Optimizing distributed firewall reconfiguration transients

Daniele Bringhenti[a,*], Fulvio Valenza[a]

[a] *Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy*

Abstract

The flexibility and dynamism brought over by softwarization for network management have increased the frequency of configuration changes. In this context, when a distributed security function is subject to a series of configuration changes, a problem that arises is the preservation of the security. The transient from the application of the first change to the last one may present unsecure temporary states, where the required security protection is missing. Establishing a safe scheduling of the configuration changes can significantly limit the number of unsecure states and decrease the time period where the network may be at risk. However, the literature challenged this problem only for centralized firewalls or SDN switches, and without applying formal methods to ensure the correctness of the computed scheduling. In order to overcome these limitations, this paper addresses the problem for distributed firewalls, aiming to satisfy the largest number of user-specified network security policies in each transient state. To this end, it proposes a formal methodology relying on the combination of three main features: automation, formal verification and optimization. This combination is achieved by pursuing a correctness-by-construction approach, based on the formulation of a Maximum Satisfiability Modulo Theories problem. A framework has been developed on the basis of this methodology, so that validation tests have been experimentally executed to assess the feasibility, efficacy and scalability of the approach.

Keywords: transient management, firewall, virtual networks security

1. Introduction

In the last decade, network softwarization has deeply revolutionized the traditional approaches for both computer network management and security configuration. The two most well-known declinations, *Network Functions Virtualization* (NFV) and *Software-Defined Networking* (SDN), have brought over important contributions, such as the agility and resilience that network and security administrators had been always searching for. The orchestration and security configuration of large-scale complex networked services have been progressively becoming feasible, since human beings are currently aided by automated software [1]. In this scenario, network updates have become extremely frequent (e.g., the creation of a mirror server or the removal of a deployed function). Consequently, the security must be updated at the same speed as the network. The redistribution of the security functions in the logical topology of the virtual network and their reconfiguration are now more frequent and happen in quite strict times.

This high dynamism characterizing the management of modern computer networks has severely impacted the preservation of the security during the updates [2]. A security reconfiguration commonly occurs when new security policies, describing the security behavior of the network, must be enforced. When this configuration update affects a distributed security function (e.g., new instances of the function must be deployed in the network, or the rules installed in some instances must be modified), a transient

*Corresponding author

Email addresses: daniele.bringhenti@polito.it (Daniele Bringhenti), fulvio.valenza@polito.it (Fulvio Valenza)

starts from the moment the first configuration change is applied to when the last one has been applied. In-between the initial and last change, there are several intermediate states where the configuration of the distributed function has already been modified with respect to the initial one, but the update is not yet complete. For example, a function instance has been already removed, but a new one has not yet been installed. The order in which the configuration changes are applied in this period of time might result in unsecure transient states, i.e., states where the security policies are violated. In particular, a type of network security function for which this problem is critical is the distributed packet filtering firewall, the most commonly used function to enforce connectivity security policies, i.e., to establish which packets are allowed or must be blocked in a network [3]. Regarding the enforcement of these policies, an incorrect scheduling of the changes for the configuration of a distributed firewall might result in a provisional service interruption or, even worse, in a temporary breach which attackers might exploit, thus endangering the systems connected to the network. As the presence of transient unsecure states may be impossible to avoid, a network administrator would want to minimize them as far as possible, depending on which policies they prioritize.

Minimizing these unsecure states is especially important for environments, such as industrial networks leveraging network virtualization [4], where the presence of safety-critical and mission-critical systems requires a guarantee for the correctness of the security management operations performed in this context. In those networks, the violation of a connectivity policy might lead to dramatic consequences (e.g., a device should always be reachable during a security update, or a network component victim of an attack should be immediately isolated from the rest of the network). Nonetheless, having the guarantee that the connectivity policies are enforced in as many transient states as possible would be important for all types of virtualized networks. Therefore, it is necessary that the problem of establishing a correct scheduling of the firewall configuration changes is challenged in a formal and provably correct way. Formal verification solutions have been proposed in literature to verify network invariants [5] [6] [7], and they might be applied after the scheduling of the configuration changes is computed. However, if then any issues are identified, the scheduling should be modified and verified again, and this process might be repeated multiple times. Instead, a better solution would be that the scheduling computation would bring alongside a formal proof of its correctness by pursuing a correctness-by-construction approach [8], so that a single occurrence of this operation would be sufficient.

In light of all these motivations, this paper faces the open problem of optimizing firewall reconfiguration transients by proposing a novel methodology, called ***Firew**All **T**ransients **O**ptimizer* (FATO). The aim is to define a scheduling of the configuration changes for a distributed virtual firewall, that maximizes the number of secure states, i.e., states where the minimum number of connectivity security policies is violated, depending on their priority. Optimality and higher confidence in the solution correctness are achieved throughout the formulation of a *Maximum Satisfiability Modulo Theories* (MaxSMT) problem, a traditional approach to pursue correctness-by-construction [9]. The optimality obtained by MaxSMT also allows to go beyond the sub-optimality provided by heuristics, which represent the most common way to face a network update problem.

The reminder of this paper is structured as follows. Section 2 overviews the related literature, and remarks the novelties of this contribution. Section 3 states the problem challenged in this work precisely. Section 4 describes the approach that is pursued to face the problem. Section 5 formalizes the models of network components and security policies. Section 6 illustrates how a ranking is generated to be associated to the security policies. Section 7 formu-

lates the MaxSMT problem representing the scheduling problem for the distributed firewall configuration changes. Section 8 describes the implementation based on the proposed approach, and shows how its effectiveness, scalability and optimization have been validated. Section 9 discusses the applicability of this approach to other contexts. Section 10 concludes the paper and outlines future work.

2. Related Work

State-of-the-art studies have broadly investigated the problem of guaranteeing that some network security properties are still valid during a network reconfiguration, as an exhaustive review of this topic discusses [2]. In particular, during a transient unfolding when a network configuration changes, there are three types of security properties that may be violated: i) when the network fails in delivering packets to their respective destinations (e.g., because an intermediate node has been temporarily shut down before installing a new function), *connectivity consistency* is not preserved; ii) when packets can still reach their destination, but not through a specific path of middleboxes during the whole transient, *policy (or path) consistency* is not preserved; iii) when availability and boundaries of resources as bandwidth and latency are not guaranteed due to transitory states, *capacity consistency* is not preserved.

The methodology presented in this paper contributes to improving this research area's state-of-the-art with a broader generalization of the connectivity consistency. All past works aim at guaranteeing the consistency of a single category of connectivity policies, called reachability policies [10]. This property namely aims at preserving the reachability among network nodes, thus avoiding service disruptions. Instead, we also consider isolation policies, in order to deal with attack vectors such as undetected intrusions, which may also lead to access control violations and privilege escalation [11].

This paper challenges this problem for reconfiguring virtual packet filtering firewalls, the most commonly used function for enforcing connectivity policies. The main gaps of the literature that we propose to fill are the following: i) to address the transient problem for distributed firewalls, as currently it has been challenged only for centralized firewalls (see Subsection 2.1 for details); ii) to extend the literature of transient management in programmable networks, overcoming the limitations deriving of existing works about the reconfiguration of SDN switches (see Subsection 2.2 for details); iii) to leverage formal modeling to define a representation of the behavior for the network where the firewall instances are installed (see Subsection 2.3 for details); iv) to apply formal verification to provide assurance of the solution correctness (see Subsection 2.4 for details).

2.1. Transients in intra-firewall reconfiguration

The literature related to the management of a reconfiguration transient for packet filtering firewalls is limited. The studies challenging this problem ([12], [13], [14], [15], [16]) exclusively focus on centralized firewalls. Therefore, the transient problem is reduced to a simpler intra-firewall policy deployment, where the firewall configuration is meant to be only its rule set. The only objective is to identify a safe scheduling of the update operations for the filtering rules of a single firewall. The operations that are considered by these studies are rule appending or deleting for firewalls of Type I ([12], [13], [14]), and additionally rule moving for firewalls of Type II ([12], [15], [16]).

This lack of studies represents a gap that needs to be filled, because the complexity of transient management for inter-firewall reconfiguration is much higher. Inter-firewall reconfiguration takes longer, so the number of transient states is higher and passing through unsecure states is more dangerous. This complexity derives from a larger number of reconfiguration operations that must be considered (e.g., deployment of new instances, removal of useless instances). Besides, the intra-firewall problem becomes

even less important in virtualized networks, as it is easier and faster to launch a new virtual firewall with the new filtering rule set to replace the old one, instead of modifying single rules. Therefore this paper aims to address the problem for inter-firewall reconfiguration.

Besides, the approaches described so far lack formal verification: they are simply based on greedy algorithms, which cannot provide correctness of the computed scheduling, and their result might not be the most efficient by their admission [12]. In all this, the only type of security policy that is considered by these approaches is an internal coherence of firewall policy states, i.e., any packet that is permitted (or denied) by both the initial and target policies is always permitted (or denied) in the intermediate policies of the reconfiguration transient. Consequently, differently from our work, they never consider connectivity policies, which would require considering the connection of a firewall to a network where multiple communications might happen among the connected devices. Lastly, these papers mainly targeted traditional firewalls: situations deriving from network softwarization, such as the usage of containers (e.g., Dockers) that require the deployment of a new process if the configuration of the previous one must be changed, are totally overlooked.

2.2. Transients in SDN network reconfiguration

The advent of SDN has revived interest in the transient management problem, casting it into the context of programmable networks [17]. Automation has become a critical factor in reducing operational times, so reconfiguration of network and security functions can easily occur with a higher frequency than with hardware-based middleboxes. This possibility has allowed reaching better scalability in network management [18], improve elasticity control of network functions [19], and automatize their configuration [20].

Managing reconfiguration transients is a problem that evidently fits the current research trend. Nevertheless, it has been scarcely studied for distributed virtual functions. In fact, the problem of transient optimization for distributed functions has been investigated in the literature only for SDN switches so far [2]. These studies are limited only to two specific types of policies. The most common type, addressed by the majority of related papers ([21], [22], [23] [24], [25], [26], [27]), is the *Per-Packet Consistency* (PPC) policy. In this case, only two paths must be crossed by the packets: one related to the original configuration of the SDN switches, the other one to the final configuration after the changes. Therefore, this kind of policy requires that every packet travels either on its initial or final path, never on intermediate ones. However, PPC might result too restrictive in some instances (e.g., sometimes it is sufficient to guarantee that the traffic passes through a firewall, independently of the other crossed functions). As such, in some papers ([28], [29]) a mitigation of PPC is represented by the *Way-Point Enforcement* (WPE) policy, for which it is just required that the packets can always cross a set of specific waypoints during the transient. There also exist studies ([30], [31]) where policies are more complex (e.g., it may be required that, in the network of a large Internet Service Provider, specific traffic flows follow certain sub-paths). However, in all the mentioned cases, the specification of connectivity policies is not addressed. This shortcoming translates into a limitation for the proposed approaches, because connectivity policies represent the most general type of security policies through which it is possible to check the reachability or isolation of traffic flow (e.g., checking if a reachability policy is satisfied in a transient state means to analyze all the possible paths where the traffic might flow, and consider the behavior and configuration of all the possibly crossed functions).

