Automatic and optimized firewall reconfiguration

Automatic and optimized firewall reconfiguration

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

***Abstract*—The continuous innovation in network softwarization has enabled higher dynamism and responsiveness in creating and deploying complex network configurations. Following this trend, several approaches have been proposed to automate the allocation and configuration of network security functions to satisfy a set of network security policies, describing the security requirements to be fulfilled in the network. In particular, many studies focused on addressing this problem for the packet filtering firewall, as it is the most common firewall technology used in computer networks. However, those proposed techniques for automatic firewall configuration are not optimized for reconfiguring an already deployed network. This results in a computation delay that is incompatible with the needs of modern networks and the timing of current network attacks. In order to overcome these limitations, this paper proposes an efficient method to reduce the computation time for reconfiguration while providing an automated, formally correct, and optimal placement and configuration of the required network security functions. The proposal has undergone validation and evaluation tests, so as to show the achieved improvements in comparison to non-optimized approaches.**

***Index Terms*—firewall, optimized reconfigurartion, virtual network orchestration**

I. Introduction

Automatic approaches for network security management have recently become popular as a consequence of the advent of virtualized networks. Nowadays, many solutions leverage the improved flexibility and dynamism introduced by virtualization for synthesizing and deploying large and complex architectures [1]. Furthermore, some of these solutions adopt a formal approach to the problem, ensuring solution correctness by construction. This is crucial because it allows to avoid possible misconfigurations, which is an essential to provide adequate network security [2] [3].

Despite the different advantages achieved by these solutions, whenever there is a change in the network topology or in the set of security policies to be enforced, most of the approaches proposed so far need to be re-executed from scratch to synthetize a new valid configuration, resulting in a wasteful process both in terms of computation power and time. Moreover, this may produce a significantly different configuration with respect to the original one. Consequently, in order to deploy the updated configuration, a large part of the network need to be shut down, updated, and restarted, adding another time delay which is not negligible (for example, OpenStack requires more than 5 s for the deployment of a single machine [4], and Open Source MANO, a well-known NFV orchestrator, requires a delay of 134 s to deploy a virtual function [5]).

This clashes also with the trend for modern networks attacks, as reported by various sources [6], [7], according to which nowadays attacks have shorter duration and rapidly mutate between multiple attack vectors within minutes.

Considering these trends, there is a clear need for a solution suitable for the timing of modern attacks that can reconfigure the network quickly and adapt to evolving attack scenarios. An automated approach based on formal methods could be a potential solution to address this new generation of network attacks, being able to proactively defend against incoming attacks through the optimized computation of an updated firewall configuration that blocks the attacker within a short time delay, while ensuring formal correctness with respect to all security policies in place.

In view of these motivations, this paper proposes a new approach for the optimized security reconfiguration of distributed packet filters for an already deployed network within a short computation time. The novelty of the proposal lies in the coexistence of three important features that, to the best of our knowledge, are not supported by any other reconfiguration approach: complete automation, optimization, and formal correctness assurance. This is made possible by the adopted technique, which is based on the resolution of a partial weighted *Maximum Satisfiability Modulo Theories* (MaxSMT) problem. This type of problem allows, with a carefully designed model of the network configuration and the desired security policies, the computation of a solution that correctly enforces the given requirements while seeking additional optimality goals, going beyond what is achievable with commonly used approaches based on heuristics. The methodology has been validated and implemented in one of the existing automated approaches: VEREFOO (VErified REFinement and Optimized Orchestration).

The remainder of this paper is structured as follows. Section II contains a summary of the related work. Section III introduces some key formalisms used to represent the network, the approach and the main algorithms which have been designed. Finally, Section IV summarizes the results of the performance tests conducted on the implemented prototype, and Section V presents the conclusion and future work.

II. Related Work

Previous related work can be divided in three main categories: 1) approaches that pursue a similar idea but are applied to a different subject with different characteristics and needs, i.e., the problem of routing management (Subsection II-A); 2) approaches that are designed for the same problem,

i.e., reconfiguration of firewalls, but lacking some of the features with respect to our approach (Subsection II-B); 3) approaches for the configuration of distributed firewalls with a similar set of features, namely automation, formal correctness assurance, and optimality, but without the support for a specific reconfiguration procedure (Subsection II-C).

A. Optimized Reconfiguration for routing problems

A small number of studies ([8], [9], [10]) adopt an approach similar to the one presented in this paper, but for a completely different problem, i.e., routing management. Indeed, they deal with routers, routing algorithms, and forwarding policies instead of network security functions and policies. In greater detail, [8] describes the design of the Control Plane Repair algorithm, an approach based on a MaxSMT problem to automatically compute correct and minimal repairs for network control planes. The solution is based on a carefully crafted formal model for the network, the routing protocols, and the exchanged traffics. It supports as optimization goal the minimization of the lines written in the configuration. [9] outlines another synthesis tool, named AED, that formally models the network and its configuration into a MaxSMT problem. The optimality goals considered in the resolution are more refined, allowing the operator to specify different management objectives, such as maintaining structural similarity across devices or minimizing the number of modified devices. Finally, [10] presents JINJING, an approach for the automatic and correct update of routing configuration on the base of intents expressed using an ad-hoc language. This approach models the network as an SMT problem, leaving as open the variables of the elements causing the inconsistencies between current configuration and desired policies while keeping the other elements as fixed. Optimality in this case is not present. Moreover, the approach could produce redundancy in the computed rules as it requires a post-processing task to minimize the lines of the computed configuration, and it just focuses on traditional networks, not allowing to modify the topology of the control plane but only its configuration.

