Politecnico di Torino

Master of Science in Computer Engineering

Master Degree Thesis

Optimized and automatic firewall reconfiguration

Supervisors
dott. Fulvio Valenza
prof. Riccardo Sisto
dott. Daniele Bringhenti

Candidate
Francesco Pizzato

Academic Year 2022-2023

Summary

Automated approaches for network security management, based on formal methods, are recently being proposed to take advantage of all the features of virtualized networks, in particular the additional flexibility and dynamicity. These solutions allow the creation and deployment of large and complex architectures with a lower latency compared to traditional approaches, which are often based on trial-and-error and human labor. These tools adopt a formal approach to the problem which guarantees the correctness of the solution by construction and so allow avoiding misconfigurations.

In this context, an interesting aspect that can be further developed is the possibility to exploit the characteristics of virtual networks for the problem of reconfiguring an already deployed network. The flexibility of an automated approach allows creating solutions which are reactive to the network, capable of generating a new configuration as an automatic response to external events. This thesis proposes a possible approach for the generation of an updated version of the configuration with the goal of achieving the result with the smallest possible computation time, without losing the formal correctness of the result. The short computation time is a feature which could be useful in different situations, especially in the field of cybersecurity, where the elapsed time from the beginning of an attack to the deployment of a solution is crucial. Having a solution which prioritize the speed would permits a rapid recovery from the event of an attack, while still being formally correct with respect to all the other security policies in place for the network.

This thesis contributed to the development of one of this automated approaches, VEREFOO (VErified REFinement and Optimized Orchestration). The framework implementing this approach can automatically generate an optimal network configuration starting from a high-level definition of desired *Network Security Requirements* (NSRs), using a problem solver and a series of clauses to model the different NSRs and network elements. The computation time required by the framework is small and its scalability over more complex networks has already been proven, however an efficient reconfiguration process is not supported by the tool and the configuration must be recomputed from scratch every time the set of NSRs is updated. The contribution of this thesis was to redesign part of the tool for generating a new formally correct configuration for an already configured network, avoiding recomputing it from zero and achieving a relevant advantage in the computation time. To achieve this result, I used traffic flows modeling solutions and adapted them to my needs, checking which elements in the network must be reconfigured based on the changes in the NSRs set. The implementation focuses, among the many possible *Network Security Functions* (NSFs), only on packet filter, which is

the most common firewall technology used to enforce security policies. To make the solution really effective also the clauses considered by the solver have been updated.

Finally, the implemented solution has been extensively tested to assess the performance improvement in different configurations, testing network of increasing sizes and different reconfiguration scenarios. The results are demonstrating the feasibility of the proposed approach and the possible advantages in terms of computation time when compared to the previous version, based on a complete reconfiguration of the whole network.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Thesis introduction

One of the major networking trend in recent years has been *Virtualized networks* and the "softwarization" of all its different components thanks to the advent of new technologies, the main one being *Software-Defined Networking* (SDN) [1] and *Network Functions Virtualization* (NFV) [2][3]. In particular, NFV is a network architecture paradigm that leverage virtualization technologies for the deployment of network functions in a virtualized manner so that they could be placed on generic hardware, whereas SDN is an emerging paradigm that promise the separation of the network's control logic from the underlying routers and switches, in favor of a logical centralization of control and introducing the ability to program the network, allowing to implement any desired network topology through software. Moreover, they ease the process of network management allowing the implementation of a centralized control point and the development of new solutions for handling large distributed networks with more flexibility.

Automated approaches, that take advantage of all the above features, have already been proposed and published in the scientific literature. These automated solutions benefit the most of the added flexibility and dynamicity of network softwarization, and they have been adopted in various scenarios such as IoT [4], Smart Home [5], and industrial networks [6]. In this context, a relevant problem is the design of the *Service Graph*(SG), a logical representation of the network including communication endpoints and network functions, and the allocation and configuration of the needed *Network Security Functions* (NSFs), such as firewalls, IPS and anti-spam filters, on the base of a given set of *Network Security Requirements* (NSRs). The classical approach is to perform these tasks manually, but it is prone to human errors and with a reaction time which is not negligible. Network Automation tools represent an alternative approach for automating the process of allocation and configuration of the required NSFs starting from a set of high-level NSRs and a logical representation of the network which is provided by the service designer [7]. Moreover, most of the proposed automated approaches are based on formal methods, so they can guarantee the correctness of the solution by construction avoiding the possible misconfigurations due to human labor. The exploit of these forms of automation is becoming essential today especially in the field of cybersecurity, like

it is expressed in [8] [9]. The complexity of modern networks can not be managed with traditional methods, which result in wrong configurations not enforcing the needed security policies and slow response time. Automated tools can help with faster reconfigurations by providing a more efficient and reactive process that is formally correct, and could improve network security management.

The scenario considered in this thesis work is indeed the one of a cybersecurity attack. In this case the time required to react and actuate a mitigation for the attack is crucial, since the period elapsed between the detection and the computation of a solution corresponds to the interval in which the system is vulnerable, the longer is this time and the higher could be the caused damage to the infrastructure. Additionally, the misconfiguration of NSFs is one of the primary cybersecurity risks nowadays, that is confirmed by recent surveys such as Verizon's most recent study [10] in which misconfiguration errors continue to be a prevailing cause of breaches, responsible for 13% of breaches over the past year. As the complexity of modern network increases, the unfeasibility of producing a correct configuration manually is clear, due to the inability of a human being to have a general vision of the whole problem. For all these reasons, Cybersecurity is one of the main area of interest for the development of automatized solutions to manage network security policy.

Even if many of these automated approaches have been proven effective and with better performances than the manual configuration approach, they are not specialized for the proposed scenario of a cybersecurity attack requiring a reconfiguration of the network. One aspect that can be further developed is exactly this, the possibility to exploit the characteristics of these automated solutions to improve the reconfiguration performance of an already configured network. Even if these solutions are already capable of reconfiguring the network in response to external events [11], like a change in the policies or a cybersecurity attack, they are not focused on the optimized resolution of this problem. The framework this thesis contributed to is VEREFOO: considering its current implementation, the configuration has to be recomputed from scratch every time there is a change in the set of requirements with an associated computational cost which is not negligible (from hundreds of seconds to several minutes depending on the network complexity). Instead, having a tool that can produce a reconfigured version of the actual configuration with the smallest possible computation time could be very important in the previously described scenario of a cybersecurity attack. Furthermore, the computed solution should not break the system but maintain the validity of all the other NSRs already in place. For this reason it has been important to maintain the formal approach to the problem used by VEREFOO that allows to guarantee the correctness of the configuration by construction.

After all these considerations, this thesis wants to study and propose a model for the reconfiguration of a network with the goal of achieving the shortest possible computation time without losing the formal correctness of the configuration, aiming to solve many of the issues present in nowadays solutions. Then the proposed model has been implemented as a part of the VEREFOO framework and it has been extensively tested in different scenarios to compare the produced solution with the original version, which is performing a complete configuration from zero every time there is a change in the security policies.

1.2 Thesis description

The remaining of the thesis is structured in the following way:

- **Chapter 2** describes how we could represent network packets, or traffic, and how we could use this representation to model the network, all its elements, and the network security requirements. Also the general behvaior of the network is modeled using an unique representation based on traffic flows. Then, this chapter describes two opposite approaches for traffic flow modeling which have been implemented in VEREFOO and are called Maximal Flows and Atomic Flows [12], describing their characteristics, advantages and disadvantages.

- **Chapter 3** describes briefly the structure and the approach of VEREFOO (VErified REFinement and Optimized Orchestration), which is the project this thesis is going to be a contribution and extension of. The focus would be only on those aspects that are needed or have been modified in this thesis work, these are in particular the two used logical representation of a network (Service Graph and Allocation Graph), the connectivity requirements, and parts of the constraints fed to the solver (only those related to the NSRs and the optimality of the solution).

- **Chapter 4** explains the objective of this thesis, introducing the general idea about the approach and the main elements of the work done for reaching the goal. Also a more detailed representation of the reconfiguration model is explained in the end of this chapter.

- **Chapter 5** contains all the contributions of this thesis. The core of the approach is an algorithm for the selection of the area of the network which needs to be updated for supporting the reconfiugration of the NSRs. Then, also the formulation of the clauses in input to the solver has been updated in this thesis for achieving a more efficient computation time. In this chapter it will be also discussed the optimality of the presented solution compared to what would be the product of a complete configuration from zero. Finally, some other implementation details are discussed and a complete and clarifying example of a reconfiguration scenario is presented.

- **Chapter 7** provides an explanation of the testing environment which has been developed and the conducted tests, presenting the achieved results with a reflection about what goals have been achieved, in which magnitude, and the further aspects that should be addressed in the future.

- **Chapter 8** contains the conclusion, which summarize the achieved goals of this thesis and some possible path that could be followed in future work to improve the solution or enrich the capabilities of the framework.

Chapter 2

Traffic Flows Modeling

The increasing trend towards network "softwarization" allows to possibly create and manage complex network environments in a few minutes or seconds, as already explained in the introduction. This additional flexibility is used by automated approaches based on formal methods, to help managing network security in a more efficient and less error prone way. The idea is that the network designer provides a set of high-level NSRs and then an automated tool is in charge of implementing them correctly, possibly applying some optimizations. This area of research is network security automation and some example in this sector are [13] [14]. One of the main problems for the advent of these technologies is the selection of an efficient and performing model for the network, or more specifically for the traffic exchanged between its nodes, which is considered the base element for modeling NSRs. This is fundamental because we should have a representation that is formally correct and at the same time computationally efficient for the subsequent operations performed by the solver. More specifically, the goal is to have an adequate formal model to represent the packets that are originated in the source of a communication and how these are forwarded or modified in a network until the destination is reached. This objective could be seen as a series of distinct sub-goals: we should be able to model a packet crossing a network, the paths that a packet can follow in a network, and the transformations that each network function can operate on a packet that traverse it. In this chapter we introduce a model for representing the network and a subset of all the possible *Network Functions* (NFs) and *Network Security Functions* (NSFs), describing the details only of those which are considered in the work conducted in this thesis (i.e., NAT, Forwarder, Packet Filter). Then, after the introduction of the *Predicate* model for representing network packets, two possible traffic flow models are presented as they were introduced in VEREFOO by a previous work [12]. These two optimized traffic flow representation models are called Atomic Flows and Maximal Flows. They are used to represent, identify, and aggregate classes of network packets (also called traffics) as required in the process for formal and automatic security management. The model allows to compute how a packet that enters the network is forwarded and transformed when crossing the various nodes, as this is necessary to find the optimal placement and configuration of NSFs, on the basis of user's NSRs.

