Automated optimal firewall orchestration and configuration in virtualized networks

Automated optimal firewall orchestration and configuration in virtualized networks

Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy, Emails: {first.last}@polito.it

***Abstract*—Emerging technologies such as Software-Defined Networking and Network Functions Virtualization are making the definition and configuration of network services more dynamic, thus making automatic approaches that can replace manual and error-prone tasks more feasible. In view of these considerations, this paper proposes a novel methodology to automatically compute the optimal allocation scheme and configuration of virtual firewalls within a user-defined network service graph subject to a corresponding set of security requirements. The presented framework adopts a formal approach based on the solution of a weighted partial MaxSMT problem, which also provides good confidence about the solution correctness. A prototype implementation of the proposed approach based on the z3 solver has been used for validation, showing the feasibility of the approach for problem instances requiring tens of virtual firewalls and similar numbers of security requirements.**

***Index Terms*—NFV, network security, firewall, optimization**

I. Introduction

Software-Defined Networking (SDN) [1] and *Network Functions Virtualization* (NFV) [2] are new technologies, designed to introduce flexibility in networking. SDN enables the run-time definition of the paths that traffic flows must cross, while NFV enables virtualized network functions, installed on general-purpose servers in the cloud. These features allow service designers to define the intended network services by means of *Service Graphs* (SGs) representing the involved service functions and their interconnection.

In a virtualized network, security automation is becoming more feasible thanks to the intrinsic agility of this environment, and to the full software-based control of each network component it allows. Nevertheless, automation of security defenses is still only partially addressed in literature [3].

A commonly time-consuming and error-prone security task that could be heavily eased by an automatic approach is the placement and configuration of the *Network Security Functions* (NSFs) [4] that must be introduced in order to satisfy some *Network Security Requirements* (NSRs) – i.e., the security constraints the network behavior must respect –. For example, isolation of a compromised network node could be required after an attack, and this new requirement can be fulfilled by properly placing and configuring NSFs that enforce it. As performing this task manually can lead to incorrect or non-optimal results, automation can be exploited not only to save human effort, but also to get provably correct and optimal solutions. Formal correctness of the achieved solution represents an added value, because it does not require further manual checks by the user, who can rely with a high level of confidence on a solution generated according to this approach. Optimality, on the other side, leads to a minimization of the employed computational resources and to a maximization of performance. In view of these motivations, in this paper we propose a new security automation methodology for virtualized networks, and we provide its validation. The main aim is to automatically define the optimal allocation scheme and configuration of virtual firewall instances, by refining a SG provided by the service designer so that it fulfills a set of given NSRs. The internal definition of this problem as a partial weighted *Maximum Satisfiability Modulo Theories* (MaxSMT) problem provides at the same time formal assurance about the correctness of the solution and optimality. To the best of our knowledge, this is the first time an approach with all these features together – i.e. automation, optimality, and formal correctness assurance – is proposed.

The focus of the proposed methodology is on packet filters, which represent the most common firewall technology and the most frequently exploited security defense in networks.

The remainder of the paper is structured as follows. Section II explains how our methodology is designed, Section III presents some performance tests carried out on a prototype implementation. Finally, Section IV and Section V contain the related works and the conclusions.

II. The Proposed Approach

A. Problem statement and solution strategy

The problem addressed in this paper is to automatically compute, in a formally correct way, the optimal allocation scheme and configuration of packet filtering firewalls in a SG, in such a way to satisfy a set of NSRs.

According to our approach, optimality means: 1) minimize the number of allocated virtual firewall instances in the SG, in order to minimize the resources consumed by the SG; 2) minimize the number of rules inside each firewall configuration, in order to reduce the memory required to store the rules and, at the same time, improve the performance of the filtering operations. The second goal has lower priority than the first, since deploying a new virtual machine requires more memory and introduces more overhead than adding a filtering rule in an already deployed firewall. Here we do not address the problem of optimal allocation of the SG onto physical servers because we already addressed it in a previous work [5].