Focusing exclusively on SDN switches is a limitation that our approach aims to overcome by addressing the transient management problem for more general packet filtering firewalls. On the one hand side, studies about SDN

switches address security issues concerning the violation of connectivity policies only partially, because the configuration of SDN switches is mainly defined to challenge networking issues with respect to firewalls. On the other hand, they overlook the analysis of the impact that the behavior of other network or security functions, which are present in the network, may cause to the reconfiguration transient. Moreover, the approach proposed in this paper can enforce optimization criteria (e.g., maximization of the secure transient states depending on the importance of each connectivity policy), thanks to the problem formulation as MaxSMT. Instead, in most of the studies analyzed so far only heuristics and greedy approaches are pursued, except for [28] and [29], where exact algorithms based on a *Mixed-Integer Program* (MIP) formulation are defined for computing an update scheme requiring the minimal number of intermediate states. Therefore, differently from most state-of-the-art approaches, the method proposed in this paper can guarantee that the computed solution is optimal with respect to the defined cost functions.

2.3. Formal models of network service graphs

The approach proposed in this paper relies on the formalization of network service graphs where the transient unfolds. The previously described studies related to the transient management problem for intra-firewall or SDN network reconfiguration do not lay their foundations on formal models. However, formalization of network graphs is common in the literature related to security optimization against cyber-attacks. Examples of relevant state-of-the-art approaches that use formal models of network service graphs are [32], [33] and [34]. However, despite the soundness and richness of their models, they differ from the ones proposed in this paper, as they could not be adapted to solve the firewall transient management problem. On the one hand, [32] and [33] model network graphs with a representation named resource graph. This formalization is syntactically equivalent to attack graphs, as those modeled in [35], but it models network resources instead of vulnerabilities differently from the latter. Resource graphs also model possible changes to be applied to the network topology, e.g., the removal of existing services, the addition of services not initially present, and the relocation of services from one host to another. On the other hand, [34] pursues a time-driven approach to capture network states at different times. However, these formal approaches are not enough for managing a transient deriving from a firewall reconfiguration. Solving this problem also requires modeling the traffic flows crossing the network, because intermediate functions can modify packets before reaching a firewall instance. Therefore, differently from them, we also model the behavior of other network and security functions (e.g., load balancers, network address translators) which compose the network service where the reconfiguration transient occurs. Besides, our approach specifically introduces models that are compliant with a ranking of the isolation and reachability policies that must be guaranteed in the network, so that they can have different relative priorities.

2.4. Application of formal verification

Most of the cited papers do not enforce formal verification techniques (e.g., model checking or state exploration) to increase the confidence in the correctness of the solution for the transient problem. Instead, the methodology proposed in this paper is inspired by the many approaches that in literature have been proposed to formally verify network security policies in computer networks ([36, 37, 38, 5, 6, 7]). However, the scope of these approaches is different with respect to the work described in this paper, because they lay their foundations on performing formal verification on static snapshots of the network data plane. However, pursuing a similar modeling strategy and embedding it in the context of a correctness-by-construction approach is helpful to solve the transient problem in a formal way, through which the behavior of the entire network is

analyzed before the solution to the problem is established. Thanks to this approach, we can provide higher confidence in guaranteeing that the computed solution is effectively correct with respect to studies that do not leverage formal methods.

3. Problem Statement

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

3.1. Characterization of a firewall reconfiguration transient

The configuration of a distributed packet filtering firewall requires a security manager to contemplate two management aspects at the same time: the establishment of the allocation scheme and the definition of the filtering rules.

The first aspect, i.e., the definition of an allocation scheme, involves deciding the positions where the firewall instances should be allocated in the network service. Instead, the second aspect consists of generating the filtering rules through which each firewall can decide if the input packets should be discarded or forwarded to the next hop in the path towards their destination. The firewall allocation scheme and filtering rules are typically established to enforce some connectivity policies, i.e., security properties for the communications that might happen in the network.

Since the configuration of a distributed packet filtering firewall requires considering both these management aspects, it is clearly a complex task. It also requires a high level of expertise if manually managed by a human being. Nevertheless, multiple approaches have been presented in the literature, where this task is performed automatically without the need of manual interventions ([39], [20]). These contributions have eased the work of security managers and, at the same time, have strengthened the security guaranteed by such a kind of network functions. In light of this, nowadays it is easier and more common to establish a new firewall configuration, when the original configuration is not valid anymore.

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

3.2. Issues of a firewall reconfiguration transient

Under real-world constraints, the security preservation of the connectivity policies during reconfiguration transients becomes an important matter when the time length of these transients is not negligible. If a transient lasts only a few seconds, the problem is less felt because, in such a short span of time, an attacker could not easily perform access control violations, privilege escalations, or other at-

tack types that undermine the connectivity policies. However, this is not the typical case of virtual networks.

From the analysis of the studies discussed in Section 2, typical transients that have been evaluated are composed of a few tens of states. Each state consists of the deployment/removal of a virtual function (e.g., a softwarized SDN switch, a Virtual Machine) or updating the function rules (e.g., all the rules of a virtual firewall). The time required for these operations may be not negligible. On the one hand, Openstack requires more than 5 seconds to deploy a single machine [40], and it has an update rate of 250 rules per second [41], if all the rules pertain to the same machine. On the other hand, a well-known NFV orchestrator, i.e., Open Source MANO, requires a Deployment Process Delay (DPD) of 134s [42], where DPD is the time the orchestrator takes to deploy and instantiate a VNF within an already booted VM and setup an operational network service. Therefore, supposing that a transient is composed of 20 states and each of them consists of the deployment of a virtual machine, the transient may require around 100 seconds in an environment based on Openstack, more than 10 minutes with Open Source MANO. Parallelization may improve these worst-case times, but not drastically. Transients requiring some minutes are common in big virtual networks, and these long times are perfect chances for intruders to exploit intermediate states where services are not protected.

During the reconfiguration transient, not all the connectivity policies (i.e., reachability and isolation policies) might be satisfied in each intermediate state, even though preserving their satisfaction as much as possible would be required to ensure a higher level of security during the configuration. In particular, two well-known issues that may occur due to this problem are the following ones.

The first issue is *service disruption*. For example, we can suppose that a service is linked to clients external to its subnetwork through two paths, guarded by a firewall each, but only one of them has an allowing rule for communications between the server and the clients. If the firewall reconfiguration establishes that the allowing rule must be removed from the original firewall node and added to the other one, if the latter operation is executed some seconds or minutes after the former, the service is not available anymore to be accessed.

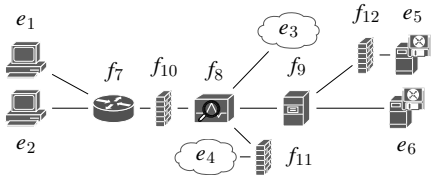


Figure 1: Network topology example

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

3.3. Motivating example

These problems can also be explained with the aid of a motivating example. Figure 1 represents a snapshot of the configuration of a distributed firewall, with three instances, in a network whose topology is a simplified version of a real one, i.e., the network of our university department. At a certain time, a cyber-attacker manages to take over node e_2, hosting a *mysql* service. This event demands to block any communication towards e_2, and at the same time to make the *mysql* service hosted by node e_5 available for any other network component, as it is a mirror of that in e_2. The security manager is thus required to perform multiple changes to the firewall configuration: (i) a new instance must be deployed between e_2 and f_7 to make the former inaccessible; (ii) f_{12} must be removed, as

e_5 must become the replacement of e_2; (iii) the rule sets of f_{10} and f_{11} must be updated, so as to allow traffic respectively between e_1 and e_4 on one side, and the *mysql* service listening to port 3306 on the other side.

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

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

3.4. Solutions to improve transient management

In light of all these considerations, it would be thus crucial to identify the ordering of the configuration changes which would maximize the security for the intermediate states of the transient from an initial configuration to a different one. In our vision, this objective translates into maximizing the number of connectivity policies that are satisfied in each transient state. The advantage of this objective definition for the transient optimization is twofold. On the one hand, each state would result to be as much secure as possible, because the majority of the security requirements for which the reconfiguration occurs are enforced in it. On the other hand, in almost all the cases, the security policies are also satisfied as early as possible, because the enunciate objective aims at their enforcement since the first states of the transient. For example, when the position of a firewall must be changed to block a specific traffic flow, the reconfiguration transient involves two operations as it is common practice in virtual networks, i.e., removing the current instance and deploying the new one in the required position. Only performing the latter operation before the former satisfies the optimization objective previously expressed, because in this way the isolation policy related to that traffic flows is already satisfied, even before removing the old instance.

However, establishing an optimal scheduling of a distributed firewall reconfiguration changes, compliant with the definition of this objective, is commonly unfeasible and unbearable for human beings, due to the composite complexity of all the involved parties (i.e., the initial and target security service topologies, the connectivity policies). Therefore, this task should be automated so that the scheduling is automatically computed based on information related to the network topology and the connectivity policies. Additionally, by automatizing this task, it is possible to enrich the process with two other important factors, i.e., optimization and formal verification. On one side, commonly some policies are more important than others. This aspect should thus be considered when computing the scheduling of the operation changes, but at the same time represents another factor that makes a manual operation impractical or deeply unoptimized. On the other side, providing higher assurance that the computed scheduling is correct would be an important feature for environments where safety-critical or mission-critical systems are present.

To summarize, automation, paired with formal methods and optimization, comes in handy for overcoming these limitations, and reaching a solution in fast times with

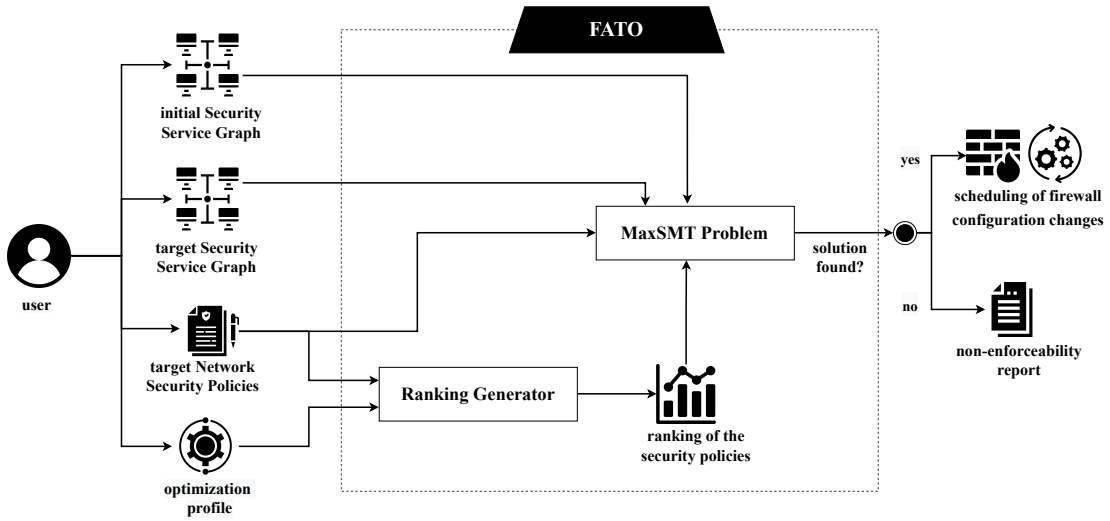


Figure 2: Workflow of the approach

higher confidence in its correctness.

4. The Proposed Approach

This section describes the approach pursued in the definition of the ***F**irew**A**ll **T**ransients **O**ptimizer* (FATO) methodology, which aims to solve the stated problem automatically. Figure 2 depicts the full workflow of the proposed approach, showing the inputs specified by the user (Subection 4.1) and the interaction among the different components of the FATO methodology (Subsection 4.2).

4.1. Inputs for FATO

FATO requires the following inputs:

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

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

source and destination ports, transport-level protocol) and the corresponding action to be enforced when the conditions are satisfied. The firewall rules belonging to the initial and target configuration are respectively called *initial Firewall Rules* (R_I) and *target Firewall Rules* (R_T). All the information captured by these inputs (e.g., the network topology, the traffic flows crossing them, the firewall configurations) is formally represented in the FATO methodology through the models illustrated from Subsection 5.1 to Subsection 5.4.