2.1 Network Model

A representation commonly used for modeling a network is a graph, whose nodes represent any possible function that could be present within the network (i.e., web client, web server, firewall, NAT etc.) and whose edges represent links between any two network nodes. In particular, according to VEREFOO, a network is modeled as a directed graph, which means that each edge represent an unidirectional channel and consequently we need two edges for a bidirectional communication channel. An additional distinction done by VEREFOO is between the Service Graph (SG) and Allocation Graph (AG), which are two possible logical representations of the network that will be presented in chapter 3.

In the graph model of a network, each node has some related properties characterizing the element which is allocated in that position. A node has a set of input and a set of output ports, each one being controlled by an *Access Control List* (ACL) that describes whether a packet with a certain header can pass through that port or not. This is used to determine what is called the domain of the node, represented as the combination of two sets: $\mathcal{I}_a$ corresponding to all packets that are allowed to pass through that node, and $\mathcal{I}_d$ representing those packets that are instead blocked. If a node has no ACL, the first set would corresponds to the set of all possible packets, while the second would be the empty set. Once a packet has entered a node, it is subject to the switching operation and transmitted to the corresponding output port accordingly to the rules present in the forwarding table, which is another defining property of each node. Finally, another important characterization is the *transformation function* $\mathcal{T}$. As we said, inside a node a packet could be possibly subject to a transformation, the most common ones being header rewriting, encapsulation, de-encapsulation, and label switching. Therefore, in addition to the forwarding table and the domains $\mathcal{I}_a$ and $\mathcal{I}_d$, each network node has another data structure which is in charge of the decision whether a packet has to be transformed or not, and if yes how it has to be transformed. This is modeled with the transformation function $\mathcal{T}$, that has one or more input domains to which correspond one or more actions and output domains. Once a packet enters a node, it is checked against the input domains of $\mathcal{T}$ and the corresponding function is applied to the packet.

2.2 Models for Network Functions and Network Security Functions

The abstract representation of a given network function which is presented here is considering only the parameters which are required for the scope of the considered problem, the design of an automated policy management system. In particular the model should describe just the forwarding behavior of the network functions and not all their details and implementation aspects. This allows to use an user-friendly high-level language for the description of the Network Security Functions, which is an important feature for this kind of approaches [15]. This section contains the brief description of the previously described node characteristics for the Network

Functions and Network Security Functions which are relevant for the work of this thesis. Starting with the first set, we consider only the following Network Functions with the explanation of their respective models.

- *FORWARDER*: it is a node whose action is to transmit all the packets without applying any modification. In this case the transformation function $\mathcal{T}$ has a single input domain $\mathcal{D}$ that includes all the packets and the function itself is modeled as the identity function since the action is to maintain the packets as they are.
- *NAT*: it has a more complicated representation. It is a network function that could perform three different transformations and consequently it has three different domains. In particular, there are domains $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$, each matching with packets subject to one of the three different operations: a packet matching $\mathcal{D}_1$ is affected by the Shadowing operation, one that matches $\mathcal{D}_2$ by the Reconversion operation, and finally one matching $\mathcal{D}_3$ is forwarded with no transformation applied. The function $\mathcal{T}$ will have three different formulations matching with the three input domains, it corresponds to the identity function for packets in the domain $\mathcal{D}_3$, to a translation of the source address into one of the public addresses of the NAT for packets in $\mathcal{D}_1$, and finally to a translation of the destination address into one of the shadowed addresses for packets in $\mathcal{D}_2$ (in this case the destination address of received packets corresponds to one of the NAT public addresses).

Instead, regarding the Network Security Functions, this thesis considers only packet filter, being the most commonly adopted firewall technology.

- *FIREWALL*: it is a node whose forwarding domain corresponds to the sets $\mathcal{I}_a$ of all the packets that are allowed to pass and the set $\mathcal{I}_d$ of all the packets which are blocked according to the filtering policy of the packet filter. In this case the transformation function $\mathcal{T}$ corresponds to the identity function for all the forwarded packets, so for the packets in $\mathcal{I}_a$, and to the zero function for all those that are blocked, so the packets in $\mathcal{I}_d$ (i.e., $\mathcal{T} : \mathcal{I}_d \to \emptyset, \mathcal{I}_a \to \mathcal{I}_a$).

Note that, even if the proposed approach consider only firewalls, it could be extended to the other Network Security Functions as it has already been done with VEREFOO and channel protection systems [16]. One important aspect to understand is how we can determine if a packet matches with a certain set or not. More precisely, the next aspect to consider is how we could identify a packet so that it could be associated with one of the input domains and the corresponding action to be performed. This is done on the base of some packet contents, usually some of the header fields, which allows to identify single packets but also classes of packets.

2.3 Predicates

A packet is modeled as a predicate computed over some of its fields, specifically parts of the header. Packets that do not differ in these fields are belonging to the

same class and are represented by the same predicate, consequently they are treated in the same way by all nodes which are encountered in the network. Note that a predicate represents what is called a network traffic. The choices for the forwarding domains and transformation behavior of a packet are based exclusively on the predicate that the packet belongs to. It is therefore necessary that also the rules inserted in the ACL of the nodes, the rules defined in the forwarding tables, and the domains of the transformation function, are represented using predicates, so using the same model adopted for network traffic. In this way the predicate describing the incoming packets can be compared with the predicates characterizing each encountered nodes and take a decision about the forwarding and transformation behavior based on the match with the rules or domains of the node. The comparison of two predicates is a fundamental aspect which should be taken into consideration when choosing a model for representing them, in particular a needed characteristic of the selected predicate model should be the usability for different comparison operations such as intersection, union and negation. The ultimate goal is to achieve a full representation of the whole network, including all of its elements, as different sets of predicates that fully characterize the forwarding and transformation behavior of it.

There are different ways to model predicates (i.e., BDD, Tuple Representation [17], Wildcards Expressions [18], FDD [19] etc.), the choice for the data structure used to represent them is crucial and can affect both space and time efficiency of the automated tools that are using them. For the work of this thesis, that aims to be a contribution and extension of an existing approach, VEREFOO, the used representation was introduced by [12] and consists in an implementation of BDD in Java. BDD stands for Binary Decision Diagram, and it is an acyclic, direct and rooted graph structure used to represent Boolean functions. The approach used to represents predicates was introduced in [14]: "a predicate is the conjunction of sub-predicates, one for each packet field that is considered, and this conjunction is denoted by the tuple of its sub-predicates". This means that, considering IP packets, we can represent them as predicates defined over the IP quintuple, i.e. IP Source, IP Destination, port source, port destination, protocol type, and each elements in this set is itself represented with a sub-predicate which describes the single field. The final predicate representing an IP packet is the conjunction of all these sub-predicates. It should be noted that each of the sub-predicate can represent either a single value, a range of values, or even the full range which is denoted with the wildcard symbol "*".

2.3.1 Predicate Model

In the considered implementation of the VEREFOO approach, a predicate is implemented as a Java class with the same name that is used to describe a class of packets. Since we consider IP packets, this class contains the useful information included in the IP header of the packet, the so-called IP quintuple (i.e., {IP source, IP destination, port source, port destination, protocol type}). As we mentioned before, the Predicate class should be able to describe either a single packet or a set of packets, in the first case the predicate fields will be filled with precise and unique values (e.g., {10.0.0.1, 20.0.0.1, 200, 80, UDP}), whereas in the second case its

fields will be filled with ranges of values or using wildcards (e.g., {10.0.0.1, 20.0.0.*, 200, [80-100], TCP}). The Predicate class is briefly introduced in picture 2.1 and in the following segment adopting a bottom-up approach, starting from the smaller classes up to the main one.

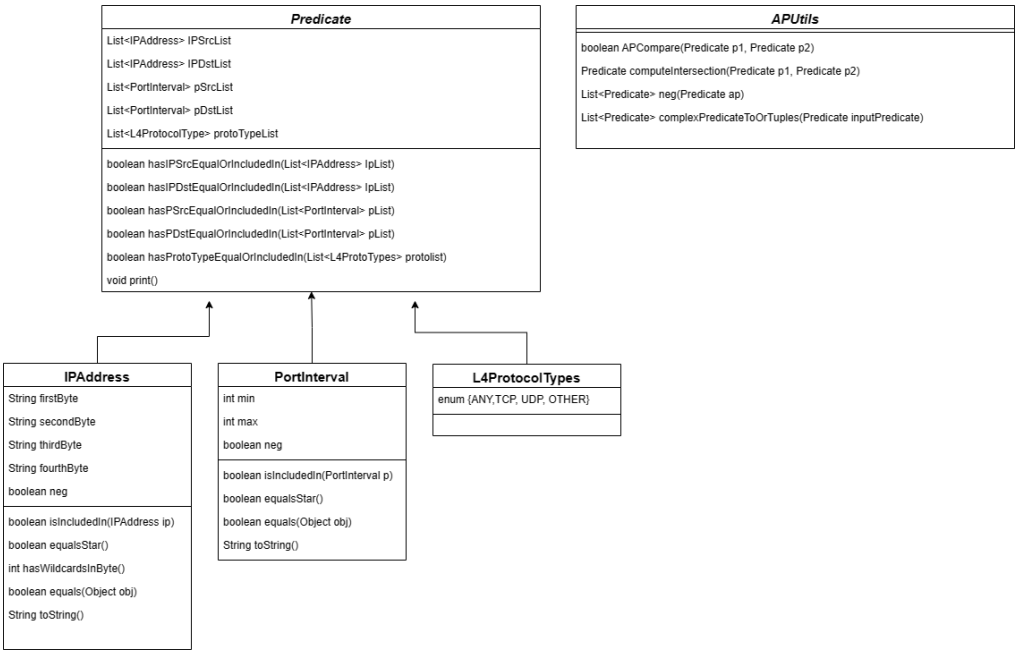

Figure 2.1: UML class diagram describing Predicate

- *IPAddress*. This class is used to model IPv4 addresses, it has four fields, one for each byte of the IP address. Each field can have a value between 0 and 255, or could use the wildcard (represented with value -1) which corresponds to the full range $[0 - 255]$.