Overall, these approaches are not suitable to the problem analyzed in this paper, even if they combine similar features (automation, formal verification, and optimization), because they apply them to a different context.

B. Automatic fixing of firewall configurations

Other studies ([11], [12], [13], [14], [15]) investigate the problem considered in this paper, i.e. automatic reconfiguration of firewalls, but they address it partially, as their proposed solutions lack some of the features which are included in our proposal. [11] proposes five algorithms to automatically reconfigure a faulty firewall after five corresponding issues (wrong rule order, missing rules, wrong condition predicates, wrong decision actions, wrong extra rules) which are detected through samples of misclassified packets, used as input of the reconfiguration process. [13] uses a dedicated calculus to formally verify if the configuration is compliant with the security policies defined by the user, and, if not, to automatically generate the optimal and correct configuration repair. [15] illustrates a methodology for configuration refinement, formal verification, and, if needed, the automatic computation of a fixing strategy in case the current configuration does not enforce correctly the user defined policies. This is based on a SMT model and, for the fixing, on a constraint refinement approach. [12] computes, whenever a misconfiguration is detected, a formally correct fixing action by resolving a carefully designed SMT problem. [14] presents another approach based on formal models and the design of an SMT problem. It follows a repair by example paradigm, providing as input of the reconfiguration a set of user defined examples of the desired filtering behavior.

Concerning their limitations, [11], [14] can not guarantee the formal correctness of the configuration with respect to a set of security policies, because they do not model all the traffics but either only those provided in the examples or those involved in a detected misclassification, and so they can not guarantee the correctness for all traffics. Almost all of the approaches [11], [12], [13], [14] are not designed for distributed firewalls but they support only single firewall instances. Moreover, they do not support the synthesis of new services from scratch but they can only modify those that have already been deployed. [12], [14] adopt limited or none optimization for the computation of the new configuration. [15] is the most complete one in terms of features but its focus is mostly on access control instead of reachability policies, and it is mostly a description of a possible approach rather than a fully functional solution.

C. Automatic, formal, and optimal firewall configuration

Finally, there are studies that propose automatic technique for the allocation or configuration of distributed firewall systems with all the features we are considering. Among all the ones that are reported in a state-of-the-art survey about automatic security configuration [1], the most relevant ones are ConfigSynth [16] and VEREFOO [17], [18]. The former automates the generation of the firewall allocation scheme (but not of the configuration) with an optimized and formal approach based on the definition of an iterative SMT problem. The idea is that, at each step of the algorithm, the architecture is tuned until it properly enforces all the security properties. In this case, the optimization criteria is the minimization of the network security functions allocated in the network. The latter proposes the definition of a MaxSMT problem to model the network and its configuration. The formal assurance is provided with a correctness-by-construction approach and the involved optimality criteria are the minimization of the number of allocated firewall and the number of firewall rules, so as to reduce the amount of consumed resources.

Despite the relevance of these two studies and other related ones in the same category, they simply regenerate the allocation scheme or configuration from scrath every time, and therefore they do not provide an optimized procedure for reconfiguration.

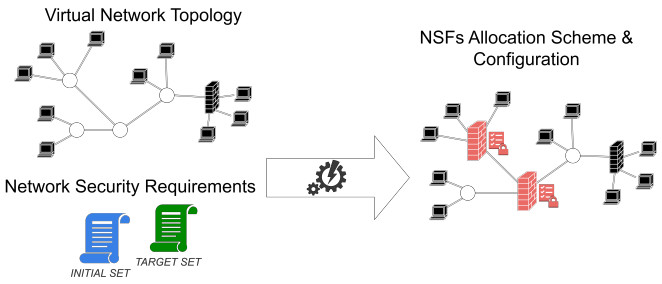


Fig. 1: general schema of the approach

III. The Proposed Approach

The approach proposed in this paper consists in a methodology that computes automatically the reconfiguration of a distributed firewall, whenever the user specifies new Network Security Requirements (NSRs) that are not yet enforced in the network. Full automation is combined with optimization and formal verification, so that the result of the reconfiguration process is computed within a short computation time, while ensuring that the given set of NSRs is satisfied.

As shown in Fig. 1, the presented approach receives two inputs from the user. The first input is the logical topology of a virtual network with an already existing distributed firewall configuration, composed of the allocation scheme of its instances and their filtering rules. The second input is a pair of NSR sets: the *Initial NSR set*, including the old NSRs already satisfied by the existing firewall configuration, and the *Target NSR set*, including the new NSRs to be enforced in the updated network configuration. The produced outputs are the updated allocation scheme and the reconfigured filtering rules of the firewall.

Starting from these inputs, the approach is composed of multiple steps. The first one involves the definition of a complete and formal model that represents the network, the configuration of the different network functions, and the traffic exchanged (Subsection III-A). Then, a central part of this proposal, representing the main novelty introduced here, is the design of an algorithm able to identify the network areas that must be reconfigured based on the intersection of the two sets of NSRs provided as input (Subsection III-B). Having computed this intersection, it is possible to discern which NSRs have been added, kept or deleted in the new set of NSRs with respect to the original one. The algorithm identifies, for all the "added" requirements, i.e., those relevant for the reconfiguration scenario, which elements of the provided network should be modified because in conflict with the new set of security requirements. The configuration of these elements is therefore put under question. Finally, the approach formulates a MaxSMT problem whose resolution allows to generate the new allocation graph and configuration rules of the needed firewalls (Subsection III-C). A MaxSMT problem differs from an SMT problem because it allows the definition of two types of clauses: the *hard constraints* that are compulsory, and the *soft constraints* that are optional and have an associated weight. The selected solution is the one that satisfy all the hard constraints and maximise the sum of the weights of the satisifed soft constraints.

