Automated Firewall Configuration in Virtual Networks

Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov

Abstract—The configuration of security functions in computer networks is still typically performed manually, which likely leads to security breaches and long re-configuration times. This problem is exacerbated for modern networks based on network virtualization, because their complexity and dynamics make a correct manual configuration practically unfeasible. This article focuses on packet filters, i.e., the most common firewall technology used in computer networks, and it proposes a new methodology to automatically define the allocation scheme and configuration of packet filters in the logical topology of a virtual network. The proposed method is based on solving a carefully designed partial weighted Maximum Satisfiability Modulo Theories problem by means of a state-of-the-art solver. This approach formally guarantees the correctness of the solution, i.e., that all security requirements are satisfied, and it minimizes the number of needed firewalls and firewall rules. This methodology is extensively evaluated using different metrics and tests on both synthetic and real use cases, and compared to the state-of-the-art solutions, showing its superiority.

Index Terms—Virtual network security, firewall, automatic security orchestration

1 Introduction

The *Network Functions Virtualization* (NFV) and *Software-Defined Networking* (SDN) paradigms increased agility in network configuration, opening the possibility for users to dynamically request the creation of virtual Service Function Graphs [1], more commonly known as *Service Graph* (SG), generalization of the *Service Function Chain* (SFC) concept. The SG is a new level of abstraction that has been enabled by decoupling computing and physical infrastructures. It is the logical representation of a virtual network, independent from the physical infrastructure where it is embedded [2].

A problem which arises in this context is how to enforce the *Network Security Requirements* (NSRs) that a SG should satisfy, like data protection and isolation. Traditionally, this task is in charge of a security manager, typically different from the network manager who defines the logical topology of the service. This separation of roles, if combined with miscommunication or lack of technical knowledge about the domain field of the other person, can lead to the enforcement of incorrect security controls [3]. Furthermore, the configuration of the *Network Security Functions* (NSFs) is commonly performed manually. This approach not only entails slower reaction when attacks on the network are detected, but it is also prone to human errors, which can lead to the introduction of vulnerabilities. For example, among the hundreds of rules that need to be defined in order to enforce a certain isolation policy, the security manager might miss one, so making isolation not actually effective.

Focusing attention on the control of traffic forwarding, the core NSFs used for this purpose are firewalls. If in traditional scenarios a single point-of-control for packet filtering was usually placed between the local area to protect and the external network where attacks could come from, the flexibility provided by NFV and SDN is encouraging distributed architectures [4], where more instances are allocated between different service functions, thus being able to address more complex security requirements and to improve efficiency and scalability. Designing and managing these complex architectures requires automation, because positioning and configuring virtual instances manually can likely lead to incorrect or non-optimal solutions, in addition to taking excessive time. Instead, automation, paired with formal verification techniques (e.g., [5] or [6]), is the key for computing provably correct and optimized solutions rapidly enough.

Unfortunately, the problem of automating network security configuration has not been sufficiently addressed in the literature so far, despite it represents a key aspect in facing the constantly increasing cybersecurity attacks. Initially, the problem was less pronounced in traditional networks, even though already perceived, because of their intrinsic limitations (e.g., difficulty in allocating multiple firewalls, or creating complex topologies) [7], [8], [9]. Later, the advent of network virtualization sharpened the sensibility for this problem [10], [11], [12], but its high complexity restrained the evolution of the state-of-the-art solutions, which are all partial. In particular, no prior formal method exists to automatically find an optimum allocation and configuration of virtual firewalls in a given SG. Consequently, automated allocation and configuration of distributed firewalls is still an open research issue.

In this context, this article proposes a new methodology that addresses the open issues. The goal is to provide an automatic

• *Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, and Fulvio Valenza are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10138 Torino, Italy. E-mail: {daniele.bringhenti, guido.marchetto, riccardo.sisto, fulvio.valenza}@polito.it.*
• *Jalolliddin Yusupov is with the Department of Automatic Control and Computer Engineering, Turin Polytechnic University, Tashkent 100095, Uzbekistan. E-mail: jaloliddin.yusupov@polito.uz.*

Manuscript received 25 Mar. 2021; revised 2 Feb. 2022; accepted 13 Mar. 2022. Date of publication 17 Mar. 2022; date of current version 14 Mar. 2023.
This work was supported in part by EU H2020 Projects ASTRID under Grant Agreement No. 786922 and in part by CyberSec4Europe under Grant Agreement No. 830929.
(Corresponding author: Fulvio Valenza.)
Digital Object Identifier no. 10.1109/TDSC.2022.3160293

way to allocate packet filters – the most common and traditional firewall technology – in a SG defined by the service designer, and to create firewall rules automatically, so as to satisfy the specified security requirements. The method is based on a formal model which provides assurance that the final solution really satisfies the security requirements (correctness-by-construction). Optimality represents another core aspect of our approach: minimizing the number of firewalls to be allocated and the number of rules to be configured in each one of them increases the performance of the overall architecture, while reducing its cost in terms of employed resources. All this is achieved by a careful formulation of the problem as a *partial weighted Maximum Satisfiability Modulo Theories* (MaxSMT) problem, which can be solved automatically, providing a solution, if one exists, that is formally guaranteed to satisfy all the hard logical constraints defined in the problem, and that is an optimum one, according to the optimization criteria defined in the problem. Our preliminary ideas about the proposed approach were presented at the NOMS 2020 conference [13]. However, in that paper, we showed just a naive solution, and without full details. Instead, this article is a comprehensive presentation of a deep re-thinking of our preliminary ideas, providing the following main new contributions:

- The way traffic flows, network functions, and network security requirements are formalized in the MaxSMT problem is totally new with respect to the preliminary ideas. The new models have been redefined in terms of flows, rather than packets, with the goal of achieving better performance and scalability. All models and algorithms are described with full technical details.
- The experimental validation has been performed much more extensively than in [13], by testing the method not only with synthetic topologies, but also with real topologies of production networks. Compared to the preliminary idea, the new framework shows greatly increased performance.
- A more extensive survey of the related literature has been carried out, and a different, more articulated, clarifying example is used here to better highlight the advantages of our proposal with respect to alternative solutions or manual strategies.

The remainder of this article is structured as follows. Section 2 presents related work. Section 3 describes the proposed methodology. Section 4 gives the details about the formal models of the network and NSRs. Section 5 presents an algorithm for the computation of the traffic flows to consider for the enforcement of the NSRs, while Section 6 presents the formalization of the firewall allocation and configuration problem as MaxSMT. Section 7 shows the results of our evaluations. Finally, Section 8 draws conclusions.

2 Related Work

The central novelty of the approach proposed in this article is that, to the best of our knowledge, it is the first methodology that performs both allocation and configuration of packet filter firewalls in a user-provided virtualized SG, achieving all the three key requirements we identified, i.e., full automation, optimization, and formal correctness assurance. Previous related approaches can be categorized into two classes: those that address automatic computation of firewall configurations *for a user-provided firewall allocation scheme*, discussed in detail in Section 2.1, and those that perform automatic synthesis or refinement of a SG, but *without synthesizing an optimized set of firewall configuration rules*, discussed in detail in Section 2.2. The idea developed in this article, of solving the firewall allocation and configuration problems jointly in an optimized and formal way, was presented for the first time in our preliminary conference article [13]. With respect to that version, the method presented here has been enhanced as already discussed in Section 1.

2.1 Automatic Firewall Configuration

The existing approaches for automatic firewall rule configuration for a given allocation scheme lack either formal correctness guarantee, or full automation, or optimization. Moreover, some of them cannot even be applied to virtualized networks. Articles [7], [8] and [9] represent the milestones for this research area, because they proposed for the first time the idea that firewall rule sets can be automatically computed by refining high-level requirements. However, as their publication date suggests, they were designed to work in traditional networks, without support for modern software networking. Moreover, they lack optimization and formal correctness assurance (only [7] provides a formally verified global filtering policy, as long as it is, however, enriched with an external verification tool [14]).

