may support a partial set of all the actions requested by the NSP. Therefore, the PEX stage is not sufficient, but a projection chaining operation is still required.

Symbol	Explanation
n	network security policy that must be enforced
S_n	action set of n
s_i	i-th action in S_n, with $i = 1, ..., \|S_n\|$
K_l	h-th subset of actions of S_n, including the same function should be in charge of, with $l = 1, ..., m$
a_{s_i}	operation of i-th action
B_{s_i}	enforcement mode set of i-th action
b_{ih}	h-th enforcement mode of i-th action, with $h = 1, ..., \|B_{s_h}\|$
P_v	projection set
p_j	j-th projection of P_v, with $j = 1, 2, ..., \|P_v\|$
x_{ij}	binary variable, whose value is set to 1 by the solver if the j-th projection is chosen as responsible for the i-th action of p, otherwise it is set to 0
y_{ij}	binary variable, whose value is set to 1 before launching the solver if the j-th projection supports the i-th action, otherwise it is set to 0
z_{ijh}	binary variable, whose value is set to 1 before launching the solver if the j-th projection supports the i-th action with h-th enforcement mode of p, otherwise it is set to 0

Table 8.2 Symbol table

8.3.2 Projection Chaining (PCH)

The Projection CHaining (PCH) operation aims to compute all the possible chains of the projections output by the PEX operation. As the PID stage is topology-independent and it works on each policy independently from the other ones, it cannot decide if a chain is more suitable than the others.

The problem of finding the projection chains has been formulated as an *Enumeration Problem* (EP) over a set of *Constraint Satisfaction Problem* (CSP)-like formulas. A chain is a solution for the EP if it contains a projection responsible for each action of the original NSP. Among all the projections that support a certain NSP action, the solver chooses a single one as responsible with the aim of avoiding redundancy. Equations (8.11)-(8.14) represent the problem constraints, and TABLE 8.2 describes the symbols used for their formulation. Among them, the output binary variable appearing in the formulas, x_{ij}, expresses if a certain projection p_j is chosen as responsible for action s_i (when $x_{ij} = 1$) or this task is assigned to another projection (when $x_{ij} = 0$). If S_n is a set, the assignment of index i to each policy action is random. Instead, if S_n is a list, the assignment naturally follows the ordering of the actions in it, so that index 1 is assigned to the first action, and index $|S_n|$ is assigned

to the last one.

$$\sum_{j=1}^{|P_v|} x_{ij} = 1, \forall i = 1, ..., |S_n| \tag{8.11}$$

$$x_{ij} \leq y_{ij}, \forall i = 1, ..., |S_n|, \forall j = 1, ..., |P_v| \tag{8.12}$$

$$|B_{s_i}| \cdot x_{ij} \leq \sum_{h=1}^{|B_{s_i}|} z_{ijh}, \forall i = 1, ..., |S_n|, \forall j = 1, ..., |P_v| \tag{8.13}$$

$$|K_l| \cdot x_{ij} \geq \sum_{i'|s_{i'j} \in K_l} x_{i'j}, \forall i = 1, .., |S_n|, \forall j = 1, .., |P_v| \tag{8.14}$$

The four quantified CSP-like formulas can be explained as follows:

1. According to formula (8.11), one and only one projection p_j is responsible for action s_i. Even though multiple projections may fulfill this task, in each enumerated solution only one is chosen.

2. According to formula (8.12), a projection p_j can be chosen as responsible for action s_i only if it supports the operation a_{s_i}, i.e., if $\exists s_k \in S_p$ such that $a_{s_k} = a_{s_i}$. This constraint is required as the PEX operation may have extracted a projection that only partially supports the actions required by the corresponding NSP.

3. According to formula (8.13), a projection p_j can be chosen as responsible for action s_i only if it supports all the enforcement modes defined in B_{s_i}. This constraint is included in the formulation of the EP problem only if $B_{s_i} \neq \emptyset$, otherwise constraint (8.12) is enough as condition of choice.

4. According to formula (8.14), if a projection p_j is chosen as responsible for action s_i and if s_i belongs to a set K_l of actions that must be applied by the same projection, then p_j must be responsible also for all the other actions in K_l as well. This constraint is included in the formulation of the EP problem only if there is the specification of at least a K_l set.

Each assignment for the x_{ij} variables represents a possible projection chain, as $x_{ij} = 1$ implies that the i-th action requires an instance of the j-th projection for its enforcement. However, the solution set computed by solving this EP problem may

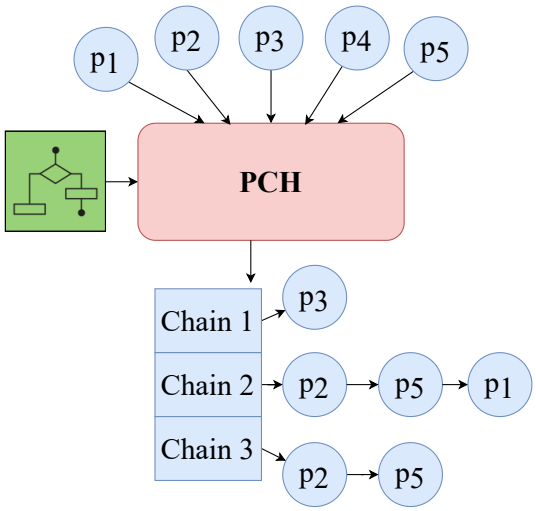


Fig. 8.3 Projection CHaining: a visual example

not be complete, and two post-processing operations may be required under specific circumstances.

First, if S_n is an (unordered) set, the assignment of index i to each policy action does not follow any strict ordering guideline. Therefore, only a possible permutation of the actions out of the $|S_n|$ possible ones is established. This deficiency is easily overcome, by computing all the other permutations.

Second, it may happen that two variables having consecutive values for the i index have the same j index, i.e., the actions require the same projections to be enforced. On the one hand, if the two actions are not part of a K_l set, then either a single instance or a pair of instances of the j-th projection may be used to enforce those actions. Therefore, both solutions must be derived from the single assignment computed by solving the EP problem. On the other hand, if the two actions are part of a K_l set, then the only possible solution is the one where a single projection is used.

A visual example of the PCH operation is shown in Fig. 8.3. The projections that are input to the EP characterizing this operation are the same ones that were presented in Fig. 8.2. They are combined in three different chains, which can enforce all the actions of the requested NSP n. As it can be seen, Chain 1 is composed of a single projection, p_3, because it can perform both the requested operations, i.e., logging the source and destination IP addresses of the packets identified by the policy conditions, and then block those packets from reaching their destination. Instead,

the other chains require more projections, because each one of them is not enough to enforce all the actions.

After the completion of the post-processing operations, these projection chains can be used for the security configuration of a virtual network service. In particular, the VEREFOO approach, already discussed in Chapter 4, may be used to allocate the projections composing these chains in an AG, ths enforcing the NSPs themselves from which the projections have been computed. This additional refinement operation (i.e., from the output of the PID algorithm to the allocation scheme and configuration of these projections that the VEREFOO approach can compute) is motivated by two reasons. First, not always a one-to-one relationship exists between a projection computed by the PID algorithm for a specific chain, and the number of instances of that same projection that must be allocated in a network. For example, if a projection requires to block all the traffic flows coming from 154.66.0.2 to any IP address: "({IPSrc = 154.66.0.2}, and if these flows do not have any intersection in their paths, then multiple projection instances must be allocated, i.e., one for each path. Second, the refinement operation performed by the VEREFOO approach is aware of the network topology and configuration, differently from the PID algorithm. Therefore, it can adapt the configuration of the projections to the characteristics of the network where they are allocated. For example, if all the packets from the source port 88 of 123.4.5.6 to 44.5.6.7 must be logged, and if the projection p derived from this NSP can only be allocated in a position that is after a NAT which changes the source IP address of those packets, then the C_p set must be modified so as the source IP address is associated to the value updated by the NAT.

8.4 Implementation and validation

The PID algorithm has been implemented as a Java framework. Human users can interact with this framework through a set of REST APIs, which can be used to specify the input VNF manifests and the NSPs in XML or JSON format. The same format is used for the representation of the output, i.e., the automatically computed projection chains. The code of the developed software is parallelized, and it allows the user to specify the number of execution threads. This design decision was possible, as both the PEX and PCH stages can work on each NSP independently

from the other. If the user does not specify the thread number, then eight threads are used by default.

In the framework implementation, the enumeration problem of the PCH stage is internally formulated and solved by employing the mathematical programming solver Gurobi[1]. This solver can work on different types of problems, e.g., linear programming, mixed-integer linear programming, quadratic programming. Gurobi offers simple APIs in multiple high-level programming languages, including Java, which is used for the framework implementation.

The experimental validation of the framework has been performed with multiple types of tests, with the aim to check: i) the generality of the models defined for the VNF manifests (Subsection 8.4.1); ii) the correctness of the algorithms employed for projection extraction and chaining (Subsection 8.4.2); iii) the scalability of the PID algorithm and its superiority with respect to the state of the art solutions (Subsection 8.4.3).

All tests have been performed on a machine with an Intel i7-6700 CPU running at 3.40 GHz and 32GB of RAM, by exploiting the version 8.1.1 of Gurobi.

8.4.1 Model generality validation

18 open-source VNFs that are currently available for enforcing security requirements have been analyzed, and their manifests have been modeled. Among the analyzed VNFs, there are packet filtering firewalls (iptables, ipfirewalls, nftables, PfSense), web-application firewalls (ModSecurity, IronBee, NAXSI, WebKnight), anti-spam filters (SpamAssassin, MailCleaner, Rspamd), VPN gateways (Strongswan, Openswan, SoftEther, OpenConnect), intrusion detection systems (Suricata, Snort, Zeek). The configuration guides and official examples have been carefully analyzed to determine the actions (i.e., the A set) and configuration fields to compose their manifests (i.e., the A set), as well as the fields for which the VNFs may take decision but they cannot be configured (i.e., the F^* set). This last modeling step required reasoning about what security properties each VNF can enforce, also by referring to the classical ISO/OSI protocol stack. For example, all the fields related to web applications (e.g., URL, domain) belong to the F^* set for each analyzed packet filtering VNF. The investigation of these open-source functions shows that our model is general enough

[1]Link: https://www.gurobi.com/. Last accessed: October 26th, 2021.

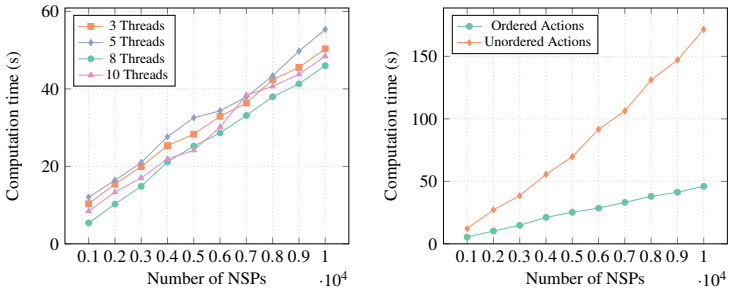
to support their representation as manifests. Besides, it can be extended for future VNFs by introducing new actions and fields in the A, F^+ and F^* sets.