This approach avoids the need to recompute the entire network from scratch, resulting in a significant reduction in computation time. The proposed strategy speeds up the process by narrowing down the space of possible solutions that are analyzed, keeping certain parts of the configuration as fixed, and modifying optimality related clauses, which are provided to the solver, to have a faster convergence towards the optimal solution. Notably, the problem still models all the traffic for the NSRs in the target set, so the union of "added" and "kept". This ensures that the computed solution is guaranteed to be correct with respect to the desired NSRs.

A. Formal models

The formal models used in this paper stem from VEREFOO [17], [18], a policy-based approach for automatic firewall configuration, which has all the features (automation, formal verification, optimization) we are interested in. Here, we report the main features of those models that are required to understand the remainder of the section, focusing on the ones that are different with respect to [17], [18]

The logical topology of the input network is modeled as a directed graph, named *Service Graph* (SG), whose nodes represent all the network functions and endpoints (e.g., web clients, web servers, firewalls, NATs) and whose edges represent directed connections. However, the SG model is not directly used by the next steps of the proposed approach, but it is preliminarily pre-processed to create an alternative representation, named *Allocation Graph* (AG). The main difference is that the AG model is characterized by an extra node type named *Allocation Place* (AP), representing a placeholder node that can be used by the MaxSMT solver to potentially allocate a firewall. The transformation of the SG model into the AG consists in adding an AP only in-between pairs of network nodes that do not contains any function that could be reused, e.g., firewalls, as the idea is to reconfigure previously allocated firewall instances whenever it is possible, rather than placing additional ones in other APs.

The packets that may cross the AG are grouped in classes, depending on the values of their header fields. Each packet class, also called traffic in this paper, is represented as a predicate computed over some variables representing the header fields. Packets whose fields have the same values belong to the same traffic, represented by the same predicate, and are therefore managed in the same way by all nodes crossed in the network. Their predicate representing the formal model of each traffic is a conjunction of sub-predicates, one for each considered packet field. Since this approach works with packet filters, the modeled fields are the five ones composing the IP 5-tuple, i.e., source and destination IP addresses, source and destination ports, and protocol type. Each sub-predicate can represent a single value, a range of values, or the range of all possible values, denoted with the "*" symbol. The set of all different packet classes crossing the AG is denoted as T.

The way each node composing the AG handles each input packet class is then modeled in a way that is as lightweight as possible, by considering only the parameters that are actually relevant for the security reconfiguration problem. In particular, it is modeled by means of two functions, representing respectively the forwarding and transformation behaviors. On the one hand, the function modeling the forwarding behavior of node n_i is $deny_i \colon T \rightarrow \mathbb{B}$. This function maps an ingress traffic t to true, if and only if n_i blocks all the packets of that class. For simplicity, we denote $\mathcal{I}_d$ the set of denied packets, and $\mathcal{I}_a$ the set of allowed packets. On the other hand, the function modeling the transformation behavior of node n_i is $\mathcal{T}_i \colon T \rightarrow T$. This function maps an input traffic t to the corresponding output traffic, after the possible modifications that it may apply. For several function types (e.g., forwarders, traffic monitors, firewalls), $\mathcal{T}_i$ is the identity function, as they cannot modify the input traffic. Instead, for functions such as NATs and load balancer, it provides the information related to the changes applied to the 5-tuple fields.

These models (i.e., the ones of packet classes and network node behavior) allow to introduce the concept of *traffic flow*. A *traffic flow* represents how a specific packet class is forwarded and transformed within its path. A flow $f \in F$ is modeled as a list of alternating nodes and packet classes $[n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_z, t_z d, n_d]$, where each node n_i in the list represents a node crossed by the flow, whereas the traffic t_{ij} is a predicate representing the class of packets transmitted from node n_i to node n_j. The definition of the traffic flows crossing the AG may differ, depending on how single packets are grouped into the corresponding classes. From this point of view, we decided to adopt the grouping strategy named Atomic Flow (AF), proposed in [19]. The reason is that, according to that study, it is the technique that provides more benefits and better performance for solving an automatic (re)configuration problem. In greater detail, this grouping strategy is based on the *Atomic Predicate* concept, proposed by [20] for the network reachability problem. The idea is that, given a set of predicates of the network, it is possible to compute a set of corresponding atomic predicates that are minimal, unique, and fully representative of the inital set. Then, a flow $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_z, t_z d, n_d]$ can be defined atomic if each traffic t_{ij} is an atomic predicate. To compute the set of atomic predicates, and then the set of AFs, we consider the "interesting" predicates extracted from the NSRs and the network configuration. The algorithms are not reported here, because they are already described in [19].