Formal verification has been introduced in this field by some subsequent articles ([11], [15], [16], [17], [18]), which underlined the importance of formal correctness assurance for the automatically computed configurations. Formal verification of firewall configurations has recently become a vital requirement for critical environments, as underlined in [6]. All these approaches miss some of the other features characterizing our approach. [15] and [16] do not include optimization, being content with the accomplishment of a formally correct firewall configuration. Moreover, a further limitation of [15] is that, differently from our approach, it cannot configure generic firewall implementations, but only firewalls based on IPChains and Cisco's PIX syntax. For what concerns the other three articles ([17], [18] and [11]), their main limitation is that they cannot configure firewalls from scratch, since they are approaches for fixing firewall misconfigurations. For this reason, they do not achieve full automation, but they require that someone provides an initial rule set. Moreover, they achieve less optimization goals than our approach: [17] and [18] only minimize rule set cardinality, while [11] only minimizes the number of functions to be updated. Additionally, [17] and [18] focus on traditional networks only, while [11] works at a higher level of abstraction, neglecting full low-level configuration information.

Other existing approaches solve less general or different problems, such as firewall rule generation for a specific architecture of industrial control networks [19], or firewall rule distribution across firewall VNFs using heuristics [20].

2.2 Automatic SG or SFC Synthesis or Refinement

The methodology proposed in this article addresses the important challenges of NFV-based operational resilience

[21], and intent-based networking [22], by automatically enriching a user-defined SG with a distributed firewall allocation and configuration that satisfies the user-defined security policies. In literature, instead, most of the related work focuses on designing a full service (SG or SFC), including all network functions, but leaving the computation of firewall rules out. Such an approach does not cover typical situations, e.g., a service designer who wants to define the SG manually and enrich it automatically with security functions, or the case of a SG that already exists, for which a new security policy has to be enforced. Moreover, as it does not compute firewall rules, this approach only solves half of the problem. The most prominent articles taking this approach are [23], [24], [25], [26], and [27]. In other cases, a similar approach is used, but using techniques different from firewalls (e.g., SDN [28] or network slicing [29]). In addition to the just highlighted main limitations of all these approaches, none of them is founded on a formal approach, and only some of them target optimization.

The approach proposed in [10] is slightly different, because it starts from a user-provided SG, which is enriched by allocating different types of NSFs (firewalls, VPN gateways, IDSs, NATs), selected and positioned in a formal and optimized way to satisfy the user NSRs. However, differently from our approach, the SG is composed of only end points and routers, without other middleboxes, and a totally different optimization problem is solved, considering security requirements as relaxable, and looking for an optimum trade-off among level of security achieved, cost of the allocated NSFs, and usability. Moreover, as in the previous cases, the rules for the allocated firewalls are not computed.

Among the few approaches that also consider firewall rules, [30] starts from a set of isolation/reachability NSRs referred to a set of endpoints, and it synthesises a SG made only of firewalls interconnecting end points, minimizing the number of rules in the allocated firewalls, under the assumption that each NSR is a firewall rule that will be installed on all the firewalls allocated on the path from the NSR source to the NSR destination. Instead, [31] and [12] synthesize SFCs of NSFs, including firewalls, for which they also generate the filtering rules. However, they cannot allocate firewalls in a user-provided SG including other middleboxes, and they cannot minimize the number of rules. Moreover, [12] is not based on a formal model, while [31] does not consider user-specified NSRs, but its goal is to filter anomalous traffic that could correspond to attacks.

This analysis confirms that our approach is the first feature-complete methodology that automatically refines an existing SG, composed of multiple function types, possibly including functions that can modify packets such as NATs and load balancers, allocating the minimum number of firewalls that are necessary to enforce the requested NSRs, and computing the optimal rule set for each allocated firewall in a formal way.

3 Approach

The optimal allocation of packet filter virtual instances in a provided SG and the auto-configuration of their rules, compliant with the provided NSRs, is achieved by solving a partial weighted MaxSMT problem. This section provides a high-level overview of the proposed methodology. First, the inputs and outputs of the method are described: the SG in Section 3.1, the NSRs in Section 3.2, and the expected outcome in Section 3.3. Then, a clarifying example is presented in Section 3.4. Finally, the problem formalization approach and the challenges arising in its definition are presented in Section 3.5, before providing, in the next sections of the article, the detailed and formal problem definition.

3.1 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. The functions do not need to be positioned in a linear combination, as in a SFC, but they can be organized in a complex architecture with multiple traffic paths from sources to destinations. A 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 a SG are *Network Functions* (NFs) implementing various functionalities, such as web caching and load balancing. 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 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 [2].

In our work, we consider a typical catalog of NFs available for SG construction, and these NFs are modeled considering that what matters for firewall configuration is only their forwarding behavior, rather than their full behavior, i.e., what is relevant is the *possibility* that a traffic flow is forwarded by a NF to its next hops, and how its relevant parts are modified, rather than exactly each operation done by the function.

The SG provided by the designer 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 tool can decide to put a firewall in order to reach the optimal allocation scheme.

However, a security skilled service designer can either force the allocation of a firewall in a specific AP without allowing a further removal by the tool, or prohibit to consider specific APs as valid firewall 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 tool must search to solve the problem. Moreover, it is useful in mixed scenarios, where firewalls are implemented not only by VNFs, but also by existing hardware packet filters, which can, in fact, be modeled as firewalls that cannot be removed by the tool. Despite these benefits, on the other hand, it is evident how this manual contribution to the configuration of the 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.

3.2 Input: Network Security Requirements

Among all the network security property types that can be defined, the focus of our methodology is on *connectivity* requirements between pairs of end points. Four different approaches are provided to the service designer for specifying the required security constraints; each one is characterized by a *default behavior*, describing how the traffic flows for which no specific requirements are formulated must be managed, and a set of specific NSRs, exclusively referred to certain types of traffic. In the first approach (*whitelisting*), the default behavior is set to block traffic flows and the user can only additionally specify *reachability* requirements, so that all traffic flows must be blocked except for those that the user explicitly allows. In the second approach (*blacklisting*), vice versa, the default behavior is set to allow traffic flows and the user can only additionally specify *isolation* requirements; in this case, all traffic flows must be allowed with the exception of those that the user specifically requests to deny.

The other two approaches, called *rule-oriented specific* and *security-oriented specific*, let the user explicitly formulate both isolation and reachability specific properties, but without manually setting a default behavior. The way the system 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* approach, the system allows only the communications that are strictly necessary in order to satisfy all user requirements. In these last two approaches, the set of the user-provided NSRs is assumed to be anomaly-free (i.e., with no conflicts and no suboptimizations). This is not a restriction because anomalies can be eliminated by means of well-known anomaly analysis techniques ([32], [33]).

In all the four approaches, the specific NSRs are expressed with a medium-level language [12], i.e., by specifying the IP 5-tuple of the flows that are allowed or prohibited. However, it is possible for an administrator to specify the NSRs in an easier and more flexible high-level language, since from that definition the corresponding medium-level NSRs can be derived with a mere translation [12].

3.3 Expected Outcome

After receiving the pre-processed SG and the NSRs, the automatic security enforcement problem is solved.

In case of positive outcome, the provided result is composed of (i) the allocation scheme of the distributed firewall instances in the SG; (ii) the *Filtering Policy* (FP) for each allocated firewall. The firewall allocation scheme specifies the APs where a firewall instance has to be allocated. The FP for each allocated firewall is a set of filtering rules that make up its configuration, expressed in a user-friendly abstract language, which is independent of specific firewall configuration languages, but fairly general. A firewall FP is composed of a default action – *whitelisting* if the firewall drops any packet unless differently specified, *blacklisting* if the firewall lets the packets pass through unless differently specified – and an anomaly-free set of auto-generated rules specifying how to manage specific kinds of traffic flows.

The allocation scheme contains the minimum number of firewall instances necessary to enforce all NSRs, so minimizing resource consumption, while the FP for each allocated firewall contains the minimum number of configured rules, thus minimizing the amount of memory needed to store them and maximizing the firewall performance.

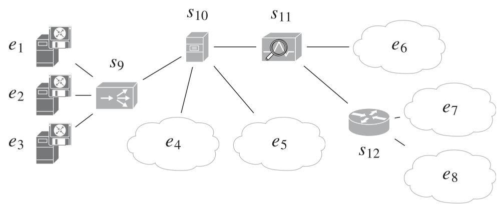


Fig. 1. Original service graph without firewalls.