- *PortInterval*. It is used to model port numbers within an IP header, both for source and destination port values. It can represent a range of ports using two different values for the fields min and max. The possible ranges must be between the interval $[0 - 65535]$. Note that also in this case the wildcard could be used, and that a single value can be represented setting min = max.

- *L4ProtocolTypes*. This is an enum class that represents all the possible values for the IP protocol type field. VEREFOO considers just two of them, which are TCP and UPD. All other possible protocols are expressed through the value OTHER. Finally, the possibility to express all protocols is given by value ANY, representing the disjunction of {TCP, UDP, OTHER} (there is no wildcard in this case).

- *Predicate*. It is the main class, representing the combination of the five fields to form the IP quintuple {source IP, destination IP, source port, destination port, protocol type}. These fields are all modeled as lists of the sub-classes introduced before. The two lists of IPAddress elements are representing the IP source and IP destination. Each element inside the list is a single address or a subnetwork, then the elements in the list are put in conjunction (i.e.,

AND) one with the others, in this way we could efficiently reproduce all possible intervals combining different sub-predicated. Also the two lists of PortInterval elements are modeled in the same way for the port source and destination attributes. The only change is for the list of L4ProtocolTypes type, which is a list composed of elements that could be taken from an enum class of just four possible values. Hence, the cardinality of the set is small and the sub-predicates inside the list are put in disjunction (i.e., OR) with the others.

Finally note that there is also another class, *APUtils*, which contains many methods to manipulate and work with Predicate objects. Two notable methods are those used to compare and compute the intersection between couple of predicates, which are both frequently used operations.

2.4 Traffic Flow

As it was said before, a class of packets can also be called a traffic, t, and it could be represented using a predicate which is defined over the IP quintuple. More precisely, a traffic is, in the more general definition, the disjunction of one or more predicates, i.e., $t = q_{t,1} \vee q_{t,2} \vee ... \vee q_{t,n}$ where $q_{t,i}$ is a predicate defined over the IP quintuple like $q_{t,i} = (IPSrc, IPDst, pSrc, pDst, tPrt)$. *Traffic Flows* are used to represent the set of all the possible flows of packets that can cross a network, and the set of all traffic flows is represented as F. Each traffic flow describes the behavior of a certain packet class along a path, taking into consideration not only how the packet class exits from the source node and how it is forwarded by the intermediate ones, but also how it is transformed passing through the various nodes encountered travelling from source to destination. Giving a definition, a Traffic Flow $f \in F$ is formally modelled as a list of alternating nodes and traffics, $[n_s, t_{sa}, n_a, t_{ab}, n_b, ..., n_k, t_{kd}, n_d]$. Each node n_i in the list represents a node belonging to the path crossed by the flow, the list starts from the source node n_s, that generates traffic t_{sa}, and reach the destination node n_d, that receives traffic t_{kd}. The generic traffic t_{ij} is the class of packets transmitted from node n_i to node n_j in the flow. The objective of using traffic flows is to exhaustively describe the behavior of an entire network only by means of the set of traffic flows F. In particular, since we want to model network traffic to refine some security properties, we are interested in a subset of all traffic flows that are possible. These are referred as the *interesting flows* and they can be selected by considering only certain sources and destinations for the flows according to the defined security policies. The interesting flows are computed by processing the given set of Network Security Requirements and the configuration of the encountered nodes. It is important to adopt a flow model that can describe in an efficient way the network behavior, in particular the model should permits to compute how a packet that enters the network is forwarded and transformed when crossing the various nodes (i.e., NAT, Load balancers, VPN gateways, firewalls etc.). Two different and alternative models for describing traffic flows have been already studied, implemented, and compared in [12]. The two approaches have an opposite initial idea adequate for the considered problem but each one of them has its advantages and disadvantages. With Maximal Flows we

have a very short computation time for their generation but ah higher complexity for their representation inside the solver, instead with Atomic Flows we have a high computation time for their generation since we must perform a pre-computation step but this approach allows to achieve a more efficent representation for the solver. These will be presented in the following sections.

2.5 Network Security Requirements

Another needed element for the verification or refinement of security policies is the model to use for the Network Security Requirements. Each requirement r is expressed as a tuple (C, a), where C is a condition and a is an action that must be performed on the traffic that satisfy the condition. The condition C is modeled as a predicate representing the IP quintuple corresponding to the packet class which is subject to that specific requirement, in other word the packets matching with the predicate representing the condition are subject to the action, a, which is either one of the two elements $\{ALLOW, DENY\}$. In particular, a traffic flow t satisfies a condition C, and consequently the requirement r, if the three following properties are satisfied:

1. its source and destination endpoints have IP addresses matching respectively $C.IPSrc$ and $C.IPDst$

2. its source traffic satisfies $C.IPSrc$ and $C.pSrc$, which means that t matches with the predicate $\{C.IPSrc, *, C.pSrc, *, *\}$

3. its destination traffic satisfies $C.IPDst$, $C.pDst$ and $C.tProto$, which means that t matches with the predicate $\{*, C.IPDst, *, C.pDst, C.tProto\}$

2.6 Atomic Flows

The first approach that has been considered makes use of *Atomic Predicates* (APs), an idea proposed in 2015 by some researchers as a model for the Network Reachability problem [20]. This idea has then been modified and adapted to the problem of verifying satisfiability of NSRs and the refinement problem in the VEREFOO framework by [12]. With this solution each complex predicate is split into a set of simpler and minimal atomic predicates, and then this set is used for generating the set of interesting flows of the network which could include only elements in the computed set of Atomic Predicates as traffics between any two nodes. Given a set of predicates, it is possible to compute a set of corresponding APs that is minimal, unique, and fully representative of the initial set. In particular, given a predicate P, the corresponding set of Atomic Predicates $A(\{P\})$ are computed as follow:

$$A(\{P\}) = \begin{cases} \{true\}, if P = false or true \\ \{P, \neg P\}, otherwise \end{cases} \tag{2.1}$$

Given two sets of Atomic Predicates $P_1 = \{b_1, ..., b_l\}$ and $P_2 = \{d_1, ..., d_m\}$, the set of Atomic Predicates corresponding to their union $P_3 = A(P_1 \cup P_2) = \{a_1, ..., , a_k\}$ is equal to:

$$\{a_i = b_{i_1} \wedge d_{i_2} | a_i \neq false, i_1 \in \{1, ..., l\}, i_2 \in \{1, ..., m\}\} \tag{2.2}$$

Then, having a generic set of predicates P, the corresponding set of Atomic Predicates $A(P)$ is computed applying in an iterative way the formula 2.2 to the sets of Atomic Predicates generated for each element in P using equation 5.1. The resultant set of Atomic Predicate represents the smallest set of disjunct predicates such that each predicate, of the set over which they are computed, can be expressed as a disjunction of a subset of them (this is expressed in the following definition).

Definition 2.6.1 *Given a set $\mathcal{P}$ of predicates, its set of Atomic Predicates $\{p_1, p_2, ...p_k\}$ satisfies these five properties:*

1. $p_i \neq false, \forall i \in \{1, ..., k\}$
2. $\vee_{i=1}^{k} p_i = true$
3. $p_i \wedge p_j = false, if\ i \neq j$
4. *each predicate $P \in \mathcal{P}, P \neq$ false, is equal to the disjunction of a subset of atomic predicates*

$$P = \cup_{i \in S(P)} p_i, where\ S(P) \subseteq \{1, ..., k\}$$

5. *k is the minumum number such that the set $\{p_1, ..., p_k\}$ satisfies the above four properties*

The *Atomic Flows* (AFs) approach make use of APs to describe each traffic that can cross the network and to configure each firewall with rules expressed using only atomic predicates, or more in general it is describing a network and its behavior using only the predicates in the set of APs. The idea is to start from those that we call "interesting" predicates and then compute the corresponding set of Atomic Predicates as described before. We consider as "interesting" all those predicates linked to nodes which are related to one of the given NSRs, so the predicates representing the source traffic (i.e., the traffic generated from the source node) and the destination traffic (i.e., the traffic reaching the destination node) of each requirement, but also predicates describing input traffic classes and transformation behavior for transformers crossed on the paths. After having computed the set of Atomic Predicates for all the "interesting" predicates of the network, we proceed to generate for each user requirement all possible AFs. Then we use them as input to the MaxSMT solver to allocate and configure the needed Network Security Functions.