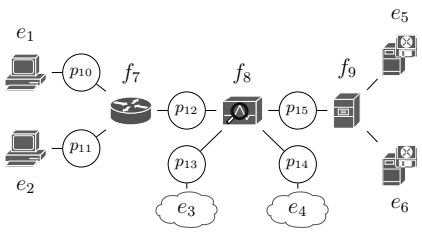


Fig. 1: Allocation Graph example

Our strategy to reach the goals of formal correctness and optimality consists of adopting a formal model of the virtual network behavior and a procedure to search for the optimal solution which guarantees correctness by construction according to this formal model. More precisely, our idea is to formulate the problem as a partial weighted *Maximum Satisfiability Modulo Theories* (MaxSMT) problem, which receives two different sets of clauses, hard and soft, for which partial satisfiability has to be achieved. Hard clauses need necessarily to be satisfied, representing essential commitments which nonetheless contribute to the reduction of the space of all the possible solutions. On the other hand, soft clauses are relaxable constraints, because their satisfiability is not strictly required; in this sense, their presence makes the SMT problem partial. Moreover, in a partial weighted MaxSMT problem each soft clause is characterized by a weight, and the problem consists not only in establishing partial satisfiability but also in finding, among the assignments that satisfy a partial set of clauses, one that maximizes the sum of the weights of the satisfied soft clauses; consequently, when an optimizer engine tries to solve a partial weighted MaxSMT instance, it assigns priority to the soft clauses characterized by higher weights. In the remainder of the paper, for simplicity, the word MaxSMT stands for partial weighted MaxSMT.

Given this context, the inputs of the problem that must be formally modeled as hard clauses are: 1) a graph, describing the network functions of the virtual network and their interconnection; 2) a set of *Network Security Requirements* (NSRs), describing which traffic flows must be allowed or blocked in the virtual network. The positioning of firewalls in the graph and their configuration, instead, have to be encoded as soft clauses, because they are subject to optimization.

If at least one of the hard clauses cannot be satisfied, then the outcome of the MaxSMT problem is UNSAT, i.e. partial satisfiability does not hold. If instead all the hard constraints can be satisfied, then the outcome is the optimal firewalls allocation scheme and the automatically computed configuration of their *Filtering Policy* (FP), that is the set of the mappings between the conditions to check on the packet fields and the actions to perform about forwarding. These outputs are achieved by targeting the two optimization objectives that have been stated at the beginning of this section.

The remainder of this section provides more details about the inputs and outputs of the methodology and their *First-Order Logic* (FOL) formulation in the MaxSMT problem. Finally, a complete example is provided to clarify how our automated approach can heavily impact the design of a network security service.

B. Service and Allocation Graph

The SG is a directed graph $G_S = (N_S, L_S)$, where N_S is the set of vertices representing the network nodes, while L_S is the set of edges representing the directed connections between the nodes. In particular, the vertex set is modeled as $N_S = E_S \cup S_S$. E_S is the set of end points that can directly correspond to a terminal, a physical server or a subnetwork in the substrate infrastructure where virtual instances of the functions will be allocated. Instead, S_S is the set of service functions; the elements in S_S are simple *Network Functions* (NFs), that do not offer any security protection from cyberattacks, but are simply exploited to create an end-to-end service.

The SG is fed to an automatic tool that generates an intermediate internal representation, called *Allocation Graph* (AG), which is obtained from the SG by adding new nodes called *Allocation Places* (APs). Each AP is a tentative position in the graph where a firewall instance could be allocated. In general, it is necessary to insert an AP for each edge of the SG in order to explore all the possible placements and hence have the assurance of eventually reaching the optimal solution. However, the service designer can introduce some placement constraints as additional inputs, clearly relying on his security knowledge. In particular, these constraints can be divided into two categories: (i) the user can explicitly forbid the generation of an AP, thus reducing the solution space (ii) the user can force the allocation of a firewall on an AP, for example because the corresponding VNF is already deployed in the virtualized network. Fig. 1 shows an example of AG, automatically generated from a SG where the service designer forbids the creation of the APs between the function f_9 and the end points e_5, and e_6.

Formally, the AG is another directed graph $G_A = (N_A, L_A)$, where N_A is the set of vertices representing the network nodes, while L_A is the set of links interconnecting them. The main difference with respect to the SG is that the vertex set is defined as $N_A = E_A \cup S_A \cup P_A$, where $E_A = E_S$ and $S_A = S_S$, while P_A is the set of the APs, that are absent in the SG. In G_A, each $n_k \in N_A$ is identified by a unique index k, so that each $l_{ij} \in L_A$, with $i \neq j$, represents the directed link from node n_i to node n_j.