It is important to note that 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 [34] (i.e., understanding how the virtual functions are placed in the physical hosts).[1] The VNE problem for virtual firewalls has been already addressed in studies such as [35], where optimized strategies for embedding functions in the hardware have been deeply investigated. Similarly, the translation of the abstract FPs into concrete configuration rules for specific packet filter functions is another necessary step that can be executed by well-known methodologies [7].

Hence, the output solution can be deployed automatically into the virtual network by means of existing technologies. After the deployment, in case a new security configuration becomes necessary, e.g., to react to an attack that has just been detected, new NSRs have to be defined by the administrator and provided to the tool, which can then automatically determine the new configuration to be deployed.

Instead, if no solution to the problem can be found, a non-enforceability report is generated to the user, who can try to guess why it has not been possible to enforce the NSRs. A possible reason for a negative outcome can be that the SG defined by the service designer does not provide adequate APs for the firewalls 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.

3.4 Clarifying Example

The most relevant features of our approach can be clarified by means of a sample scenario. This case study is also useful to explain the advantages of our methodology with respect to manual configuration or state-of-the-art automated approaches.

Let us consider Fig. 2, which is the AG generated from the input SG shown in Fig. 1, supposing that the service designer did not set any allocation constraint. In these figures – and in all the next ones –, symbol a is used to denote the APs automatically generated in the AG, while each undirected link represents two adjacent directed links. Table 1 shows the IP addresses and the function type of each SG node, where a node can also be a subnetwork. Table 2 lists the NSRs to satisfy, defined through a *security-*

1. Even though the VNE problem is not faced in the proposed approach, solving the FW allocation problem in the SG is itself complex and challenging.

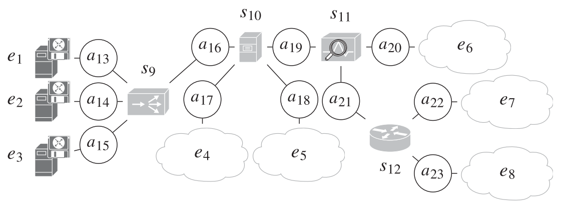


Fig. 2. Allocation graph with allocation places.

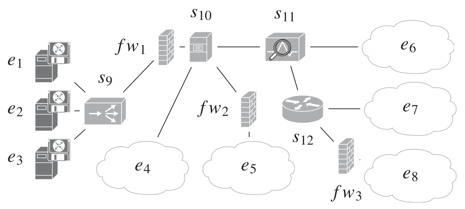


Fig. 3. Final service graph with allocated firewalls.

TABLE 1
IP Addresses and Function Types

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 2
Network Security Requirements

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.$*$	$*$	$\neq$80	TCP
Deny	192.168.1.$*$	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.$*$	$*$	$\neq$80	TCP
Deny	40.40.41.$*$	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 3
Filtering Policy Rules for Allocated Firewalls

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	$*.*.*.*$	$*.*.*.*$	$*$	$*$	$*$

oriented specific approach, where the symbol $*$ has the usual meaning of wildcard.

Our approach takes all the decisions to enforce the requested NSRs automatically, correctly, and optimally, producing the solution shown in Fig. 3 (allocation) and in Table 3 (configuration). Instead, a manual approach is prone to both human errors and sub-optimizations. For example, human beings may fail to understand that placing 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 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 administrator has decided to allocate a firewall in AP a_{16} with default action deny. In order to satisfy the NSR 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 3. These issues increase dramatically when the network topology size and the number of NSRs increase.

Even the adoption of simple heuristic strategies that try to trade optimization for correctness is not immune from these problems. For example, a possible simple unoptimized strategy that could be adopted is to allocate a firewall in each AP, with *deny* default action, and one allow rule for each reachability NSR. However, because of the presence of the NAT and of the load balancer, these rules cannot be just the original NSRs, but the addresses need to be modified in the correct way. This solution is also the worst one from the optimization point of view, because it allocates the maximum number of firewalls and installs the maximum number of rules. Another possible strategy, which yields a more optimized result, is to use firewalls with *allow* default actions, and, for each isolation NSR, allocate a firewall in each AP that is closest to the source specified by the NSR, with a rule that enforces the NSR. In our example, this strategy would lead to allocate firewalls in APs a_{17}, a_{18}, a_{20}, a_{22}, and a_{23}, with a total of 8 specific rules installed, which is a solution more optimized than the previous one, but still less than the optimum solution found by our approach, which just allocates 3 firewalls including a total of 6 specific rules. Moreover, even in this case, the rules have to be synthesized taking into account the presence of NAT and load balancer, which can lead to human errors. Finally, this solution does not minimize permissions, as our solution does, because it is based on a blacklisting approach.

Another possible way that could be tried to solve the problem is to use a combination of the automated state-of-the-art methods that are available to solve the allocation or configuration problem. However, even a combination of existing methods could not solve the problem for this use case automatically, because they work under assumptions that are either more restrictive than or totally different from the ones we are making in this use case. As discussed in Section 2, there is no refinement approach that can refine user-provided SGs including functions that can modify packets (e.g., NATs modifying the IP addresses), while in the topology used for the case study, s_9 and s_{12} are functions of this type.

Let us analyze now what happens when the NSRs cannot be enforced. Let us suppose, for example, that the user imposes an allocation constraint, specifying that APs a_{17} and a_{18} cannot host any firewall. In this case, the solver fails to solve the MaxSMT problem, and a non-enforceability report is returned. At this point, the user may try to change the allocation constraints on the SG, or the definition of the NSRs, so that a next run of the automated procedure can be successful.

In conclusion, the approach proposed in this article can reach the optimal solution automatically, with a complete overview on both the SG and the NSRs, saving time and resources, and avoiding the errors that could be made manually.

3.5 Problem Formulation and Modeling

The formally correct and optimal solution for the firewall allocation and auto-configuration problem is obtained by formulating and solving a partial weighted MaxSMT problem. As a generalization of the traditional SMT problem (which consists of determining if it is possible to satisfy at the same time all the first-order logic constraints in a given set), this problem introduces optimization by distinguishing between two different sets of input constraints: hard clauses and soft clauses. Hard clauses are not relaxable, i.e., they have to be satisfied in order to get a solution of the problem; their presence also determines the *partiality* of the optimization problem, since they are not subject to optimization. On the other hand, soft clauses are assigned a weight – whence the adjective *weighted* derives – and their satisfaction is not strictly required, but it is subject to the optimization objective of maximizing the sum of the weights assigned to the satisfied relaxable clauses. From now on, for simplicity, the term MaxSMT will be used to refer to partial weighted MaxSMT.

In terms of computational complexity, MaxSMT is *NP-complete* [36]. Nevertheless, despite this discouraging worst-case complexity, the state-of-the-art solvers can solve many instances of this problem in polynomial time with respect to the problem dimension [37]. Consequently, a convenient formulation of the problem is crucial to achieve scalability.

The MaxSMT formulation is key to achieve all the three main objectives of the proposed 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 to perform 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 requirements. For this reason, the next sections of the article provide the full definition of the MaxSMT problem starting from the definition of the models of network components and security requirements (Section 4). Our modeling approach is based on the concept of maximal traffic flows. The way these flows can be computed is illustrated in Section 5. Finally, the additional soft and hard constraints required for computing the optimum firewall allocation scheme and configuration, and for modeling the filtering behavior of automatically configured firewalls, are presented in Section 6.

4 Network and Requirements Model

This section defines the formal modeling of the following components: the SG and AG in Section 4.1, the traffic and network functions in Section 4.2, the traffic flows in Section 4.3, and the NSRs in Section 4.4. These models differ substantially from the one adopted in [13], where individual packets were modeled instead of traffic flows. Table 4 includes the main formal notations (symbols, functions, predicates, operators) used in the next sections.