Definition 2.6.2 *A flow $f = [n_s, t_{sa}, n_a, ..., n_h, t_{hi}, n_i, ..., n_k, t_{kd}, n_d]$ is defined atomic if each traffic $t_{ij} \in B$, where B is the set of Atomic Predicates computed from the set of interesting predicates.*

The main advantage is that the predicates are unique by definition, so each one can be associated to a different integer identifier. In this way, the solver could work using simple integers as representation of the traffics instead of more complex representations like the Predicate class presented before. Consequently, the resolution performances of the solver is greatly improved using this solution since it allows to define a leaner model of the problem. Moreover, we already said that many operations are based on intersections and unions, which are far less complex if performed over sets of integer compared to sets of more complex predicate representations. Using a complex Java class, as the one showed previously, would imply giving the solver multiple variables corresponding to the different fields of the class and so producing a more complex definition of the problem. Another advantage of using APs it is the easier configuration of Network Security Functions. This is important while configuring a firewall's rule set, each configured rule is associated to a given input traffic, but considering atomic predicates which are by definition disjoint, we can be sure that each configured rule is blocking only an atomic portion of the traffic and has no effect on the others, which would be totally disjunct from the predicate used in the rule. On the opposite side, working with integers is causing the inability of the solver to merge multiple configured rules into a single one, and so it will generate a larger number of configured rules. This is because the solver has no visibility on the IP addresses and other details of the predicates, in fact it is working with opaque integer identifiers. It is however possible to apply a post-processing task to aggregate the rules after reconverting the integer identifiers to the IP-quintuple. In this sense an approach using AFs would not produce the smallest possible number of configured rules in an absolute sense but it achieves the smallest number of disjointed rules, hence not the absolute optimal solution. Another disadvantage of this approach is the initial computation time required for generating the set of Atomic Predicates from the set of interesting predicates. This step is computationally intensive because it requires to process one interesting predicate at the time and compute its intersection with all the others APs in the set before adding it, so that the set contains only disjointed traffics.

2.6.1 Brief description of the computation of Atomic Flows

We briefly present the idea behind the algorithm for computing the set of atomic flows. The starting point is the set of NSRs, as we have seen each requirement r can be seen as the tuple (C, a) made of a condition and an action, where the condition C is a predicate like $\{IPSrc, IPDst, pSrc, pDst, tProto\}$. For each requirement we evaluate all the possible paths connecting $C.IPSrc$ with $C.IPDst$. Then, we compute the set of interesting predicates for the network considering the requirements and all the transformer nodes encountered in at least one path, from this set we generate the set of Atomic Predicates. Having completed this, which is the most computational heavy phase, we consider one requirement and one path at the time and generate the Atomic Flows for the selected combination. The traffic generated by the source of the requirement is grouped in a set B_0, this contains the disjunction of all the Atomic Predicates whose source IP and port are equal to those expressed in the condition of the requirement. Each single Atomic Predicate in B_0 is propagated along the path and possibly modified by the middleboxes until the

destination is reached. Note that with the term middleboxes we consider "any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and a destination host" [21]. In this way we obtain all the possible of Atomic Flows, associating to the set of requirements all the possible flows from the paths and the traffics exchanged between each couple of nodes. However some pruning is needed, not all flows are part of the solution, either because they reach the destination with an incorrect predicate (i.e., different from $\{*, C.IPDst, *, C.pDst, C.tProto\}$) or because they reached the destination without being the endpoint of the path (i.e., they arrive at another endpoint or are dropped along the path). Remarkably, the most demanding phase of this process is the computation of the Atomic Predicates, since the generation of Atomic Flows could be easily executed in parallel and implies a simple traversal of a graph. For the computation of this set we have to generates all the interesting predicates and then process them one by one to compute the final set of APs, this operation become very demanding as the number of interesting predicates increases, which is correlated to the the number of NSRs and transformers.

2.7 Maximal Flows

The second and opposite model is called *Maximal Flows*(MFs), which details can be found in [22]. With the previous solution we were trying to split the traffic flows into smaller ones, reaching the highest level of granularity but also an higher number of traffic flows. Instead, with this second approach we try to do the opposite, reducing the number of generated flows aggregating different sub-flows into a single Maximal Flow, which is still representative for all the ones that have been joined. All traffic flows which have been merged in the same Maximal Flow must behave in the same way when crossing the various nodes of the network, in this way we reach a larger granularity and a smaller number of flows representative of the network. Also in this case the traffic flows are modeled as a list of alternating nodes and predicates, however the used predicates are no longer atomic but express the disjunction of several IP quintuples. The set F_r^M of Maximal Flows is defined as a subset of F_r containing only the flows that are not subflows of any other flow in F_r.

Definition 2.7.1 *Called F_r the set of all possible flows of the network, the corresponding set of Maximal Flows F_r^M matches the following definition:*

$$F_r^M = \{f_r^M \in F_r | \nexists f \in F_r . (f \neq f_r^M \wedge f_r^M \subseteq f\}$$

We aggregate into the same Maximal Flow all the flows behaving in the same way, that should be treated in the same manner by the network. Then, the design and resolution of the MaxSMT problem is modeled using only the set F_r^M which has a smaller size than F_r but the same expressiveness. The main advantage of this solution is that the algorithm for computing the set F_r^M is much faster than the one to compute the set of Atomic Flows, mostly because this algorithm, with respect to the previous approach, does not require any initial computation time for computing the traffic flows. The main disadvantage is that the traffics exchanged

between nodes for each Maximal Flow are not disjointed and unique, and this implies that they can not be associated with an integer identifier like it could be done with Atomic Predicates. The solver has to work with a representation of the predicate, in our case the Predicate Java class presented before. This solution requires in total 13 fields: 4 integers representing the source IP address, 4 integers representing the destination IP address, 2 integers for representing the range of source ports, 2 integers for the range of destination ports, and finally a string for the protocol type. There are many more variables given in input to the solver and this has a significant impact in the final resolution performance.

2.7.1 Brief description of the computation of Maximal Flows

For the resolution of the MaxSMT problem, the necessary step is to compute all the Maximal Flows for the given network security requirements. The set F_r^M of Maximal Flows is computed for each Network Security Requirement r on the basis of an algorithm that consider one requirement at the time and compute the set of paths in the allocation graph which satisfy the condition of the requirements. Then, for each path, all the Maximal Flows $f \in F_M^r$ correlated to the selected path are computed and added to the result set, this is conducted by an iterative process. The starting point is the largest traffic that satisfies the source component of the requirement's condition, this is then propagated forward on the path and updated accounting for the possible transformations performed on traffic by the encountered network functions, and it is split into sub-traffics when needed. In particular the flow is divided into smaller sub-flows only when it encounters a node which forwarding domain and transformation behavior require this operation. Then, the set of computed flows has to be restricted in order to select only those satisfying the destination predicate of the requirement and this is done in a backward traversal of the path, by propagating all the computed elements in F backwards. This process is possibly repeated for several iterations until the final set of Maximal Flows is found.

Chapter 3

VEREFOO

In the context of security automation, new solutions have been recently proposed for implementing an automated approach to policy-based network security management systems, an example of research in this field is [23]. These solutions exploit emerging technologies like *Network Functions Virtualization* (NFV) and *Software-Defined Networking* (SDN) to improve network management and access to new degrees of networking flexibility. The configuration of security functions is an operation commonly performed manually, making it likely to lead to incorrect configurations and long timing required for the application of any change in the configuration. As the complexity of modern network increases, the unlikelihood of producing a correct, and possibly optimized, configuration manually is obvious. Automated approaches would allow to compute a configuration more efficiently and also avoiding human errors. The majority of these approaches are based on formal methods which are guaranteeing the solution correctness by construction. Additionally some of these tools allow other features, the main one being to seek for optimality in the solution. This is for example used to produce a configuration which is not only correct but also optimized to minimize resource usage. The remaining part of this chapter will introduce one of these security automation approaches based on formal method. The VEREFOO approach is based on the design of a partial weighted Maximum Satisfiability Module Theories (maxSMT) problem, which resolution would provide a formally correct configuration of the input Network Security Requirements and also the minimization of the number of used firewalls and configured firewall rules. The general structure of the tool with all its modules will be presented in this chapter, with an emphasis on the aspects this thesis has contributed to the development. Also an overview of some more theoretical concepts are presented, in particular the distinction between Service Graph and Allocation Graph, and a final overview of the most relevant constraints defined for the MaxSMT problem.

3.1 VEREFOO

VEREFOO (VErified Refinement and Optimized Orchestration) is a framework that manages the creation, configuration and orchestration of a complete end-to-end network security service following an approach based on formal methods.

VEREFOO manages the optimal allocation and configuration of the needed Network Security Functions on a provided Service Graph in order to fulfill the input Network Security Requirements expressed using a high-level language. The framework uses the z3 solver for the resolution of a MaxSMT problem in order to find the correct and optimized configuration.

The VEREFOO architecture is showed in figure 3.1 and a brief description of all its modules is now presented.

- the user interact with the framework through the **Policy GUI**, which receives the definition of the Network Security Requirements which must be enforced. The NSRs could be expressed either in an user-friendly language with the *High-Level Policies* (HLP) or with a higher level of details using the *Medium-Level Policies* (MLP). Note that policies expressed using HLP are then translated automatically into MLP by the **High-To-Medium (H2M)** module, so that they could be processed in the same way for the creation and configuration of NSFs.

- the given requirements are subject to a first processing phase by the **Policy Analysis Module (PAN)**. This module has the goal of detecting errors and possible conflicts in the input set of network security policy. It will return the minimum set of requirements that must be satisfied or produce a report if the conflicts were not possible to be solved automatically.

- Considering the given NSRs, the **NF Selection Module (SE)** is in charge of deciding which are the NSFs necessary to satisfy them. Note that the possible functions are selected from a pre-built catalogue of all the available ones, this is represented by the **NF catalogue** element in the picture.

- the second input received by the framework comes through the **Service GUI**. Using this component the user could define the Service Graph (or directly the Allocation Graph) that is later adopted for enforcing the requirements. Note that also this module is linked with the catalogue of all the NFs, so that the user could select those to be included in the graph.

- the core element is the **Allocation, Distribution and Placement Module (ADP)** which receives the set of conflict-free NSRs, the selected NFs, and either the SG or the AG (if it receives the SG it will automatically generate the corresponding AG). The scope of this module is to produce the final graph with the allocated and configured NSFs. This is the central element, the one using the z3 problem solver for computing the solution of a partial weighted MaxSMT problem which would produce the optimized configuration. The NSRs are introduces as hard constraints which must be always satisfied by the solver, while other specifications and optimizations are introduced as weighted and optional soft constraints. We will see later this step in more details.