8.4.2 Correctness verification

The correctness of the framework has been validated by applying it on several simple use cases. For each use case, a different set of VNF manifests and NSPs has been created, the framework has been fed with these inputs to compute the output projection chains, and the results have been checked manually against the expected ones. Variations of the use cases have been obtained by changing the set of available VNF manifests. Some peculiar use cases have also been created, to test specific features of the PID algorithm. For example, the following use cases have been investigated:

- VNFs with the same manifest are used to enforce an NSP, to check that the framework creates the same projection for them and uses it once in creating the projection chains;
- an inadequate number of VNFs is used to enforce an NSP, to check that the framework produces a non-enforceability report stating that a feasible solution for the security configuration problem does not exist;
- NSPs with unordered actions have been written, to check that the framework can also consider the more complex case where all the chains derived from the solution of the enumeration problem are subject to a post-processing step to consider all the possible permutations of the projections;
- NSPs with actions that must be applied on packet fields have been written, to check that the framework uses only the VNF manifests having all those fields in the F^+ for creating a projection.

Then, the VEREFOO approach has been fed with the output of the PID algorithm, to automatically configure network security for the use cases related to packet filtering and communication protection. As it has been done to check the correctness of the VEREFOO approach in other validation tests, the compliance of the obtained output with the user-specified NSPs has been checked by using the Mininet emulator.



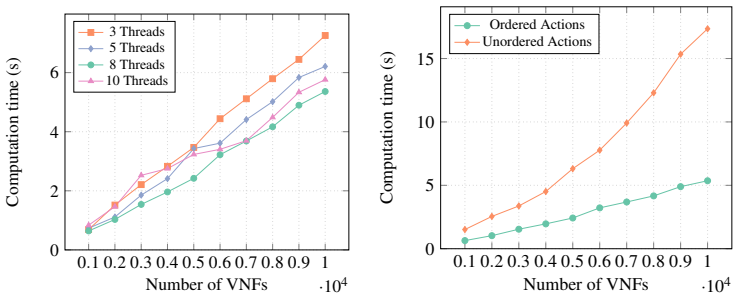
(a) Thread number (b) Action ordering

Fig. 8.4 Scalability versus number of Network Security Policies

8.4.3 Scalability evaluation

The scalability of the framework has been validated against two dimensions, which are the number of NSPs and the number of VNFs the projections must be derived from.

Fig. 8.4 shows the results for the scalability tests related to the number of NSPs. Each dotted plot in the charts represents the average value computed over 100 iterations on a use case, where the number of VNFs is fixed to 100, whereas the number of NSPs is progressively increased from 1000 to 10000. On the one hand, 8.4a plots the results related to the investigation of scalability against NSPs with ordered actions, while varying the number of threads used for the execution of the PID algorithm. For the machine that has been employed for the execution of the tests, the least computation time is almost always achieved when eight threads are used. If a higher number of threads is employed, the performance gets worse, because of the thread creation overhead. This experimental result is also the reason why in the PID algorithm eight threads are used by default, in the absence of a user specification. On the other hand, Fig. 8.5b shows the impact of ordering the actions of the user-specified NSPs. The performance of the framework gets worse if the actions characterizing each NSP are unordered. The computation time for the worst use case that has been considered for creating this chart ranges from less than one minute if the actions are ordered, to almost three minutes if the actions are unordered. This difference is explained by the fact that, in the PCH algorithm, all the possible permutations of the actions must be considered, when the NSPs have unordered actions. Anyhow, it is also important to remark that the case where all 10000 NSPs require unordered actions has been artificially created to put the framework under



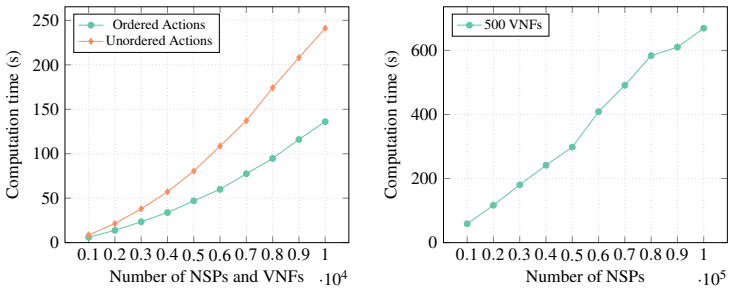
(a) Thread number (b) Action ordering

Fig. 8.5 Scalability versus number of Virtual Network Functions

great stress, as a worst-case. However, in reality, it can be expected that the NSPs that must be enforced in a network have a mixed nature, i.e., some require ordered actions while others do not. Besides, the result achieved for the worst case is significant by itself, especially if compared with the time that can be expected to do the same tasks manually. Indeed, it can be expected that manual approaches would struggle in dealing with so large numbers of NSPs, probably leading to mistatkes, and taking a time much higher than the three minutes taken in the worst case by the framework.

Fig. 8.5 shows the results for the scalability tests related to the number of VNFs. Each dotted plot in the charts represents the average value computed over 100 iterations on a use case, where the number of NSPs is fixed to 100, whereas the number of VNFs is progressively increased from 1000 to 10000. Considerations related to these results are similar to the ones concerning the results of Fig. 8.4. On the one hand, Fig. 8.5a again shows that the optimal number of threads to execute the algorithm in the employed machine is eight. On the other hand, Fig. 8.5b shows that the cases where NSP actions are unordered require a higher computation time than those where actions are ordered. In addition, it can be noted that scalability against VNFs is better than scalability against NSPs. This result can be explained because for each NSP the whole PID algorithm must be executed, whereas each additional VNF simply represents an additional decision variable, and the number of times the PID algorithm is executed remains the same.

Fig. 8.6 shows the results for the scalability tests related to two peculiar scenarios. Agains, each dotted plot in the charts represents the average value computed over 100 iterations on the two use cases. On the one hand, Fig. 8.6a shows the behavior of the framework when the numbers of VNFs and NSPs are equally increasing till 10000. Even though the computation time is higher than the ones shown in Fig.



(a) Equal increasing number of NSPs and VNFs

(b) Very high scalability versus number of NSPs

Fig. 8.6 Validation under two peculiar scenarios

5.1b and Fig. 5.2b, the trend of the plot is not exponential, but it follows the same growth of the previous charts, also when NSPs have unordered actions. On the other hand, Fig. 5.3a shows the behavior of the framework when run on a stressing scenario, consisting of 500 available VNFs, and a number of NSPs to be enforced in the network progressively increasing from 10000 to 100000 NSPs. Even though such high numbers may be extreme, the fast growth of modern virtual networks may create circumstances where they are not so uncommon. The developed framework is able to manage even this case, as the time required for computing the projection chains is much less than the time a manual approach would require, and again the trend of the plot is not exponential.

Chapter 9

VEREFOO Integration with Orchestrators and Applications

This chapter discusses how the VEREFOO approach, proposed in Chapter 4, Chapter 5 and 6 to solve the automatic configuration problem for different types of NSFs, can be integrated with state-of-the-art network orchestrators and applications. Specifically, VEREFOO has been integrated with two orchestrators, i.e., Docker Compose (Section 9.1) and Kubernetes (Section 9.2), and its application in the context of IoT networks has been demonstrated (Section 9.3).

9.1 Integration with Docker Compose

According to the official Docker documentation[1], Docker Compose is a container orchestration technology that can be used to run multiple containers on a single host machine. In fact, it is a tool for defining and running multi-container Docker applications. Docker Compose allows the specification of a YAML file to configure each service of an application. Then, with a single command, it is possible to create and start all the services from the configuration specified in the YAML file, as Docker Compose supports an automated deployment of the service. It is also flexible, as the environment can be deployed or removed in a few seconds. Common use cases of Docker Compose are development environments, automated testing environments, and single host deployments.

[1]Link: https://docs.docker.com/compose/. Last accessed: October 18th, 2022.

Docker Compose has been chosen as a network orchestrator for the integration of the VEREFOO approach, because it is suitable for testing purposes, it supports a declarative approach for creating network topologies, and it eases the extension of an already created network to a more ramified topology. Specifically, the VEREFOO application that has been integrated and tested with Docker Compose is the one related to packet filtering firewalls (Chapter 5).

Considering the output of the VEREFOO framework, i.e., the network topology enriched with the firewall allocation scheme and configuration, the following implementation choices have been made for the integration with Docker Compose, so as to maximize efficiency, minimize resource consumption for the deployment of the functions, and keep only the minimal network functionalities that are really required:

- Each end point of the network topology output by the VEREFOO approach is implemented as a container with an Alpine Linux image, which is a very stripped down OS, with a dimension of only 6 MB. The Alpine Linux image contains most of the functionalities needed for end points, without overloading the virtual environment with useless complex features.
- Each router of the network topology is implemented as a container with an Alpine Linux image as well, as it is enough to perform routing.
- For more complex functions such as network address translators and load balancers, the Alpine Linux image is enriched with the installation of the iptables-firewall function, so as to fulfill their behavior.
- For firewalls, three different types of functions have been integrated: 1) iptables-firewall is the easiest to deploy, since it just requires an Alpine Linux image with iptables installed on it. This makes the iptables virtual network environment the most lightweight in terms of resource consumption; 2) bpf-firewall requires a larger image and consumes more resources, due to the several required dependencies; 3) openvswitch is derived from the Docker bridge, with some changes at the host OS level to adapt its normal behavior.

By means of a translator that has been implemented, the network topology output by the VEREFOO framework is automatically translated into a set of containers as described above, and the routes are defined so as to guarantee connection, as required by the directed links of the graph.

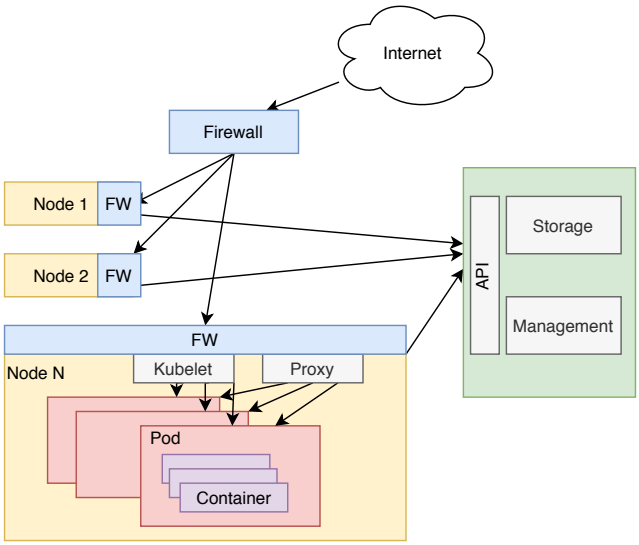


Fig. 9.1 Kubernetes architecture

This integration has been tested with the hping3 command, so as to check if the user-specified NSPs have been correctly enforced in the virtual environment managed by Docker Compose.

9.2 Integration with Kubernetes

A limitation of Docker Compose is that it is designed for deployment testing purposes, instead of production. Therefore, the VEREFOO approach has also been integrated with another state-of-the-art orchestrator, Kubernetes, which provides higher flexibility and additional features.

9.2.1 Kubernetes

Kubernetes[2], also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

As shown in Fig. 9.1, a Kubernetes cluster is composed of multiple nodes, which can be virtual or physical. A Pod is a minimal management unit which can

[2]Link: https://kubernetes.io/. Last accessed: October 18th, 2022.