4.1 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 middleboxes (i.e., 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. We define $index(n_k) = k$. Moreover, each $n_k \in N_S$ 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,

TABLE 4
Notation

Symbol/Function/Predicate/Operator	Definition
$\mathbb{B}$ = {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
$\mathcal{I}_i^d,\ \mathcal{I}_i^a$	packets that are dropped/allowed in n_i
$\mathcal{D}_{i,j}$	traffic that is transformed by $\mathcal{T}_{i,j}$
$f \in F$	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 = {DENY, 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: $N_S \rightarrow 2^I$	maps $n \in N_S$ to its set of IP addresses
$\mathcal{T}_i$: $T \rightarrow T$	maps an input traffic to the corresponding output traffic
$\mathcal{T}_{i,j}$: $T \rightarrow T$	maps part of $\mathcal{D}_{i,j}$ to the corresponding output traffic
π: $F \rightarrow (N_A)^*$	maps a flow to the ordered list of nodes that are crossed by that flow
τ: $F \times N_A \rightarrow T$	maps a flow and a node to the ingress traffic
ν: $N_A \times F \rightarrow N_A + \{n_0\}$	maps a network node n and a traffic flow f to the next node crossed by f after n
$\sigma : A_A \rightarrow \mathbb{Z}$	model the sign of the weights
allocated: $N_A \rightarrow \mathbb{B}$	true $\Leftrightarrow$ a FW is allocated in a_h
forbidden: $L_S \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the creation of an AP on $l_{i,j}$ prohibited
forced: $L_S \rightarrow \mathbb{B}$	true $\Leftrightarrow$ allocation of a firewall on $l_{i,j}$ is required
deny$_i$: $T \rightarrow \mathbb{B}$	true $\Leftrightarrow$ n_i drops all the packets
wlst: $A_A \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the def. act. of FW allocated in the AP is DENY
enforces: $A_T \times R \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the FW def. act. enforces requirement r
configured: $P_h \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the placeholder rules included in U_h
matchAll: $P_h \times Q \rightarrow \mathbb{B}$	true $\Leftrightarrow$ the rule conditions completely match the 5-tuple
matchNone: $P_h \times Q \rightarrow \mathbb{B}$	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)

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 $n_k \in E_A \cup S_A$, *allocated*(n_k) is true by definition, whereas for each $a_h \in A_A$, the automatic procedure decides whether $allocated(a_h)$ 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 $l_{i,j} \in L_S$, $forbidden(l_{i,j})$ is true if and only if the creation of an AP on $l_{i,j}$ has been prohibited, while $forced(l_{i,j})$ is true if and only if the allocation of a firewall on $l_{i,j}$ has been required. The constraint (1) expresses that the two requirements cannot coexist for the same $l_{i,j}$

$$\forall l_{i,j} \in L_S. \neg(forbidden(l_{i,j}) \wedge forced(l_{i,j})). \quad (1)$$

According to the requirements expressed by the *forbidden* predicate, A_A and L_A are computed as the smallest sets that satisfy the conditions (2) and (3)

$$\forall l_{i,j} \in L_S.(\neg forbidden(l_{i,j}) \Rightarrow a_h \in A_A \wedge l_{i,h} \in L_A \wedge l_{h,j} \in L_A) \quad (2)$$

$$\forall l_{i,j} \in L_S.(forbidden(l_{i,j}) \Rightarrow l_{i,j} \in L_A). \quad (3)$$

According to (2), for each SG edge $l_{i,j}$, if the creation of an AP on it is not prohibited, i.e., if $forbidden(l_{i,j}) = \text{false}$, 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 (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, (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

$$\forall l_{i,j} \in L_S.\ (forced(l_{i,j}) \Rightarrow allocated(a_h)). \quad (4)$$

4.2 Traffic and Network Functions Model

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 $q_{t,1} \vee q_{t,2} \vee \cdots \vee q_{t,n}$, 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

$$q_{t,i} = (IPSrc, IPDst, pSrc, pDst, tPrt), \quad (5)$$

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 $\mathcal{I}_i^d$ specifies the packets that are dropped by the function, i.e., $deny_i(t)$ is true if and only if $t \subseteq \mathcal{I}_i^d$. Instead, the traffic $\mathcal{I}_i^a$ is the complement of $\mathcal{I}_i^d$, since it specifies the packets that are allowed (i.e., not dropped) by the function. Clearly, $\mathcal{I}_i^d \vee \mathcal{I}_i^a = \text{true}$ and $\mathcal{I}_i^d \wedge \mathcal{I}_i^a = \text{false}$.

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

One may argue that function $\mathcal{T}_i$ alone would be enough, e.g., by setting $\mathcal{T}_i(t) = t^0$ for all t such that $deny_i(t)$ is true. However, keeping these two functions distinct and independent brings some advantages to our framework. 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 $\mathcal{T}_i(t) = t$ (i.e., $\mathcal{T}_i$ 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, we can 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, $\mathcal{T}_i$ is the identity function, and the following constraint is applied to the *deny* predicate:

$$deny_i(t) = \text{false}. \tag{6}$$

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).[2] 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 $\mathcal{T}_i(t) = \vee_j(\mathcal{T}_{i,j}(\mathcal{D}_{i,j} \wedge t))$, where $\mathcal{T}_{i,j}$: $T \rightarrow T$ is the transformer applied for the packet class defined by predicate $\mathcal{D}_{i,j}$. In the case of NAT, we have $\mathcal{T}_i(t) = \mathcal{T}_{i,1}(\mathcal{D}_{i,1} \wedge t) \vee \mathcal{T}_{i,2}(\mathcal{D}_{i,2} \wedge t) \vee \mathcal{T}_{i,3}(\mathcal{D}_{i,3} \wedge t)$, where $\mathcal{T}_{i,1}$ is the shadowing transformer, $\mathcal{T}_{i,2}$ is the reconverting transformer and $\mathcal{T}_{i,3}$ is the identity transformer that is applied in all other cases. Let us denote $p_1, \ldots, p_m$ the predicates representing the shadowed IP addresses and $a_1, \ldots, a_l$ the predicates representing the public IP addresses of the NAT. Then, considering a generic traffic $t = \vee_{k=1}^{h}(q_{t,k})$, the predicates $\mathcal{D}_{i,j}$ and the transformers $\mathcal{T}_{i,j}$ can be defined as follows:

$$\mathcal{D}_{i,1} = \vee_{x=1}^{m}(p_x, \neg(\vee_{z=1}^{m}(p_z)), *, *, *) \tag{7}$$

$$\mathcal{T}_{i,1}(t) = \vee_{y=1}^{l}\vee_{k=1}^{h}(a_y, q_{t,k}.IPDst, q_{t,k}.pSrc, q_{t,k}.pDst, q_{t,k}.tPrt) \tag{8}$$

$$\mathcal{D}_{i,2} = \vee_{y=1}^{l}(\neg(\vee_{x=1}^{m}(p_x)), a_y, *, *, *) \tag{9}$$

$$\mathcal{T}_{i,2}(t) = \vee_{x=1}^{m}\vee_{k=1}^{h}(q_{t,k}.IPSrc, p_x, q_{t,k}.pSrc, q_{t,k}.pDst, q_{t,k}.tPrt) \tag{10}$$

$$\mathcal{D}_{i,3} = \neg(\mathcal{D}_{i,1}) \wedge \neg(\mathcal{D}_{i,2}) \tag{11}$$

$$\mathcal{T}_{i,3}(t) = t. \tag{12}$$

4.3 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 $n_a, n_b, \ldots \in N_A$ 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 $[n_s, t_{s,a}, n_a, t_{a,b}, n_b, \ldots, n_k, t_{k,d}, n_d]$, 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., $\forall t_{i,j} \neq t_{s,a}.\ t_{i,j} = \mathcal{T}_i(t_{k,i})$, 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

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

$\tau(f,n) = t^0$; 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. 2. 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:

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

$$\begin{aligned} t_{12,21} &= t_{21,11} = t_{11,19} = t_{19,10} = t_{10,16} = t_{16,9} \\ &= (220.124.30.1, 130.10.0.4, *, 80, \mathrm{TCP}) \end{aligned}$$

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

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 $\forall n \in \pi(f_1).\tau(f_1, n) \subseteq \tau(f_2, n)$.

4.4 Network Security Requirements Model

The NSRs for a SG include a default behavior D, which is an element of the set {blacklisting, whitelisting, rule-oriented-specific, security-oriented-specific}, and a set of specific NSRs, R_s. Each $r \in R_s$ is expressed in medium-level language as a pair $r = (C, a)$. C is a condition and a is the action that must be performed on the flows that satisfy C.