The third input is a set of connectivity policies, in this paper also called *target Network Security Policies* (P_T), which the target firewall configuration must enforce. The connectivity policies establish which packet flows, identified by the values of the IP 5-tuple fields, must reach their destination, and which ones must instead be blocked. Therefore, they can be respectively divided into reachability and isolation policies. The P_T set includes both persistent and non-persistent policies. The formal model for P_T is illustrated in Subsection 5.5.

The fourth input is an optimization profile, i.e., a compact indication for FATO about the relative priority of the connectivity policies. Some profiles additionally require that the user specifies a partial or total order relationship for P_T (i.e., the specification of the fact that each policy or group of policies is more or less important than one or a group of other policies). In particular, the following profiles have been defined, but more can be defined:

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

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

4.2. FATO Methodology

The FATO methodology works as follows.

Firstly, from the specification of the optimization profile and, optionally, the partial or total order relationship for the policies, FATO defines a ranking for the input policies (with the exclusion of the persistent policies, because they must be enforced in any intermediate state), as it comes handy for the definition of the optimization problem. The same rank can be assigned to multiple policies, if they have the same relative priority. The user could have personally defined this ranking. Still, such an operation would not have been suitable to be manually performed by a human being. Possible reasons may be that the number of policies may be high, or the person in charge of this task may want to specify only a partial order relationship between the policies instead of a complete ranking. Additional information about the ranking generation is provided in Section 6.

Then, the initial and target security graphs with the firewall configurations, the target policies and their ranking are used by FATO to formulate a *Maximum Satisfia-*

bility Modulo Theories (MaxSMT) problem. With respect to the traditional Satisfiability (SAT) problem which is well-known in mathematical logic, an SMT formulation is based on predicate logic, and it exploits additional theories than the only Boolean algebra (e.g., integer and string theories). Additionally, a MaxSMT formulation differs from a standard SMT formulation because it is composed of two types of constraints. A first category is made by hard constraints, which must always be satisfied to find a correct solution to the problem. If at least a hard constraint is not fulfilled, then the problem is unsatisfiable. Other clauses, called soft constraints, do not strictly require satisfaction for reaching a solution. They are given a weight, and the objective is to maximize the sum of the weights assigned only to the soft constraints that are effectively fulfilled.

On the basis of this definition, the FATO methodology builds a MaxSMT problem composed of the hard and soft constraints illustrated in Section 7. Throughout this formulation, the aim of the methodology is to maximize the number of intermediate states in the transient from G_I to G_T where the policies of the P_T set are enforced. After solving this optimization problem, FATO identifies the optimal order of configuration changes, in such a way that the optimization fulfills the criteria derived from the ranking (i.e., from the optimization profile and the order relationship for the policies, specified by the user). This scheduling can be followed by a human who manages the virtual network, or a state-of-the-art orchestrator can exploit it to perform the required actions.

According to the correctness-by-construction principle enabled by MaxSMT, the computed solution is correct, as long as the formal models defined for the problem (i.e., the set of first-order logic formulas that express them) correctly model the real problem. Specifically, the formal models whose correctness impacts the correctness of the solution itself are those of network functions and traffic. About the former, in literature approaches, such as [43], are already defined for automatically extracting formal models of network functions starting from a representation, expressed in a high-level programming language like Java, of a given virtual function. With these approaches, it is possible to get models that provide high confidence about their adherence to the function behavior. About the latter, the traffic model proposed in Subsection 5.3 is intuitively coherent with the representation of network packets, as it is based on the five traditional fields of the IP 5-tuple. For all these reasons, the FATO methodology can provide higher confidence in the solution correctness than a solution computed manually or with a standard algorithm.

Finally, there might be cases where the FATO methodology cannot compute a solution. A first case of error occurs when the initial and target security graphs specified by the user have totally different topologies. For FATO, the problem is formulated in a way that, whenever a scheduling can be successfully computed for the firewall reconfiguration changes, the number of transient states is finite and predetermined, as it corresponds to the number of reconfiguration changes itself. In case the topologies of the initial and final graphs are totally different, then the transient length would not be predetermined. However, this specific case represents an error condition for the problem, because we are interested in managing reconfiguration transients specifically for changes related to distributed firewalls, not to other functions that are present in the network. A second case of error occurs when no solution that satisfies the user requirements can be computed for a problem instance, e.g., because the input connectivity policies are conflicting or a persistent policy is violated in at least a transient state. Under these circumstances, FATO notifies the user with a non-enforceability report about the unsatisfiability of the problem, so that they can modify the inputs accordingly.

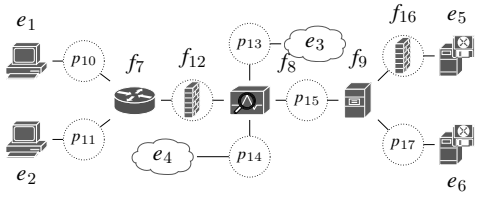


Figure 3: Security Service Graph with firewall allocation scheme

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

Table 1: Notation

5. Formal Models

A formalization of the network components and the security policies is required so that the solution computed for the MaxSMT problem can be formally correct. Therefore, this section deals with the illustration of their formal models. The formal models of traffic flows, of network functions and partially of security policies are mutated from the modeling approach proposed in [7]. TABLE 1 includes the main formal notations (symbols, functions, predicates, operators) used in the next sections.

5.1. Security Service Graphs model

The initial Security Service Graph is modeled as a directed graph $G_I = (N_I, L_I)$ where N_I is the set of vertices, representing the network nodes, while L_I is the set of edges, representing directed connections between nodes. This graph represents how the allocation scheme of the distributed firewall is defined at the initial state of the transient, and it is internally represented in the FATO methodology as follows. Assuming that Figure 3 depicts a Security Service Graph, the dotted circled boxes appearing in the picture, called *Allocation Places* (APs), represent all the logical positions where a firewall instance may be positioned. In this example, two of them are effectively filled with firewalls (i.e., f_{12} and f_{16}). The others are empty, but some instances may be deployed there in a Security Service Graph representing a different state of the transient (e.g., the final one).

Focusing on the two sets composing the G_I model, the N_I set includes end points among which communications might happen, service functions such as load balancers and traffic monitors, the APs that have not been filled with a firewall instance, and the firewalls that instead have been allocated. Each element of N_I is uniquely identified by a non-negative integer index k, and n_k denotes the element of N_I identified by k. In this way, each element of L_I is uniquely identified by a pair of non-negative integers, i.e., $l_{ij} \in L_S$, with $i \neq j$, is the edge from n_i to n_j. To this end, it can be defined that $index(n_k) = k$.

The target Security Service Graph is similarly defined as a directed graph $G_T = (N_T, L_T)$. As for N_I, each element of N_T is uniquely identified by a non-negative integer index k, and $index(n_k) = k$. It is possible that an element of N_T has the same index as an element of N_I, and it is possible that an element of L_T is denoted by the same pair of non-negative integers as an element of L_I. This simply means that the elements are the same for both the graphs.

At this stage, the concept of *union Security Service Graph* (G_U) is introduced. It is a directed graph modeled as $G_U = (N_U, L_U)$, where $N_U = N_I \cup N_T$ and $L_U = L_I \cup L_T$. It basically represents the union of the initial and target graphs, where a single instance of the nodes with the same

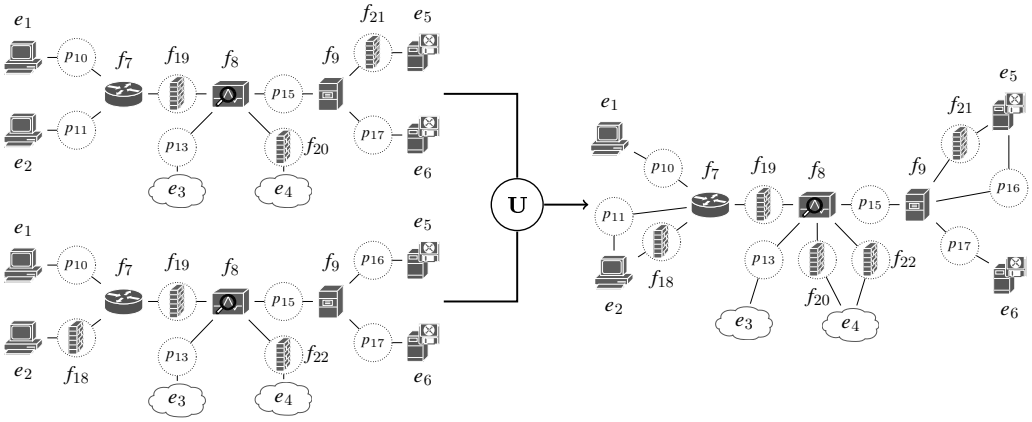


Figure 4: Generation of the union Security Service Graph

index which appear in both is kept, and the same is applied to the links. It is evident that, in this graph, nodes which are not active at the same time might be present (e.g., a firewall which was present in G_I, but has been removed in G_T). Nevertheless, this representation is useful to consider all the possible paths in any situation, independently of the moment when a node might be active.

A clarifying example of how G_U is computed from the input G_I and G_T is represented in Figure 4. In their models, the APs that have not been filled with firewall instances are still present, but they have the simple role of forwarders, i.e., it is as if they forward each received packet to the next hop. Their presence eases the formalization of the state models and the hard constraints of the MaxSMT problem. In the automatically derived G_U, it is possible that two firewalls, with different indexes, refer to the same logical position (as for nodes f_{20} and f_{22}). This means that, even though they share the same position in the allocation scheme, they have different rules, so they are distinct entities in the model.

To this regard, in some types of virtualized networks (e.g., where firewalls are implemented as Virtual Machines, and not as containers), the configuration of a single instance is allowed, without the need to instantiate a new process replacing it. This possibility is modeled with the $configUpdate : N_U \times N_U \rightarrow \mathbb{B}$ predicate, which is true if the two input nodes of N_U share the same AP and they represent two different configurations of the same virtual firewall instance. Referring to the previous example of Figure 4, if G_U derives from the representation of an NFV-based network, and f_{20} and f_{22} represent two different configurations for the same instance, then $configUpdate(f_{20}, f_{22}) = true$.

5.2. State Sequence model

In the transformation of G_I into G_T, four elemental configuration changes have been considered: (1) deployment of a new firewall instance, (2) removal of an existing firewall instance, (3) update of the filtering rules for a firewall instance, (4) deviation of the traffic that was sent to a removed firewall instance to another instance that has been deployed. In modeling the sequence of states characterizing the transient due to the distributed firewall reconfiguration, however, it is enough to consider the number of changes of type (1) and (2), i.e., deployment and removal of single instances. The reasons why the other change types are not explicitly represented for the state sequence models are the following.

Regarding the update of the filtering rules for a firewall instance, this event cannot be always performed instantaneously. If a firewall implemented as a virtual machine in a NFV-based networks could be effectively updated, the same does not apply to a container, where there is instead the necessity to launch a new firewalling process, with the

newly required rule set. Therefore, the rule update can be represented as a combination of firewall deployment and removal operations, and for the NFV-based example it is enough to introduce a specific hard constraint in the MaxSMT problem so that during the rule update other configuration changes are not performed (more details are presented later, in Subsection 7.2).
Regarding the deviation of a traffic flow so that it can reach a different firewall located in the same AP where a removed one was previously present, this operation can be represented as well through a hard constraint. This constraint can impose that the new instance becomes effectively active (i.e., it can receive and forward packets) only after the previous one has been deactivated (i.e., it cannot receive anymore the traffic).