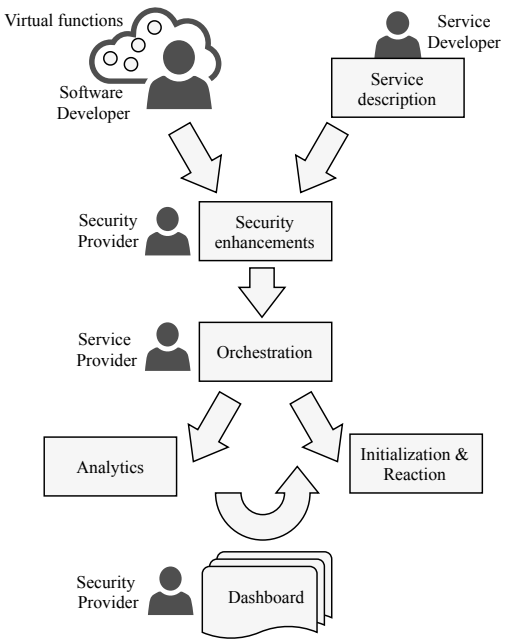


Fig. 9.2 Overall workflow of the ASTRID framework

accommodate one or more containers. Each Pod is protected by a packet filter (i.e., FW in Fig 9.1). Pods are assigned with network addresses and are allocated to nodes. Containers inside a Pod share resources, such as volumes where they can write and read data. Clients contact the cluster through another firewall, which distributes requests to nodes according to load-balancing rules. The proxy receives requests from this firewall and forwards them to Pods. Each node has a proxy installed. If a Pod is replicated, the proxy distributes the load among the replicas. A kubelet is a component that manages Pods, containers, images and other elements in the node. A kubelet forwards data on the monitoring of containers to the main node, which acts when necessary.

9.2.2 ASTRID Security Orchestrator

Kubernetes is an orchestrator that is mainly designed to address networking issues in cloud environments. However, network security is a major concern in such a context. For example, if the underlying infrastructure of the cloud is unreliable (or configured in a vulnerable manner), there is no way to guarantee the security of a Kubernetes cluster built on this foundation.

A project funded by the European Union, *AddreSsing ThReats for virtualIseD services* (ASTRID)[3], aimed to address these technological security gaps in the scope of cloud infrastructures. The project proposed a novel cyber security framework, named ASTRID Security Orchestrator, to provide situational awareness for cloud applications and NFV services. The overall workflow of the framework is presented in Fig. 9.2. According to the workflow, the ASTRID framework allows software and service developers to provide a description of the service request, which is enriched with security policies by the security provider entity. The code is publicly available at the following link: https://github.com/astrid-project.

The ASTRID Security Orchestrator is responsible for the support of the dynamic adaptation of the behaviour of the network security functions and of the dynamic implementation of reaction and mitigation actions on the deployed security service. The Security Orchestrator has been designed as a stand-alone component, which does not require any modification to the main Service Orchestrator (i.e., Kubernetes), but only requires the availability of an API for remote invocation of orchestration actions. The Security Orchestrator must fulfill two main goals, related to supporting human operations in the dynamic configuration of virtual security services and reacting to possible cyber-attacks. The two tasks it must fulfill are the following:

1. The Security Orchestrator should change the configuration/behavior of local security agents, according to the evolving context. The fulfillment of this task requires dynamism and readiness, especially when the network security functions must be updated to avoid that they are compromised by a detected cyber attack. The configuration/behavior update involves many low-level operations as well (e.g., provide requested data to a newly started security service, enable a new feature that is required to identify a new form of attacks), for which the different components of the ASTRID architecture should be properly interfaced.

2. The Security Orchestrator is conceived as an orchestrator-agnostic framework so that any software service orchestration tool available from commercial solutions or open-source projects can be easily integrated for the management of service orchestration operations (e.g., life-cycle management and deployment for virtual functions).

[3]Link: https://www.astrid-project.eu/. Last accessed: October 18th, 2022.

In order to achieve these main objectives, the architecture of the Security Orchestrator has been designed to include three main building blocks: Context Broker, AAA and Security Controller.

- The Context Broker provides the interfaces between the ASTRID framework and the network security functions deployed in the service graph by the Service Orchestrator. In order to cope with the heterogeneity of the network security functions (in terms of functionality and configurability), this component of the architecture uses suitable abstraction models, e.g., a generic context model, which defines the type, position, and capabilities of the security functions deployed in the service graph.
- The AAA module includes identity provisioning as well as the core capabilities to authenticate, authorize and audit authorization, authentication and access operations. It also verifies operations between the different components/services/microservices of the ASTRID framework.
- The Security Controller automates the response to security events; it is conceived as a policy-based framework (which is able to load a set of rules and evaluates them at run-time) that elaborates notifications and alerts provided by the security services and automatically undertakes control/management actions (e.g. for remediation, mitigation, investigation).

The Security Orchestrator and, more specifically, the Security Controller must communicate with other components of the ASTRID architecture in order to properly fulfill their tasks:

- The ASTRID dashboard is used as GUI for interaction between the ASTRID framework and the network and security administrator. For instance, the ASTRID dashboard can load a Kubernetes template and start the deployment, edit and select the set of security policies, show and change the configuration of ASTRID security agents.
- The ASTRID algorithms enable ASTRID stakeholders to assess the security properties of their services, to detect, to manage, and to react to vulnerabilities, threats, attacks and anomalies.

9.2.3 Security Controller

The ASTRID Controller is the core element of the ASTRID architecture. The implementation of the Security Controller is based on the *Event-Condition-Action* (ECA) paradigm. Any algorithm based on the ECA paradigm is characterized by a set of rules, where each rule is made by three main components: Event which may trigger the rule, Condition to be satisfied so that the rule is effectively applied in reaction to the triggering Event, and the Action to be performed. In the ASTRID framework, an Event may be triggered by the data plane, the management plane (i.e., manual indications from the dashboard, notifications from the service orchestrator), or the control plane (i.e., an algorithm or a security service). A Condition typically considers context information as graph topology, security configurations, data source, date/time, user, past events, etc. Finally, an Action might not be limited to simple commands, but can implement complex logics, also including some form of processing on the run-time context (e.g., to derive firewall configuration for the running instance), to respond, mitigate, or prevent attacks. Based on the ECA paradigm, the range of possible operations performed by the Security Controller is very broad. Most of the ECA rules of the Security Controller, are not pre-computed a priori, but they are automatically generated by a set of requirements, defined or chosen by security administrator through the dashboard.

Among all the possible operations that can be performed by the Security Controller, there is the application of the VEREFOO framework to incorporate programmability and automation in the synthesis of virtual firewall rules from user-specified connectivity policies. For its applications, the Security Controller needs to work in close coordination along with the Service Orchestrator, which is in charge of providing a description of the service graph as well as the infrastructure information. The infrastructure information includes the actual number of launched virtual network functions and parameters assigned after the enforcement process such as IP and port addresses.

The interaction of the Security Controller with the VEREFOO framework and the Service Orchestrator (i.e., Kubernetes) occurs as explained below.

1. After a set of connectivity policies is pushed by the user to the Security Controller, they are delivered to the VEREFOO framework together with

the run-time configuration of the network (i.e., IP addresses) retrieved from Kubernetes.

2. The VEREFOO framework computes the optimal configuration parameters of each firewall instance so as to enforce the user-specified connectivity policies.

3. The low-level configuration parameters of each firewall are delivered to the Security Controller, after they are derived with a translation step from the medium-level language used by the VEREFOO framework.

4. The Security Controller receives the firewall configuration via REST API and delivers them to the Service Orchestrator, that finally performs the enforcement of the rules on the allocated firewalls.

Next, the Security Controller listens for alerts that may trigger the need of a new firewall reconfiguration, e.g., the notification about a new instance of the virtual function that is added or removed in the network, or the notification about the suspicious misbehaving of function in the network. At that point, the interaction with the other modules continues as explained below.

5. The Security Controller updates the original use-specified connectivity policies, so as to take into account the received alert.

6. The Security Controller sends the modified connectivity policies to the VEREFOO framework, together with the new run-time network configuration.

7. The previous 2), 3) and 4) steps are repeated.

9.2.4 Use case scenario

Here, a practical use case is described to show how the Security Controller works, and how it interacts with other components of the ASTRID frameworks by means of a REST API interface. Specifically, the Security Controller exposes a number of resource endpoints to the Service Orchestrator, which will use them to deliver the service graph and infrastructure information and to retrieve the automatically generated firewall rules.

In the analyzed scenario, an administrator predefines the logical service graph presented in Fig. 9.3a and feeds it to the dashboard of the ASTRID framework. This

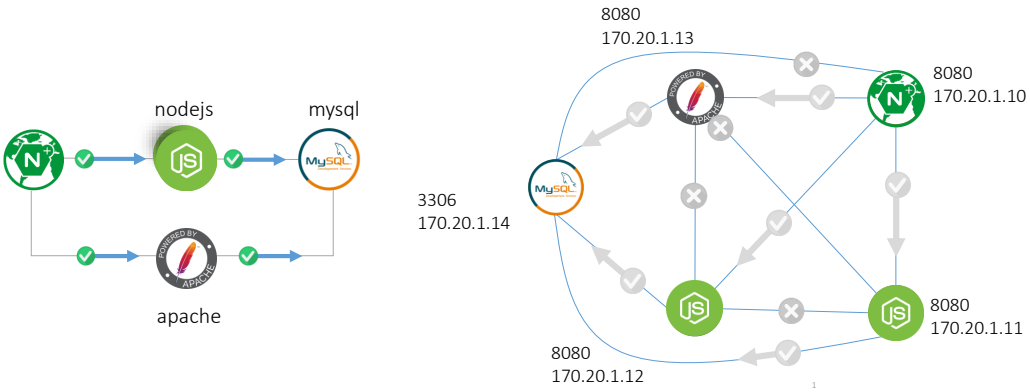


Fig. 9.3 a) a logical service graph b) enriched service graph after the deployment

service graph represents a realistic scenario where the *nginx* web server is made public to the Internet and functions as a reverse proxy to fetch dynamic data from multiple instances of *nodejs* and *apache* servers. In this case, both servers can acquire data from a *mysql* database. As it can be seen from the figure, reachability policies required by the use case are rather obvious (i.e., highlighted with arrows). Instead, the isolation property required by the service graph is not evident. For instance, all the communications not highlighted with arrows must be isolated. Considering the fact that each service in the graph is associated with a firewall, firewalls are preconfigured with *deny-all* rules, in order to satisfy this policy. This ensures that all other interactions within the service graph are isolated, except the ones predefined by the user (i.e., arrows).

The Service Orchestrator of the ASTRID framework is in charge of deploying the service graph onto the infrastructure and generating the enriched service graph shown in Fig. 9.3b. During this enforcement phase, all the services are assigned corresponding IP addresses and ports where these services can be reached. The multiple instances of the services are deployed in separate Pods and each will have its own IP addresses. In this scenario, the user specified to have two instances of the *nodejs* server to handle the load. To illustrate the complexity introduced by this simple use case, all the links connecting each service in the infrastructure are included in Fig. 9.3b. Taking into account the *deny-all* rules of each firewall of the service, there is no reachability between the Pods in this phase. Instead, the user policy that needs to be satisfied is illustrated by means of the arrows in the figure. As an example, *apache* server needs to be configured to allow traffic from itself to

a *mysql* database and allow communication from *nodejs*. However, it needs to be isolated from each instance of the *nodejs* servers.