Lastly, the security requirements that must be enforced in the AG are modeled as the combination of two elements: a set of specific NSRs R and a general behavior. On the one hand, each specific NSR $r \in R$ is formally modeled as a tuple (a, C), where a is the action that must be applied on packets matching with the condition predicate C. The NSR is defined isolation requirement if the action a is deny, reachability requirement if instead the action a is allow. The set of specific NSRs provided by the user is assumed to be anomaly-free, which is not a limitation because there are many well-known anomaly analysis techniques ([21], [22]) that easily allows to derive anomaly-free policy sets. On the other hand, the general behavior adopted in this proposal is a "don't care" approach, which allows users to define both reachability and isolation requirements without imposing any restriction on other packet classes (i.e., users are not concerned about the reachability and isolation of packets for which they have not specified a specific NSR).

The formalization of the specific reachability and isolation NSRs in terms of traffic flows management is as follows: an isolation NSR is satisfied if for any associated traffic flow there is at least one node with an allocated function configured to block the traffic in input for that flow (eq. 2), whereas a reachability NSR is satisfied if there is at least one associated traffic flow that is not blocked from source to destination by any of the crossed nodes (eq. 1). These will be modeled in the MaxSMT problem as hard constraints, making their satisfaction mandatory. Note that the reported equations are using some utility functions whose meaning is as follows: $\pi(f)$ returns the nodes belonging to flow f (excluding the source), $allocated(n)$ returns true if there is a firewall allocated in node n, and finally $\tau(f, n)$ returns the traffic in input to node n for flow f.

$$\exists f \in F_r . \forall i . (n_i \in \pi(f) \wedge allocated(n_i) \implies \neg deny_i(\tau(f, n_i))) \tag{1}$$

$$\forall f \in F_r . \exists i . (n_i \in \pi(f) \wedge allocated(n_i) \wedge deny_i(\tau(f, n_i))) \tag{2}$$

Moreover, the input NSR set R is composed of two subsets: the *Initial set* R_i, including the NSRs that are already enforced in the existing network modeled by the AG, and the *Target set* R_t, including the new NSRs to be enforced in the updated network configuration.

B. Algorithm for detection of network area to reconfigure

Starting from the formal models of all the input components (i.e., network topology, function behavior, traffic flows, and NSRs), our approach envisions the execution of an algorithm, designed to detect the network areas that actually require firewall reconfiguration for the satisfaction of the Target NSRs included in R_t.

First, this algorithm classifies each NSR $r \in \{R_t \cup R_i\}$ to one of the following groups: (*i*) $\mathcal{R}_d = \{r \in R_i | r \notin R_t\}$, the "deleted" NSRs which are no more needed in the final configuration but are present in the initial one, (*ii*) $\mathcal{R}_a = \{r \in R_t | r \notin R_i\}$, the "added" NSRs that should be enforced in the final configuration and are not present in the initial one, and (*iii*) $\mathcal{R}_k = \{r \in R_t | r \in R_i\}$, the "kept" NSRs that are already configured in the provided network and should continue to be enforced in the final configuration.

Second, the algorithm detects, for all "added" requirements $\mathcal{R}_a$, i.e., those relevant to the reconfiguration scenario, which elements of the provided network should be modified because in conflict with at least a new requirement. Due to the different formulations of the two requirement types, this part of the

Algorithm 1 Algorithm for selecting network area to reconfigure for each added isolation requirement

Input: an isolation requirement r, and an AG $\mathcal{G}_A$
Output: nodes to be reconfigured $N_{reconfigure}$
1: **for** $f \in F_r$ **do**
2: $found \leftarrow False$
3: **for** $n_i \in \pi(f) = [n_1, n_2, ..., n_d]$ **do**
4: **if** $allocated(n_i)$ & $deny_{n_i}(\tau(f, n_i))$ **then**
5: $found \leftarrow True$
6: break
7: **end if**
8: **end for**
9: **if** $found == False$ **then**
10: $N_{reconfigure} \leftarrow \pi(f)$ {All nodes in the path should be reconfigured}
11: **end if**
12: **end for**
13: **return** $N_{reconfigure}$

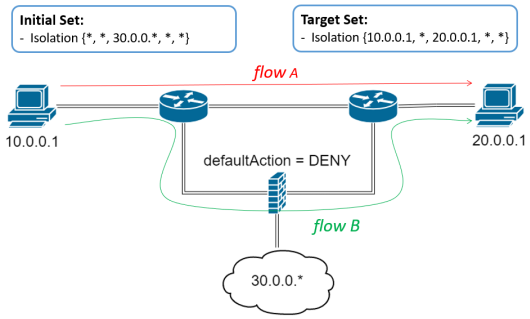


Fig. 2: Example of addition of an isolation requirement

Algorithm 2 Algorithm for selecting network area to reconfigure for each added reachability requirement

Input: a reachability requirement r, and an AG $\mathcal{G}_A$
Output: nodes to be reconfigured $N_{reconfigure}$
1: $tmpReconfigured \leftarrow \emptyset$
2: **for** $f \in F_r$ **do**
3: $tmpFlow \leftarrow \emptyset$
4: **for** $n_i \in \pi(f) = [n_1, n_2, ..., n_d]$ **do**
5: **if** $allocated(n_i)$ & $deny_{n_i}(\tau(f, n_i))$ **then**
6: $tmpFlow \leftarrow n_i$
7: **end if**
8: **end for**
9: **if** $tmpFlow.isEmpty()$ **then**
10: $tmpReconfigured \leftarrow \emptyset$
11: break
12: **else**
13: $tmpReconfigured \leftarrow tmpFlow$
14: **end if**
15: **end for**
16: **return** $N_{reconfigured} \leftarrow tmpReconfigured$