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

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

Once computed c, a state sequence S is created, defined as a list $S = [s_0, s_1, ..., s_{c-1}, s_c]$. Each $s \in S$ is a state of the transient, and in-between two consecutive states a single action (i.e., firewall deployment or removal) is performed.

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

5.3. Traffic Flows model

A class of packets (also called traffic in this paper) $t \in T$, where T is the set of all the possible packet classes, is modeled as a conjunction of five predicates that are defined over the IP 5-tuple packet fields. The formal representation of a traffic t is the following:

$$t = (IPSrc, IPDst, pSrc, pDst, tProto)$$

According to this formalization, a packet belongs to class t if and only if each field of its 5-tuple satisfies the condition imposed by the corresponding predicate ($IPSrc$ stands for the predicate upon the source IP address, $IPDst$ stands for the predicate upon the destination IP address, $pSrc$ stands for the predicate upon the source transport-level port, $pDst$ stands for the predicate upon the destination transport-level port, and $tProto$ stands for the predicate upon the transport-level protocol). Additionally, given two

packet classes $t_1 \in T$ and $t_2 \in T$, $t_1 \subseteq t_2$ (i.e., t_1 is a sub-traffic of t_2), if t_1 represents a subset of the packets represented by t_2.

The traffic model represents the basic building block for modeling the concept of traffic flows. Given the set of all the possible traffic flows F, a flow $f \in F$ represents a class of packets that are generated by a source endpoint $n_s \in N_U$, directed to a destination endpoint $n_d \in N_U$, and steered to pass through an ordered list of intermediate nodes $n_a, n_b, ... \in N_A$ that may forward them at each hop, possibly changing them, or dropping them. Accordingly, a flow is formally modeled as a list: $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d]$, where t_{ij} represents the traffic (i.e., class of packets) transmitted from n_i to n_j, and each t_{ij} is the result of the transformation of the previous traffic in the flow by node n_i. In this formalization, all the packets belonging to the class t_{ij} are handled in the same way by n_j, i.e., either all of them or none of them are dropped and, if n_j applies different transformations to different classes of packets, they belong all to the same class.

Two auxiliary functions based on the traffic flows models and required for the definition of some hard constraints in the MaxSMT problem are illustrated in the following:

- π: $F \rightarrow (N_U)^*$ associates a flow f to the ordered list of nodes that are traversed by packets belonging to f;
- τ: $F \times N_U \rightarrow T$ maps a flow f and a node n to the packet class, belonging to f, that is received by n. In case f does not cross n, $\tau(f, n) = t_0$, where t_0 is a special element of T that represents absence of traffic.

5.4. Network Functions model

The network functions model covers two main aspects: their configuration and their behavior.

The configuration model varies on the basis of the different kinds of network function. As an example, the configuration of a firewall instance $n \in N_U$ is modeled as a pair (d_n, R_n). In this pair, R_n is a set of filtering rules that are evaluated whenever the firewall received a packet, while d_n represent the default action to be applied in case the rules in R_n do not indicate how a certain type of traffic should be managed. Each $r \in R_n$ is modeled as $r = (c_r, a_r)$, where c_r is the rule condition, while a_r is the action that must be applied when the condition is satisfied[1]. In greater detail, each c_r is a predicate, given by the conjunction of five predicates:

$$c_r = (IPSrc, IPDst, pSrc, pDst, tProto)$$

Similarly as for the traffic model, each predicate composing c_r expresses a condition on a field of the IP 5-tuple. The default action d_n and each rule action a_r can be "allow" or "deny".

The network functions behavior is then modeled by means of two functions, which correspond to the forwarding behavior and the transformation behavior. The former is related to establishing which types of traffic are allowed or dropped by the network function, the latter expresses how the packets might be modified by the function.

The function that models the forwarding behavior of the VNF in node $n \in N_U$ is the predicate *deny*: $N_U \times T \rightarrow \mathbb{B}$ which is true for node $n \in N_U$ and ingress traffic $t \in T$, if and only if n drops all the packets represented by t. Instead, the transformation behavior of the VNF in node $n \in N_U$ is modeled by the function *transform*: $N_U \times T \rightarrow T$, which maps an input traffic to the corresponding output traffic.

5.5. Network Security Policies model

Each policy p of the P_T set (i.e., the set of all the target Network Security Policies) is formally defined as a

[1] The proposed model supports firewall that do not require order dependencies among the rules except for the default action (i.e., firewalls that work only in blacklisting or whitelisting mode). However, as reported in literature, the configuration of a firewall with ordered rules can be always equivalently expressed in our model, with algorithms such as the one described in [44].

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

Table 2: Target Network Security Policies

pair $p = (C, a)$. In this definition, C represents the policy condition, whereas a is the action that is applied on all the traffic flows satisfying the condition.

Each condition C is a predicate similar to the ones defined for modeling packet classes:

$$C = (IPSrc,\ IPDst,\ pSrc,\ pDst,\ tProto)$$

The predicates *IPSrc* and *pSrc* specify the traffic sources the policy refers to. Instead, the predicates *IPDst*, *pDst*, and *tProto* specify the traffic destinations and the protocol the policy refers to. A flow $f = [m_s, t_{sa}, ..., t_{kd}, n_d]$ satisfies C if the following three conditions are satisfied: 1) the source and destination endpoints n_s, n_d have IP addresses respectively matching *IPSrc* and *IPDst*; 2) its source traffic satisfies *IPSrc* and *pSrc*, i.e., $t_{sa} \subseteq (C.IPSrc, *, C.pSrc, *, *)$; 3) its destination traffic satisfies *IPDst*, *pDst*, and *tProto*, i.e., $t_{kd} \subseteq (*, C.IPDst, *, C.pDst, C.tProto)$. Let then $F_p \subseteq F$ denote the set of flows that satisfy $p.C$.

Each action a is one of the two elements of the set $A = \{allow, deny\}$. When $p.a = allow$, then p is defined a reachability policy: a policy of this type is satisfied in an intermediate state of the transient if, in that state, there exists at least a flow satisfying the policy condition that reaches its destination. Instead, when $p.a = deny$, p is defined an isolation policy: in this case, the policy is satisfied in a state if all the flows satisfying its conditions cannot reache their destination.

An example P_T set is shown in Table 2.

The P_T set is split into two subsets, i.e., $P_T = P_T^P \cup P_T^N$. The elements of P_T^P are the persistent policies, for which the MaxSMT problem requires the formulation of hard constraints, because these requirements must be satisfied in all the intermediate states of the reconfiguration transient. Instead, the element of P_T^N are all the other non-persistent policies. They must be organized in a ranking and their satisfaction is not strictly required for any intermediate state; therefore, their presence is reflected to a set of soft constraints.

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

6. Ranking Generation

The FATO methodology requires that a ranking is defined for all the policies of the P_T^N set, before the formulation of the optimization problem. As anticipated in Section 4, the generation of this ranking is performed by FATO as long as the user provides the methodology with an *optimization profile*, i.e., a working mode for the ranking computation. Some profiles additionally require that the user specifies a partial or total order relationship for the elements in P_T^N (i.e., the specification of the fact that each policy or group of policies is more or less important than one or a group of other policies).

The computation of the ranking is then performed on the basis of this information provided by the user throughout two steps: 1) computation of a matrix, called dominance matrix, which captures the information about the relationships between any pair of policies in P_T^N (Subsection 6.1); 2) generation of the ranking on the basis of the dominance matrix previously computed (Subsection 6.2). Alternatively, the ranking might be directly defined by the

user, if she has the required skills (e.g., she has total control on the network, and a high level of security expertise). In this particular case, the usage profile and the relationship set for the policies are not required by FATO, which directly receives the ranking from the user.

6.1. Dominance Matrix Computation

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

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

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

The relationships among the elements of P_T^N are established depending on the working profile selected by the user for the Ranking Generation algorithm. As explained in Subsection 4.2, there are four available profiles: *security-max*, *service-max*, *policy-max* and *state-max*.

Security-max profile. This profile determines two types of constraints. On the one hand, each isolation policy dominates any reachability policy. This constraints is represented in (1), according to which, given a policy p whose action $p.a$ is *deny* and a policy p' whose action $p'.a$ is *allow*, the former must have higher priority than the latter. On the other hand, two policies of the same type are independent, unless the user specifically forces a dominance relationship between them, as it is in their facility.. This constraints is represented in (2), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other.

$$\forall p \in P_T^N \mid p.a = deny.\ \forall p' \in P_T^N \mid p.a = allow.\ (r \succ r') \tag{1}$$

$$\forall p, p' \in P_T^N \mid p.a = p'.a.\ (p \parallel p') \tag{2}$$

Service-max profile. This profile determines two types of constraints. On the one hand, each reachability policy dominates any isolation policy. This constraints is represented in (3), according to which, given a policy p whose action $p.a$ is *allow* and a policy p' whose action $p'.a$ is *deny*, the former must have higher priority than the latter. On the other hand, two policies of the same type are independent, unless the user specifically forces a dominance relationship between them, as it is in their facility. This constraints is represented in (4), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other.

$$\forall p \in P_T^N \mid p.a = allow.\ \forall p' \in P_T^N \mid p.a = deny.\ (p \succ p') \tag{3}$$

$$\forall p, p' \in P_T^N \mid p.a = p'.a.\ (p \parallel p') \tag{4}$$

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

State-max profile. Each pair of policies, independently of their types, is characterized by an independence relationship. This constraints is represented in (5), according to which, given two policies p whose actions $p.a$ and $p'.a$ are the same (both are *deny* or *allow*), neither of them has necessarily higher priority than the other. This guarantees that, when trying to enforce their satisfiability in each transient state, no policy has a higher priority than another. Therefore, the global objective of the FATO methodology, i.e., maximizing the number of states where each policy is satisfied, translates into maximizing the number of policies satisfied in each intermediate state.

$$\forall p, p' \in P_T^N.\ (p \parallel p') \tag{5}$$

Note that FATO can be easily extended to support other profiles. It is enough to specify how the dominance and independence relationships are specified for the policies in P_T^N for each new profile, in a similar say as it has been explained for the four ones that are currently supported.

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

Table 3: First matrix example

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

Table 4: Second matrix example

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

Table 5: Third matrix example

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

- if $p \succ p'$, then $M_D[p, p'] = 1$;
- in all the other cases, $M_D[p, p'] = 0$.

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

- the user decides to adopt a security-oriented profile, without specifying any dominance relationship for policies of the same type (i.e., isolation or reachability). The resulting dominance matrix is shown in Table 3;
- the user decides to adopt a security-oriented profile, and specifies that p_1 dominates all the other isolation policies, while p_5 is dominated by all the other reachability policies. The resulting dominance matrix is shown in Table 4;
- the user decides to adopt a complete profile. The resulting dominance matrix is shown in Table 5.

6.2. Ranking Generation Algorithm

Supposing that the cardinality of P_T^N is n, the associative array *rank*, composed by n elements, associates a policy $p \in P_T^N$ to the corresponding rank. The value of each element $rank[p]$ is computed on the basis of the dominance matrix M_D throughout Algorithm 1.

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

Next (line 5), the policies are ordered according to the descending order of the number of policies each one dominates. As such, the *descendingOrder* function works on the P_T^N set and the *dominated* array, so as to compute another array, identified by l_P. If two or more policies dominates the same number of policies, their relative ordering is indifferent. This array can be accessed by means of integer indexes from 1 to n, and in particular $l_P[i]$ returns the policy in i-th position in the computed descending ordering.