- finally, the last module is the **Medium-to-Low (M2L) module** which takes in input the list of medium-level policy rules produced by the solver and translates them into low-level language, depending on the actual implementation of the network functions.

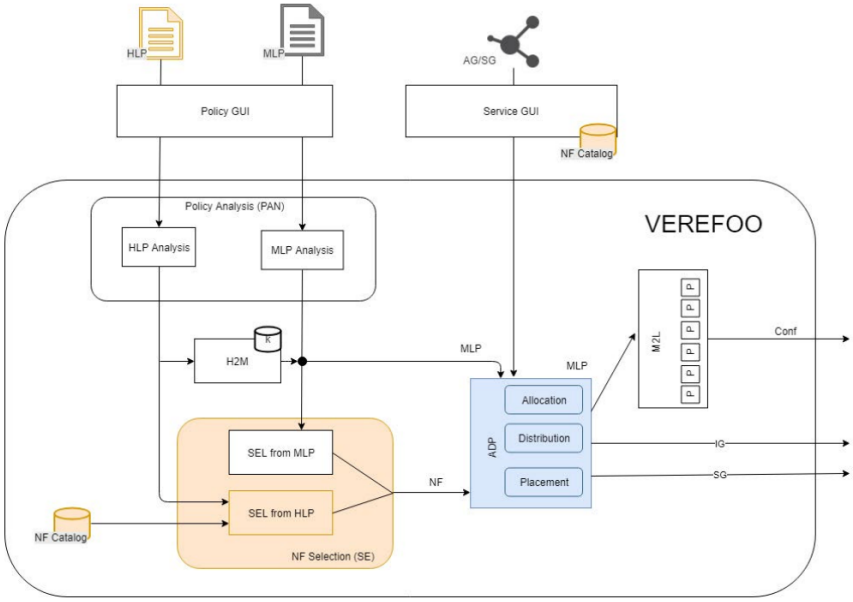

Figure 3.1: VEREFOO Model

3.2 Service Graph

Service Graph (SG) is a logical topology of a virtual network. It represents an interconnection of service functions and network nodes providing a complete end-to-end network service. This is a generalization of the *Service Function Chain* (SFC) concept, which instead is the representation of a linear and ordered set of abstract service functions that must be applied on selected packets/flows [24]. The difference is that in a Service Graph the functions do not need to be positioned in a linear combination but they can be organized in a more complex architecture with multiple paths from source to destination and also with loops, just like in a generic network model. The Service Graph could exploit only a defined set of services, the *Network Functions* (NFs), which include various functionalities such as load balancing and web-caching, but also other simpler functions like NATs and forwarders. Note that low level elements like routers and switches are not represented in the Service Graph, they are implicit in the representation.

In the context of VEREFOO, the Service Graph could be represented as $G_s = (N_s, L_s)$ and it is characterized by just two sets: N_s which is the set of network nodes including both the endpoints of the communications and the middleboxes, and L_s which is the set of the links interconnecting pairs of nodes. The result is an abstract view of the network including all the possible paths a packet could follow and some functions that are uncorrelated with the security policies to be enforced.

3.2.1 XML schema of the Service Graph

The Service Graph is represented by the XML schema of a *graph* element. This is characterized by an unique identifier, a boolean attribute named *ServiceGraph*, and a sequence of *node* elements. The boolean attribute is used to specify if the

XML schema represents a Service Graph (value true) or an Allocation Graph (value false), since the same representation could be used for both.

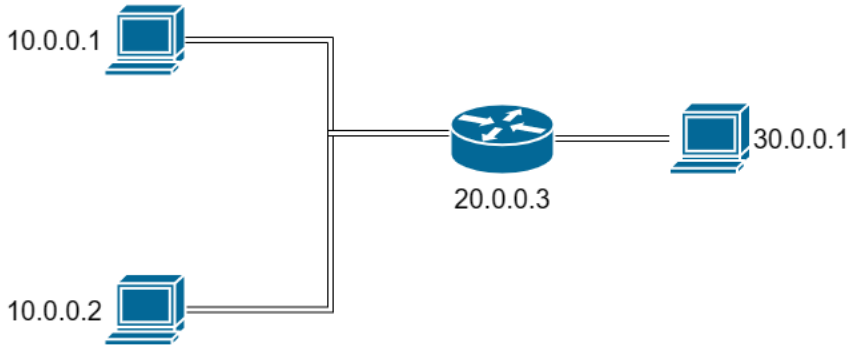

Figure 3.2: Graphical example of a Service Graph

Listing 3.1: XML example of a Service Graph

```
<graph id="0" serviceGraph="true">
    <node functional_type="WEBCLIENT" name="10.0.0.1">
        <neighbour name="20.0.0.3"/>
        <configuration description="WebClient_description" name="WC_Conf">
            <webclient nameWebServer="30.0.0.1"/>
        </configuration>
    </node>
    <node functional_type="WEBCLIENT" name="10.0.0.2">
        <neighbour name="20.0.0.3"/>
        <configuration description="WebClient_description" name="WC_Conf">
            <webclient nameWebServer="30.0.0.1"/>
        </configuration>
    </node>
    <node functional_type="FORWARDER" name="20.0.0.3">
        <neighbour name="10.0.0.1"/>
        <neighbour name="10.0.0.2"/>
        <neighbour name="30.0.0.1"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node functional_type="WEBSERVER" name="30.0.0.1">
        <neighbour name="20.0.0.3"/>
        <configuration description="WebServer_description" name="WS_Conf">
            <webserver>
                <name>30.0.0.1</name>
            <webserver/>
        </configuration>
    </node>
</graph>
```

As we see in the listing 3.1, that contains the XML representation of the network in 3.2, each one of the *node* elements represent either a service function or a communication endpoint and it is characterized by different elements:

- a *name* attribute representing the IP address of the node (or an unique string).

- a *functional_type* attribute which signals which is the function assigned to the node.
- a list of *neighbour* elements, each one representing one linked node. Note that the connection is unidirectional, the other node must have another neighbour element with the specification of the current node in order to create a bidirectional channel.
- a *configuration* attribute that represent the behavior of the selected network function. Its content depends on the specific functional type selected for the node.

3.3 Allocation Graph

The Service Graph model does not allow the allocation of new firewalls or other network security functions, because every node is already characterized and occupied by a specific function, for this reason we need a second model. The *Allocation Graph* (AG) is a logical topology that could be either produced from zero or automatically generated starting from the Service Graph. AG and SG are internally correlated since they are characterized by the same set of network functions, but the difference is that the AG could be enriched with some additional nodes. Indeed, the allocation graph representation adds to the previous one an extra placeholder node type called *Allocation Place* (AP). An AP is considered as an empty spot which could be used by the automated tool and either be filled with a Network Security Function (i.e., firewall) if it is an optimal position, or left unused. In the implementation, if an AP is left unused but it is part of the path of at least one input requirement, its place would be occupied by a forwarder since its behavior would be to forward each received packet.

The process implemented in VEREFOO to automatically generate an AG starting from a SG consists in adding a new AP on ever link between any two nodes. However, the service designer has the power to impose some constraints on the generation process, either forcing the allocation of a NSF on a specific AP or prohibiting the placement of a new AP in a specific location, and then no network function will be possibly placed. This additional possibility allows to adapt the approach also to traditional networks in which the allocation for the security functions is limited by the physical hardware.

3.3.1 XML schema of the Allocation Graph

The model is represented similarly to the previous one, $G_A = (N_A, L_A)$, with the difference that the set of nodes N_A contains all endpoints, all the service functions, and additionally all allocation places. Considering the XML model, the starting point is the same structure of the SG with the possibility to specify some *node* elements without any *functional_type* and *configuration*, these correspond to the allocation places. Considering the network already used in 3.2 and the relative SG, the automatically generated AG (without any additional constraint) is represented

in listing 3.2 and the picture 3.3. Note also that the flag *serviceGraph* must be set to false in this case.

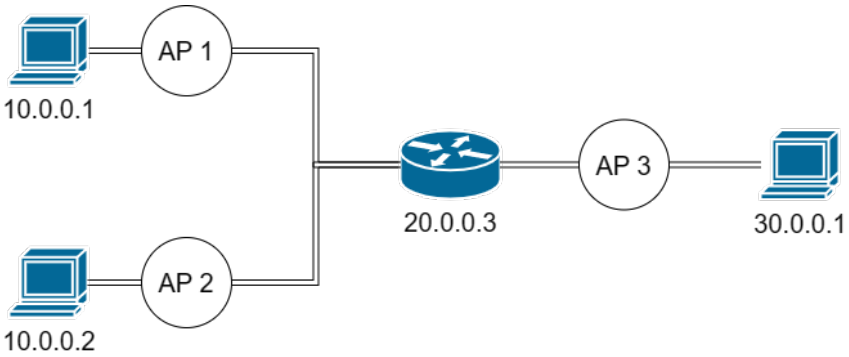

Figure 3.3: Graphical example of an Allocation Graph

Listing 3.2: XML example of an Allocation Graph

```
<graph id="0" serviceGraph="false">
    <node functional_type="WEBCLIENT" name="10.0.0.1">
        <neighbour name="40.0.0.1"/>
        <configuration description="WebClient_description" name="WC_Conf">
            <webclient nameWebServer="30.0.0.1"/>
        </configuration>
    </node>
    <node functional_type="WEBCLIENT" name="10.0.0.2">
        <neighbour name="40.0.0.2"/>
        <configuration description="WebClient_description" name="WC_Conf">
            <webclient nameWebServer="30.0.0.1"/>
        </configuration>
    </node>
    <node functional_type="FORWARDER" name="20.0.0.3">
        <neighbour name="40.0.0.1"/>
        <neighbour name="40.0.0.2"/>
        <neighbour name="40.0.0.3"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node functional_type="WEBSERVER" name="30.0.0.1">
        <neighbour name="40.0.0.3"/>
        <configuration description="WebServer_description" name="WS_Conf">
            <webserver>
                <name>30.0.0.1</name>
            <webserver/>
        </configuration>
    </node>

    <node name="40.0.0.1">
        <neighbour name="20.0.0.3"/>
        <neighbour name="10.0.0.1"/>
    </node>
    <node name="40.0.0.2">
        <neighbour name="20.0.0.3"/>
        <neighbour name="10.0.0.2"/>
```

```
        </node>
        <node name="40.0.0.3">
            <neighbour name="30.0.0.1"/>
            <neighbour name="20.0.0.3"/>
        </node>
    </graph>
```