To obtain the low-level configuration of each firewall component, the Security Controller accepts as an input the infrastructure information and logical service graph. Infrastructure information contains the IP and port addresses of each service that is shown in Fig. 9.3b. This information is required to define the firewall rules, which allow blocking specific packet flows involving specific Pods. In the next step, the Security Controller automatically generates an output with a low-level configuration of each firewall component. As an example, the partial output format and the actual configuration parameters generated by the Security Controller are presented in Listing 9.1.

Listing 9.1 Automatic Configuration Output for mysql

```
<node name="172.20.1.34" functional_type="FIREWALL">
<neighbour name="172.20.1.14"/>
<neighbour name="172.20.1.30"/>
<neighbour name="172.20.1.31"/>
<neighbour name="172.20.1.32"/>
<neighbour name="172.20.1.33"/>
<configuration name="mysql" description="172.20.1.14">
   <firewall defaultAction="DENY">
      <elements>
         <action>ALLOW</action>
         <source>172.20.1.13</source>
         <destination>172.20.1.14</destination>
         <protocol>ANY</protocol>
         <src_port>*</src_port>
         <dst_port>*</dst_port>
      </elements>
      <elements>
         <action>ALLOW</action>
         <source>172.20.1.11</source>
         <destination>172.20.1.14</destination>
         <protocol>ANY</protocol>
         <src_port>*</src_port>
         <dst_port>*</dst_port>
      </elements>
      <elements>
         <action>ALLOW</action>
         <source>172.20.1.12</source>
         <destination>172.20.1.14</destination>
         <protocol>ANY</protocol>
         <src_port>*</src_port>
         <dst_port>*</dst_port>
      </elements>
   </firewall>
</configuration>
```

```
35 </node>
```

Listing 9.1 shows the configuration parameters generated for the firewall component of the *mysql* service. It includes all the neighbors of the firewall in the infrastructure network and firewall rule entries. According to the output, the firewall needs to be configured with three rules.

The first rule states that the packets arriving from the Pod with an IP address 172.20.1.13 need to be allowed. The rest of the rules are associated with the two instances of the *nodejs* server of the service graph. Due to the default action set by the firewall in line 8, Listing 9.1, all the other packets arriving from the network is dropped. For instance, intruders from the Internet are not able to access the *mysql* database in accordance with these rules. This, in fact, ensures the satisfiability of the initial service graph policy defined by the user.

An important feature of the Security Controller is the possibility of firewalls without any configuration as in the use case or with partial configuration, giving the tool itself the task of providing the missing configurations as an output. The tool generates the configuration with the objective of satisfying all the requested policies while minimizing the number of generated rules in order to achieve it. In the case of partial configuration, a firewall may include static rule entries that will not be changed in the output. This is useful when the service graph is updated according to an event when a Pod is terminated or a new Pod has been created to handle the overhead to the service. In this scenario, in order not to recompute the configuration parameters of all the other services, their rules can be provided in a static manner, meaning that they can be left unchanged. This process not only generates a set of configuration parameters but also provides an optimal set of rules to satisfy the user policy. Optimality is achieved by minimizing the number of rules inside each firewall to improve the performance of the virtual network functions.

9.3 VEREFOO applications in IoT networks

In the context of the EU project CyberSec4Europe[4], the VEREFOO approach has been selected as one of the assets promoted by the project and employed in a demonstration related to a specific smart city security use case.

[4]Link: https://cybersec4europe.eu/. Last accessed: October 18th, 2022.

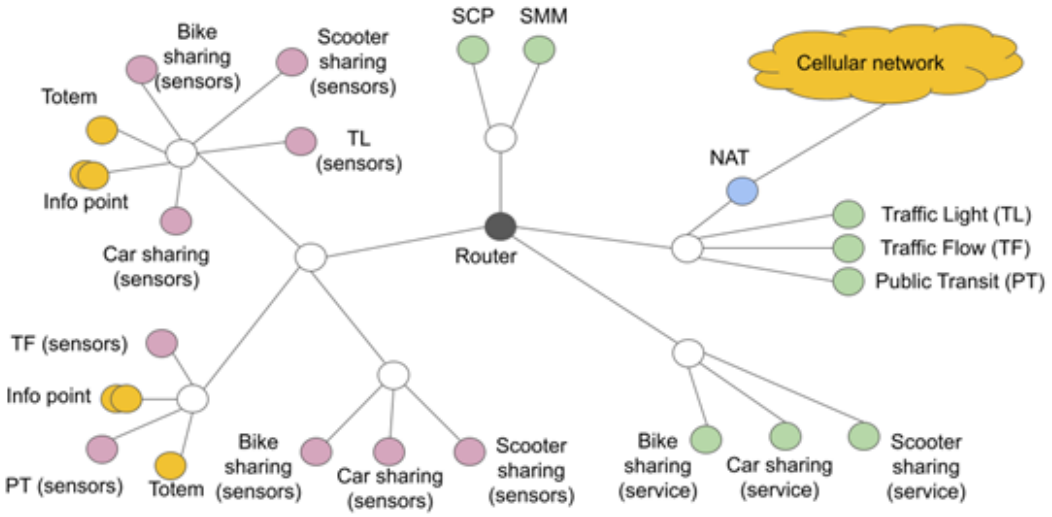


Fig. 9.4 The smart city use case.

The smart city scenario is characterized by a high level of complexity, as it involves IoT aspects. Moreover, such scenario envisions complex interactions between human users and different types of heterogeneous services that rely on the security of lower level technologies, including the communication network. Therefore, the security of a smart city is strictly dependent to a correct definition and enforcement of the security policies that must be applied to the network. The use case that has been built to show this complexity of smart city security is based on the network illustrated in Fig. 9.4. This scenario represents a platform for the urban mobility management in a smart city. The communication network is provided by the smart city which, through its administrators, has complete control over it, and thus can configure it. The service components, instead, can be provided by third-party service providers and thus they are not fully controlled by the smart city management.

In this use case, multiple services are provided to the citizens by the municipality, such as the "public transit" (PT) service, the "traffic light" (TL) service, the "traffic flow" (TF) service. The PT service provides information to citizens about timetables of public transportation means. It relies on several sensors (PTS) distributed in the city. The TL service provides information to the city management personnel about the traffic lights of the city. The TF service monitors the congestion status of the city streets and provides data to the city management. Both TL and TF rely on

corresponding sensors (TFS) distributed in the city. These three services are also connected to the "Smart City Platform" (SCP), which collects data and provides information to citizens and other applications.

In this use case, there are also three additional third-party services: the "bike-sharing" (BS) service, the "car-sharing" (CS) service, and the "e-scooters sharing" (ES) service. These services collect data from several sensors distributed in the city, respectively called BSS, CSS, and ESS. These three services interact with the "Smart Multi-modal Mobility" (SMM), which is in charge of managing the mobility in the city, combining use of public transportation and private sharing companies.

Users in the smart city can access all these services from different access points, e.g., physical multimedia totems distributed in the city, specific locations with information point nodes, their smartphones.

The connectivity security policies that must be enforced in this network are the following ones:

- all access points can communicate with high level services (SCP and SMM);
- SCP can communicate with the 3 municipality services (PT, TL, TF);
- PT, TL, TF can communicate with their respective sensors (PTS, TLS, TFS);
- SMM can communicate with third-party services (BS,CS,ES);
- BS, CS, ES can communicate with their sensors (BSS,CSS,ESS);
- the smartphone network can communicate with third-party services (BS,CS,ES);
- the BS, CS, ES services can communicate one with each other;
- the BSS, CSS, ESS sensors can communicate one with each other.

The VEREFOO approach has been employed to automatically refine these connectivity policies into the security configuration of the smart city communication networks. Given a description of the network as a graph, including the typical network functions such as switches, firewalls, NATs, load balancers, etc, and the aforementioned set of policies about reachability and isolation in the network, VEREFOO established the firewall allocation scheme in the network graph and found their optimal configuration that guarantees that all policies are satisfied.

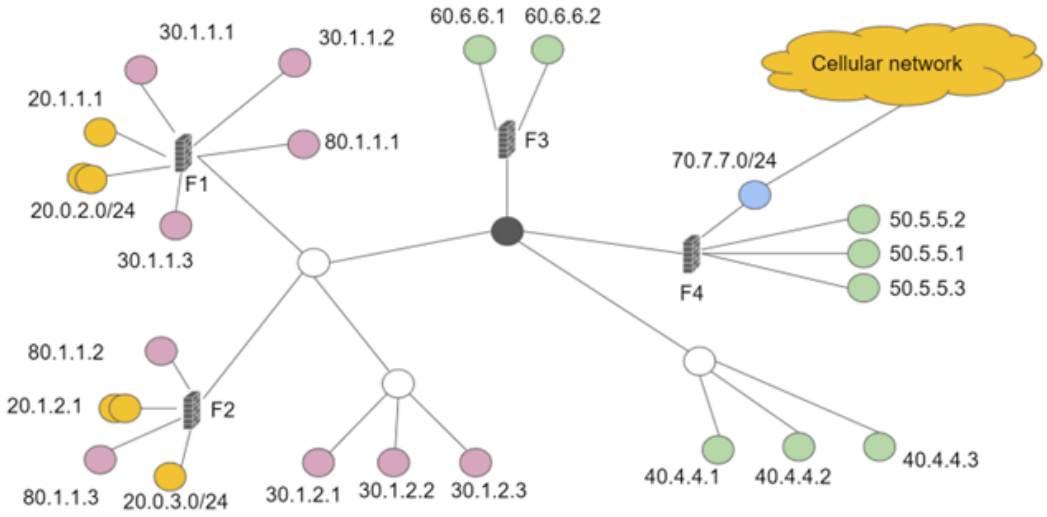


Fig. 9.5 The VEREFOO output for the smart city use case.

In particular, when VEREFOO is applied to this use case, it generates the allocation scheme of the firewalls shown in Fig. 9.5, and their configuration described in Table 9.1. This output is produced according to the objectives of minimizing the number of allocated firewalls and configured rules that were already enunciated in Chapter 5.

Outside the context of the CyberSec4Europe project, the VEREFOO approach has been also integrated with a state-of-the-art SDN orchestrator, ONOS, to solve the security allocation and configuration problems for other types of IoT-aware networks, as documented in [14].

9.4 Final considerations

As discussed in Section 3.3.3, state-of-the-art orchestrating platforms used to address mostly network-oriented problems (e.g., balancing of traffic floods, and assurance for the availability of applications and services). The few security orchestrators that are present in literature mainly embed on ECA-based strategies for the mitigation of cyber attacks, without focusing on how the security service is reorganized after an event triggering their ECA-cased rule engine.