Each condition C is a predicate similar to the ones defined for modeling packet classes, i.e., $(C = \mathit{IPSrc}, \mathit{IPDst}, \mathit{pSrc}, \mathit{pDst}, \mathit{tPrt})$. The predicates *IPSrc* and *pSrc* specify the traffic sources the requirement refers to. Instead, the predicates *IPDst*, *pDst*, and *tPrt* specify the traffic destinations and the protocol the requirement refers to. A flow $f = [e_s, t_{s,a}, \ldots, t_{k,d}, e_d]$ 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., $\alpha(e_s) \subseteq C.\mathit{IPSrc}$ and $\alpha(e_d) \subseteq C.\mathit{IPDst}$; 2) its source traffic satisfies *IPSrc* and *pSrc*, i.e., $t_{sa} \subseteq (C.\mathit{IPSrc}, *, C.\mathit{pSrc}, *, *)$; 3) its destination traffic satisfies *IPDst*, *pDst*, and *tPrt*, i.e., $t_{kd} \subseteq (*, C.\mathit{IPDst}, *, C.\mathit{pDst}, C.\mathit{tPrt})$. Let then $F_r \subseteq F$ denote the set of flows that satisfy $r.C$.Therefore, 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 $A_T = \{\mathrm{DENY}, \mathrm{ALLOW}\}$. If $r.a = \mathrm{DENY}$, we say r is an isolation requirement, meaning that all flows that satisfy $r.C$ are blocked, i.e., their destination is reached by no packet of the flow, otherwise we say it is a reachability requirement, meaning that at least one flow that satisfies $r.C$ is allowed to reach its destination. R_s is assumed to be conflict-free.

Formally, an isolation requirement r can be expressed as

$$\forall f \in F_r.\exists i.(n_i \in \pi(f) \wedge \mathit{allocated}(n_i) \wedge \mathit{deny}_i(\tau(f, n_i))), \tag{13}$$

while a reachability requirement r can be expressed as

$$\exists f \in F_r.\forall i.(n_i \in \pi(f) \wedge \mathit{allocated}(n_i) \Rightarrow \neg \mathit{deny}_i(\tau(f, n_i))). \tag{14}$$

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. 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 $r.a = \mathrm{ALLOW}$ if $D = \text{blacklisting}$ and $r.a = \mathrm{DENY}$ if $D = \text{whitelisting}$. 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 Maximal Flows Computation

The conditions expressed by (13) and (14), associated with a NSR r, depend on the set of flows F_r. However, in order to improve the efficiency of our 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 $\mathrm{deny}_i(\tau(f, n_i))$ 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 (13) and (14) respectively:

$$\forall f \in F_r^M.\exists i.(n_i \in \pi(f) \wedge \mathit{allocated}(n_i) \wedge \mathit{deny}_i(\tau(f, n_i))) \tag{15}$$

$$\exists f \in F_r^M.\forall i.(n_i \in \pi(f) \wedge \mathit{allocated}(n_i) \Rightarrow \neg \mathit{deny}_i(\tau(f, n_i))). \tag{16}$$

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. This concept represents an important novelty with respect to our preliminary attempt [13], where all models, including network functions and NSRs, were defined considering packets rather than maximal flows. Moving to 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.