For each NF in the AG, a formal model of the NF forwarding behavior is defined. The forwarding behavior is the only aspect of NFs that is really relevant for the definition of firewall placement and configuration in the SG. Other aspects of the packet processing performed by the NFs that do not influence the forwarding behavior can be safely neglected. More precisely, in order to keep the models simple, we track only the *possibility* of the various forwarding actions taken by each NF for each packet, rather than representing all the details of the decision algorithm. Hence, these models are based on two predicates, that represent the possibility that each packet is received or forwarded by any node. These predicates are

defined for each $n_i, n_j \in N_A$ and $p_0 \in P$, where P is the set of all packets: (i) $recv(n_i, n_j, p_0)$ which is true if node n_j can receive a packet p_0 from node n_i; (ii) $send(n_i, n_j, p_0)$ which is true if node n_i can send a packet p_0 to node n_j.

In view of this consideration, some of the NFs have an extremely simplified model, according to which each packet can be forwarded to each out-port without modifications. For example, traffic monitors belong to this class of NFs, because they forward all packets to their out port without modifying them. Another example is a load balancer, because, even though it implements a specific algorithm by means of which it distributes the traffic to different servers of a cluster, it is not possible to establish beforehand how each flow will be effectively managed. On the other hand, other functions, such as a NAT, have more complex behaviors that require specific models. This kind of modeling of NFs, expressed by means of sets of FOL clauses, has already been proposed and validated in literature for network verification [6]–[8]. The same approach is reused in our work, by feeding the FOL formulas that represent each NF forwarding behavior to the MaxSMT solver as hard clauses. For firewalls, the model is slightly different, because each firewall can be present or not. This will be detailed in section II-D. Finally, graph edges are also expressed as hard clauses involving the *send* and *recv* predicates.

C. Network Security Requirements

Concerning the security requirements to be enforced in the network service, our methodology focuses on *connectivity* requirements, i.e., the specification of which traffic flows must be allowed (or prohibited) between any pair of end points in the SG. These security constraints represent the second input of the framework and are characterized by two elements: (i) a *general behavior* representing the default rule applied to traffic flows for which the user does not specify any further indication; (ii) a set of specific Network Security Requirements (NSRs), each one specifying whether a traffic flow must be allowed (*reachability* requirement), or must be blocked by a firewall (*isolation* requirement).

There are three approaches a service designer can adopt for the definition of the security constraints. Two are based on the traditional *whitelisting* and *blacklisting* methods, i.e., all traffic flows must be blocked (in the former) or allowed (in the latter) with the exception of the communications for which the user explicitly defines some reachability (in the former) or isolation (in the latter) requirements. In the third available approach, called *specific*, the service designer must explicitly formulate only the requirements – both isolation and reachability specific properties – he is interested in. Therefore, the optimal solution will be computed in order to satisfy exclusively this specific set of constraints, while for the unspecified cases the system will automatically decide whether to allow or forbid the flow. Within this approach, we assume that the entire set of security requirements is conflict-free, since this can be easily obtained from a general set by means of well-known conflict analysis techniques proposed in literature [9]–[11]; in this way, the security constraints do not require a priority criterion in their formulation.

Formally, if R is the set of all the NSRs that must be fulfilled, each $r \in R$ is modeled as a 6-tuple *r = (type, IPSrc, IPDst, pSrc, pDst, tProto)* where *type* is the requirement type, which can be isolation or reachability, while the other elements are the typical IP 5-tuple values (source and destination IP addresses, source and destination port numbers, transport-level protocol) that specify a packet flow.

The NSRs contribute to the definition of the hard clauses of the MaxSMT problem. Before presenting their FOL formulas, however, two notations must be introduced. The first one, $addr(e_k)$, is the function that maps an endpoint $e_k \in E_A$ to its IP address if it is a single host or to its IP address range – e.g. 10.1.∗.∗ – if it is a collection of end points. The second notation, $r.match(p)$ is the predicate that is true if and only if packet $p \in P$ matches requirement $r \in R$, i.e. if each requirement component positively matches or includes – depending if it is a single value or a range – the corresponding packet field.

Having introduced these notations, the hard clauses for enforcing the NSRs in the AG are defined as follows. On one side, if $r \in R$ is an isolation property, all the pairs of end points $e_i, e_j \in E_A$ such that $addr(e_i) \subseteq r.IPSrc \wedge addr(e_j) \subseteq r.IPDst$ are identified. For each pair of nodes e_i and e_j thus identified, then the following constraints must be satisfied:

$$\forall k \mid n_k \in N_A \wedge l_{ik} \in L_A.\ \exists p_0.\ (send(e_i, n_k, p_0) \wedge r.match(p_0)) \quad (1)$$

$$\begin{gathered}\forall k \mid n_k \in N_A \wedge l_{kj} \in L_A.\ \forall p_0.\ (recv(n_k, e_j, p_0) \wedge \\ p_0.IPDst = addr(e_j) \implies \neg\, r.match(p_0))\end{gathered} \quad (2)$$