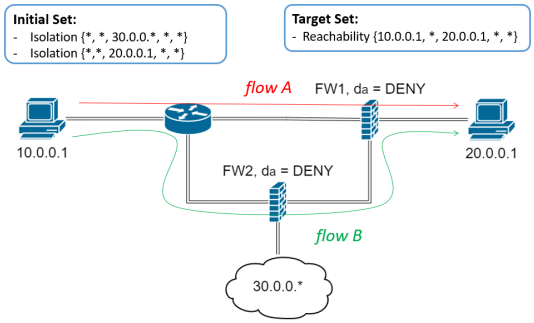


Fig. 3: Example of addition of a reachability requirement

algorithm is differently formulated for the case of isolation requirements, and the case of reachability requirements. Anyhow, it is important to underline that the algorithm selects as reconfigurable all the nodes which can potentially be used to fulfill the NSRs in $\mathcal{R}_a$, since it can not decides a priori which is either the optimal node for blocking a traffic or the optimal traffic flow which must be allowed from source to destination.

Considering a new isolation requirement $r \in \mathcal{R}_a$ and a given AG $\mathcal{G}_A$, the procedure to compute the network elements to be reconfigured is presented in Algorithm 1. This procedure starts considering for each isolation requirement, r, all the correlated atomic flows, F_r. The algorithm checks whether there is a node along the flow path that is currently blocking the incoming traffic for that flow (lines 3-7). If no such node is found for a given flow, then all the nodes crossed along the flow path are designated as eligible for reconfiguration (lines 9-11). This would allow the solver to subsequently decide in which node a firewall should be allocated to enforce the isolation requirement r. Fig. 2 clarifies this with an example, where the algorithm takes as inputs the two sets of Initial and Target NSRs, along with a partially configured network that comprises a firewall and two forwarders. The the new requirement that should be enforced is the isolation requirement for the traffic from the web client 10.0.0.1 to the web server 20.0.0.1. In this case, there are two paths associated with this requirement, each with only one AF. These two flows are labeled A and B, respectively. The algorithm analyzes each flow to determine if there exists a node blocking the traffic. In this scenario, flow B encounters a firewall that blocks all traffic through its default action, consequently the condition is satisfied. Instead, flow A does not crosses any network function that is blocking the traffic. So, the nodes belonging to its path must added to the set of nodes to be reconfigured, $N_{reconfigure}$ (i.e., the two forwarders in this case).

Considering a new reachability requirement $r \in \mathcal{R}_a$ and an AG $\mathcal{G}_A$, the procedure for selecting the network elements to be reconfigured is presented in Algorithm 2. In this case, the satisfiability condition is the logical negation of the prior scenario. The algorithm has to look for the existence of an atomic flow that is not blocked from the source up to the destination. If such a flow is found to satisfy the reachability condition, the algorithm may terminate before having checked all flows. For all the flows F_r correlated with the reachability requirement r, the algorithm examines whether the nodes along the path are blocking the incoming traffic for the given flow. If this is the case, these nodes are added to a temporary list (lines 4-8). In the end, if the algorithm does not find an atomic flow satisfying the reachability condition, all the nodes in the temporary list are selected for reconfiguration and added to the set $N_{reconfigure}$. Considering the example in Fig. 3,

we encounter a situation analogous to the previous scenario but with different inputs. In this case, the network consists of two firewalls configured in whitelisting mode and a forwarder. The new requirement that should be added is the reachability from node 10.0.0.1 to node 20.0.0.1, encompassing all possible ports and protocol types. This requirement is associated with two paths, each with an associated atomic flow. These are referred as A and B. In this instance, the algorithm checks whether at least one of these flows does not block the traffic. If this is not the case, the algorithm proceeds to select for reconfiguration, in each flow, the nodes that are blocking the traffic. In this specific case, both flows encounter a firewall that is blocking their traffic. As a result, the algorithm selects the nodes that are blocking both flows, as the solution would be to reconfigure either FW1 or FW2. This ensure that at least one atomic flow can reach the destination, thus meeting the reachability requirement condition.

C. MaxSMT Problem formulation

After the algorithm has identified all the firewall instances that may potentially require reconfiguration, this information is used, jointly with the formal models of the input, for the formulation of the MaxSMT problem. The core of this formulation is mutuated from the one proposed in [17], [18] for automatic firewall configuration from scratch. However, some key changes have been introduced to contextualize that formulation to the optimized reconfiguration scenario.

A first difference is that the approach proposed in this paper restricts the set of APs that are available for the solver to allocate firewall instances, and keeps some of them as static elements, which cannot be updated within the security configuration.

Moreover, to further optimize this approach, also the soft constraints involved in the optimization phase have been adjusted, leading to an additional performance improvement. The guiding principle is that the optimal reconfiguration is the one that does not require any update to the network, thus causing the lowest possible delay. Following this idea, the soft constraints have been updated to prefer an already allocated firewall instead of deploying a new one, as well as an already configured rule must be preferred with respect to a newly generated one.

The same classes of soft constraint have been employed to express both objectives, namely the minimization of resource usage and the preference of reusing the original firewall configuration. However, the weights associated to these constraints differ in relation to the considered subject, whether it is an empty AP or reconfigured firewall, or a newly generated firewall rule with respect to an already configured one. This difference is such that the usage of a reconfigured node, as any of its rules, would result in an higher sum of weight and a preferred solution. As a result, the final configuration will be the one that not only minimizes the number of consumed resources in general, but also the one that produces the smallest possible number of changes with respect to the initial configuration.