3.4 Network Security Requirements

The second input that the service designer must provide to VEREFOO is the set of Network Security Requirements. The focus of this thesis is just on connectivity requirements between pairs of nodes, more specifically we consider reachability and isolation requirements. These respectively represent the need of blocking or allowing a communication between two endpoints or subnetworks. In this work the NSRs are expressed with a medium-level language, i.e., each network security requirement, independently from the type, is modelled specifying the IP 5-tuple of the flows that are allowed or prohibited. This is not a limitation since it would still be possible for an administrator to use an high-level language and delegating the framework to their translation in medimum-level requirements using well-known approaches [25]. Therefore, a NSR is modeled as the combination of six elements:

$$[ruleType, IPSrc, IPDst, portSrc, portDst, transportProto]$$

- *ruleType* express which kind of constraint should be satisfied, in our case the possible values are "reachability property" or "isolation property"
- *IPSrc* is the source IP address of the communication
- *IPDst* is the destination IP address of the communication
- *portSrc* is the source port of the communication
- *portDst* is the destination port of the communication
- *transportProto* is the transport-level protocol of the communication

The IP addresses could use the wildcard symbol to represent a subnetwork, if the wildcard is used for one or both of the port attributes it represents the full range of ports from 0 to 65535.

Other than the set of input network security requirements, the approach presented in VEREFOO allows the service designer to specify one out of four possible general behaviors. Each one is characterized by a *default behavior* describing how the framework should manage all the traffic flows for which there is not an explicitly formulated requirement. There are three main approaches and the third one has two further characterizations:

1. *whitelisting*, if all the communications for which no specific requirement is formulated should be blocked, in this case the default behavior is to block all traffic flows and the user can only specify additional reachability requirements.

2. *blacklisting*, if all the communications for which no specific requirement is formulated should be allowed, the default behavior is set to allows all traffic flows and the user can specify only isolation requirements.

3. *specific*, it the service designer does not care about how the communications for which no specific requirement is formulated are handled. In this case the user is interested only in enforcing the requirements which have been explicitly provided, without caring about the behaviour of the other communications. The user could specify both isolation and reachability requirements in this case and the way in which the other traffic flows are handled is automatically decided by the framework to achieve other goals, such as:

   - *rule-oriented specific*, the goal is to minimize the number of configured rules.
   - *security-oriented specific*, the goal is to increase the security of the system and consequently all communications that are not strictly necessary to satisfy the requirements are blocked.

Note that using the third approach, in both variants, the requirements should be conflict free, which is something that should be handled by the PAN module of VEREFOO. In the other two approaches this is not needed since the requirements would be all reachability or all isolation depending on the selected one. In this thesis work we consider the approach *specific*, supposing that the service designer should explicitly express which communications should be blocked and which should be allowed. Moreover, since the PAN module is not the focus of this work and because anomalies can be eliminated with well-known anomaly detection techniques [26] [27], a second assumption is to have an input set of NSRs that is always conflict-free.

3.4.1 XML model of the Network Security Requirements

The XML schema for the input NSRs is represented by the *PropertyDefinition* element, which is internally made by a list of *Property* elements, each representing a single requirement. Every requirement is characterized by an identifier for the correlated Service Graph, or Allocation Graph, and the attributes presented before.

Listing 3.3: XML example of a Network Security Requirement

```
<PropertyDefinition>
    <Property graph="0" name="ReachabilityProperty" src="10.0.0.-1"
dst="20.0.0.2" dst_port="80" />
    <Property graph="0" name="IsolationProperty" src="10.1.1.1"
        dst="130.192.-1.-1" src_port="[2000-3000]" dst_port="80" />
</PropertyDefinition>
```

Note that the source and destination ports are optional fields, if they are not specified the whole range of ports will be considered. Listing 3.3 represents two requirements, one for each kind. The first one represent a reachability requirement for communications starting from the subnet 10.0.0.0/24 (expressed using wildcard, correspondent to value -1) to IP address 20.0.0.2 on port 80. Instead, the second one

is an isolation requirement for the communications starting from node 10.1.1.1 with a port ranging from 2000 to 3000, and with destination the subnet 130.192.0.0/16 listening on port 80.

3.5 Formulation of the MaxSMT problem

VEREFOO uses *z3* [28], a state of the art solver, to resolve an instance of a *partially weighted Maximum Satisfiability Modulo Theories (MaxSMT)* problem, which is a generalization of the SMT problem (i.e., which consist in determining if it is possible to satisfy at the same time all the given first-order logic constraints) that introduce optimization by distinguishing two different sets of input clauses. The solver uses *hard constraints* and *soft constraints*. The first ones are not relaxable and must be satisfied in any case to get a correct solution for the problem, they do not have an associated weight and for this reason we have a "partially" MaxSMT problem. By contrast, soft constraints are relaxable, which mean that their satisfaction is not strictly required, and they have an assigned weight. Hard constraints are used to model the input Network Security Requirements and all the other constraints imposed by the service designer (i.e., the allocation of a firewall in a specific location, prohibiting the allocation of any service function in a certain link). Instead, Soft constraints are used for the optimization objective, since the solver will select among the many solutions the one which satisfy all the hard constraints and produce the maximum value for the sum of the weights of the satisfied soft constraints. In the case of VEREFOO, the problem is modeled with the optimization objective of minimizing resource consumption, preferring the solution with the minimum number of used firewalls and the minimum number of configured rules for each one of them. The formulation of the MaxSMT problem is key for achieving the objectives of VEREFOO, namely full automation, optimization, and formal correctness. Specifically, the approach is fully automatic since it is the solver which produces the correct configuration with minimum human intervention (the user has to provide only the security policies), it is also optimized because the soft constraints are expressing the optimization objectives, and finally it is formally correct because the formal correctness requirements, i.e. the satisfaction of the network security requirements, are expressed with the hard constraints.

This thesis worked mainly with the set of soft constraints for minimization of resource consumption. These have been modified and for this reason they will be now described in more details. In order to present them we need to introduce some auxiliary functions:

$allocated(n) : N \rightarrow B$. This Boolean function returns true if a firewall must be allocated in allocation node n, or false otherwise.

$deny(t) : T \rightarrow B$. Another Boolean function, it returns true if the ingress traffic t is dropped by the node. It is a function used to model the forwarding behavior of a node.

$\pi(f) : F \rightarrow (N)^*$. This function maps a flow f to the ordered list of nodes crossed by that flow, including the destination but not the source.

$\tau(f, n) : F \times N \rightarrow T$. It maps a flow and a node to the ingress traffic of that node for that flow. In case the flow does not cross that node, the empty set is returned.

$enforces(d_k, r) : A \times R \rightarrow B$. This Boolean function returns true if the default action, d_k, of a firewall (allocated in the allocation place k) enforces the requirement r. Basically if the default action is aligned with the requirement, i.e., if default action is "ALLOW" and the requirements is of type reachability or if default action is "DENY" and requirement is isolation.

Optimality of Firewall Allocation

Moving to the soft constraints, the first one regards the minimization of allocated firewalls. In this case, a constraint is defined for each allocation place to express the preference that no firewall is allocated in that position.

$$\forall a_{ij} \in A.Soft(allocated(a_{ij}) = false, c_k) \tag{3.1}$$

Optimality of Firewall Rules Configuration

The second optimality aspect is to minimize the number of configured rules inside each allocated firewall. In this case we adopt the function *enforces* since a rule has to be configured only if it is not already enforced by the default action. The idea is to identify only the security requirements which could need an explicit rule in the firewall (i.e., those rules producing false if used as input for *enforces*):

$$\forall r \in R. \forall f \in F_r. (a_k \in \pi(f) \wedge \neg enforces(d_k, r) \implies (\forall q \in \tau(f, a_k).q \in Q_k)) \tag{3.2}$$

The generated set Q_k corresponds to all possible Predicates describing the traffic incoming to a potential firewall in node k for which a rule is needed. For each one of them it should be created a rule, called placeholder rule p, that could potentially become a configured rule for the firewall but this will be decided by the solver which states if it is necessary or not. The solver will resolve the problem and decide which placeholder rules will be configured. Another useful function is indeed the one that maps each placeholder rule to a boolean value, true if it is configured and false otherwise:

$configured(p) : P \rightarrow B$. This function returns true if the placeholder rule needs to be configured.

Finally, the actual soft constraint which implements this second optimization goal is:

$$\forall p_i \in P_k.Soft(\neg configured(p_i), c_{ki}) \tag{3.3}$$

Note that the constraint states that it is preferred to not configure any placeholder rule. If the solver decides that a placeholder rule gets configured for an allocation place k, then there is an additional hard constraint to force also the allocation of a firewall in that position:

$$\exists p_i \in P_k.configured(p_i)) \implies allocated(a_k) \tag{3.4}$$

Finally, it is important to consider the relationship between the weights of the two soft constraints. It is clearly evident that the second goal should have less priority than the first one, since the non-allocation of a firewall must be always preferred with respect to the non allocation of one of its rules. In general, the cost of allocating a new firewall must be greater than the sum of the costs related to all other soft constrains. If the solver establishes that an allocation place should not be used, it does not make sense to consider the cost of satisfying all the other constraints about the firewall configuration. As a consequence, the weight for the second soft clause should be less than the weight for the first one.

$$\sum_{i:p_i \in P_k} (c_{ki}) < c_k \tag{3.5}$$

3.6 Reachability Requirements Hard Constraint

A Reachability requirement is used to request that a source node, or a source subnetwork, can communicate with a destination node, or a destination subnetwork. This implies that packets belonging to this communication are not discarded by any middlebox, in particular the packet filters, along the path. For a correct satisfaction of the reachability requirement, using the traffic flows concepts defined in chapter 2, there should be at least one traffic flow which is not blocked from source to destination. In this case we consider as input the set of flows $F_r \subseteq F$ which satisfy C, the condition of the requirement r. The hard constraint which must be satisfied for the reachability requirement r is then expressed in the following way:

$$\exists f \in F_r . \forall i . (n_i \in \pi(f) \wedge allocated(n_i) \implies \neg deny_i(\tau(f, n_i))) \tag{3.6}$$

The requirement r is satisfied if exists at least a traffic flow computed for r, for which all nodes in the path of that flow do not block the ingress traffic for that node for that flow.