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

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 $p = [n_0, n_1, \dots, n_{m+1}] \in paths(r, G_A)$ do
3:      $F \leftarrow \{[n_0, t^r_{0,1}, n_1, \text{true}, n_2, \dots, \text{true}, n_{m+1}]\}$
4:      for $i = 1, 2, \dots, m$ do
5:          $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' | l + [b_i, n_i] + l' \in F, b'_i \in \{\mathcal{I}^a_i, \mathcal{I}^d_i\}\}$
6:          $F \leftarrow \{l + [b_i \wedge b'_i, n_i] + l' | l + [b_i, n_i] + l' \in F, b'_i \in \{\mathcal{D}_{ij}\}\}$
7:          $F \leftarrow \{l + [b_i, n_i, b_{i+1} \wedge \mathcal{T}_i(b_i), n_{i+1}] + l' | l +$
            $[b_i, n_i, b_{i+1}, n_{i+1}] + l' \in F\}$
8:      $F' \leftarrow \{l + [t^r_{m,m+1} \wedge b_{m+1}, n_{m+1}] \mid l + [b_{m+1}, n_{m+1}] \in F\}$
9:      for $i = m, m-1, \dots, 1$ do
10:         $F' \leftarrow \{l + [b_i \wedge \mathcal{T}_i^{-1}(b_{i+1}), n_i, b_{i+1}] + l' | l +$
            $[b_i, n_i, b_{i+1}] + l' \in F'\}$
11:     if $F \neq F'$ then
13:         $F \leftarrow F'$
13:         goto line 4
14:     $F_r^M \leftarrow F_r^M \cup F$
15: return $F_r^M$
```

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 $p = [n_0, \dots, n_{m+1}]$, 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 $[n_0, t^r_{0,1}, n_1, \text{true}, n_2, \dots, \text{true}, n_{m+1}]$ (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 4-7), 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 sublists 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 $\mathcal{I}^a_i$, $\mathcal{I}^d_i$, and D_{ij}, i.e., the classes that can be distinguished by the $deny_i$ predicate and by the $\mathcal{T}_i$ transformer respectively (lines 5-6). In these formulas, the operator + means list concatenation. Note that, for the packet filters in the APs, $\mathcal{I}^a_i$ and $\mathcal{I}^d_i$ are unknown, because their configuration is not yet decided. For these nodes, $\mathcal{I}^a_i$ and $\mathcal{I}^d_i$ 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 $\mathcal{T}_i(b_i)$, 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 8). 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 9-10), 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.

6 Firewall Allocation and Configuration

Soft constraints are used to find the optimal firewall allocation and configuration choices. They 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 (17), where $\text{Soft}(f, c)$ stands for a soft clause with formula f and weight c

$$\forall a_h \in A_A. \ \text{Soft}(allocated(a_h) = \text{false}, c_h). \tag{17}$$

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), the FP of the firewall that can be placed in $a_h \in A_A$ is characterized by a default action d_h and a set of specific rules U_h.[3] 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.

Given the potentially high number of FP rules that can be configured for each firewall instance, limiting the number of soft clauses that have to be generated to reach goal 2) 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

3. 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.

maximal flows F_r^M contains at least a traffic flow that passes through the AP a_h (i.e., such that (18) holds)

$$\exists f \in F_r^M .\ a_h \in \pi(f). \tag{18}$$

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 (18) 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 (18) 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 the firewall 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., $\text{enforces}(d_h, r)$ is false. The set Q_h is thus computed as the smallest set of the 5-tuples, such that (19) holds

$$\begin{aligned}\forall r \in R. \forall f \in F_r^M .\ (a_h \in \pi(f) \wedge \neg enforces(d_h, r)\\ \Rightarrow (\forall q \in \tau(f, a_h).\ q \in Q_h)).\end{aligned} \tag{19}$$

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.

Each placeholder rule will be then actually included in U_h only if the optimizer engine will establish 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 (20) is thus exploited to represent the ideal condition in which no firewall rule needs to be configured

$$\forall p_i \in P_h.\text{Soft}(\neg configured(p_i), c_{h,i}). \tag{20}$$

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:

$$(\exists p_i \in P_h.configured(p_i)) \Rightarrow allocated(a_h). \tag{21}$$

If the consequent of this hard constraint is true, the soft clause defined by (17) 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 (17) must be higher than the sum of the weights of the soft clauses (20) related to all the firewall placeholder rules

$$\sum_{i:p_i \in P_h} (c_{h,i}) < c_h. \tag{22}$$

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 (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 (24), is exploited to model the sign of the weights

$$\forall p_i \in P_h. \forall j \in \{1, 2, 3, 4\}.\text{Soft}(p_i.IPSrc_j = *, \sigma(a_h) \cdot c_{h,i,j}) \tag{23}$$

$$\sigma(a_h) = \begin{cases} -1 & \text{if } \ wlst(a_h) \\ +1 & \text{if } \ \neg wlst(a_h) \end{cases}. \tag{24}$$

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 (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 FP 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 $T_h = \{\tau(f, a_h) | f \in F_r^M \text{for some } r \in R\}$. 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, if it must drop t, and another one to define how the same policy must be configured if 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 returns true if the rule conditions completely include the values of the 5-tuple fields, i.e., if (25) holds. Instead the *matchNone* predicate returns true if the packet classes expressed by the rule conditions and by the 5-tuple fields are disjunct, i.e., if (26) holds

$$matchAll(p_i, q) \Leftrightarrow q \subseteq p_i.C \tag{25}$$

$$matchNone(p_i, q) \Leftrightarrow \neg(q \wedge p_i.C). \tag{26}$$

Then, for each $t \in T_h$, the two hard clauses in (27) and (28) are stated. 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

$$\begin{aligned}&allocated(a_h) \wedge deny_h(t) \Rightarrow \text{(a)} \vee \text{(b)}\\&\text{(a)} = wlst(a_h) \wedge \forall q \in t.(\forall p_i \in P_h.(\neg configured(p_i) \vee matchNone(p_i, q)))\\&\text{(b)} = \neg wlst(a_h) \wedge \forall q \in t.(\exists p_i \in P_h.(configured(p_i) \wedge matchAll(p_i, q)))\end{aligned} \tag{27}$$

$$\begin{aligned}&allocated(a_h) \wedge \neg deny_h(t) \Rightarrow \text{(a)} \vee \text{(b)}\\&\text{(a)} = wlst(a_h) \wedge \forall q \in t.(\exists p_i \in P_h.(configured(p_i) \wedge matchAll(p_i, q)))\\&\text{(b)} = \neg wlst(a_h) \wedge \forall q \in t.(\forall p_i \in P_h.(\neg configured(p_i) \vee matchNone(p_i, q))).\end{aligned} \tag{28}$$

The truth of the antecedents directly depends on the consequences of the implications represented by (15) and (16). If the firewall is the function identified by (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 (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.

6.1 Summary of MaxSMT Problem Formulation

Having described the problem modeling constraints in Sections 4 and 6, here we summarize how they are used in the formulation of the MaxSMT problem.

For what concerns hard constraints, on one side, for each requirement r, hard constraint (15) (for isolation) or (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 Section 4.2. For example, for each function in node n_i that cannot drop flows, (6) is introduced, while for each allocation place a_h where a firewall can be allocated, (27) and (28) are introduced to define the forwarding behavior of this firewall.

For what concerns soft constraints, for each allocation place a_h, (17) is used to minimize the number of allocated firewalls, while constraints (20) are used to minimize the number of rules in each allocated firewall, and constraints (23) to prefer aggregate rules. The weights of soft constraints are decided so as to satisfy constraints (22) and (24). For each allocation place a_h, the additional hard constraint (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 (4), which forces the allocation of firewalls in the positions where the administrator necessarily requests the presence of a firewall.

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 6.1. *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.*

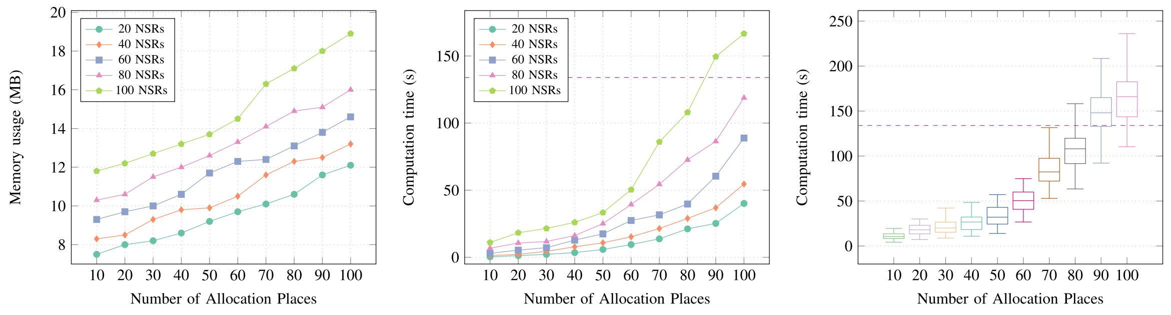
Proof. By contradiction, let us assume that s' exists. The sum of weights of s's true soft constraints is $c_h(|A_A| - n) + \delta$, where $c_h(|A_A| - n)$ is the contribution from soft clauses (17) and $\delta < \sum_{i:p_i \in P_h}(c_{h,i})$ the one from soft clauses (20). Similarly, the same sum for 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 $c_h \le c_h(n' - n) < \sum_{i:p_i \in P_h}(c_{h,i})$, which contradicts (22). □

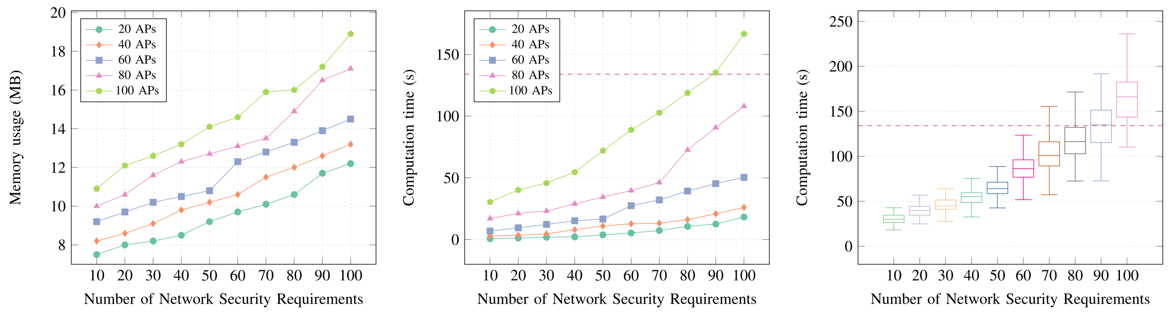
A similar theorem can be proved for the minimization of the number of rules.

From what concerns correctness, we do not present a formal proof, which would be too complex, but we provide its intuition. 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, we can prove that the solution is correct, as long as we can prove or assume that the hard constraints of the problem presented in this paper, 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 [38], 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.



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

Fig. 4. Scalability for increasing number of allocation places.



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

Fig. 5. Scalability for increasing number of network security requirements.

7 Implementation and Validation