Both clauses are needed to enforce an isolation property: (1) imposes that the source can send at least one packet matching the requirement to every neighbor; (2) imposes that all the packets that can be received and accepted by the destination do not match the requirement. On the other hand, if $r \in R$ is a reachability property, for each pair of nodes $e_i, e_j \in E_A$ such that $addr(e_i) \subseteq r.IPSrc \wedge addr(e_j) \subseteq r.IPDst$, the following constraints must be satisfied:

$$\exists k \mid n_k \in N_A \wedge l_{ik} \in L_A.\ \exists p_0.\ (send(e_i, n_k, p_0) \wedge r.match(p_0)) \quad (3)$$

$$\begin{gathered}\exists k \mid n_k \in N_A \wedge l_{kj} \in L_A.\ \exists p_0.\ (recv(n_k, e_j, p_0) \wedge \\ p_0.IPDest = addr(e_j) \wedge\ r.match(p_0))\end{gathered} \quad (4)$$

These two clauses introduce different commitments than those defined for an isolation property: (3) imposes that at least a packet that can be sent by the source to one of its neighbors matches the requirement; (4) imposes that at least a packet that can be received and accepted by the destination matches the requirement.

D. Firewalls allocation and configuration

In case of success, the first outcome must be the optimal allocation scheme of the firewalls in the AG. This result is achieved by considering the possibility to allocate an instance in each available AP. Since the best solution would be to allocate the least number of firewalls, in the MaxSMT problem

a soft constraint is formulated for each $p_k \in P_A$ so that the optimal value of the *allocated(p_k)* predicate – which is true if a firewall is allocated in p_k – is false. This clause is formalized by (5), where the notation $Soft(x, c_k)$ specifies a soft constraint with formula x and weight c_k.

$$\forall k \mid p_k \in P_A.\ Soft(allocated(p_k) = false,\ c_k) \tag{5}$$

The second expected outcome is the automatic configuration of the allocated firewalls; in this context, the firewall FP is characterized by a default action and a set of more specific 5-tuple-based rules.

First, the default action is established so that the number of filtering rules is minimized, as it has been explained beforehand, according to the approach the service designer exploits for the formulation of the NSRs. Then, for each firewall allocated in $p_k \in P_A$, a set of placeholder rules Π_k must be identified, i.e., the maximum number of rules that could be needed in its FP is established with respect to the input security requirements. This step, which is critical to achieve good scalability, is performed by means of a number of pruning strategies. The main two ones are: (i) given a specific NSR, in a firewall policy a corresponding placeholder rule is not needed if the traffic flow related to this requirement cannot cross the AP on which the packet filter is tentatively allocated; (ii) given a specific NSR whose traffic flow can cross the AP, a placeholder rule is not needed anyway if the default action of the firewall that would be allocated there already enforces the requirement (e.g. a whitelisting firewall guarantees the satisfiability of an isolation property, if the specific rules are properly configured, as they are in an optimal configuration).

Moreover, the *wildcards* feature has been introduced to further reduce the cardinality of the maximum set of placeholder rules. This feature allows us to represent both an IP address and the netmask in a joint expression: for instance, the $10.0.0.*$ statement refers to the network 10.0.0.0/24. Besides, it can be also applied to transport-level ports and protocols. In view of this consideration, if some NSRs that would effectively require corresponding filtering rules in the same firewall can be merged in a single one by means of wildcards, then it is possible to assign a single placeholder rule for all these requirements, as long as this decision does not have any impact on the satisfiability of the other ones.

After identifying the maximum number of placeholder rules by means of the aforementioned algorithms, two different classes of soft clauses are defined for policy configuration. First, in order to minimize the total number of configured rules, for each placeholder rule $\pi_i \in \Pi_k$ of a firewall that can be allocated in $p_k \in P_A$, a soft constraint is defined so that the optimal value of the function *configured(π_i)* – which returns true if π_i is configured in the policy – is false:

$$\forall i \mid \pi_i \in \Pi_k.\ Soft(configured(\pi_i) = false,\ c_{ki}) \tag{6}$$

In order to enforce the wanted priority between the two minimization objectives, the weights of these soft clauses are decided so as to satisfy constraint (7).