In particular, the soft constraint regulating the allocation of a firewall for each node n in the set of APs $\mathcal{A}$, is formulated as in equation 3. This instructs the solver to prefer a solution that does not allocate a firewall for any APs. Indeed, the non-allocation soft constraint produces a contribution to the sum of weights equal to c_k.

$$\forall a_i \in \mathcal{A}.\ Soft(\neg allocated(a_i), c_k) \tag{3}$$

Similarly, the soft constraint regulating the configuration of firewall rules is presented in equation 4. In this case a different weight c_{ki}, smaller than the previous, is assigned to the non configuration of each potential firewall rule p_i. The set P_k contains all the rules that should be potentially configured in a firewall if it is allocated.

$$\forall p_i \in P_k.\ Soft(\neg configured(p_i), c_{ki}) \tag{4}$$

In the approach optimized for reconfiguration, the weights associated to these soft constraints are modified for the nodes in $N_{reconfigured}$. The first soft constraint has been modified in such a way that the non-allocation of each firewall in $N_{reconfigured}$ produces a contribution c_k^R to the sum of weights, such that $c_k^R < c_k$. In this way, the non-allocation of an empty AP would be preferred to the non-allocation of a reconfigured firewall because it has an higher weight. The same principle is applied for the second soft constraint.In this case, for each firewall in $N_{reconfigured}$, any of the configured rules p_i has an associated weight c_{ki}^R, such that $c_{ki}^R < c_{ki}$. As before, this implies that the non-configuration of a new potential rule is preferred with respect to the non-configuration of a previously used one.

Finally, it is worth noting that the proposed approach may only produce a solution that is optimal with respect to the network areas identified as to be reconfigured, and not an optimal solution in a global sense. Nevertheless, this limitation is compensated by the improved computation time, which represent a more critical parameter in the proposed scenario of a cybersecurity attack.

IV. Implementation and Validation

The proposed approach has been implemented as a Java-based framework, and the Z3 theorem prover [23] has been used to solve the formulated MaxSMT problem. The framework has been extensively tested to assess its correctness and its performance improvement with respect to the traditional approach for firewall configuration from scratch. The validation was carried out on synthetic networks of increasing sizes and various reconfiguration scenarios. The performance tests were designed to evaluate how much the results obtained with the proposed optimized reconfiguration approach deviate from those obtained with a state-of-the-art approach lacking support for optimized reconfiguration. For this purpose, the comparison has been done with the official implementation of VEREFOO [24]. The evaluation also covered the achievement of the optimality goals, quantifying the deviations of the proposed approach from the global optimum in terms of resource consumption. As mentioned in III-C, this approach

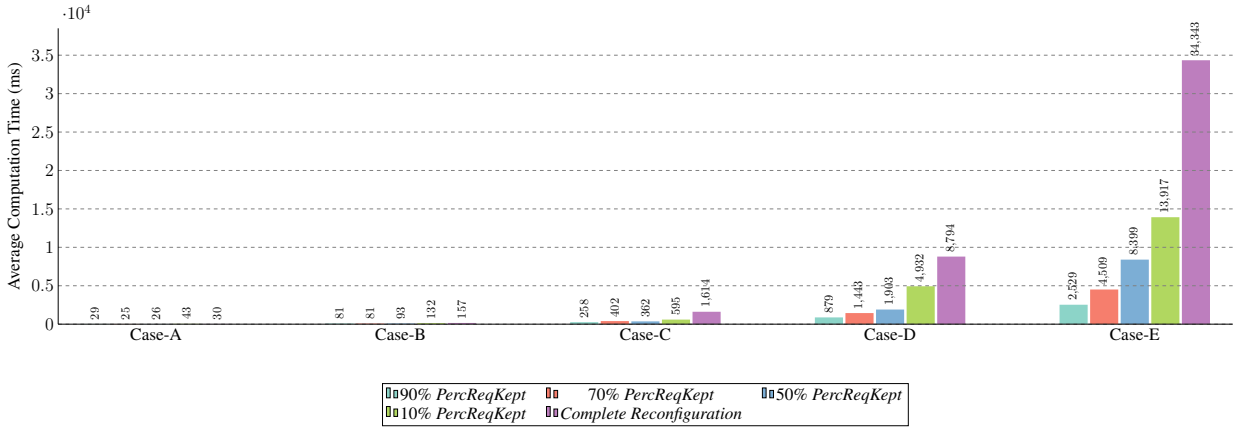


Fig. 4: Performance tests

considers a limited subset of the solution space, thus it may compute a configuration that is locally optimal concerning the reconfigured nodes but not in a global sense.