The final operation is the computation of the value of $rank[p]$ for each $p \in P_T^N$. To this end, after initializing the auxiliary variable *rankCounter* to 1 (line 6), the policies are analyzed one at a time, according to the previously computed ordering. At each i-th iteration step, the *rankCounter* variable is incremented by one unit, if the following conditions are satisfied (lines 8-9): (1) the policy returned by $dominated[l_P[i]]$ dominates a larger number of policies than the policy $dominated[l_P[i+1]]$ (because they

Algorithm 1 computation of the ranking

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

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

might have the same value of dominated policies, and in that case the i+th policy antecedes the (i+1)-th one in this ordering is casual); and (2) the i-th policy dominates the (i+1)-th one (because it might happen that they are independent, so there is not relative priority between them).

After the decision on increasing the *rankCounter* variable, its updated value is assigned to *rank*[p] (line 11). After this operation is repeated for all the policies, the value of *rank*[p] for each $p \in P_T^N$ has been computed. These values will come in hand for the formalization of the soft constraints of the MaxSMT problem.

7. MaxSMT Problem Formulation

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

7.1. Hard constraints on boundary states

The presence of a node in the N_I or N_T sets (i.e., respectively in the G_I or in the G_T graphs) determines some boundary conditions on the initial and final states of the reconfiguration transient. These conditions are formalized as three classes of hard constraints, represented by equations (6), (7) and (8). In these formulas, the outcome of the *active* predicate is constrained when applied to the initial state s_0 and the final state s_c, because the conditions of each node in those states are already known from the inputs of the approach. In light of these considerations, the three classes of hard constraints are formulated as follows:

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

$$\forall\, n \in N_U \backslash N_T\,.\,(active(n, s_0) \wedge \neg active(n, s_c)) \tag{6}$$

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

$$\forall\, n \in N_U \backslash N_I\,.\,(\neg active(n, s_0) \wedge active(n, s_c)) \tag{7}$$

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

$$\forall\, n \in N_I \cap N_T\,.\,(active(n, s_0) \wedge active(n, s_c)) \tag{8}$$

7.2. Hard constraints on intermediate states

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

In greater detail, three classes of hard clauses are defined for the conditions on the intermediate states of the transient, and they are represented by equations (9), (10) and (11).

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

$$\begin{aligned}\forall\, n \in N_U\,.\,((active(n, s_0) \wedge active(n, s_c)) \implies \\ (\forall s \in S\,.\,(active(n, s))))\end{aligned} \tag{9}$$

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

$$\forall\, n \in N_U.\ ((active(n, s_0) \neq active(n, s_c)) \implies \left(\sum_{s_i \in (S \setminus s_c)} (|bti(active(n, s_{i+1})) - bti(active(n, s_i))|) = 1 \right) \tag{10}$$

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

$$\forall\, s_i, s_{i+1} \in S.\ \exists! n \in N_U.\ (active(n, s_i) \neq active(n, s_{i+1})) \tag{11}$$

Additionally, the case where two nodes $n, n' \in N_U$ share the same AP and their update is allowed without the need of instantiating a new software process (i.e., as explained in Subsection 5.1, $configUpdate(n, n') = true$) is formulated with hard constraints as well. This translates into the hard constraint shown in (12), stating that, when the node n is not active anymore in a state s_i, then in the next state s_{i+1}, the node that has become active must be n'. Thus, even though in the modelization the nodes are distinct, the result is the same as if they were a single one, and only the configuration was changed.

$$\forall\, n, n' \in N_U.\ (configUpdate(n, n') \wedge active(n, s_0) \wedge \neg active(n', s_0) \implies (\exists i.\ \neg active(n, s_i) \implies active(n', s_{i+1}))) \tag{12}$$

7.3. Hard constraints on the forwarding behavior

The forwarding behavior of the network functions (i.e., the decision if a middlebox must drop the input traffic or forward it) is formalized with some hard constraints impacting on the outcome of the *deny* predicate. The truth or falseness of this predicate, in turn, impacts the satisfiability of the hard constraints related to the security policies, as it will be described in Subsection 7.4. Each network function type is characterized by different hard constraints, in accordance with the approach that has been pursued for modeling the network functions in [7]. In the following, we present two representative examples.

Simple functions such as normal forwarders never block packets, because they do not have filtering functionalities. This behavior characterizes also the APs that have not been filled with a firewall instance in G_I or G_T, and therefore are present in G_U as well. For a node $n \in N_U$ of such a type, it is enough to force the outcome of the *deny* predicate to false for each possible traffic in input to n. As shown in (13), the condition that node N may receive the traffic is checked by verifying if the node belongs to the path of the traffic flow, i.e., if $n \in \pi(f)$.

$$\forall\, p \in P_T.\ \forall\, f \in F_p.\ (n \in \pi(f) \implies \neg deny(n, \tau(f, n))) \tag{13}$$

A firewall instance has, instead, a more complex behavior, because it can block the traffic depending on its configuration (i.e., its default action and filtering rules). Considering a packet filter $n \in N_U$, the hard constraint imposed upon the *deny* predicate to represent the forwarding behavior of this kind of function is represented in (14).

$$\begin{aligned} deny(n, t) &\iff (a) \vee (b) \\ (a) &:= (d_n = deny) \wedge (\nexists r \in R_n.\ match(r, t)) \\ (b) &:= (d_n = allow) \wedge (\exists r \in R_n.\ match(r, t)) \end{aligned} \tag{14}$$

In the definition of this hard constraint, the $match\colon R_n \times T \to \mathbb{B}$ predicate is employed. This predicate returns true if the input traffic positively matches the conditions of the input filtering rule. The relation of the *match* outcome with the rule conditions is shown in (15).

$$\begin{aligned} \text{match}(r, t) = t.IPSrc \subseteq r.IPSrc \wedge t.IPDst \subseteq r.IPDst \wedge \\ t.pSrc \subseteq r.pSrc \wedge t.pDst \subseteq r.pDst \wedge t.tProto \subseteq r.tProto \end{aligned} \tag{15}$$

In light of the definition of this predicate, the hard constraint (14) can be explained as follows. A firewall instance

n may show two alternative behaviors. If n has a *deny* default action, then it blocks a traffic t if there is not any specific rule, with *allow* action, that matches that traffic. Alternatively, if n has an *allow* default action, then it blocks a traffic t only it there is a specific rule, with *deny* action, that matches that traffic.

7.4. Hard constraints on the security policies

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

$$\forall\, p \in P_T\,.\,(satisfied(p, s_c)) \tag{16}$$

For all the intermediate states of the transient, a set of hard constraints must be introduced to map the outcome of the *satisfied* predicate to the forwarding behavior of the functions that are present in the paths crossed by the flows satisfying the conditions of each policy. The hard constraints are different depending on the policy type (i.e., reachability or isolation). The formalization of the hard constraints for a reachability policy $p \in P_T$ is shown in (17). In a state s the policy p is satisfied if there exists at least a flow f satisfying $p.C$ such that all the nodes of the paths crossed by f are active in that state and do not block the incoming traffic of f.

$$\begin{aligned}\forall s \in S.(satisfied(p, s) \iff (\exists f \in F_p.\\ (\forall n \in \pi(f).(active(n, s) \wedge \neg deny(n, \tau(f, n))))))\end{aligned} \tag{17}$$

Instead, the formalization of the hard constraints for an isolation policy $p \in P_T$ is shown in (18). In a state s the policy p is satisfied if for each flow f satisfying $p.C$ there exists at least a node of the path of f that is not active in that state (i.e., it cannot receive traffic), or it is active and blocks the incoming traffic of f.

$$\begin{aligned}\forall S \in S.(satisfied(p, s) \iff (\forall f \in F_p.\\ (\exists n \in \pi(f).(\neg active(n, s) \vee (active(n, s) \wedge deny(n, \tau(f, n)))))))\end{aligned} \tag{18}$$

Algorithm 2 computation of the weights for the soft constraints of the MaxSMT problem

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

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

Finally, if $p \in P_T^P$, i.e., it is a persistent policy, then it must be satisfied in each intermediate state. Therefore, the class of constraints shown in (19) imposes that the *satisfied* predicate must be true when applied to a persistent policy p and to any state, because that policy must be satisfied in any intermediate state of the transient.

$$\forall\, p \in P_T^P\,.\,(\forall s \in S.(satisfied(p, s))) \tag{19}$$

7.5. Soft constraints

The optimization objective of the MaxSMT problem is to maximize the number of transient states where each policy $p \in P_T^N$ is satisfied, while considering their relative priority expressed throughout the ranking computed as illustrated in Section 6. The persistent policies are excluded from this objective, because they are already managed as shown in (19). The achievement of this objective requires the formalization of some weighted soft constraints, so that the MaxSMT solver gives priority in the satisfaction of the clauses with the highest weight, trying to maximize the sum of the weights assigned to the satisfied soft clauses.

Each soft constraint is related to the application of the *satisfied* predicate to a pair composed of a policy p and a state s. Therefore, if the number of states is c (excluding the final state), the total number of soft constraints is $c \cdot |P_T^N|$. The computation of their weights is described in Algorithm 2, and these weights are returned by means of the associative array *weight*, indexed by the elements of P_T^N. Supposing that the number of different ranks is m, the

algorithm starts to assign the weights to policies having the lowest rank, i.e., the m-th rank (line 2). The weight that is assigned to all the policies within the same rank must be the same as well. Initially, the weight assigned to the policies, identified by the auxiliary variable *weightValue*, is set to a conventional number, which is 1 for simplicity (line 1). Whenever an element of the *weight* array is assigned with the proper value (line 5), a counter (*weightSum*) is incremented by the product of the current weight value (*weightValue*) and the number of states c (line 6). The reason is that this same weight will be used for that policy for c soft constraints, for each transient state. Then, when the algorithms iterates to a higher rank, the value of the variable of *weightValue* is incremented by the sum of all the previous weights (*weightSum*) plus 1 (line 7), so that the soft constraints will be characterized by a weight that is higher than the sum of all the weights assigned to soft clauses for policies of lower rank.

The formulation of the soft constraints is represented in (20). In this representation, the $Soft(c, w)$ notation identifies a soft clause, expressing the constraint c and having weight w. Each soft constraint simply assigns $weight[p]$, computed as previously explained, to the *satisfied* predicate, applied to the specific policy p and to any transient state s.

$$\forall p \in P_T^N . \forall s \in S \backslash \{s_c\} . (Soft(satisfied(p, s), weight[p])) \quad (20)$$

7.6. Solution Computation

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

8. Implementation and validation

The FATO methodology has been implemented as a Java-based framework, employing a state-of-the-art theorem prover called Z3 for the formulation and resolution of the MaxSMT problem. This framework offers REST APIs for the interaction with other tools, e.g., a tool for the automatic computation of firewall configurations in virtualized networks [20]. Through this RESTful interface, the framework can also interact with NFV MANO (Management and orchestration) tools, such as Open Source MANO. More specifically, the framework can retrieve information about the virtual network from the MANO. Then, after running the FATO methodology, it provides the MANO with information about the order the different operations composing the distributed firewall reconfiguration must be executed in the network.

The framework has been validated in different ways: i) the effectiveness and feasibility of the approach have been proved with some realistic use cases, based on computer networks having varying topologies (Subsection 8.1); ii) the computation time and memory usage of the approach has been evaluated with an extensive series of scalability tests (Subsection 8.2); iii) the optimization provided by our approach has been evaluated to assess how optimized the scheduling of the configuration changes effectively is (Subsection 8.2). All the tests have been carried out on an 8-core Intel Core i7-10700E CPU @ 2.90GHz workstation with 32 GB RAM. For these tests, the adopted Z3 version is 4.8.5.