3.7 Isolation Requirements Hard Constraint

An Isolation requirement is used to block any communication between a source and a destination, each one being either a single node or a subnetwork. All packets belonging to the communication should be discarded by some middlebox in the path. Using traffic flows, an isolation requirement r is satisfied if all flows in the set $F_r \subseteq F$, so all those satisfying the condition C, are blocked. The hard constraint representation of the isolation requirement r is the following:

$$\forall f \in F_r . \exists i . (n_i \in \pi(f) \wedge allocated(n_i) \wedge deny_i(\tau(f, n_i))) \tag{3.7}$$

The requirement r is satisfied if, for each possible traffic flows computed for r, exists on the path of that flow at least one firewall that is allocated and is configured to block the ingress traffic for that flow.

Chapter 4

Thesis objective

Stated how important is the development of automated approaches for network security management, this chapter will present which is the objective of this thesis. As mentioned in the introduction, one aspect which could be improved in current automated solutions is the optimized resolution of the reconfiguration problem, which has an impact especially in cybersecurity related scenarios. The situation of a cyberattack would require a reconfiguration of the currently enforced Network Security Requirements, potentially for isolating a malicious node or blocking some undesired traffic, and this operation should be actuated in the shortest possible time. The longer is the interval elapsed from the beginning of an attack to the deployment of a solution, the higher is the risk for the system to be compromised. Unfortunately current solutions for automated network security management are not focused on producing an updated version of a current configuration but instead they have to recompute the complete configuration starting from zero every time, with a consequent impact on the final computation time. In chapter 3 it was introduced the model and some general information about the automated approach this thesis has contributed to, which is VEREFOO. The information introduced in that chapter will be useful in the following ones to better understand which are the novelties proposed by this thesis, in which context they are placed, and their implementation inside the framework. Also the digression done in chapter 2 about network and traffic flow models will be fundamental to understand the approach which is illustrated in this thesis work.

The functioning procedure of VEREFOO starts with the definition of the requirements by the user, who is also in charge of providing a Service Graph model (or directly an Allocation Graph) for the considered network. Then, the framework will perform some computations to generate all the possible traffic flows which are affected by at least one requirement and then it uses this set as input to the solver. The z3 solver is the core component for the formal and automated allocation and configuration of the Network Security Functions. Considering the various Network Functions already deployed in the network and the computed traffic flows associated to the set of Network Security Requirements, the solver could automatically produce the optimized configuration by solving an instance of a partially weighted MaxSMT problem. The optimality goals of VEREFOO are correlated to the minimization of resource consumption, the produced result should have the minimum number of allocated firewalls and the minimum number of configured rules in each

firewall.

The problem of the original approach used by VEREFOO is the absence of a dedicated resolution of the reconfiguration problem, which was handled by re-computing the whole configuration from zero even for the smallest change in the set of requirements. The main goal of this thesis is to propose a new model for optimizing the reconfiguration of an already configured network and implement this as a contribution to the VEREFOO approach, possibly relaxing the optimality constraints to improve the computation time. This goal has been subdivided into different steps which had to be be achieved to reach the main objective. First, the reconfiguration problem is studied in its characteristics and also modeled with two sets of requirements, the Initial and Target sets. Secondly, the selected approach is presented in depth. The notions of traffic flow modeling are adopted to introduce an algorithm for detecting which network areas must be reconsidered by the solver according to the changes in the set of requirements. In this step it has been important to study in more details the process adopted by VEREFOO that starting from the requirements, proceeds generating the traffic flows and produces the constraints given to the solver. It has been important in this step to also reconsider the constraints which are fed to the solver, introducing some changes to the Soft Constraints to preferably maintain the state of the network, i.e., its current configuration, and consequently achieving a faster resolution of the problem. Finally, it has been crucial to test and compare the implemented solution with the previous approach, highlighting the advantages which this scenario could bring as well as its limitations, describing also the possible further developments.

4.1 Introduction to the Reconfiguration Problem

Stated that this thesis wants to find a more efficient solution for the reconfiguration of a network, the first step is to clearly identify the problem. As already said, the presented solution is intended as a contribution to VEREFOO and represent an alternative approach for the reconfiguration of an already configured network, consequently the subject of this study is a network which has already been configured by a previous run of VEREFOO and which needs to be modified as a response to a change in the policy set which has been made necessary to contrast a cybersecurity attack (situation that, as we said, would require a fast response and mitigation action). However, this scenario is not strictly the only possible one since the proposed solution works, and produce a relevant advantage, in general with any configured network, even if the configuration was performed in a different way. This thesis considers only a change in the set of Network Security Requirements as possible reason for reconfiguration, instead a possible change in the connections or topology is not yet considered in this work.

For the modeling of the problem, it is expected that the user (i.e., the network designer) provides in input to the framework the partially configured virtual representation of the network and two different sets of security policies: the *Initial Set* containing all the NSRs which are already configured in the provided service graph, and the *Target Set* containing the new set of NSRs which must be enforced in the updated configuration. In particular, the second set could contains both new

requirements and others which are already configured correctly in the provided network, these should be maintained as they are also in the new configuration. This situation is shown in the picture 4.1, representing the model of the general scenario in which the Initial and Target sets are partially overlapped. In this situation we can distinguish three different sets of requirements:

- the **deleted** NSRs which are no more needed in the final configuration. These are the requirements present only in the Initial Set I and not in the Target Set T
$$deleted = I \setminus T = \{r \in I : r \notin T\}$$
- the **added** NSRs which are the requirements that should be added in the updated configuration, these are not present in the Initial Set I but added in the Target Set T
$$added = T \setminus I = \{r \in T : r \notin I\}$$
- the **kept** NSRs are those already enforced in the given configured network and should be still enforced in the final configuration. These are the requirements belonging to both sets
$$kept = I \cap T = \{r \in T : r \in I\}$$

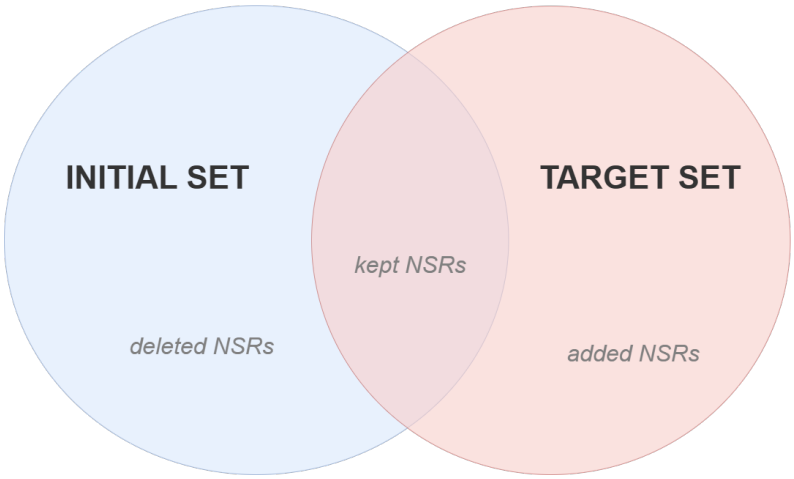

Figure 4.1: Representation of Initial and Target Sets

Notice that this general case has two opposite extreme situations. The first one is the case in which one of the two sets is completely included in the other, which could imply either that all the configured requirements are maintained and some are added (i.e., $I \subset T$), or that only some requirements are kept from the initial configuration and the others are no more needed and so deleted (i.e., $T \subset I$). An even more extreme case is the complete equality of the two sets (i.e., $I = T$), which has no meaning for the problem of reconfiguration since represent the absence of a change in the set of network security requirements. The second extreme situation is the one with no intersection between the two sets, which implies that the new set of requirements has no elements in common with the original configuration (i.e. $I \cap T = \emptyset$). However, in the validation and testing process these extreme cases are

not considered since they are far from the common scenario we are considering. The expected common case of a response to a cybersecurity attack is the one in which only few requirements are added, many are kept, and a few others are possibly deleted.

The idea for the new approach will be presented in the following chapters. The proposed solution is to identify the changes between the Initial and Target sets of Network Security Requirements and then, especially considering the security policies which must be added or deleted, select only the portions of the network whose configuration has to be re-discussed in order to support these modified requirements. The approach aims to enable VEREFOO to detect and reconfigure only the parts of the configuration which needs to be updated while maintaining some other parts as static elements in the configuration.