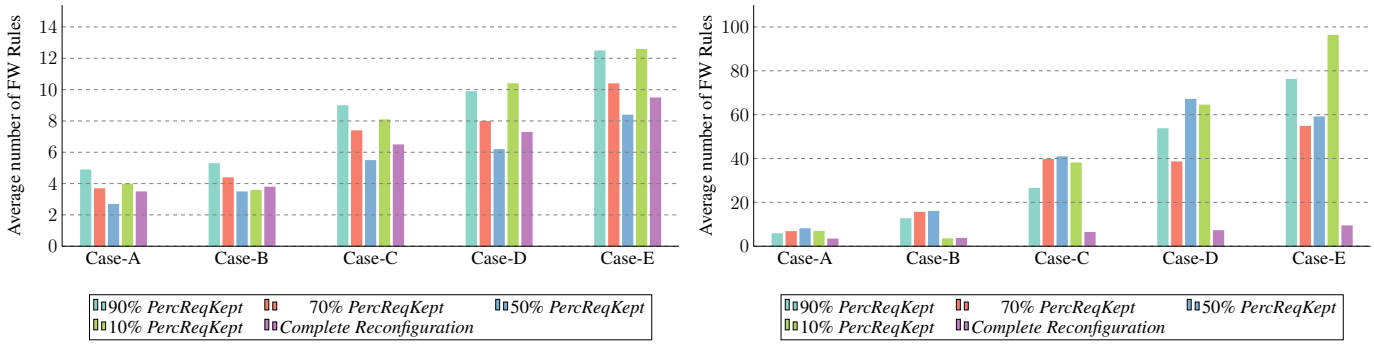
The main parameters used in the different test cases are the number of NSRs, the number of endpoints, and number of NATs (which introduce an additional complexity factory, ad they can modify the crossing traffic). Another important parameter is *PercReqKept*, which represents the percentage of requirements in the set $\mathcal{R}_k$ with respect to the complete set of NSRs. In other words, it represents the proportion of kept requirements with respect to the total number of defined NSRs. Clearly, if the percentage of requirements maintained between the Initial and Target sets of NSRs increases, then the number of NSRs in the added ($\mathcal{R}_a$) and deleted ($\mathcal{R}_d$) groups will consequently decrease.

The first analysis, shown in Fig. 4, compares the performance of the algorithm in five different types of networks, differentiated by an increasing number of endpoints, NSRs and NATs, and with four reconfiguration scenarios, differentiated by the value of the *PercReqKept* parameter. In particular, the five differents classes of networks that have been tested are: *case-A* with 10 NSRs, 60 enpoints and 5 NATs, *case-B* with 15 NSRs, 80 endpoints, 10 NATs, *case-C* with 20 NSRs, 100 endpoints, 15 NATs, *case-D* with 25 NSRs, 120 endpoints, 20 NATs, and *case-E* with 30 NSRs, 140 endpoints and 25 NATs. Then, the application of the framework to each of these network classes has been tested in four different reconfiguration scenarios, each characterized by a decreasing value of *PercReqKept*, and, as a consequence, an increased number of modified NSRs. The values adopted for this parameter are as follows: 90%, 70%, 50%, and 10%. Note that the tested scenarios are more concentrated on higher values of *PercReqKept*, as justified by multiple sources. For instance, [25] reports that updates in the Facebook infrastructure affect on average 157 lines of configuration, considering only the backbone, or 738 lines, if also data centers and edge servers are considered. Both numbers of lines are relatively small, when compared to the huge scale of their network. Instead, [26] questioned different large online service providers and found out that, for 75% of the networks operated by them, the median change includes only three devices.

For each value of *PercReqKept*, and for each type of network, the algorithm has been executed 100 times. Moreover, this analysis aimed to highlight the improvement versus an unoptimized approach, which is referred in the tests as the *Complete Reconfiguration* case and it corresponds to the vanilla version of VEREFOO.

The observed trend is that the computation time is directly proportional to the number of endpoints and NSRs, and inversely proportional to the percentage of kept requirements. Every considered reconfiguration scenario achieves an average computation time significantly lower than the approach adopted in the vanilla VEREFOO. The obtained results highlight that the main parameters increasing the computation time are the number of endpoints, the number of NSRs and, just for the reconfiguration case, the percentage of NSRs which are maintained, i.e., the *PercReqKept* parameter. Indeed, the Initial and Target NSR sets, once overlapped, form a shared region representing the group $\mathcal{R}_k$. The larger this area, the smaller the sets $\mathcal{R}_a$ and $\mathcal{R}_d$, representing the added and deleted NSRs, and fewer NSRs must be processed, making the reconfiguration process less expensive in terms of computation time.

These results show that the highest advantage in terms of computation time is obtained when the reconfiguration regards a small subset of the total NSRs. This is an expected result. In fact, the optimization improvement of the presented approach is mainly achieved by limiting the solution space that is considered by the solver. This reduction is achieved by fixing the configurations of some network elements which are unaffected by the new NSRs, shrinking the set of variables whose values must be determined with the resolution of the problem. If the modified NSRs represent a major part of the whole set, then the unaffected area is reduced, and the optimization effect is limited. This validation phase also assessed that the impact of the designed algorithm could be considered irrelevant when compared to the overall computation time, since its contribution always ranged between 0 to 100 ms, with most of the runs being under 10 ms. In general, the



(a) Results with ratio 2

(b) Results with ratio 10

Fig. 5: Optimality comparison

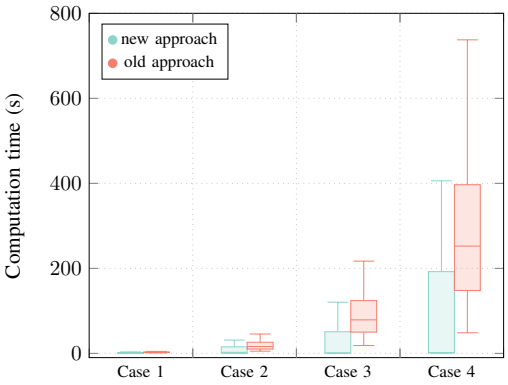


Fig. 6: Scalability tests

results confirm the feasibility of the proposed approach and its relevant advantages in terms of computation time when compared to the previous solution, based on a complete reconfiguration of the whole network.