Therefore, the integration of the VEREFOO approach within Docker Compose, Kubernetes, ONOS and a smart city application represents a step forward in the

Firewall F1

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	40.4.4.2	30.1.1.3	∗	∗	∗
2	Allow	30.1.1.1	40.4.4.3	∗	∗	∗
3	Allow	40.4.4.1	30.1.1.2	∗	∗	∗
4	Allow	30.1.1.3	40.4.4.2	∗	∗	∗
5	Allow	40.4.4.3	30.1.1.1	∗	∗	∗
6	Allow	30.1.1.2	40.4.4.1	∗	∗	∗
7	Allow	20.∗.∗.∗	60.6.6.1	∗	∗	∗
8	Allow	60.6.6.1	20.∗.∗.∗	∗	∗	∗
9	Allow	30.1.∗.∗	30.1.∗.∗	∗	∗	∗
8	Allow	80.∗.∗.∗	50.5.5.2	∗	∗	∗
8	Allow	50.5.5.2	80.∗.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall F2

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	80.1.1.2	50.5.5.1	∗	∗	∗
2	Allow	50.5.5.3	80.1.1.3	∗	∗	∗
3	Allow	50.5.5.1	80.1.1.2	∗	∗	∗
4	Allow	80.1.1.3	50.5.5.3	∗	∗	∗
5	Allow	20.∗.∗.∗	60.6.6.1	∗	∗	∗
6	Allow	60.6.6.1	∗.∗.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall F3

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	60.6.6.2	40.∗.∗.∗	∗	∗	∗
2	Allow	70.∗.∗.∗	60.6.6.1	∗	∗	∗
3	Allow	60.6.6.1	20.∗.∗.∗	∗	∗	∗
4	Allow	50.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗
5	Allow	40.4.4.∗	60.6.6.2	∗	∗	∗
6	Allow	∗.∗.∗.∗	50.5.∗.∗	∗	∗	∗
7	Allow	20.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗
8	Allow	60.6.6.1	70.7.7.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall F4

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	70.7.7.∗	40.∗.∗.∗	∗	∗	∗
2	Allow	40.4.∗.∗	70.7.7.∗	∗	∗	∗
3	Allow	50.5.∗.∗	80.1.∗.∗	∗	∗	∗
4	Allow	60.6.∗.∗	∗.∗.∗.∗	∗	∗	∗
5	Allow	∗.∗.∗.∗	∗	60.6.6.1	∗	∗
6	Allow	80.1.∗.∗	50.∗.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Table 9.1 Filtering Policy rules for allocated firewalls

automatic enforcement of security in network orchestration. It shows that existing

frameworks can be extended to manage automatically reaction mechanisms, so as to optimize their operations. This achievement if even more important in cloud infrastructures, such as the ones managed by Kubernetes, where limitations of the policies that can be defined there makes security management an open challenge in general [175].

Besides, the feasibility of this integration proved that the VEREFOO approach can be effectively applied to manage the security of real environments. As both the required inputs and produced outputs simply require a translation step for being exchanged with other modules of the comprehensive orchestrating platform, this means that the medium-level language characterizing the data in the VEREFOO approach really prove all the information that are useful to (re)configure a security service during an orchestration process.

Final Discussion on Network Security Automation

Chapter 10

Conclusions and Future Work

Since the beginning of the last decade, computer networks have been undergoing an incessant revolution, determined by the advent of softwarization technologies (e.g., Network Functions Virtualization and Software-Defined Networking) and the pervasive diffusion of distributed technologies (e.g., Internet of Things, and smart home networking). As part of this ongoing revolution of networking, the size, complexity, and dynamics of modern computer networks are constantly increasing too, ruling out the traditional manual ways of performing network security management. Old manual approaches are becoming unfeasible and unbearable for human operators, who likely make errors and sub-optimizations in performing their tasks and take more time than available in rapidly evolving dynamic networks.

As a possible solution to this problem, this dissertation studied the introduction of automation into network security management. On the one hand, automation reduces the number of human interventions, as automatic approaches just require input specification and assistance during their independent work, so reducing the possibility of human error. On the other hand, as shown in this thesis, automation can be paired with two important features for security, i.e., formal verification, to further improve correctness assurance of the management operations, and optimization, to improve their results from the efficiency point of view. The study illustrated in this thesis started from a state of the art characterized by just few attempts made to introduce automation in network security management, the attention of researchers having been focused much more on the automation of network-related management

problems. Besides, formal verification and optimization had rarely been leveraged to improve automatic security management approaches.

This dissertation contributed to improve the state of art of network security automation by proposing novel formal and optimized approaches related to two main thematic areas: automatic security configuration and automatic security orchestration.

For what concerns automatic security configuration, this dissertation proposed the VEREFOO approach, which combines automation, formal verification and optimization to compute the allocation scheme and configuration rule set of multiple types of network security functions in the logical topology of a virtual computer network. In this approach, the automatic configuration problem has been formalized as a Maximum Satisfiability Modulo Theories problem, thus pursuing correctness by construction, integrating optimization objectives, and avoiding the application of time-consuming a-posteriori formal verification techniques. The VEREFOO approach has been successfully applied to solve the configuration problem for packet filtering firewalls and VPN gateways. The most challenging goal of this work has been the definition of formal models that are detailed enough to solve the problem automatically, but lightweight enough to make the automatic resolution algorithm efficient and scalable to networks of significant size. This was a central objective of this work, as formal verification and optimization are features that may worsen the performance of a technique where they are embedded, if they are not correctly managed. The experimental results confirmed the achievement of this goal and the correctness and efficiency/scalability of the approach.

For what concerns automatic orchestration, this dissertation addressed three main problems. First, it proposed the FATO formal methodology for the optimization of distributed firewall reconfiguration transients, so as to minimize the number of transitory states where undesired intrusions and service disruptions may occur. Second, it described a novel abstraction of network security functions, so as to abstract from their vendor-dependent characteristics and simplify function selection before their deployment in the physical network infrastructure, allowing more optimized results. Third, it discussed how the VEREFOO approach can be integrated with state-of-the-art orchestrators, such as Docker Compose and Kubernetes. The proposed solutions for these three problems jointly concur to improve the comprehensive security orchestration workflow, as they are strictly dependent to each other to introduce automation

in this field. Specifically, an automated security orchestrator based on the above proposals may use the projection abstraction to identify the vendor-independent security functionalities that must be enforced, a (re)configuration mechanism like VEREFOO to allocate and configure them in the logical network topology, an optimized strategy for VNF selection, where only the really required ones to enforce the allocated security projections are chosen, and a transient optimization mechanism like FATO to schedule the deployment of the new configuration in the optimum way. This dissertation investigated the solution of open problems that are essential to make this orchestration process possible. The experiments that have been carried out showed that these security orchestration strategies can cooperate in close synergy with orchestrators oriented to solve networking problems. This achievement represents an important step ahead towards full autonomy in network security.

These results leave space for further research in both research areas. First, the Maximum Satisfiability Modulo Theories formulation is flexible enough to be extended to support other network and security function types, such as web-application gateways, anti-spam filters, intrusion detection systems, stateful network address translators and more. Second, currently the VEREFOO approach can only work on a service graph devoid of network security functions, thus creating the security configuration from scratch even when it is not necessary, e.g., when a distributed firewall is already configured and only some of the user-specified security policies are modified. Therefore, another possible future research direction is the study of an optimized version of the VEREFOO approach, which can manage the reconfiguration of distributed security functions in an optimized way, i.e., by identifying the network components that are affected by the changes, and by using this information to compute the new configuration starting from the previous one rather than starting again from scratch. Third, a pending problem of the Maximum Satisfiability Modulo Theories formulation is that, even if it scales to large networks, it cannot manage the largest networks composed of tens of thousands nodes. Therefore, a heuristic algorithm could be investigated to be used as an alternative strategy. Finally, reaction and mitigation strategies can be investigated for a possible integration with the VEREFOO approach and the network orchestrators, with the aim of making further steps in the direction of full autonomy in network security management.

References

[1] Nick Feamster, Jennifer Rexford, and Ellen W. Zegura. The road to SDN: an intellectual history of programmable networks. *Comput. Commun. Rev.*, 44(2):87–98, 2014.

[2] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Commun. Surv. Tutorials*, 18(1):236–262, 2016.

[3] Gang Mei, Nengxiong Xu, Jiayu Qin, Bowen Wang, and Pian Qi. A survey of internet of things (iot) for geohazard prevention: Applications, technologies, and challenges. *IEEE Internet Things J.*, 7(5):4371–4386, 2020.

[4] Daniele Bringhenti and Fulvio Valenza. A twofold model for VNF embedding and time-sensitive network flow scheduling. *IEEE Access*, 10:44384–44399, 2022.

[5] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *IEEE SDN for Future Networks and Services, SDN4FNS 2013, Trento, Italy, November 11-13, 2013*, pages 1–7. IEEE, 2013.

[6] Dinesh Verma. Simplifying network administration using policy-based management. *IEEE Netw.*, 16(2):20–26, 2002.

[7] David D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 27-29, 1987*, pages 184–195. IEEE Computer Society, 1987.

[8] Raouf Boutaba and Issam Aib. Policy-based management: A historical perspective. *J. Netw. Syst. Manag.*, 15(4):447–480, 2007.

[9] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Serena Spinoso, Fulvio Valenza, and Jalolliddin Yusupov. Improving the formal verification of reachability policies in virtualized networks. *IEEE Trans. Netw. Serv. Manag.*, 18(1):713–728, 2021.

[10] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Automated optimal firewall orchestration and configuration in virtualized networks. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–7. IEEE, 2020.

[11] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Automated firewall configuration in virtual networks. *IEEE Tran. on Dep. and Sec. Comp.*, pages 1–18, 2022. in press.

[12] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza. Short paper: Automatic configuration for an optimal channel protection in virtualized networks. In *CYSARM@CCS '20: Proceedings of the 2nd Workshop on Cyber-Security Arms Race, Virtual Event, USA, November, 2020*, pages 25–30. ACM, 2020.

[13] Daniele Bringhenti and Fulvio Valenza. Optimizing distributed firewall reconfiguration transients. *Comput. Networks*, 215:109183, 2022.

[14] Daniele Bringhenti, Jalolliddin Yusupov, Alejandro Molina Zarca, Fulvio Valenza, Riccardo Sisto, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Automatic, verifiable and optimized policy-based security enforcement for sdn-aware iot networks. *Comput. Networks*, 213:109123, 2022.

[15] Daniele Bringhenti, Fulvio Valenza, and Cataldo Basile. Toward cybersecurity personalization in smart homes. *IEEE Secur. Priv.*, 20(1):45–53, 2022.

[16] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza. A novel approach for security function graph configuration and deployment. In *7th IEEE International Conference on Network Softwarization, NetSoft 2021, Tokyo, Japan, June 28 - July 2, 2021*, pages 457–463. IEEE, 2021.

[17] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Introducing programmability and automation in the synthesis of virtual firewall rules. In *6th IEEE Conference on Network Softwarization, NetSoft 2020, Ghent, Belgium, June 29 - July 3, 2020*, pages 473–478. IEEE, 2020.

[18] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Towards a fully automated and optimized network security functions orchestration. In *2019 4th International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, October 10-12, 2019*, pages 1–7. IEEE, 2019.

[19] Fulvio Valenza. *Modelling and Analysis of Network Security Policies*. PhD thesis, 2017.

[20] Ehab Al-Shaer, Hazem H. Hamed, Raouf Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE J. on Sel. Areas in Commun.*, 23(10), 2005.