$$\sum_{i \mid \pi_i \in \Pi_k} (c_{ki}) < c_k \tag{7}$$

A second class of soft clauses is, instead, introduced to specify that using *wildcards* has to be preferred for each single component of each filtering rule; in fact, wildcards are useful not only to reduce the number of placeholder rules, which is done in the pre-processing phase, but also to reduce the number of rules in the solution of the MaxSMT problem. (8) and (9) define the soft clauses related to *wildcards* usage in each one of the four components of IP addresses in quad-dotted notation. Similar soft clauses are defined also for the transport-level ports and protocol.

$$\forall i \mid \pi_i \in \Pi_k.\ \forall j \in \{1,2,3,4\}.\ Soft(\pi_i.IPSrc_j = *,\ c_{kij1}) \tag{8}$$

$$\forall i \mid \pi_i \in \Pi_k.\ \forall j \in \{1,2,3,4\}.\ Soft(\pi_i.IPDst_j = *,\ c_{kij2}) \tag{9}$$

In our approach, the use of wildcards for each rule component has lower priority than the absence of rule itself. Consequently, constraint (10) must be respected for each $\pi_i \in \Pi_k$.

$$\sum_{j=1}^{4} (c_{kij1} + c_{kij2}) < c_{ki} \tag{10}$$

The set of clauses so built is analyzed by the MaxSMT solver. If partial satisfiability holds, the optimal allocation and configuration of firewalls is returned. Instead, if partial satisfiability does not hold, a non-enforceability report is returned. A possible reason for this condition is that the APs are not sufficient because of additional constraints introduced by the user to prohibit their creation. This report can then be exploited for a next run of the tool, after the inputs have been properly updated.

E. Clarifying example

The most relevant features of our approach can be clarified by means of a sample scenario, where a manual configuration would be easily prone to human errors. For this purpose, let us consider Fig. 1 as the AG generated from an input SG. It is worth mentioning that the end points e_3 and e_4 are not single hosts, but subnetworks. Table I illustrates: (i) how each SG node is mapped to an equivalent single IP address or address range and the function type of the node; (ii) the NSRs to satisfy, defined through the *specific* approach.

First of all, let us focus only on the first two constraints of the NSRs list in Table I, that are the isolation requirements for the end points e_1 and e_2, shadowed by the NAT f_7. Since the service designer requires that they are isolated from the two services e_5 and e_6, at least a firewall is needed. Considering for the moment only the APs p_{10}, p_{11} and p_{12} and supposing that no other requirements are specified, then our methodology would place a single whitelisting firewall on p_{12}, since it would be able to filter all the packets coming from the NAT, thus reducing the number of firewalls – the non-optimal alternative which a service designer may instead consider to adopt would be to place two firewalls, one in p_{10} and the other one in p_{11}.

Then let us consider also the other NSRs, except the last one of the list in Table I. On one side, e_3 must be able to reach the HTTP web server e_5 at the TCP destination port 80, e_4 must be able to reach the POP3 mail server e_6 at the TCP

TABLE I: Input

IP addresses and function type

Identifier	IP address	Function type
e_1	192.168.0.2	Web client
e_2	192.168.0.3	Web client
e_3	130.192.225.∗	Network of end points
e_4	130.192.120.∗	Network of end points
e_5	220.226.50.2	HTTP web server
e_6	220.226.50.3	POP3 mail server
f_7	120.0.2.2	NAT
f_8	120.0.2.3	Traffic monitor
f_9	120.0.2.4	Web cache

Network Security Requirements

Type	IPSrc	IPDst	pSrc	pDst	tProto
Isol	192.168.0.2	220.226.50.∗	∗	∗	∗
Isol	192.168.0.3	220.226.50.∗	∗	∗	∗
Isol	130.192.225.∗	220.226.50.3	∗	∗	∗
Reach	130.192.225.∗	220.226.50.2	∗	80	TCP
Isol	130.192.225.∗	220.226.50.2	∗	≠80	TCP
Isol	130.192.225.∗	220.226.50.2	∗	∗	UDP
Reach	220.226.50.2	130.192.225.∗	∗	∗	∗
Isol	130.192.120.∗	220.226.50.2	∗	∗	∗
Reach	130.192.120.∗	220.226.50.3	∗	110	TCP
Isol	130.192.120.∗	220.226.50.3	∗	≠110	TCP
Isol	130.192.120.∗	220.226.50.3	∗	∗	UDP
Reach	220.226.50.3	130.192.120.∗	∗	∗	∗
Isol	130.192.120.∗	130.192.225.∗	∗	∗	∗

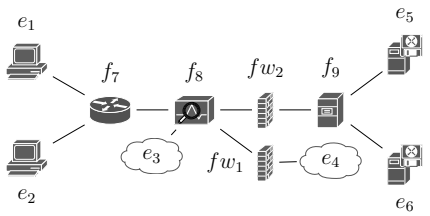


Fig. 2: Final Service Graph with allocated firewalls

destination port 110; all the other traffic between these pairs – i.e. TCP with different destination port or UDP – must be blocked. Since the paths from e_3 and e_4 towards the server intersect in p_{15} with the paths from e_1 and e_2, the optimal solution would be to allocate a firewall in that position.