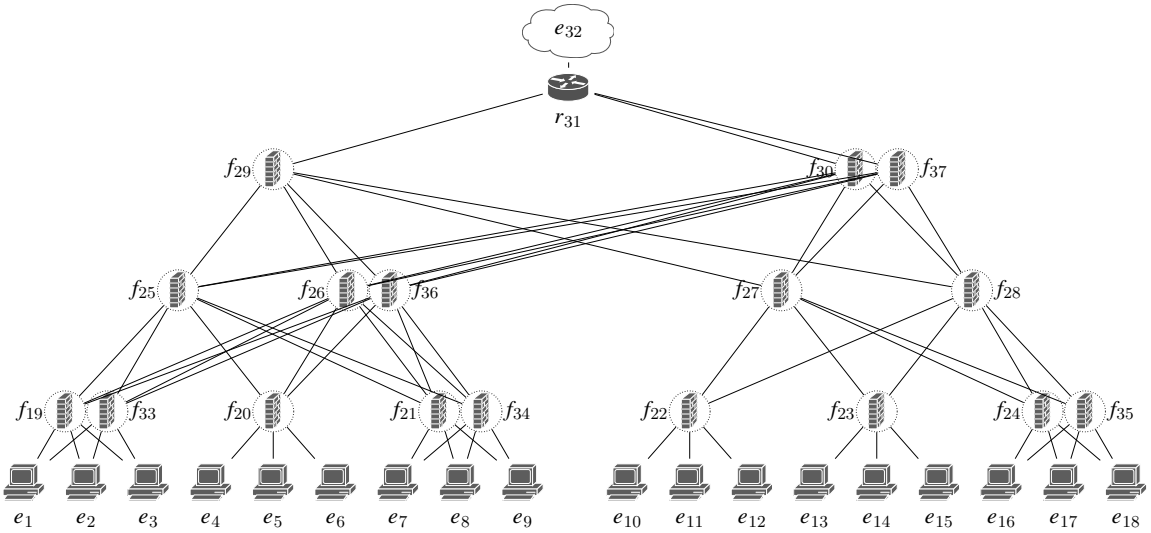


Figure 5: Topology of the three-layer data center network

8.1. Validation of the effectiveness

The effectiveness of the approach has been checked with some use cases, based on different networks having topological structures that vary from simple to complex. Four networks topologies have been considered: i) the network of our university department; ii) a three-tier data center network; iii) the GÉANT[2] topology; iv) the Internet2[3] topology.

Both the solution optimization and correctness have been checked. About the optimization, we have enumerated all the possible solutions in solving the transient reconfiguration problem for the networks. Then, we have run the framework and checked that the output produced by the MaxSMT solver corresponds to the optimal solution, i.e., it minimizes the number of intermediate transient states where policies are violated. About the correctness, we have simulated the intermediate states of the transient derived from the solution computed by the framework by using GNS3, a software that allows real-time network simulation for pre-deployment testing without the need for network hardware[4]. Under those conditions, we have tried to establish some communications in the network and we have checked if the specified security policies were violated or not in the transient states.

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

Table 6: Network Security Policies

Here we report the way the effectiveness validation has been performed for a fairly complex network topology, i.e., the three-tier data center network depicted in Figure 5. We can suppose that the human user, i.e., a network administrator, has specified four security policies and has assigned them with the rank shown in Table 6. The table also reports the firewall configuration changes required for the full enforcement of the corresponding policy.

The FATO methodology establishes that the following order of the configuration changes must be scheduled: $f_{19} \rightarrow f_{33}$, $f_{21} \rightarrow f_{34}$, $f_{26} \rightarrow f_{36}$, $f_{30} \rightarrow f_{37}$, and $f_{24} \rightarrow f_{35}$. In the following, we explain how the intermediate states of the transient of the computed solution still maintaining security and service availability according to the requirements deriving from specified ranking.

State after $f_{19} \rightarrow f_{33}$: the network administrator required that e_1 and e_{32} must be isolated as soon as possi-

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

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

State after $f_{21} \rightarrow f_{34}$: the network administrator required that hosts of the subnetworks e_4, e_5 and e_6 must be isolated one from each other, e.g., because Dockers belonging to different companies have been launched and they cannot interact due to privacy. As the security level is thought inferior than the previous policy, f_{21} is replaced by f_{34} after the $f_{19} \rightarrow f_{33}$ configuration change. At this point, the intermediate state fully maintains the desired level of security, as all traffic flows that must be blocked cannot reach their destinations. However, service availability is not complete, because both the reachability policies (between e_1 and e_{10}, between e_{16} and e_{18}) are not satisfied.

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

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

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

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

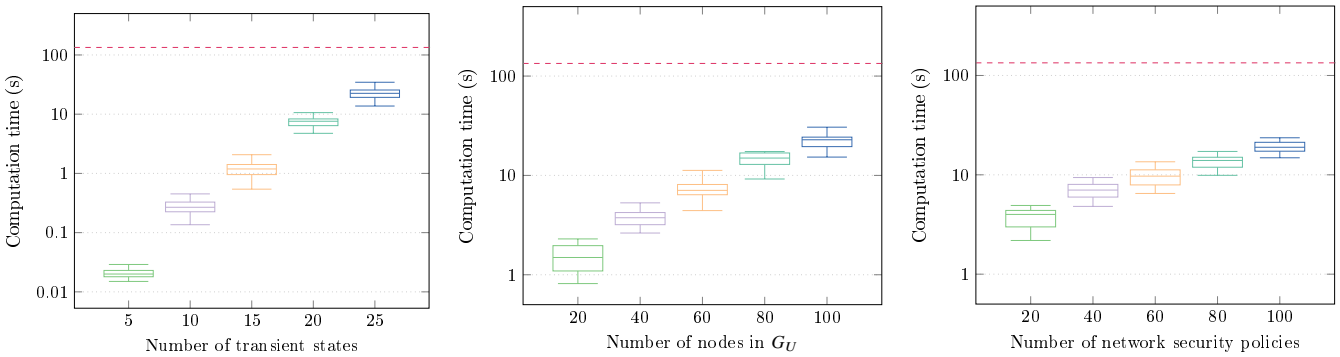
Topology	# firewalls	Time (10 states)	Time (20 states)
University department	6	0.64 s	7.73 s
Three-tier data center	17	1.47 s	12.88 s
GÉANT	35	3.83 s	15.10 s
Internet2	40	6.61 s	23.91 s

Table 7: Computation times for the four network topologies

Additionally, Table 7 reports the time that was required for computing the optimal scheduling for the four different topologies, alongside their number of firewalls. The experiments were carried out under two different transient lengths, respectively 10 and 20 states. In both cases, the FATO methodology succeeded in computing the solution in a reasonable time with the requirements of virtual networks, as already discussed in Section 8. This consideration also holds for more complex and varied topologies (e.g., GÉANT and Internet2), where the number of filtering functions is much higher and the structure of their allocation scheme is not trivial.

8.2. Evaluation of scalability

The scalability performance with respect to computation time and memory usage has been validated with a series of tests to show the feasibility of the framework com-



(a) Time scalability versus number of transient states

(b) Time scalability versus number of network nodes

(c) Time scalability versus number of network security policies

Figure 6: Time scalability

patibly with the requirements of virtualized networks. The four metrics that have been considered for the scalability evaluation are: i) the number of transient states; ii) the number of nodes composing the network topology; iii) the number of network security policies that determined the reconfiguration transient; iv) the number of filtering rules in each instance of the distributed firewall.

Figure 6 and Figure 7 report the results of the time scalability tests. First, Figure 6a analyzes the performance of the implementation when the number of transient states (i.e., the number of changes occurred in the transient from G_I to G_T, as formalized in Subsection 5.2) progressively increases. For those tests, each scenario characterized by a certain number of transient states is based on a topology of corresponding dimension (e.g., when the number of states is 20, in the resulting G_U the number of firewalls subject to configuration changes is 20). The enforcement of a congruent number of network security policies is requested as well. Second, Figure 6b analyzes how the framework behaves for increasing dimension of the network on which it is applied (i.e., the G_U), while keeping the number of transient states fixed to 20 and the number of policies fixed to 50. Third, Figure 6c evaluates the scalability versus the number of network security policies that should be enforced in the reconfiguration transient, while keeping the number of transient states fixed to 20 and the network dimension fixed to 50 nodes. Fourth, Figure 7 depicts the experimental results for the scalability tests related to the number of filtering rules (from 25 to 100) in each firewall instance. The percentage of firewall instances is 2/3 for testing the scalability versus firewall rules, so that the framework is evaluated in situations that might be stressful for its application. The number of intermediate states of the transients is again fixed to 20.

For all these time scalability tests, the topology is an extension synthetically derived from the network represented in Figure 4, whereas the policies are similar to the examples shown in Table 2. Besides, for each scenario, the corresponding whisker plot shows minimum, 5th percentile, median, 95th percentile and maximum values, computed over the results of 500 iterations. Many iterations have been performed because the computation time of Z3 for the resolution of optimization problems involving the integer theory may differ depending on the effective integers employed in the definition of the IP addresses. Therefore, each iteration differs from the others for each scenario only because different addresses are assigned to the networks nodes, both end points and middleboxes. The previously mentioned figures are semi-logarithmic plots, with a logarithmic scale for the Y-axis ("Computation time") and a linear scale for the X-axis. Through this representation, whisker plots depicted for low numbers of transient states can be better visualized, while they would be otherwise stretched to the bottom part of the plot.

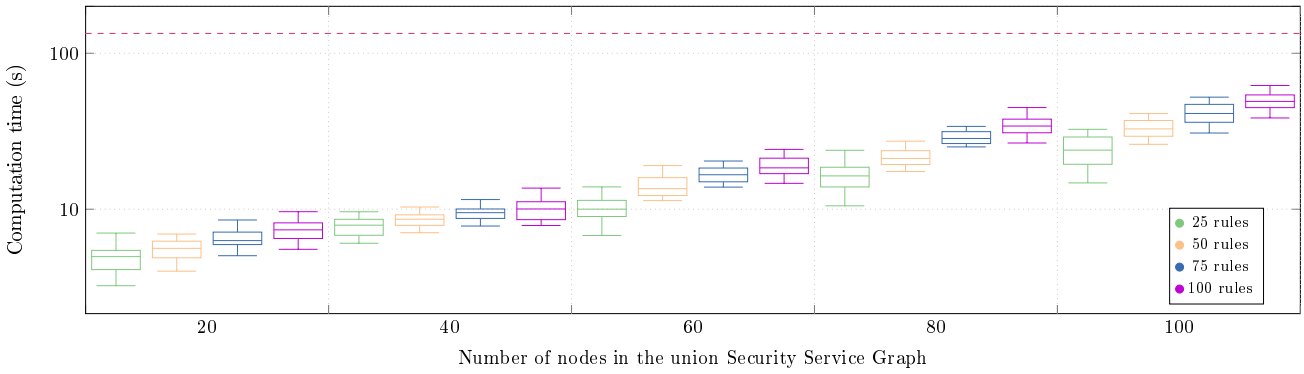


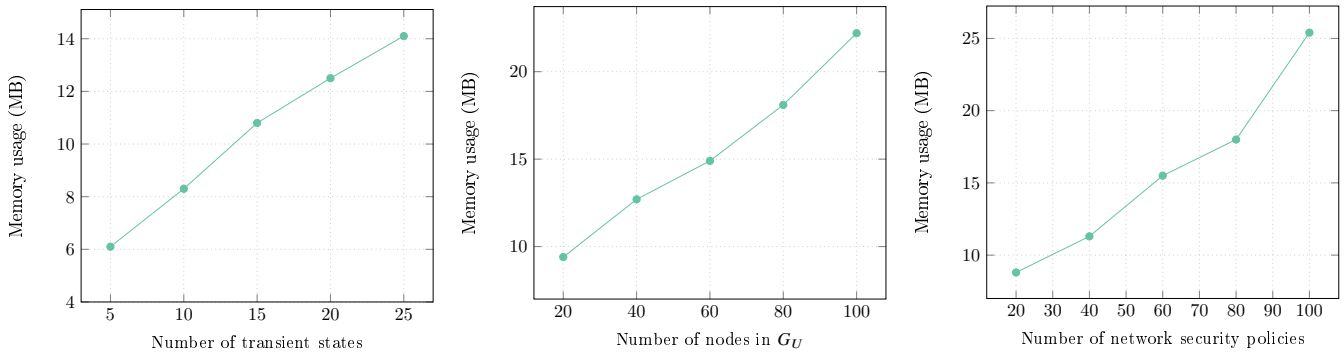
Figure 7: Time scalability versus number of rules in each firewall instance

As it has been discussed in the problem statement of Section 3, the number of transient states is equal to the number of the distributed firewall configuration changes. In virtualized networks, the best practice is to deploy a new instance with the required configuration instead of the previous one. Therefore, the number of states is also directly proportional to the number of firewall instances that are subject to the reconfiguration. In light of these considerations, the numbers of transient states and security function instances characterizing the scenarios under which our approach has been tested are in line with the experimental tests carried out for the validation of related approaches ([28][29][45]). In those studies, the maximum number of nodes composing the distributed function architecture of SDN is 20-30. Considering that not all of the instances are usually subject to changes during a reconfiguration, then this shows that the transient length we considered as reference is also the most recurrent one.