[21] Hazem Hamed and Ehab Al-Shaer. Taxonomy of conflicts in network security policies. *IEEE Commun. Mag.*, 44 (3), 2006.

[22] Fulvio Valenza, Cataldo Basile, Daniele Canavese, and Antonio Lioy. Classification and analysis of communication protection policy anomalies. *IEEE/ACM Trans. Netw.*, 25(5):2601–2614, 2017.

[23] Kresimir Popovic and Zeljko Hocenski. Cloud computing security issues and challenges. In *Proc. of the 33rd Inter. Convention MIPRO*, 2010.

[24] Ipsos MORI and University of Portsmouth. Cyber Security Breaches Survey 2018, 2018. Available: https://www.ipsos.com/ipsos-mori/en-uk/cyber-security-breaches-survey-2018, Visited: 2022-10-18.

[25] Oracle Communications. Enterprise Networks in Transition - Taming the Chaos, 2018. Available: http://www.oracle.com/us/industries/communications/enterprise-networks-transition-5073874.pdf, Visited: 2022-10-18.

[26] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S. Yu. A survey of heterogeneous information network analysis. *IEEE Trans. Knowl. Data Eng.*, 29(1), 2017.

[27] Ponemon Institute and IBM Security. Cost of a Data Breach Report 2019, 2019. Available: https://www.ibm.com/security/digital-assets/cost-data-breach-report/, Visited: 2022-10-18.

[28] Ken Goldberg. What is automation? *IEEE Trans. Autom. Sci. Eng.*, 9(1), 2012.

[29] Deloitte. The future of cyber survey 2019, 2019. Available: https://www.marsh.com/us/insights/research/marsh-microsoft-cyber-survey-report-2019.html, Visited: 2022-10-18.

[30] Marsh and Microsoft. 2019 Global Cyber Risk Perception Survey, 2019. Available: https://www.marsh.com/us/insights/research/marsh-microsoft-cyber-survey-report-2019.html, Visited: 2022-10-18.

[31] Enterprise Strategy Group. Network Security Operations Transformation: Embracing Automation, Cloud Computing, and DevOps, 2018. Available: https://www.tufin.com/network-security-operations-esg-research, Visited: 2022-10-18.

[32] Enterprise Strategy Group. 2018 It Spending Intentions Survey, 2018. Available: https://research.esg-global.com/reportaction/2018ITSpendingIntentions/Marketing, Visited: 2022-10-18.

[33] Ponemon Institute and IBM Security. The fourth annual study on the Cyber Resilient Organization, 2019. Available: https://www.techrepublic.com/resource-library/whitepapers/

2019-ponemon-institute-study-on-the-cyber-resilient-organization/, Visited: 2022-10-18.

[34] Donald Norman. The 'problem' with automation: Inappropriate feedback and interaction, not 'over-automation'. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 327, 1990.

[35] Enrique Dávalos and Benjamín Barán. A survey on algorithmic aspects of virtual optical network embedding for cloud networks. *IEEE Access*, 6:20893–20906, 2018.

[36] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecur.*, 2(1):20, 2019.

[37] Seungwon Shin, Phillip A. Porras, Vinod Yegneswaran, Martin W. Fong, Guofei Gu, and Mabry Tyson. FRESCO: modular composable security services for software-defined networks. In *Proc. of the 20th Network and Distributed System Security Symp.*, 2013.

[38] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using SDN. In *Proc. of the ACM SIGCOMM Conf.*, 2013.

[39] Andrey Gushchin, Anwar Walid, and Ao Tang. Scalable routing in sdn-enabled networks with consolidated middleboxes. In *Proc. of the ACM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, 2015.

[40] Nicolas Schnepf, Remi Badonnel, Abdelkader Lahmadi, and Stephan Merz. Rule-based synthesis of chains of security functions for software-defined networks. *ECEASST*, 76, 2018.

[41] Thomas Szyrkowiec, Michele Santuari, Mohit Chamania, Domenico Siracusa, Achim Autenrieth, Victor Lopez, Joo Cho, and Wolfgang Kellerer. Automatic intent-based secure service creation through a multilayer sdn network orchestration. *J. Opt. Commun. Netw.*, 10(4), 2018.

[42] Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, and Lisandro Zambenedetti Granville. Refining network intents for self-driving networks. In *Proc. of the Workshop on Self-Driving Networks (SelfDN18)*, 2018.

[43] Nicolas Schnepf, Remi Badonnel, Abdelkader Lahmadi, and Stephan Merz. Automated factorization of security chains in software-defined networks. In *Proc. of the IFIP/IEEE Int. Symp. on Integrated Network Management (INM19)*, 2019.

[44] Eder J. Scheid, Cristian Cleder Machado, Ricardo Luis dos Santos, Alberto E. Schaeffer Filho, and Lisandro Zambenedetti Granville. Policy-based dynamic service chaining in network functions virtualization. In *IEEE Symp. on Computers and Communication (ISCC16)*, 2016.

[45] Eder J. Scheid, Cristian Cleder Machado, Muriel Figueredo Franco, Ricardo Luis dos Santos, Ricardo J. Pfitscher, Alberto E. Schaeffer Filho, and Lisandro Zambenedetti Granville. Inspire: Integrated nfv-based intent refinement environment. In *Proc. of the IFIP/IEEE Symp. on Integrated Network and Service Management (IM17)*, 2017.

[46] Zheng Hao, Zhaowen Lin, and Ran Li. A sdn/nfv security protection architecture with a function composition algorithm based on trie. In *Proc. of the 2nd Intern. Conf. on Computer Science and Application Engineering (CSAE18)*, 2018.

[47] Woosik Lee and Namgi Kim. Security policy scheme for an efficient security architecture in software-defined networking. *Information*, 8(2), 2017.

[48] Younghee Park, Pritesh Chandaliya, Akshaya Muralidharan, Nikash Kumar, and Hongxin Hu. Dynamic defense provision via network functions virtualization. In *Proc. of the ACM Intern. Workshop on Security in Software Defined Networks & Network Function Virtualization, (SDN-NFVSec17)*, 2017.

[49] Xin Li and Chen Qian. An NFV orchestration framework for interference-free policy enforcement. In *Proc. of the 36th IEEE Intern. Conf. on Distributed Computing Systems, (ICDCS16)*, 2016.

[50] Yi Liu, Hongqi Zhang, Jiang Liu, and Yingjie Yang. A new approach for delivering customized security everywhere: Security service chain. *Sec. and Commun. Netw.*, 2017, 2017.

[51] Yicen Liu, Yu Lu, Wenxin Qiao, and Xingkai Chen. A dynamic composition mechanism of security service chaining oriented to sdn/nfv-enabled networks. *IEEE Access*, 6, 2018.

[52] Alireza Shameli Sendi, Yosr Jarraya, Makan Pourzandi, and Mohamed Cheriet. Efficient provisioning of security service function chaining using network security defense patterns. *IEEE Trans. Services Comput.*, 12(4), 2019.

[53] Andrés F. Ocampo, Juliver Gil-Herrera, Pedro Heleno Isolani, Miguel C. Neves, Juan Felipe Botero, Steven Latré, Lisandro Zambenedetti Granville, Marinho P. Barcellos, and Luciano Paschoal Gaspary. Optimal service function chain composition in network functions virtualization. In *Proc. of the Intern. Conf. on Autonomous Infrastructure, Management, and Security, (AIMS17)*, 2017.

[54] Cataldo Basile, Antonio Lioy, Christian Pitscheider, Fulvio Valenza, and Marco Vallini. A novel approach for integrating security policy enforcement

with dynamic network virtualization. In *Proceedings of the 1st IEEE Conference on Network Softwarization, NetSoft 2015, London, United Kingdom, April 13-17, 2015*, pages 1–5. IEEE, 2015.

[55] Cataldo Basile, Fulvio Valenza, Antonio Lioy, Diego R. Lopez, and Antonio Pastor Perales. Adding support for automatic enforcement of security policies in NFV networks. *IEEE/ACM Trans. Netw.*, 27(2), 2019.

[56] Dhanu Dwiardhika and Takuji Tachibana. Optimal construction of service function chains based on security level for improving network security. *IEEE Access*, 7, 2019.

[57] MyungKeun Yoon, Shigang Chen, and Zhan Zhang. Minimizing the maximum firewall rule set in a network with multiple firewalls. *IEEE Trans. Comput.*, 59(2), 2010.

[58] Mohammad Ashiqur Rahman and Ehab Al-Shaer. Automated synthesis of distributed network access controls: A formal framework with refinement. *IEEE Trans. Parallel Distrib. Syst.*, 28(2), 2017.

[59] Yair Bartal, Alain J. Mayer, Kobbi Nissim, and Avishai Wool. *Firmato*: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4), 2004.

[60] Frédéric Cuppens, Nora Cuppens-Boulahia, Thierry Sans, and Alexandre Miège. A formal approach to specify and deploy a network security policy. In *Proc. of the 2nd IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST)*, 2004.

[61] Joshua D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. of the IEEE Symp. on Security and Privacy*, 1997.

[62] Angelos Keromytis, Kostas Anagnostakis, Sotiris Ioannidis, Michael Greenwald, and Jonathan Smith. Managing access control in large scale heterogeneous networks. In *Proc. of the NATO Consultation, Command and Control Interoperable Networks for Secure Communication Symp.*, 2003.

[63] Pavan Verma and Atul Prakash. FACE: A firewall analysis and configuration engine. In *Proc. of the IEEE/IPSJ Intern. Symp. on Applications and the Internet (SAINT05)*, 2005.

[64] John Govaerts, Arosha K. Bandara, and Kevin Curran. A formal logic approach to firewall packet filtering analysis and generation. *Artif. Intell. Rev.*, 29(3-4), 2008.

[65] Padmalochan Bera, Soumya Kanti Ghosh, and Pallab Dasgupta. Policy based security analysis in enterprise networks: A formal approach. *IEEE Trans. Netw. Service Manag.*, 7(4), 2010.

[66] Nicolas Stouls and Marie-Laure Potet. Security policy enforcement through refinement process. In *Proc. of the 7th Intern. Conf. of B Users, Besançon*, 2007.

[67] Arosha K. Bandara, Antonis C. Kakas, Emil C. Lupu, and Alessandra Russo. Using argumentation logic for firewall configuration management. In *Proc. of the 11th IFIP/IEEE Intern. Symp. on Integrated Network Management*, 2009.

[68] Pedro Adão, Claudio Bozzato, G. Dei Rossi, Riccardo Focardi, and Flaminia L. Luccio. Mignis: A semantic based tool for firewall configuration. In *Proc. of the IEEE 27th Computer Security Foundations Symp.*, 2014.

[69] Dinesha Ranathunga, Matthew Roughan, Phil Kernick, and Nick Falkner. The mathematical foundations for mapping policies to network devices. In *Proc. of the 13th Intern. Joint Conf. on e-Business and Telecommunications*, 2016.

[70] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin T. Vechev. Network-wide configuration synthesis. In *Proc. of the 29th Intern. Conf. on Computer Aided Verification CAV17*, 2017.

[71] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin T. Vechev. Netcomplete: Practical network-wide configuration synthesis with autocompletion. In *Proc. of the 15th USENIX Symp. on Networked Systems Design and Implementation, NSDI18*, 2018.