Finally, according to the last NSR of Table I, e_4 must not be able to contact e_3. However, neither the path between them crosses p_{15}, where the previous discussion led to the decision to place a firewall, nor it is possible to identify a single other intersection between all the paths that the possible traffic flows that must be considered pass through. Consequently, the only solution is to allocate an additional firewall, either in p_{13} or in p_{14}, which would block the packets from e_4.

In this process, if each decision is taken manually by the designer, several mistakes can be made while defining firewall allocation and configuration. For example, since among the same pairs of end points different constraints are defined for different traffic flows, a manual approach could likely introduce shadowing or correlation anomalies [9], which would lead to an incorrect security service. Moreover, even though the designer manages to reach a correct solution, it could be a non-optimal one.

TABLE II: Policy rules

Firewall fw_1

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	220.226.50.3	130.192.120.∗	∗	∗	∗
2	Allow	130.192.120.∗	220.226.50.3	∗	110	TCP
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Firewall fw_2

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
1	Allow	130.192.225.∗	220.226.50.2	∗	80	TCP
2	Allow	130.192.120.∗	220.226.50.3	∗	∗	∗
3	Allow	220.226.50.∗	130.192.∗.∗	∗	∗	∗
D	Deny	∗.∗.∗.∗	∗.∗.∗.∗	∗	∗	∗

Instead, using the framework we developed according to the methodology we proposed, it is possible to reach the optimal and formally correct solution. For the sake of completeness, Fig. 2 shows the final logical topology of the SG computed by our approach, whereas Table II describe the FPs of the two introduced firewall instances. In these table, the *D* letter is used to identify the firewall default action.

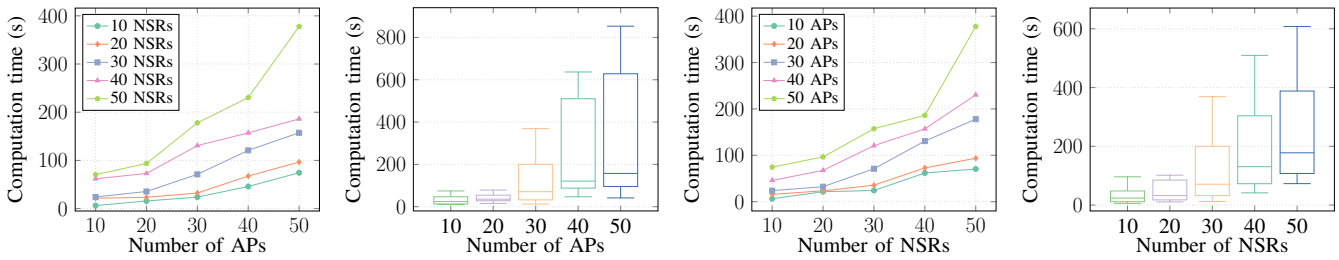
III. Implementation and Validation

We implemented our approach by means of a Java framework, which exploits the APIs offered by the z3 theorem prover [12] to formulate and solve the MaxSMT problem. The framework offers a REST APIs, so that it can be easily integrated as a component of more complex architectures.

The validation of the developed framework has been performed by means of scalability tests, which have been run on a machine with Intel i7-6700 CPU running at 3.40 GHz and 32GB of RAM. The parameters that have been considered are the ones that mostly affect the complexity of the problem (i.e. the number of clauses): (i) the number of APs where the firewall instances can be allocated (ii) the number of NSRs. We cannot compare our approach with alternative existing approaches because, as explained in section IV, no other approach solving the same problem exists.

The charts in Fig. 3a and 3c present the results of the tests performed to evaluate execution time versus number of APs and number of NSRs. The security requirements considered for the tests are defined in the context of a *specific* approach and only functions that do not modify packets are considered, so that the validation is focused on the two metrics of interest. Besides, for each test case with a given number of APs and NSRs, we compute the median computation time on 30 runs, where the service and the requirements are the same but only the IP addresses are different. This is motivated by the experimental observation that computation time can vary if the IP addresses are changed, which is due to how z3 internally manages the integer theory. For this reason, we also show the experimental results by means of the whisker plots in Fig. 3b and 3d. The number of NSRs in Fig. 3b, and the number of APs in Fig. 3d are fixed to 30.