We implemented the methodology proposed in this article by means of a Java framework, which exploits the APIs offered by the z3 solver [39] 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 external tools as a component of a more complex architecture. In particular, we tested its interaction with some NFV and cloud orchestrators, such as Open Baton and Kubernetes [40].

The framework has been validated in different ways: on synthetically generated networks, to prove the scalability of the approach (Section 7.1), and on topologies inspired by production networks, to validate its feasibility in realistic and complex environments (Section 7.2). It has also been compared to the state-of-the-art approaches and to our preliminary approach [13] (Section 7.3). Finally, the optimization provided by our approach has been evaluated (Section 7.4). 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.

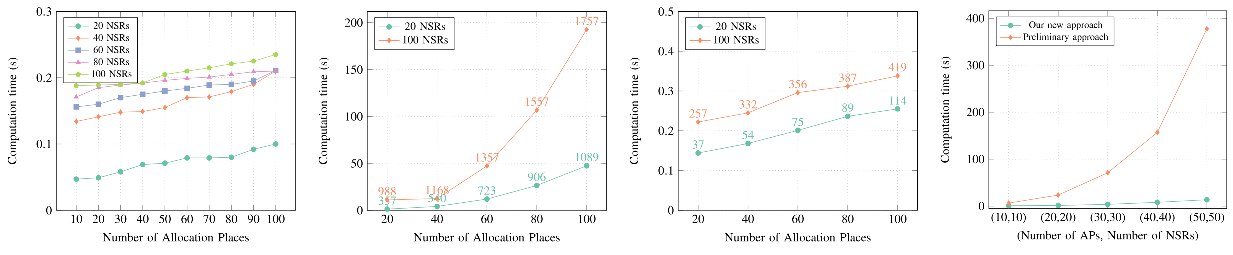
7.1 Scalability Validation

The scalability of the approach has been evaluated varying the following 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. They have also been compared with those of our preliminary approach [13].

7.1.1 Scalability versus Number of APs and NSRs

The charts in Figs. 4 and 5 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. 2. 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



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

Fig. 6. Other tests related to the scalability validation.

choice is motivated by the consideration, confirmed by our experiments, that this is the worst case.

Figs. 4a and 5a 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 our framework can work without any worrisome limitation due to memory.

Figs. 4b and 5b 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 [37]. Second, we have defined models that 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 [39].

Figs. 4c and 5c show the value distribution of the computation time measures by means of whisker plots; the number of NSRs is fixed to 100 in Fig. 4c, and the number of APs is fixed to 100 in Fig. 5c. 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 Figs. 4b, 5b, 4c, and 5c, 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 [41], this time is 134ms. The figures of our experiments show that the time taken by our 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.

7.1.2 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 we study the cases of non-enforceability, 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. 4, 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. 4a, 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. 6a. More precisely, managing non-enforceability cases requires a computation time that is two magnitude orders less than the other cases, under the same dimension of the network 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 [39]. This feature is beneficial to our 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.

7.1.3 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

TABLE 5
Number of Hard and Soft Constraints

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

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 in this paper.

The results of the scalability tests related to the number of constraints are presented in Fig. 6b. 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, where the exact division between hard and soft constraints is included.

From the results depicted in Fig. 6b, 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 dimension of the overall solution space. These clauses are also the ones where the highest number of free variables appear. 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.

7.1.4 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, 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 implement the flow algorithm.

The results of the scalability tests related to the maximum flows algorithm are presented in Fig. 6c. 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.

7.1.5 Comparison With Preliminary Approach

The performance of the proposed methodology has been also compared with the one of the preliminary version of this work [13], which was not based on the computation of maximal flows and was characterized by very different formal models and constraints of the MaxSMT problem. Fig. 6d reports the average computation time over 100 iterations, for both approaches, for progressive numbers of APs and NSRs. From these results, it is evident that the new approach presented in this article improves the performance by at least one magnitude order in all the evaluated test cases.

TABLE 6
Test Results for GÉANT and Internet2 AGs

	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

7.2 Validation on Production Network Topologies

The framework has been also validated with GÉANT and Internet2, two AGs inspired by the production networks.[4] 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.

Table 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 inspired from the GÉANT and Internet2 features. A second reason is that in these AGs

4. https://geant3plus.archive.geant.net/, https://www.internet2.edu/

TABLE 7
Comparison With Most Related Approaches (Features versus Scalability)

Approach	Network	Allocation	Configuration	Formal	Optimization	Scalability
[7]	Traditional	X	✓	✓(with [14])	X	No Info
[8]	Traditional	X	✓	X	X	No Info
[9]	Traditional	X	✓	X	X	10 devices - No info
[10]	Both	✓(SG)	X	✓	✓(**allocation**: minimize deployment cost, and maximize security and usability)	20FW - 80s
[12]	Virtual	✓(SFC)	✓	X	X	20 devices - 4s
[15]	Traditional	X	✓	✓	X	No Info
[16]	Both	X	✓	✓	X	5FW - 50s
[31]	Both	✓(SFC)	✓	✓	X	No Info
[30]	Both	✓(SG)	X	X	✓(**allocation**: minimize rule set cardinality)	60FW - No Info
Our approach	Both	✓(SG)	✓	✓	✓(**allocation & configuration**: minimize number of FWs and of rules)	100FW - 90s

links are bidirectional, so each bidirectional link counts as two directed links. More detailed information about the exact topologies of the two AGs and the set of NSRs that have been exploited for these tests is provided in the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2022.3160293.

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 the time needed for a manual configuration and that this automated approach avoids human errors.

7.3 Comparison With State-of-the-Art Approaches

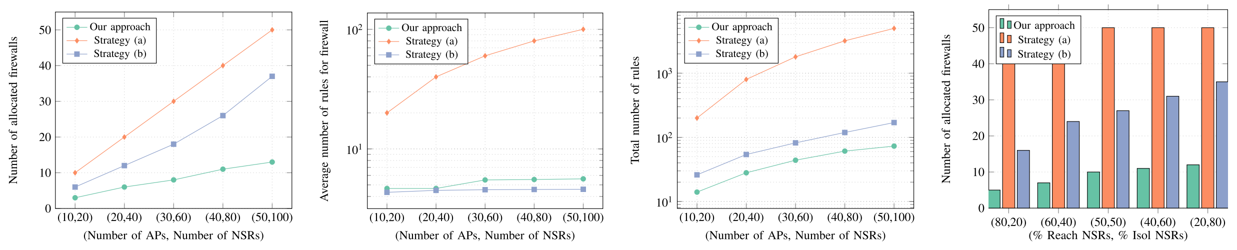
Table 7 shows a comparison of our approach with the most related state-of-the-art approaches available in the literature. The table compares the following main features of each approach: "Network" specifies if the approach can work only on traditional physical networks, only on virtual networks, or on 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 our previous discussions in Sections 2 and 3.4, i.e., that no other existing approach jointly computes the firewall allocation scheme and the configuration starting from a provided SG, as we do. Also, no prior work achieves all the three features of full automation, optimization, and formal correctness, with the exception of [10], 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, as we already discussed, even no combination of existing approaches can be used to obtain automatically the same results that our approach can obtain.

For these reasons, there is no alternative equivalent approach to which we can compare the performance of our tool. Nevertheless, it may be interesting to make a rough comparison between the scalability data reported by the other approaches and our data (the "Scalability" column of Table 7). Our 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 [7], [8], and [15], 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 [9], [10], [12], and [16], can scale up to small sized networks (between 5 and 20 firewalls). An approach that has been tested on bigger networks is [30]. 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. Our method, 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 [30] solves a problem that is much simpler than our 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.

7.4 Evaluation of Optimization

The optimization that our 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 we consider the configuration strategies introduced in Section 3.4. However, in order to have a uniform target, we consider all cases targeting a blacklisting approach. Specifically, we consider (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 *allow* default action in each AP that is closest to the source specified by the NSR, with a rule that enforces the NSR.



(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. 7. Optimization tests, in comparison with configuration strategies.

Figs. 7a, 7b and 7c 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 our own, adopt a blacklisting target. Fig. 7a shows that our 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 Figs. 7b and 7c, our 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 Figs. 7b and 7c 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. 7d 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, we have also changed the target of our approach to the security-oriented case, which is more complex than the blacklisting one. Instead, the other reference strategies remain as previously described, as they cannot support our security-oriented mode. Therefore, the number of reachability NSRs does not influence their results. From Fig. 7d, it is evident how both strategy (b) and our method generate bigger allocation schemes and rule sets when the number of isolation NSRs is higher, but the increase for our approach is much more mitigated than the one for approach (b). Finally, the behavior of our 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 our method, whatever approach is employed.

8 Conclusion