[72] Chiara Bodei, Pierpaolo Degano, Letterio Galletta, Riccardo Focardi, Mauro Tempesta, and Lorenzo Veronese. Language-independent synthesis of firewall policies. In *Proc. of the IEEE European Symp. on Security and Privacy*, 2018.

[73] Manuel Cheminod, Luca Durante, Lucia Seno, Fulvio Valenza, and Adriano Valenzano. A comprehensive approach to the automatic refinement and verification of access control policies. *Comp. & Sec.*, 80, 2019.

[74] Abhijeet Sahu, Patrick Wlazlo, Nastassja Gaudet, Ana Goulart, Edmond Rogers, and Katherine Davis. Generation of firewall configurations for a large scale synthetic power system. In *Proc. of the IEEE Texas Power and Energy Conference*, 2022.

[75] Arthur Selle Jacobs, Ricardo J. Pfitscher, Rafael Hengen Ribeiro, Ronaldo A. Ferreira, Lisandro Zambenedetti Granville, Walter Willinger, and Sanjay G. Rao. Hey, lumi! using natural language for intent-based network management. In *Proc. of the USENIX Annual Technical Conference*, 2021.

[76] Manel Smine, David Espes, Nora Cuppens-Boulahia, Frédéric Cuppens, and Marc-Oliver Pahl. A priority-based domain type enforcement for exception management. In *Proc. of the Inter. Symp. on Foundations and Practice of Security*, 2021.

[77] Erisa Karafili, Fulvio Valenza, Yichen Chen, and Emil C. Lupu. Towards a framework for automatic firewalls configuration via argumentation reasoning. In *Proc. of the IEEE/IFIP Network Operations and Management Symp.*, 2020.

[78] Dinesha Ranathunga, Matthew Roughan, and Hung X. Nguyen. Verifiable policy-defined networking using metagraphs. *IEEE Trans. Dependable Secur. Comput.*, 19(1), 2022.

[79] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. V. Surendran, and D. M. Martin. Automatic management of network security policy. In *Proc. of the DARPA Information Survivability Conf. and Expos.*, volume 2, 2001.

[80] Mohamed G. Gouda and Alex X. Liu. Firewall design: Consistency, completeness, and compactness. In *Proc. of the 24th Intern. Conf. on Distributed Computing Systems (ICDCS04)*, 2004.

[81] Frédéric Cuppens, Nora Cuppens-Boulahia, and Joaquin Garcia-Alfaro. Detection of network security component misconfiguration by rewriting and correlation. *Conf. on Security in network Architectures and Security of Information Systems*, 2006.

[82] Sruthi Bandhakavi, Sandeep N. Bhatt, Cat Okita, and Prasad Rao. Analyzing end-to-end network reachability. In *Proc. of the 11th IFIP/IEEE Intern. Symp. on Integrated Network Management*, 2009.

[83] Sanjai Narain, Rajesh Talpade, and Gary Levin. *Network Configuration Validation*. 2010.

[84] Nihel Ben Youssef and Adel Bouhoula. A fully automatic approach for fixing firewall misconfigurations. In *Proc. of the 11th IEEE Intern. Conf. on Computer and Information Technology, CIT11*, 2011.

[85] Fei Chen, Alex X. Liu, JeeHyun Hwang, and Tao Xie. First step towards automatic correction of firewall policy faults. *TAAS*, 7(2), 2012.

[86] Aaron Gember-Jacobson, Aditya Akella, Ratul Mahajan, and Hongqiang Harry Liu. Automatically repairing network control planes using an abstract representation. In *Proc. of the 26th Symp. on Operating Systems Principles*, 2017.

[87] Kamel Adi, Lamia Hamza, and Liviu Pene. Automatic security policy enforcement in computer systems. *Comp. & Sec.*, 73, 2018.

[88] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Proc. of the IEEE Intern. Conf. on Network Protocols*, 2012.

[89] Md. Mazharul Islam, Qi Duan, and Ehab Al-Shaer. Specification-driven moving target defense synthesis. In *Proc. of the 6th ACM Workshop on Moving Target Defense, MTD@CCS19*, 2019.

[90] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proc. of the 1st workshop on Hot topics in software defined networks, HotSDN12*, 2012.

[91] Adrian Lara and Byrav Ramamurthy. Opensec: Policy-based security using software-defined networking. *IEEE Trans. Netw. Service Manag.*, 13(1), 2016.

[92] Vijay Varadharajan, Kallol Krishna Karmakar, and Udaya Kiran Tupakula. Securing communication in multiple autonomous system domains with software defined networking. In *Proc. of the IFIP/IEEE Symp. on Integrated Net. and Serv. Manag.*, 2017.

[93] Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid. Transiently policy-compliant network updates. *IEEE/ACM Trans. Netw.*, 26(6), 2018.

[94] V. Varadharajan and U. Tupakula. Counteracting attacks from malicious end hosts in software defined networks. *IEEE Trans. Netw. Service Manag.*, 2019.

[95] Mudassar Hussain, Nadir Shah, and Ali Tahir . Graph-based policy change detection and implementation in sdn. *Electronics*, 8(10), 2019.

[96] Zhi Fu and Shyhtsun Felix Wu. Automatic generation of ipsec/vpn security policies in an intra-domain environment. In *Proc. of the 12th Intern. Workshop on Distributed Systems, DSOM01*, 2001.

[97] Yanyan Yang, Charles U. Martel, and Shyhtsun Felix Wu. On building the minimum number of tunnels: an ordered-split approach to manage ipsec/vpn policies. In *Proc. of the IEEE/IFIP Network Operations and Management Symp.*, 2004.

[98] Yanyan Yang, Zhi (Judy) Fu, and Shyhtsun Felix Wu. BANDS: an inter-domain internet security policy management system for ipsec/vpn. In *Proc. of the IFIP/IEEE 8th Intern. Symp. on Integrated Network Management (IM03),*, 2003.

[99] Chi-Lan Chang, Yun-Peng Chiu, and Chin-Laung Lei. Automatic generation of conflict-free ipsec policies. In *Proc. of the 25th IFIP WG 6.1 Intern. Conf. Formal Techniques for Networked and Distributed Systems - FORTE*, 2005.

[100] Mohammad Mehdi Gilanian Sadeghi, Borhanuddin Mohd Ali, Hossein Pedram, Mehdi Dehghan, and Masoud Sabaei. A new method for creating efficient security policies in virtual private network. In *Proc. of the 4th Intern. Conf. Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 08,*, 2008.

[101] Michael Rossberg, Guenter Schaefer, and Thorsten Strufe. Distributed automatic configuration of complex ipsec-infrastructures. *J. Network Syst. Manage.*, 18(3), 2010.

[102] Salman Niksefat and Masoud Sabaei. Efficient algorithms for dynamic detection and resolution of ipsec/vpn security policy conflicts. In *Proc. of the 24th IEEE Conf. on Advanced Information Networking and Applications*, 2010.

[103] Lotfi Firdaouss, Ayoub Bahnasse, Belkadi Manal, and Yazidi Ikrame. Automated VPN configuration using devops. In *Proc. of the Inter. Conf. on Emerging Ubiquitous Systems and Pervasive Networks*, 2021.

[104] Ricardo Neisse, Gary Steri, and Gianmarco Baldini. Enforcement of security policy rules for the internet of things. In *Proc. of the IEEE 10th Inter. Conf. on Wireless and Mobile Computing, Networking and Communications*, 2014.

[105] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, Cinzia Cappiello, and Alberto Coen-Porisini. Security policy enforcement for networked smart objects. *Comput. Net.*, 108, 2016.

[106] S. Sicari, A. Rizzardi, L.A. Grieco, G. Piro, and A. Coen-Porisini. A policy enforcement framework for internet of things applications in the smart health. *Smart Health*, 3-4, 2017.

[107] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, and Alberto Coen-Porisini. Security towards the edge: Sticky policy enforcement for networked smart objects. *Inf. Syst.*, 71, 2017.

[108] Gokhan Sagirlar, Barbara Carminati, and Elena Ferrari. Decentralizing privacy enforcement for internet of things smart objects. *Comput. Net.*, 143, 2018.

[109] Vasudevan Nagendra, Arani Bhattacharya, Vinod Yegneswaran, Amir Rahmati, and Samir Ranjan Das. An intent-based automation framework for securing dynamic consumer iot infrastructures. In *Proc. of the Web Conference*, 2020.

[110] Mohammad Ashiqur Rahman, Amarjit Datta, and Ehab Al-Shaer. Automated configuration synthesis for resilient smart metering infrastructure. *EAI Endorsed Trans. Security Safety*, 8(28), 2021.

[111] Sian En Ooi, Razvan Beuran, Yasuo Tan, Takayuki Kuroda, Takuya Kuwahara, and Norihito Fujita. Secureweaver: Intent-driven secure system designer. In *Proc. of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2022.

[112] Sébastien Ziegler, Antonio F. Skarmeta, Jorge Bernal Bernabé, Eunsook Eunah Kim, and Stefano Bianchi. ANASTACIA: advanced networked agents for security and trust assessment in CPS iot architectures. In *Proc. of the IEEE Glob. Internet of Things Summit*, 2017.

[113] Alejandro Molina Zarca, Jorge Bernal Bernabé, Ivan Farris, Yacine Khettab, Tarik Taleb, and Antonio F. Skarmeta. Enhancing iot security through network softwarization and virtual security appliances. *Int. J. Netw. Manag.*, 28(5), 2018.

[114] Alejandro Molina Zarca, Jorge Bernal Bernabé, Antonio F. Skarmeta, and Jose M. Alcaraz Calero. Virtual iot honeynets to mitigate cyberattacks in sdn/nfv-enabled iot networks. *IEEE J. Sel. Areas Commun.*, 38(6), 2020.

[115] Alejandro Molina Zarca, Miloud Bagaa, Jorge Bernal Bernabé, Tarik Taleb, and Antonio F. Skarmeta. Semantic-aware security orchestration in sdn/nfv-enabled iot systems. *Sensors*, 20(13), 2020.

[116] Joshua D. Guttman and Amy L. Herzog. Rigorous automated network security management. *Int. J. Inf. Sec.*, 4 (1), 2005.

[117] Tomás E. Uribe and Steven Cheung. Automatic analysis of firewall and network intrusion detection system configurations. *J. of Comp. Sec.*, 15(6), 2007.

[118] Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, and Stere Preda. MIRAGE: A management tool for the analysis and deployment of network security policies. In *Proc. of the 5th Intern. Workshop, Data Privacy Management and Autonomous Spontaneous Security DPM10*, 2010.

[119] Patrick Lingga, Jeonghyeon Kim, Jorge David Iranzo Bartolomé, and Jaehoon Jeong. Automatic data model mapper for security policy translation in interface to network security functions framework. In *Proc. of the IEEE Inter. Conf. on Information and Communication Technology Convergence*, 2021.

[120] Alain J. Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proc. of the Symp. on Sec. and Priv.*, 2000.

[121] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *Proc. of the Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2007.

[122] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. In *Proc. of the 16th ACM SIGPLAN Intern. Conf. on Functional Programming*, 2011.

[123] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. Policycop: An autonomic qos policy enforcement framework for software defined networks. In *Proc. of the IEEE SDN for Future Networks and Services (SDN4FNS)13*, 2013.