The most evident result from this validation is that, even though the MaxSMT problem belongs to the *NP-complete* class in terms of worst-case computational complexity, the



(a) Scalability on APs (b) Whisker plot for APs (c) Scalability on NSRs (d) Whisker plot for NSRs

Fig. 3: Results of scalability tests on APs and NSRs

framework can scale to SGs of tens of APs and a number of NSRs that is expected in a service of this dimension. It is also possible to notice that an increment of the NSRs number produces a computation time comparable to the one produced by the same increment of the APs number. Furthermore, the two whisker plots show how most of the values are gathered around the expected median value. Moreover, memory consumption is not an issue because, in the worst case which has been considered, it is only 10.3 MB. All these positive results that have been showed in this section are mainly due to the correct tuning of the optimization parameters and the pruning strategies we adopted, which reduce the solution space. In fact, even though some possible solutions are not evaluated by the MaxSMT solver because of these strategies, nevertheless they would not be considered optimal.

IV. Related Works

A. Automatic configuration and verification of firewall policies

In literature, the automatic configuration of firewalls and the formal verification of their policies represent a central research area in the network security field. A milestone is represented by *Firmato* [13], a firewall management toolkit that performs a refinement of high-level filtering requirements. Other similar works are [14] and [15], which can automatically generate rule sets also in distributed firewall architectures. Despite the relevance of these works, they are mainly targeted to traditional networks, rather than to NFV environments. Moreover, they do not provide formal correctness assurance.

Formal methods have been exploited in more recent works, such as [16]–[19]. However, they have a number of limitations. [16] lacks optimality but also generality, since it is bound to IPChains and Cisco PIX. [17], [18] and [19] can only fix firewall misconfigurations rather than allowing the creation of rules from scratch. Furthermore, [17] and [18] do not focus on virtualized networks, while [19] works at an abstraction level higher than the actual policy rules.

Finally, in all the works mentioned so far, the decision about where to allocate the packet filters in the logical topology is not made by the tool, but it is assumed as input.

B. Automatic synthesis and refinement of a Service Graph

Other works address the automatic creation and refinement of a SG, according to a set of constraints, before its deployment. This topic is becoming central because of the growing interest in operational resilience based on NFV and intent-based networking [20] [21]. Among the works regarding automatic synthesis of SGs, [22]–[25] define methodologies for intent-based generation of network services in virtualized environments. However, not only they lack formal correctness assurance of the achieved solutions, but the approaches described in [22] and [25] do not even target optimality.

The most relevant works that provide optimal or sub-optimal automatic placement of firewalls in a SG are [26] and [27]. However, none of them can also optimally synthesize the rules of each placed firewall. [26] approximately minimizes the maximum number of rules for each firewall by means of a heuristic algorithm, without providing formal correctness assurance, while [27] computes the optimal placement using a formal model, also taking other aspects into account, but using an iterative approach where the constraints are tuned after each failed attempt.

V. Conclusion and Future Works

This paper presents a new methodology for automated firewall allocation and configuration that can be used to exploit the flexibility provided by virtualized networks. The proposed approach suits the work of a service designer, replacing manual tasks, and contributes to achieving a correct security configuration, by means of its formal approach, also finding the optimal solution among all the possible ones. Up to our knowledge, this is the first time an approach with these features is proposed. From the validation of the framework developed according to the described methodology, the approach has been shown to be feasible for problem instances requiring tens of virtual firewalls and similar numbers of security requirements.

Our purpose for a near future is to further refine the methodology, addressing the automatic allocation and configuration of other security functions, such as web application firewalls, anti-spam filters and VPN gateways. Besides, we are planning to improve the performance, by pursuing a trade-off between optimality of configurations and required computational complexity. Finally, we are planning experiments with real SGs.

Acknowledgment

This work has been partially supported by the EU H2020 Projects ASTRID (Grant Agreement no. 786922) and CyberSec4Europe (Grant Agreement no. 830929).

References

[1] D. Kreutz, F. M. V. Ramos, P. J. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015. [Online]. Available: https://doi.org/10.1109/JPROC.2014.2371999

[2] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016. [Online]. Available: https://doi.org/10.1109/COMST.2015.2477041

[3] 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, 2019. [Online]. Available: https://doi.org/10.1109/TNET.2019.2895278

[4] Verizon, "Data Breach Investigations Report," 2019.