From the results shown in Figure 6a, the computation time of the MaxSMT solver progressively increases with a quadratic behavior when the number of states of the transients gets bigger. On the one hand, the scalability is worse when the number of states is bigger than 20. On the other hand, the framework is particularly fast for transients composed of at most 20 states: all the MaxSMT instances representing the evaluated scenario are successfully solved in less than 10 seconds. Considering that related approaches work on a similar number of transient states as mentioned before, this is a significant result for validating the methodology illustrated in this paper. It can also be noted that in [28] some problem instances with more than 20 functions could not be solved in 600 seconds, while in [29] the time required to find a first feasible solution for the optimization problem is in the same magnitude order as the results of our validation tests. Differently from those papers, our solution also provides the additional benefits that have been described in the discussion of related work in Section 2.

Similar considerations apply to the scalability versus the number of nodes in G_U and the number of network security policies. The results plotted in Figure 6b and Figure 6c respectively show that the proposed approach can successfully manage networks of fairly big dimension and it can check the satisfiability of a large set of policies. The scalability with respect to the network dimension and the policy set cardinality is even better than the scalability with respect to the number of transient states. This is due to the fact that the increment of soft constraints in the formulation of MaxSMT problem is lower. Additionally, Figure 7 shows that, even if the number of rules in each firewall instance increases, the difference among the resulting computation times is not significantly large. As such, the methodology is also suitable for managing firewalls with a high number of rules.

This time scalability is also in line with the times that state-of-the-art approaches required for the security man-



(a) Memory scalability versus number of transient states

(b) Memory scalability versus number of network nodes

(c) Memory scalability versus number of network security policies

Figure 8: Memory scalability

agement of virtualized networks. The common workflow to enforce security in a virtualized networks is composed of multiple steps [46]: i) establishing the configuration of the security functions; ii) determining the embedding scheme of their virtual implementations in the physical infrastructure; iii) the instantiation of the virtual security service; iv) the effective deployment of the virtual functions. The same process should be also followed in case of a reconfiguration, as such the one that determines the transients studies in this paper. By looking at the related literatures, the computation times that are commonly needed to perform those tasks are bigger than the ones we can reach for the management of the reconfiguration transient. [47] reports that configuring a security service with a distributed firewall composed of 100 instances requires around 3 minutes. [48] underlines that establishing the embedding scheme of 10 virtual functions on a physical network composed of 50 nodes varies according to the adopted methodology, from 1400s for the resolution of the exact problem to around 100ms when the heuristic that is proposed there ix executed. [49] experimentally checked that the instantation time of a network security service takes more than 100 seconds, when the service is composed of around 30 virtual functions, for the Virtual Infrastructure Managers of two-well known orchestrators: Open Source MANO and Openstack. [42] states that DPD time related to the deployment of a single virtual function is 134s. If these numbers are combined, it is clear that the time introduced by our framework does not represent a delay, as the scheduling of the reconfiguration changes may be easily computed by FATO while another step, such as the service instantation, is performed. This is even more evident when the DPD time is compared to the time scalability of our approach. In transients composed of 20 states, more than a single virtual function must be deployed and instantiated, but just deploying one takes more time than running our methodology. Therefore, we have decided to represent the value of the DPD time for reference as a red dotted horizontal line in Figure 6 and Figure 7.

Finally, the peak memory scalability has been evaluated by running the framework under the same conditions of Figure 6. This analysis was required, because memory may be a critical parameter for the solver, which is highly memory-demanding. The results are reported in Figure 8, where whisker plots are not used because memory usage is not influenced by the different IP addresses used in the MaxSMT formulation, differently from the computation time. Two considerations can be derived from these plots. On the one hand, the memory usage increases linearly with respect to all the analyzed metrics. On the one hand, even the worst case that was identified (i.e., when 100 network security policies are formalized in the MaxSMT problem) is inferior to 26 MB. Therefore, all these results shows that the implementation of our methodology can work without

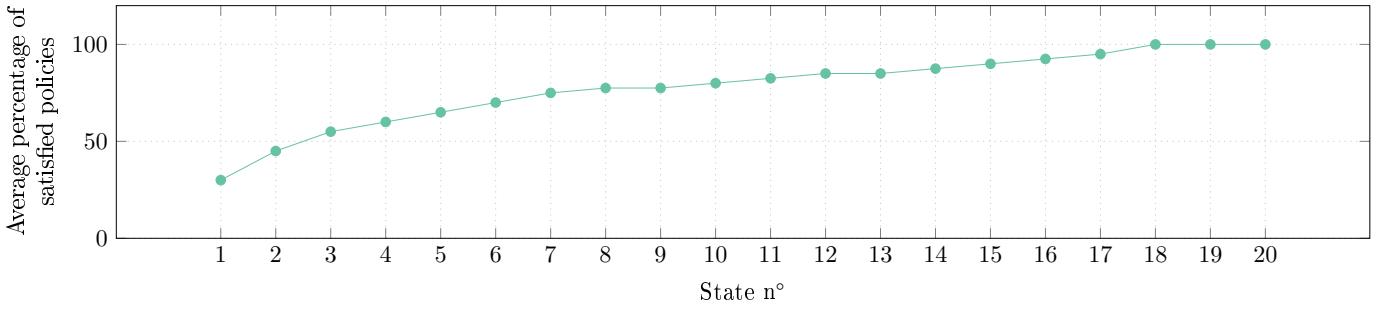


Figure 9: Evaluation of optimization

any worrisome limitation due to memory.

8.3. Evaluation of optimization

The objective of the FATO methodology is to maximize the number of connectivity security policies that are satisfied in each transient state. Therefore, the optimization with respect to this goal has been evaluated by checking how many policies are effectively satisfied during the whole transient, so as to assess the benefits that the proposed approach can bring over unoptimized strategies. For this validation, we have run the framework on 10 different networks topologies, that are synthetically generated as extension of the topology shown in Figure 4, as we have also done for the tests of Figure 6. Each network topology has a different structure (e.g., different connections among nodes, different paths crossed by traffic flows). The number of transient states is kept fixed to 20, and the number of networks ecurity policies that have to be enforced is kept fixed to 40. After running the framework on each network topology, we have computed the number of policies satisfied in each state. We have derived this information by extracting the value of the *satisfied(p,s)* predicate for each pair (policy p, state s) from the output of the Z3 solver.

Figure 9 plots the average percentage number of satisfied policies in each transient state. As it can be seen, most of the policies are already enforced in the early states of the transient: over 50% are satisfied in average in the third state, and over 75% in the seventh state. Therefore, this experimental result shows that the proposed approach largely improves the security level during a reconfiguration transient. It also confirms the indirect benefit provided by our optimization objective, already stated in Subsection 3.4, i.e., maximizing the satisfactions of the policies in each state leads to satisfy them as early as possible. This result adds to the other ones of this section, proving that FATO is a full-fledged methodology that automatically applies optimization to a state-of-the-art problem, ensuring the correctness of its solution and guaranteeing times that are compatible with the management of virtualized networks.

9. Final discussion

The FATO methodology has been designed to exploit the main characteristics of virtualized networks. The definition of formal models takes into account best practices for management of virtualized networks (e.g., during a reconfiguration, a new virtual instance is commonly deployed with the new configuration, in place of an existing one), and it introduces the formal correctness that state-of-the-art orchestrators often lack. Besides, as it has been proved with an extensive validation, the proposed methodology can successfully reach the promised optimization objective, i.e., maximizing the number of connectivity policies satisfied in each transient state, with computation times that do not introduce delay in the process of security reconfiguration.

However, there may be scenarios where this strategy would not be the best, and it would require extensions and modifications to be applied. For example, industrial computer networks have been subject to new challenges

in terms of security management in recent years, as new paradigms like Industry 4.0 and Internet-of-Things revolutionized their organization. There, guaranteeing service continuity is often the most critical requirement, as servers should be reachable in any moment of the day to provide vital assistance. Our approach would require some changes to support this context. At the same time, it is flexible enough to make these changes feasible. On the one hand, some formal models of network functions may require extension, e.g., to introduce the peculiarities of industrial switches, but some other ones (network graphs, connectivity policies, state sequence, traffic flows) would be already compliant. On the other hand, new optimization objectives must be included, e.g., related to latency and bandwidth. Nevertheless, since the optimization objectives are modeled in our strategy with soft constraints, which are distinct entities from hard constraints in the MaxSMT formulation, if a certain context required different objectives, they could be modeled without impacting most of the hard constraints of the problem definition.

In light of these considerations, the FATO methodology has been designed to be efficiently optimized for virtualized networks. At the same time, future research developments may adapt it to more peculiar environments. This confirms that the proposed approach is a valid research path that is worth being further investigated also in the future for the transient management problem.

10. Conclusions

This paper presented a novel methodology to compute the scheduling of the reconfiguration changes for a virtual distributed packet filter, so as to automatically manage the problems deriving from the reconfiguration transient. The problem formulation as MaxSMT is founded on a correctness-by-construction approach to provide higher confidence in the correctness of the computed scheduling. Optimization objectives are integrated within the proposed methodology (e.g., an objective is to maximize the number of intermediate transient states in which each connectivity security policy is satisfied). This is the first time a method featuring all these properties has been proposed to manage transients due to the reconfiguration of a distributed packet filter.

This represents a first step forward in the literature, but some limitations still exist. In particular, the proposed approach can only support stateless packet filtering firewalls, characterized only by "deny" and "allow" actions, that represent the most commonly used type. Therefore, in the future, a first work will be to extend the presented method to optimize transients due to the reconfiguration of other types of firewalling functions, e.g., web-application firewalls, stateful firewalls, or anti-spam filters. Other types of network security functions will be explored as well. Finally, a direct integration of this methodology with mitigation and reaction mechanisms is planned, with the objective to create an autonomic process for the dynamic orchestration and reconfiguration of virtual security functions.

References

[1] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. S. Santos, C. E. Rothenberg, Network service orchestration: A survey, Comput. Commun. 142-143 (2019) 69–94. doi:10.1016/j.comcom.2019.04.008.

[2] K. Foerster, S. Schmid, S. Vissicchio, Survey of consistent software-defined network updates, IEEE Commun. Surv. Tutorials 21 (2) (2019) 1435–1461. doi:10.1109/COMST.2018.2876749.