[124] Celio Trois, Marcos Didonet Del Fabro, Luis Carlos Erpen De Bona, and Magnos Martinello. A survey on SDN programming languages: Toward a taxonomy. *IEEE Commun. Surveys Tuts.*, 18(4), 2016.

[125] Günter Karjoth, Matthias Schunter, and Michael Waidner. Privacy-enabled services for enterprises. In *Proc. of the 13th IEEE Inter. Work. on Database and Expert Systems Applications*, 2002.

[126] Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent software-defined network updates. *IEEE Commun. Surv. Tutorials*, 21(2):1435–1461, 2019.

[127] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *IEEE SDN for Future Networks and Services, SDN4FNS 2013, Trento, Italy, November 11-13, 2013*, pages 1–7. IEEE, 2013.

[128] Charles C. Zhang, Marianne Winslett, and Carl A. Gunter. On the safety and efficiency of firewall policy deployment. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*, pages 33–50, 2007.

[129] Zeeshan Ahmed, Abdessamad Imine, and Michaël Rusinowitch. Safe and efficient strategies for updating firewall policies. In *Proc. of the 7th Inter. Conf. Trust, Privacy and Security in Digital Business,*, pages 45–57, 2010.

[130] A. Kartit and M. E. Marraki. An enhanced algorithm for firewall policy deployment. In *Proc. of the Inter. Conf. on Multimedia Computing and Systems*, pages 1–4, 2011.

[131] F. Bezzazi, A. Kartit, M. E. Marraki, and D. Aboutajdine. Optimized strategy of deployment firewall policies. In *Proc. of the 2nd Inter. Conf. on the Innovative Computing Technology (INTECH12)*, pages 46–50, 2012.

[132] Zaid Kartit, H. Idrissi, Kartit Ali, and Mohamed El marraki. Improvement of algorithm for updating firewall policies. *Journal of Theoretical and Applied Information Technology*, 66:158–289, 08 2014.

[133] Bruno Astuto A. Nunes, Marc Mendonca, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Commun. Surv. Tutorials*, 16(3):1617–1634, 2014.

[134] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: enabling innovation in network function control. In *Proc. of the ACM SIGCOMM Conf. (SIGCOMM'14)*, pages 163–174, 2014.

[135] Juan Deng, Hongda Li, Hongxin Hu, Kuang-Ching Wang, Gail-Joon Ahn, Ziming Zhao, and Wonkyu Han. On the safety and efficiency of virtual firewall elasticity control. In *Proc. of the 24th Network and Distributed System Security Symp., NDSS*, 2017.

[136] Naga Praveen Katta, Jennifer Rexford, and David Walker. Incremental consistent updates. In *Proc. of the 2nd ACM SIGCOMM Work. on Hot Topics in Software Defined Networking*, pages 49–54, 2013.

[137] Rick McGeer. A correct, zero-overhead protocol for network updates. In *Proc. of the 2nd ACM SIGCOMM Work. on Hot Topics in Software Defined Networking*, pages 161–162, 2013.

[138] Jingyu Hua, Xin Ge, and Sheng Zhong. FOUM: A flow-ordered consistent update mechanism for software-defined networking in adversarial settings. In *Proc. of the 35th IEEE Inter. Conf. on Computer Communications, (INFOCOM16)*, pages 1–9, 2016.

[139] Pavol Cerný, Nate Foster, Nilesh Jagnik, and Jedidiah McClurg. Optimal consistent network updates in polynomial time. In *Proc. of the 30th Inter. Symp. Distributed Computing, DISC16*, pages 114–128. Springer, 2016.

[140] Stefano Vissicchio, Laurent Vanbever, Luca Cittadini, Geoffrey G. Xie, and Olivier Bonaventure. Safe update of hybrid SDN networks. *IEEE/ACM Trans. Netw.*, 25(3):1649–1662, 2017.

[141] Weijie Liu, Rakesh B. Bobba, Sibin Mohan, and Roy H. Campbell. Inter-flow consistency: A novel SDN update abstraction for supporting inter-flow constraints. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 469–478. IEEE, 2015.

[142] Radhika Sukapuram and Gautam Barua. PPCU: proportional per-packet consistent updates for software defined networks. In *24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016*, pages 1–2. IEEE Computer Society, 2016.

[143] Arne Ludwig, Matthias Rost, Damien Foucard, and Stefan Schmid. Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies. In *Proc. of the 13th ACM Work. on Hot Topics in Networks, (HotNets14)*, pages 15:1–15:7, 2014.

[144] Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid. Transiently secure network updates. In *Proc. of the ACM SIGPLAN Inter. Conf. on Measurement and Modeling of Computer Science*, pages 273–284, 2016.

[145] Jedidiah McClurg, Hossein Hojjat, Pavol Cerný, and Nate Foster. Efficient synthesis of network updates. In *Proc. of the 36th ACM Conf. on Programming Language Design and Implementation*, pages 196–207, 2015.

[146] Stefano Vissicchio and Luca Cittadini. FLIP the (flow) table: Fast lightweight policy-preserving SDN updates. In *Proc. of the 35th IEEE Inter. Conf. on Computer Communications, (INFOCOM16)*, pages 1–9, 2016.

[147] Bernd Jäger. Security orchestrator: Introducing a security orchestrator in the context of the ETSI NFV reference architecture. In *Proc. of the IEEE TrustCom/BigDataSE/ISPA*, 2015.

[148] Liang Xia, John Strassner, Cataldo Basile, and Diego R. Lopez. Information model of nsfs capabilities. Rfc, RFC Editor, 2019.

[149] Kostas Giotis, Yiannos Kryftis, and Vasilis Maglaris. Policy-based orchestration of NFV services in software-defined networks. In *Proceedings of the 1st IEEE Conference on Network Softwarization, NetSoft 2015, London, United Kingdom, April 13-17, 2015*, pages 1–5. IEEE, 2015.

[150] Sangwon Hyun, Jinyong Kim, Hyoungshick Kim, Jaehoon Jeong, Susan Hares, Linda Dunbar, and Adrian Farrel. Interface to network security functions for cloud-based security services. *IEEE Commun. Mag.*, 56(1):171–178, 2018.

[151] Alejandro Molina Zarca, Dan García Carrillo, Jorge Bernal Bernabé, Jordi Ortiz Murillo, Rafael Marín-Pérez, and Antonio F. Skarmeta. Enabling virtual AAA management in sdn-based iot networks. *Sensors*, 19(2):295, 2019.

[152] Susan Hares, Diego R. López, Myo Zarny, Christian Jacquenet, Rakesh Kumar, and Jaehoon (Paul) Jeong. Interface to network security functions (I2NSF): problem statement and use cases. *RFC*, 8192:1–29, 2017.

[153] Nathan F. Saraiva de Sousa, Danny Alex Lachos Perez, Raphael Vicente Rosa, Mateus A. S. Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *Comput. Commun.*, 142-143:69–94, 2019.

[154] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.*, 159:102595, 2020.

[155] Preeti Mishra, Emmanuel S. Pilli, Vijay Varadharajan, and Udaya Kiran Tupakula. Intrusion detection techniques in cloud environment: A survey. *J. Netw. Comput. Appl.*, 77:18–47, 2017.

[156] Tao Han, Syed Rooh Ullah Jan, Zhiyuan Tan, Muhammad Usman, Mian Ahmad Jan, Rahim Khan, and Yongzhao Xu. A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers. *Concurr. Comput. Pract. Exp.*, 32(16), 2020.

[157] Montida Pattaranantakul, Ruan He, Qipeng Song, Zonghua Zhang, and Ahmed Meddahi. NFV security survey: From use case driven threat analysis to state-of-the-art countermeasures. *IEEE Commun. Surv. Tutorials*, 20(4):3330–3368, 2018.

[158] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutorials*, 18(2):1153–1176, 2016.

[159] Jun-feng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chen-meng Wang, and Yunjie Liu. A survey of machine learning techniques applied to software defined networking (SDN): research issues and challenges. *IEEE Commun. Surv. Tutorials*, 21(1):393–430, 2019.

[160] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, Khalifa Toumi, and Hervé Debar. Adaptive policy-driven attack mitigation in sdn. In *Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures*, XDOMO'17, New York, NY, USA, 2017. Association for Computing Machinery.

[161] Luiz Fernando Carvalho, Taufik Abrão, Leonardo de Souza Mendes, and Mario Lemes Proença Jr. An ecosystem for anomaly detection and mitigation in software-defined networking. *Expert Syst. Appl.*, 104:121–133, 2018.

[162] Jinyong Tim Kim, Eunsoo Kim, Jinhyuk Yang, Jaehoon Paul Jeong, Hyoungshick Kim, Sangwon Hyun, Hyunsik Yang, Jaewook Oh, Younghan Kim, Susan Hares, and Linda Dunbar. IBCS: intent-based cloud services for security applications. *IEEE Communications Magazine*, 58(4):45–51, 2020.

[163] J. Garay, J. Matias, J. Unzilla, and E. Jacob. Service description in the nfv revolution: Trends, challenges and a way forward. *IEEE Commun. Mag.*, 54(3), 2016.

[164] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[165] Mohamed El Halaby. On the computational complexity of maxsat. *Electron. Colloquium Comput. Complex.*, 2016.

[166] Robert Robere, Antonina Kolokolova, and Vijay Ganesh. The proof complexity of SMT solvers. In *Computer Aided Verification*. Springer International Publishing, 2018.

[167] Guido Marchetto, Riccardo Sisto, Matteo Virgilio, and Jalolliddin Yusupov. A framework for user-friendly verification-oriented VNF modeling. In *Proc. of the 41st IEEE Annual Computer Software and Applications Conference*, 2017.

[168] Girma M. Yilma, Faqir Zarrar Yousaf, Vincenzo Sciancalepore, and Xavier Pérez Costa. Benchmarking open source NFV MANO systems: OSM and ONAP. *Comput. Commun.*, 161:86–98, 2020.

[169] Aaron Paradowski, Lu Liu, and Bo Yuan. Benchmarking the performance of openstack and cloudstack. In *17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2014, Reno, NV, USA, June 10-12, 2014*, pages 405–412. IEEE Computer Society, 2014.

[170] Maciej Kuzniar, Peter Peresíni, and Dejan Kostic. What you need to know about SDN flow tables. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement - 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings*, volume 8995 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2015.

[171] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proc. of the ACM SIGCOMM Conf. (SIGCOMM'12)*, pages 323–334, 2012.

[172] Sahel Sahhaf, Wouter Tavernier, Matthias Rost, Stefan Schmid, Didier Colle, Mario Pickavet, and Piet Demeester. Network service chaining with optimized network function embedding supporting service decompositions. *Comput. Networks*, 93:492–505, 2015.

[173] Ignazio Pedone, Antonio Lioy, and Fulvio Valenza. Towards an efficient management and orchestration framework for virtual network security functions. *Secur. Commun. Networks*, 2019:2425983:1–2425983:11, 2019.

[174] Chuan Pham, Nguyen H. Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach. *IEEE Trans. Serv. Comp.*, 13(1), 2020.

[175] Francesco Minna, Agathe Blaise, Filippo Rebecchi, Balakrishnan Chandrasekaran, and Fabio Massacci. Understanding the security implications of kubernetes networking. *IEEE Secur. Priv.*, 19(5):46–56, 2021.