[5] G. Marchetto, R. Sisto, J. Yusupov, and A. Ksentini, "Virtual network embedding with formal reachability assurance," in *14th International Conference on Network and Service Management, CNSM 2018, Rome, Italy, November 5-9, 2018*, 2018, pp. 368–372. [Online]. Available: http://ieeexplore.ieee.org/document/8584921

[6] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, and R. Sisto, "Formal verification of virtual network function graphs in an sp-devops context," in *Service Oriented and Cloud Computing*, S. Dustdar, F. Leymann, and M. Villari, Eds. Cham: Springer International Publishing, 2015, pp. 253–262.

[7] A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker, "Verifying reachability in networks with mutable datapaths," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 699–718. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154630.3154687

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

[9] E. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005. [Online]. Available: https://doi.org/10.1109/JSAC.2005.854119

[10] C. Basile, D. Canavese, C. Pitscheider, A. Lioy, and F. Valenza, "Assessing network authorization policies via reachability analysis," *Computers and Electrical Engineering*, vol. 64, pp. 110 – 131, 2017.

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

[12] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[13] 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, Nov. 2004. [Online]. Available: http://doi.acm.org/10.1145/1035582.1035583

[14] P. Verma and A. Prakash, "FACE: A firewall analysis and configuration engine," in *2005 IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005), 31 January - 4 February 2005, Trento, Italy*, 2005, pp. 74–81. [Online]. Available: https://doi.org/10.1109/SAINT.2005.28

[15] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, 1997, pp. 120–129. [Online]. Available: https://doi.org/10.1109/SECPRI.1997.601327

[16] 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. [Online]. Available: https://doi.org/10.1007/s10462-009-9147-0

[17] N. B. Youssef and A. Bouhoula, "A fully automatic approach for fixing firewall misconfigurations," in *11th IEEE International Conference on Computer and Information Technology, CIT 2011, Pafos, Cyprus, 31 August-2 September 2011*, 2011, pp. 461–466. [Online]. Available: https://doi.org/10.1109/CIT.2011.84

[18] K. Adi, L. Hamza, and L. Pene, "Automatic security policy enforcement in computer systems," *Computers & Security*, vol. 73, pp. 156–171, 2018. [Online]. Available: https://doi.org/10.1016/j.cose.2017.10.012

[19] A. Gember-Jacobson, A. Akella, R. Mahajan, and H. H. Liu, "Automatically repairing network control planes using an abstract representation," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, 2017, pp. 359–373. [Online]. Available: https://doi.org/10.1145/3132747.3132753

[20] S. Dobson, D. Hutchison, A. Mauthe, A. E. S. Filho, P. Smith, and J. P. G. Sterbenz, "Self-organization and resilience for networked systems: Design principles and open research issues," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 819–834, 2019. [Online]. Available: https://doi.org/10.1109/JPROC.2019.2894512

[21] 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 Network*, vol. 32, no. 6, pp. 152–159, 2018. [Online]. Available: https://doi.org/10.1109/MNET.2018.1700467

[22] E. J. Scheid, C. C. Machado, M. F. Franco, R. L. dos Santos, R. J. Pfitscher, A. E. S. Filho, and L. Z. Granville, "Inspire: Integrated nfv-based intent refinement environment," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, May 8-12, 2017*, 2017, pp. 186–194. [Online]. Available: https://doi.org/10.23919/INM.2017.7987279

[23] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong, "An intent-based network virtualization platform for SDN," in *12th International Conference on Network and Service Management, CNSM 2016, Montreal, QC, Canada, October 31 - Nov. 4, 2016*, 2016, pp. 353–358. [Online]. Available: https://doi.org/10.1109/CNSM.2016.7818446

[24] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," *Computer Communication Review*, vol. 48, no. 5, pp. 55–63, 2018. [Online]. Available: https://doi.org/10.1145/3310165.3310173

[25] Z. Hao, Z. Lin, and R. Li, "A sdn/nfv security protection architecture with a function composition algorithm based on trie," in *Proceedings of the 2Nd International Conference on Computer Science and Application Engineering*, ser. CSAE '18. New York, NY, USA: ACM, 2018, pp. 176:1–176:8. [Online]. Available: http://doi.acm.org/10.1145/3207677.3277992

[26] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Trans. Computers*, vol. 59, no. 2, pp. 218–230, 2010. [Online]. Available: https://doi.org/10.1109/TC.2009.172

[27] 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, 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2016.2585108