[3] A. X. Liu, A. R. Khakpour, J. W. Hulst, Z. Ge, D. Pei, J. Wang, Firewall fingerprinting and denial of firewalling attacks, IEEE Trans. Inf. Forensics Secur. 12 (7) (2017) 1699–1712. doi:10.1109/TIFS.2017.2668602.

[4] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. M. Khan, N. Meskin, Cybersecurity for industrial control systems: A survey, Comput. Secur. 89 (2020). doi:10.1016/j.cose.2019.101677.

[5] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P. B. Godfrey, Veriflow: Verifying network-wide invariants in real time, in: Proc. of the 10th USENIX Symp. on Networked Systems Design and Implementation, 2013, pp. 15–27.

URL https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/khurshid

[6] A. Panda, O. Lahav, K. J. Argyraki, M. Sagiv, S. Shenker, Verifying reachability in networks with mutable datapaths, in: 14th USENIX Symposium on Networked Systems Design and Implementation, 2017, pp. 699–718.
URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/panda-mutable-datapaths

[7] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, J. Yusupov, Improving the formal verification of reachability policies in virtualized networks, IEEE Trans. Netw. Serv. Manag. 18 (1) (2021) 713–728. doi:10.1109/TNSM.2020.3045781.

[8] T. Runge, T. Thüm, L. Cleophas, I. Schaefer, B. W. Watson, Comparing correctness-by-construction with post-hoc verification - A qualitative user study, in: Proc. of the Inter. Works.on Formal Methods (FM19), 2019, pp. 388–405. doi:10.1007/978-3-030-54997-8_25.

[9] R. Sebastiani, P. Trentin, On optimization modulo theories, maxsmt and sorting networks, in: Proc. of the Inter. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, 2017, pp. 231–248. doi:10.1007/978-3-662-54580-5_14.

[10] R. Mahajan, R. Wattenhofer, On consistent updates in software defined networks, in: D. Levine, S. Katti, D. Oran (Eds.), Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, College Park, MD, USA, November 21-22, 2013, ACM, 2013, pp. 20:1–20:7. doi:10.1145/2535771.2535791.

[11] S. Scott-Hayward, G. O'Callaghan, S. Sezer, Sdn security: A survey, in: IEEE SDN for Future Networks and Services, SDN4FNS 2013, Trento, Italy, November 11-13, 2013, IEEE, 2013, pp. 1–7. doi:10.1109/SDN4FNS.2013.6702553.

[12] C. C. Zhang, M. Winslett, C. A. Gunter, On the safety and efficiency of firewall policy deployment, in: Proc. of the IEEE Symp. on Security and Privacy (S&P), 2007, pp. 33–50. doi:10.1109/SP.2007.32.

[13] Z. Ahmed, A. Imine, M. Rusinowitch, Safe and efficient strategies for updating firewall policies, in: Proc. of the 7th Inter. Conf. Trust, Privacy and Security in Digital Business,, 2010, pp. 45–57. doi:10.1007/978-3-642-15152-1_5.

[14] A. Kartit, M. E. Marraki, An enhanced algorithm for firewall policy deployment, in: Proc. of the Inter. Conf. on Multimedia Computing and Systems, 2011, pp. 1–4. doi:10.1109/ICMCS.2011.5945704.

[15] F. Bezzazi, A. Kartit, M. E. Marraki, D. Aboutajdine, Optimized strategy of deployment firewall policies, in: Proc. of the 2nd Inter. Conf. on the Innovative Computing Technology (INTECH12), 2012, pp. 46–50. doi:10.1109/INTECH.2012.6457775.

[16] Z. Kartit, H. Idrissi, K. Ali, M. El marraki, Improvement of algorithm for updating firewall policies, Journal of Theoretical and Applied Information Technology 66 (2014) 158–289.

[17] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutorials 16 (3) (2014) 1617–1634. doi:10.1109/SURV.2014.012214.00180.

[18] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, Opennf: enabling innovation in network function control, in: Proc. of the ACM SIGCOMM Conf. (SIGCOMM'14), 2014, pp. 163–174. doi:10.1145/2619239.2626313.

[19] J. Deng, H. Li, H. Hu, K. Wang, G. Ahn, Z. Zhao, W. Han, On the safety and efficiency of virtual firewall elasticity control, in: Proc. of the 24th Network and Distributed System Security Symp., NDSS, 2017.

[20] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, Automated optimal firewall orchestration and configuration in virtualized networks, in: NOMS 2020 - IEEE/IFIP Network Operations and Management Symp., 2020, pp. 1–7. doi:10.1109/NOMS47738.2020.9110402.

[21] N. P. Katta, J. Rexford, D. Walker, Incremental consistent updates, in: Proc. of the 2nd ACM SIGCOMM Work. on Hot Topics in Software Defined Networking, 2013, pp. 49–54. doi:10.1145/2491185.2491191.

[22] R. McGeer, A correct, zero-overhead protocol for network updates, in: Proc. of the 2nd ACM SIGCOMM Work. on Hot Topics in Software Defined Networking, 2013, pp. 161–162. doi:10.1145/2491185.2491217.

[23] J. Hua, X. Ge, S. Zhong, FOUM: A flow-ordered consistent update mechanism for software-defined networking in adversarial settings, in: Proc. of the 35th IEEE Inter. Conf. on Computer Communications, (INFOCOM16), 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524499.

[24] P. Cerný, N. Foster, N. Jagnik, J. McClurg, Optimal consistent network updates in polynomial time, in: Proc. of the 30th Inter. Symp. Distributed Computing, DISC16, Springer, 2016, pp. 114–128. doi:10.1007/978-3-662-53426-7_9.

[25] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, O. Bonaventure, Safe update of hybrid SDN networks, IEEE/ACM Trans. Netw. 25 (3) (2017) 1649–1662. doi:10.1109/TNET.2016.2642586.

[26] W. Liu, R. B. Bobba, S. Mohan, R. H. Campbell, Interflow consistency: A novel SDN update abstraction for supporting inter-flow constraints, in: 2015 IEEE Conference on

Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015, IEEE, 2015, pp. 469–478. doi:10.1109/CNS.2015.7346859.

[27] R. Sukapuram, G. Barua, PPCU: proportional per-packet consistent updates for software defined networks, in: 24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016, IEEE Computer Society, 2016, pp. 1–2. doi:10.1109/ICNP.2016.7784460.

[28] A. Ludwig, M. Rost, D. Foucard, S. Schmid, Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies, in: Proc. of the 13th ACM Work. on Hot Topics in Networks, (HotNets14), 2014, pp. 15:1–15:7. doi:10.1145/2670518.2673873.

[29] A. Ludwig, S. Dudycz, M. Rost, S. Schmid, Transiently secure network updates, in: Proc. of the ACM SIGPLAN Inter. Conf. on Measurement and Modeling of Computer Science, 2016, pp. 273–284. doi:10.1145/2896377.2901476.

[30] J. McClurg, H. Hojjat, P. Cerný, N. Foster, Efficient synthesis of network updates, in: Proc. of the 36th ACM Conf. on Programming Language Design and Implementation, 2015, pp. 196–207. doi:10.1145/2737924.2737980.

[31] S. Vissicchio, L. Cittadini, FLIP the (flow) table: Fast lightweight policy-preserving SDN updates, in: Proc. of the 35th IEEE Inter. Conf. on Computer Communications, (INFOCOM16), 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524419.

[32] D. Borbor, L. Wang, S. Jajodia, A. Singhal, Surviving unpatchable vulnerabilities through heterogeneous network hardening options, J. Comput. Secur. 26 (6) (2018) 761–789. doi:10.3233/JCS-171106.

[33] D. Borbor, L. Wang, S. Jajodia, A. Singhal, Optimizing the network diversity to improve the resilience of networks against unknown attacks, Comput. Commun. 145 (2019) 96–112. doi:10.1016/j.comcom.2019.06.004.

[34] S. Y. Enoch, J. B. Hong, M. Ge, K. M. Khan, D. S. Kim, Multi-objective security hardening optimisation for dynamic networks, in: 2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019, IEEE, 2019, pp. 1–7. doi:10.1109/ICC.2019.8761984.

[35] R. Dewri, N. Poolsappasit, I. Ray, L. D. Whitley, Optimal security hardening using multi-objective optimization on attack tree models of networks, in: P. Ning, S. D. C. di Vimercati, P. F. Syverson (Eds.), Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007, ACM, 2007, pp. 204–213. doi:10.1145/1315245.1315272.

[36] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, S. T. King, Debugging the data plane with anteater, in: Proc. of the ACM SIGCOMM Conf. (SIGCOMM'11), ACM, New York, NY, USA, 2011, pp. 290–301. doi:10.1145/2018436.2018470.

[37] R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu, Symnet: Scalable symbolic execution for modern networks, in: Proc. of the ACM SIGCOMM Conf. (SIGCOMM'16), 2016, pp. 314–327. doi:10.1145/2934872.2934881.
URL http://doi.acm.org/10.1145/2934872.2934881

[38] M. Canini, D. Venzano, P. Peresíni, D. Kostic, J. Rexford, A NICE way to test openflow applications, in: S. D. Gribble, D. Katabi (Eds.), Proc. of the 9th USENIX Symposium on Networked Systems Design and Implementation, 2012, pp. 127–140.
URL https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/canini

[39] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, The mathematical foundations for mapping policies to network devices (technical report), CoRR abs/1605.09115 (2016).

[40] A. Paradowski, L. Liu, B. Yuan, Benchmarking the performance of openstack and cloudstack, in: 17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2014, Reno, NV, USA, June 10-12, 2014, IEEE Computer Society, 2014, pp. 405–412. doi:10.1109/ISORC.2014.12.

[41] M. Kuzniar, P. Peresíni, D. Kostic, What you need to know about SDN flow tables, in: J. Mirkovic, Y. Liu (Eds.), Passive and Active Measurement - 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings, Vol. 8995 of Lecture Notes in Computer Science, Springer, 2015, pp. 347–359. doi:10.1007/978-3-319-15509-8_26.

[42] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, X. P. Costa, Benchmarking open source NFV MANO systems: OSM and ONAP, Comput. Commun. 161 (2020) 86–98. doi:10.1016/j.comcom.2020.07.013.

[43] G. Marchetto, R. Sisto, M. Virgilio, J. Yusupov, A framework for user-friendly verification-oriented VNF modeling, in: Proc. of the 41st IEEE Conf. Computer Software and Applications, 2017, pp. 517–522. doi:10.1109/COMPSAC.2017.16.

[44] H. Hu, G. Ahn, K. Kulkarni, Detecting and resolving firewall policy anomalies, IEEE Trans. Dependable Secur. Comput. 9 (3) (2012) 318–331. doi:10.1109/TDSC.2012.20.

[45] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for network update, in: Proc. of the ACM SIGCOMM Conf. (SIGCOMM'12), 2012, pp. 323–334. doi:10.1145/2342356.2342427.

[46] B. Jäger, Security orchestrator: Introducing a security orchestrator in the context of the ETSI NFV reference architecture, in: 2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1, IEEE, 2015, pp. 1255–1260. doi:10.1109/Trustcom.2015.514.

[47] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, Automated firewall configuration in virtual networks, IEEE Tran. on Dep. and Sec. Comp. (2022) 1–18doi:10.1109/TDSC.2022.3160293.

[48] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, P. Demeester, Network service chaining with optimized network function embedding supporting service decompositions, Comput. Networks 93 (2015) 492–505. doi:10.1016/j.comnet.2015.09.035.

[49] I. Pedone, A. Lioy, F. Valenza, Towards an efficient management and orchestration framework for virtual network security functions, Secur. Commun. Networks 2019 (2019) 2425983:1–2425983:11. doi:10.1155/2019/2425983.