We presented a new methodology to compute, in a fully automatic way, the optimal allocation scheme and configuration of packet filter firewalls that enforces a given set of NSRs on a given SG, with formal correctness assurance. This technique allows a service designer who wants to enrich a SG with packet filters in order to enforce a set of NSRs in an NFV environment, to avoid human errors, to save resource consumption in the servers where the enriched SG will be deployed, and to perform the enrichment operation rapidly enough even with SGs with several tens of APs and with several tens of NSRs. This is the first time a method featuring all these properties has been proposed.

In the future, our purposes are to extend the presented methodology to the allocation and auto-configuration of other NSFs, such as web application firewalls, anti-spam filters and intrusion detection systems, to further enrich the set of security requirements which can be specified by the user of the framework. Moreover, we are planning to integrate this methodology in mitigation mechanisms, so that the security orchestration is performed not only to satisfy user-specified policies, but also as automatic reaction to cyber attacks.

References

[1] J. Halpern and C. Pignataro, “Service function chaining (SFC) architecture,” *RFC 7665*, pp. 1–32, 2015.

[2] J. Garay, J. Matias, J. Unzilla, and E. Jacob, “Service description in the NFV revolution: Trends, challenges and a way forward,” *IEEE Commun. Mag.*, vol. 54, no. 3, pp. 68–74, Mar. 2016.

[3] S. Singh, Y. Jeong, and J. H. Park, “A survey on cloud computing security: Issues, threats, and solutions,” *J. Netw. Comput. Appl.*, vol. 75, pp. 200–222, 2016.

[4] R. Mijumbi, J. Serrat, J.-I. Gorricho, S. Latré, M. Charalambides, and D. López, “Management and orchestration challenges in network functions virtualization,” *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[5] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, “Improving the formal verification of reachability policies in virtualized networks,” *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 713–728, Mar. 2021.

[6] Y.-M. Kim and M. Kang, “Formal verification of SDN-based firewalls by using TLA+,” *IEEE Access*, vol. 8, pp. 52100–52112, 2020.

[7] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato: A novel firewall management toolkit,” *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, 2004.

[8] P. Verma and A. Prakash, “FACE: A firewall analysis and configuration engine,” in *Proc. IEEE/IPSJ Int. Symp. Appl. Internet*, 2005, pp. 74–81.

[9] J. D. Guttman, “Filtering postures: Local enforcement for global policies,” in *Proc. IEEE Symp. Secur. Privacy*, 1997, pp. 120–129.

[10] M. A. Rahman and E. Al-Shaer , “Automated synthesis of distributed network access controls: A formal framework with refinement,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 416–430, Feb. 2017.

[11] A. Gember-Jacobson , A. Akella, R. Mahajan, and H. H. Liu, “Automatically repairing network control planes using an abstract representation,” in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 359–373.

[12] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 707–720, Apr. 2019.

[13] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, 2020, pp. 1–7.

[14] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 177–187.

[15] J. Govaerts, A. K. Bandara, and K. Curran, "A formal logic approach to firewall packet filtering analysis and generation," *Artif. Intell. Rev.*, vol. 29, no. 3/4, pp. 223–248, 2008.

[16] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner, "The mathematical foundations for mapping policies to network devices," in *Proc. 13th Int. Joint Conf. E-Bus. Telecommun.*, 2016, pp. 197–206.

[17] N. B. Y. B. Souayeh and A. Bouhoula, "A fully automatic approach for fixing firewall misconfigurations," in *Proc. 11th IEEE Int. Conf. Comput. Inf. Technol.*, 2011, pp. 461–466.

[18] K. Adi, L. Hamza, and L. Pene, "Automatic security policy enforcement in computer systems," *Comput. Secur.*, vol. 73, pp. 156–171, 2018.

[19] A. Tsuchiya, F. Fraile, I. Koshijima, Á. Ortiz, and R. Poler, "Software defined networking firewall for industry 4.0 manufacturing systems," *J. Ind. Eng. Manage.*, vol. 11, 2018, Art. no. 318.

[20] P. Salva-Garcia , E. Chirivella-Perez , J. B. Bernabé, J. M. Alcaraz-Calero , and Q. Wang, "Towards automatic deployment of virtual firewalls to support secure mMTC in 5G networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2019, pp. 385–390.

[21] S. Dobson, D. Hutchison, A. Mauthe, A. Schaeffer-Filho , P. Smith, and J. P. G. Sterbenz, "Self-organization and resilience for networked systems: Design principles and open research issues," *Proc. IEEE*, vol. 107, no. 4, pp. 819–834, Apr. 2019.

[22] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling efficient service function chaining by integrating NFV and SDN: Architecture, challenges and opportunities," *IEEE Netw.*, vol. 32, no. 6, pp. 152–159, Nov./Dec. 2018.

[23] E. J. Scheid *et al.*, "INSpIRE: Integrated NFV-based intent refinement environment," in *Proc. IFIP/IEEE Symp. Integr. Netw. Serv. Manage.*, 2017, pp. 186–194.

[24] Y. Han, J. Li, D. Hoang, J.-H. Yoo, and J. W.-K. Hong, "An intent-based network virtualization platform for SDN," in *Proc. 12th Int. Conf. Netw. Serv. Manage.*, 2016, pp. 353–358.

[25] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," *Comput. Commun. Rev.*, vol. 48, no. 5, pp. 55–63, 2018.

[26] Z. Hao, Z. Lin, and R. Li, "A SDN/NFV security protection architecture with a function composition algorithm based on trie," in *Proc. 2nd Int. Conf. Comput. Sci. Appl. Eng.*, 2018, Art. no. 176.

[27] Y. Liu, Y. Lu, W. Qiao, and X. Chen, "A dynamic composition mechanism of security service chaining oriented to SDN/NFV-enabled networks," *IEEE Access*, vol. 6, pp. 53918–53929, 2018.

[28] A. Pasias, T. Kotsiopoulos, G. Lazaridis, A. Drosou, D. Tzovaras, and P. G. Sarigiannidis, "Enabling cyber-attack mitigation techniques in a software defined network," in *Proc. IEEE Int. Conf. Cyber Secur. Resilience*, 2021, pp. 497–502.

[29] A. Matencio-Escolar , J. M. Alcaraz-Calero , P. Salva-Garcia , J. B. Bernabé, and Q. Wang, "Adaptive network slicing in multi-tenant 5G IoT networks," *IEEE Access*, vol. 9, pp. 14048–14069, 2021.

[30] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 218–230, Feb. 2010.

[31] N. Schnepf, R. Badonnel, A. Lahmadi, and S. Merz, "Rule-based synthesis of chains of security functions for software-defined networks," *Electron. Commun. EASST*, vol. 76, pp. 1–19, 2018.

[32] E. Al-Shaer , H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 10, pp. 2069–2084, Oct. 2005.

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

[34] J. Gil-Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[35] K. Hida and S. Kuribayashi, "Joint deployment of virtual routing function and virtual firewall function in NFV-based network with minimum network cost," in *Proc. 21st Int. Conf. Adv. Netw.-Based Inf. Syst.*, 2018, pp. 333–345.

[36] M. E. Halaby, "On the computational complexity of MaxSAT," *Electron. Colloq. Comput. Complex.*, p. 34, 2016. [Online]. Available: https://eccc.weizmann.ac.il/report/2016/034

[37] R. Robere, A. Kolokolova, and V. Ganesh, "The proof complexity of SMT solvers," in *Proc. Int. Conf. Comput. Aided Verification*, 2018, pp. 275–293.

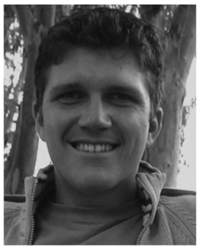
[38] G. Marchetto, R. Sisto, M. Virgilio, and J. Yusupov, "A framework for user-friendly verification-oriented VNF modeling," in *Proc. 41st IEEE Annu. Comput. Softw. Appl. Conf.*, 2017, pp. 517–522.

[39] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. 14th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2008, pp. 337–340.

[40] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Towards a fully automated and optimized network security functions orchestration," in *Proc. 4th Int. Conf. Comput. Commun. Secur.*, 2019, pp. 1–7.

[41] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. P. Costa, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Comput. Commun.*, vol. 161, pp. 86–98, 2020.

Daniele Bringhenti received the MSc (summa cum laude) degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2019, where he is currently working toward the PhD degree in control and computer engineering. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, formal verification of network security policies.



Guido Marchetto received the PhD degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2008, where he is currently an associate professor with the Department of Control and Computer Engineering. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.

Riccardo Sisto received the PhD degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 1992. Since 2004, he has been a full professor of computer engineering with the Politecnico di Torino. He has authored and coauthored more than 100 scientific papers. His research interests include formal methods, applied to distributed software and communication protocol engineering, distributed systems, and computer security. He is a senior member of the ACM.

Fulvio Valenza received the MSc (summa cum laude) and PhD (summa cum laude) degrees in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively, where he is currently an associate professor. His research interests include network security policies, orchestration and management of network security functions in SDN/NFV-based networks, and threat modeling.

Jalolliddin Yusupov received the MS and PhD degrees in computer engineering from Politecnico di Torino, Italy, in 2016 and 2020, respectively. He is currently the head of the Academic Department, Turin Polytechnic University, Tashkent, Uzbekistan. His research interests include formal verification of security policies in automated network orchestration, modeling, cyber physical systems, and cloud computing systems.