In this phase, also the optimality of the solution has been analyzed. The results show that the reconfiguration approach achieves a slightly higher number of allocated firewalls and configured rules. The extent of this difference changes depending on the ratio between the weight assigned to a reconfigured node and that used for a new one. This is demonstrated by the additional tests shown in Fig. 5. Two different cases are represented here, one in which the ratio between the weight assigned to a new AP and the weight assigned to a reconfigured node is equal to 2, in Fig. 5a, and another case in which the same ratio is equal to 10, Fig. 5b. As we can see, increasing this ratio results in an increase for the number of generated firewall rules, and the same applies to the number of firewalls (even if not represented here). The sub-optimality of the result is due to two factors: first, the reduction of the solution space for the solver which is limited to the subset of allocation places that could be modified, and second, the soft constraints which force the preference of reusing old configuration elements even if a completely new configuration would result in a slightly better optimality. Note that the results for performance and scalability have been conducted using the value 2 for the ratio, which allows to reduce the computation time while achieving a nearly optimal usage of resources.

The second validation analysis tested the approach with larger networks and considering just a single reconfiguration scenario in which 70% of NSRs are kept from the Initial to the Target set. Fig. 6 compares the obtained average computation time for the proposed approach compared with the previous one. The considered networks types have increasing sizes, specifically the considered cases are from left to right: 200 NSRs and 40 endpoints, 300 NSRs and 60 endpoints, with 400 NSRs and 80 endpoints, and 500 NSRs and 100 nodes. As we can see, even considering the high variability of the obtained values, the reconfiguration approach performs well when compared to the previous one also in terms of scalability.

V. Conclusion and Future Work

This paper presented a novel approach for the optimized reconfiguration of distributed firewall systems. To the best of our knowledge, the proposed approach is the first one in literature to address that problem while combining three main features: full automation in computing the firewall reconfiguration, formal correctness assurance of the computed configuration, and optimizations in terms of resource consumption. The proposal was designed considering use cases of network attacks, requiring the computation of a new formally correct and secure solution within a short computation time, so as to limit the exposure of the systems to the attack. The proposed strategy has been implemented as a framework, whose validation showed benefits in terms of performance with respect to a state-of-the-art technique that automatically computes the firewall configuration from scratch.

As future work, the current methodology may be extended to the reconfiguration of other network security functions, such as anti-spam filters and web application firewalls. A longer-term work is also to exploit this solution as a starting point for the design of a parallelized approach for the resolution of the problem, dividing the network into independent areas that could be configured separately (and in parallel), and consequently breaking one single and complex problem into multiple smaller ones, each requiring a shorter computation time.

REFERENCES

[1] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, aug 2023.

[2] S. Singh, Y. Jeong, and J. H. Park, "A survey on cloud computing security: Issues, threats, and solutions," *J. Network and Computer Applications*, vol. 75, pp. 200–222, 2016.

[3] H. Tabrizchi and M. K. Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The Journal of Supercomputing*, vol. 76, p. 9493–9532, 2020.

[4] A. Paradowski, L. Liu, and B. Yuan, "Benchmarking the performance of openstack and cloudstack," in *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2014, pp. 405–412.

[5] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source nfv mano systems: OSM and ONAP," *Computer Communications*, vol. 161, pp. 86–98, 2020.

[6] Cloudflare, "DDoS attack trends for 2022 Q2," 2022. [Online]. Available: https://blog.cloudflare.com/ddos-attack-trends-for-2022-q2/

[7] Proton, "A brief update regarding ongoing DDoS incidents," 2022. [Online]. Available: https://proton.me/blog/a-brief-update-regarding-ongoing-ddos-incidents

[8] 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.

[9] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: Incrementally synthesizing policy-compliant and manageable configurations," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 482–495.

[10] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, D. Yu, C. Tian, H. Zheng, and B. Y. Zhao, "Safely and automatically updating in-network acl configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 214–226.

[11] F. Chen, A. X. Liu, J. Hwang, and T. Xie, "First step towards automatic correction of firewall policy faults," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, 7 2012.

[12] 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.

[13] K. Adi, L. Hamza, and L. Pene, "Automatic security policy enforcement in computer systems," *Computers & Security*, vol. 73, pp. 156–171, 2018.

[14] W. T. Hallahan, E. Zhai, and R. Piskac, "Automated repair by example for firewalls," *Formal Methods in System Design*, vol. 56, pp. 127–153, 12 2020.

[15] M. Cheminod, L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A comprehensive approach to the automatic refinement and verification of access control policies," *Computers & Security*, vol. 80, pp. 186–199, 1 2019.

[16] M. A. Rahman and E. Al-Shaer, "Automated synthesis of distributed network access controls: A formal framework with refinement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 416–430, 2017.

[17] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7.

[18] ——, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023.

[19] S. Bussa, R. Sisto, and F. Valenza, "Security automation using traffic flow modeling," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 486–491.

[20] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, pp. 1–11.

[21] 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.

[22] 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.

[23] 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.

[24] VEREFOO, "Verefoo github page." [Online]. Available: https://github.com/netgroup-polito/verefoo/

[25] Y.-W. E. Sung, X. Tie, S. H. Wong, and H. Zeng, "Robotron: Top-down network management at Facebook scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 426–439.

[26] A. Gember-Jacobson, W. Wu, X. Li, A. Akella, and R. Mahajan, "Management plane analytics," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 395–408.