Chapter 5

Approach for the Reconfiguration problem

The approach elaborated in this thesis is mixing some heuristic techniques with some reformulations of the constraints given to the solver with the final goal of achieving a faster computation time for a reconfiguration while maintaining the formal correctness of the solution. The general idea is to consider the different regions resultant from the comparison of the Initial and Target sets, and then use them for detecting the network nodes which must be put in discussion so that their configuration, or even their allocation, could be re-evaluated by the solver considering the modified set of Network Security Requirements, the Target set. This would allows to maintain parts of the configuration unchanged and consider only a subset of the network as subject to the reconfiguration. The number and complexity of the constraints in the MaxSMT problem are determinant for the performance of the solver, considering only a subset of the nodes in the service graph permits to limit the number of constraints and achieve a faster resolution. The first part of this discussion will include some considerations about the traffic flow models presented in chapter 2 and the selection of the one which is more suitable for the proposed reconfiguration approach. Then, the Refinement process adopted by VEREFOO (i.e., the process of finding the optimal allocation and configuration of security mechanisms on the basis of the requirements expressed in a high-level language) will be briefly covered, following the steps which brings from the Network Security Requirements to the hard and soft constraints which are ultimately fed to the solver. Studying these aspects, the design of an algorithm for the detection of network areas to be reconfigured is presented, as this aspect constitutes the core of this work. This algorithm should select the elements in the network which are effected by the reconfiguration, making their allocation and configuration modifiable by the solver which could evaluate them accordingly to the updated set of requirements. In the last part the changes introduced to the soft constraints are presented, describing their idea and usage.

5.1 Traffic Flow model

Chapter 2 introduced two opposite but equally valid models for traffic flows, which are Maximal Flows and Atomic Flows. Now their differentiating factors are analyzed in more details in order to chose which would suit better for the presented approach, as this is the first step toward the design of a solution. Summarizing what has already been explained, using Maximal Flows we would obtain the smallest number of generated flows because the basic idea is to group all possible subflows into a larger one, so considering a subset of flows which is smaller but equally representative. This approach would produce also the minimum number of configured rules. The main disadvantage is that the traffics within a Maximal Flow are the result of aggregating together multiple traffics, i.e., they are the disjunction of different predicates, and they cannot be "atomic". For this reason the predicates within Maximal Flows should be represented using the Predicate Java class proposed in chapter 2, making it more complex and, as a result, creating a slower and more difficult representation of the traffics in z3. Instead, using the Atomic Flows model we would obtain an higher number of flows since the idea is the opposite one, splitting each traffic flow into multiple minimal and disjoint ones. Consequently using this approach would produce an higher number of configured firewall rules, because it produces a larger number of traffic flows. The main disadvantage of this approach is that it is necessary to execute a long pre-processing task to compute the set of Atomic Predicates for the network, which are needed for computing the set of traffic flows. The advantage is that, being the adopted traffics atomic and disjunct, it is possible to represent each Atomic Predicate with an integer identifier instead of a more complex representation. Consequently the solver could use these integer values for modeling the constraints and the traffic flows, producing a much simpler formulation of the MaxSMT problem.

In our situation, the objective is to use traffic modeling for detecting the network area associated to specific sets of requirements, and then use the selected elements for the reconfiguration steps. The goal is to minimize such area in order to achieve the greatest improvement in the computation time. We must select the traffic modeling solution which allows to identify the smallest set of nodes which need to be reconfigured according to the changes in the Network Security Requirements. This task involve considering the input requirements and the traffic flows associated with them, then the algorithm should select the traffics involved with the added or delete requirements and which portions of the actual configuration should be recomputed because they are in conflict with the updated set of requirements. Using the Maximal Flow approach, since it uses predicates that aggregate multiple traffics, they may include larger portions of traffics and it would be more likely to cause an enlargement of the network area which is selected as to be reconfigured, not obtaining an optimal selection. For example, a configured rule in a firewall that is associated to one Maximal Flow could cover at the same time multiple traffics and it would be more difficult to distinguish the parts of the configuration which are associated just to a single requirement. Instead, using the Atomic Flow approach, each traffic is atomic and disjointed from the others, consequently this modeling solution allows to work on a finer grain. In this way it would be possible to identify for each added or deleted requirement the smallest portion of the network's

configuration which is correlated with the associated traffic flows of the requirement and that should be reconfigured. Moreover, for the aforementioned properties of Atomic Predicates, the firewall configuration results much easier to work with. Each firewall's rule that is blocking or allowing a traffic is specified only for one specific Atomic Predicate, and thanks to the property of being totally disjunct, the configured rule has no effect on any other traffic but only on the selected one. Considering the same example of before, if there is a firewall rule configured on the base of an Atomic Flow, this rule will be associated to only one atomic traffic. After all these considerations, the solution which best fit the presented approach for the reconfiguration problem is the Atomic Flows, which has been selected since it is the one that best suit the purpose of this work allowing for a finer and more precise selection of the area of the network which should be reconsidered, selecting the optimal subset of nodes.

5.2 Definition of added, deleted, and kept sets

The approach presented in this thesis suppose that the service designer provides to the framework two different sets of Network Security Requirements, the Initial set containing all the already configured security policies, and the Final set containing the updated security policies which must be configured in the network. The first necessary step is to determine from these two sets the different groups of requirements relevant for the reconfiguration problem, the added, deleted and kept ones. The used strategy is to compute the intersection of the two sets of NSRs, corresponding to the *kept* group, and then derive the groups of *added* and *deleted* requirements as:

$$added = Target \setminus kept = \{r \in T : r \notin kept\}$$

$$deleted = Initial \setminus kept = \{r \in I : r \notin kept\}$$

In this way we can compute in a simple and fast manner all the required categories. The algorithm for computing the intersection is presented in algorithm 1. This is considering two sets of input requirements R_1 and R_2, and each element is characterized by type (i.e., Isolation or Reachability), source IP, destination IP, source port, destination port, and protocol type. i.e., $[Type, IPSrc, IPDst, pSrc, pDst, tProto]$. Note that each attribute is referenced with the dot notation, i.e, $r.IPSrc$ returns the IP source for the requirement r. The algorithm returns the set of requirements which are present in both sets by comparing each element of R_1 to all the elements of R_2 until a match is found. A match means that all the attributes are identical between the pair of considered requirements.

5.3 Logical Formulation of the Connectivity Requirements

It is important to understand the process which starting from the input set of Network Security Requirements could ends up with the definition of the soft and

Algorithm 1 Algorithm for computing the intersection of two sets of requirements

Input: two sets of requirements R_1 an R_2
Output: the intersection of the two sets $R_{res} = R_1 \cap R_2$

```
1: R_res ← ∅
2: for r_i = [Type, IPSrc, IPDst, pSrc, pDst, tProto] ∈ R_1 do
3:     for r_j ∈ R_2 do
4:         if Compare(r_i, r_j) then
5:             R_res ← r_i
6:             break
7:         end if
8:     end for
9: end for
10: return R_res

11: function Compare(r_1, r_2)
12:     if r_1.Type ≠ r_2.Type then
13:         return false
14:     end if
15:     if r_1.IPSrc ≠ r_2.IPSrc then
16:         return false
17:     end if
18:     if r_1.IPDst ≠ r_2.IPDst then
19:         return false
20:     end if
21:     if r_1.pSrc ≠ r_2.pSrc then
22:         return false
23:     end if
24:     if r_1.pDst ≠ r_2.pDst then
25:         return false
26:     end if
27:     if r_1.tProto ≠ r_2.tProto then
28:         return false
29:     end if
30:     return true
31: end function
```

hard constraints which are provided to the solver. Of course this whole process is studied considering the Atomic Flow modeling solution for the traffic flows, for all the reasons provided in the previous section. The process linking the NSRs to the generated traffic flows is the following: (i) compute the interesting predicates of the network starting from the requirements and from the configured transformers (interesting predicates are the source and destination predicates for each requirement and the domains of transformation of the middleboxes), (ii) from the previous set calculate the corresponding set of Atomic Predicates and use this result to fill the

transformation maps of all transformers with atomic traffics identified with integers, and (iii) for each NSR generate all the related Atomic Flows as described in 2, by passing the atomic traffics through the paths and considering the transformation behavior of each encountered node. After the computation of the associated flows, the constraints that should be fed to the solver must be prepared. In particular we should consider the allocation and configuration soft constraints as presented in 3, which are the two main clauses relevant for this thesis.

As said, this work is limited to the reconfiguration of firewalls due to a change in the Network Security Requirement set. In particular, the requirements which are considered are the connectivity ones, so Isolation and Reachability requirements. These are formulated with an hard requirements which must be satisfied and they have respectively the following logical formulations:

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

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

Considering Isolation requirement, the formula states that there must be an allocated firewall for each Atomic Flow which is blocking the associated Atomic Predicate received in input. In other word, for satisfying an isolation requirement all the flows should be considered and for all of them there must be a node blocking the traffic. Instead for a reachability requirement it is enough that at least one Atomic Flow is not blocked (i.e., all nodes belonging to that flow do not block the relative Atomic Predicate received in input for that flow). To satisfy a reachability requirement all the flows must be considered and it must be checked that there is at least one of them which is not interrupted from source to destination.

The new algorithm, which is now presented, is based on the model of the reconfiguration problem presented in chapter 5 and on the aforementioned logical formulations of the connectivity requirements. The guiding principle is to distinguish between kept, added, and deleted Network Security Requirements, and for each element in these groups detect which parts of the configuration should be reconfigured, either because they are no more needed or because they must be changed for allowing the addition of a new requirement. Note that for the Network Security Requirements in the kept set no operation is necessary, since they are already configured in the provided network and so no nodes should be update for permitting their satisfaction.

5.4 Added Network Security Requirements

The first and most important set is the one of the added Network Security Requirements, which are those belonging to the Target set but not to the Initial set. Each requirement in this group, in principle, is not yet enforced in the actual configuration of the network, however it could be that some of them are already satisfied while others require some changes. The proposed approach consists in considering each requirement in this group and check if it is already satisfied by the provided configuration, and if this is not the case, the algorithm would detect which parts

should be changed because in conflict. This operation has a different meaning depending on the kind of requirement which is considered, for this reason we split the presented reasoning in two separate parts.

5.4.1 Isolation Requirements

Considering an isolation requirement which belongs to the set of added NSRs, for it to be satisfied there must be an already allocated firewall for each Atomic Flow which is blocking the associated Atomic Predicate in input. If there is an Atomic Flow which traffic is not blocked by any of the nodes, then the part of the network configuration which is associated with this flow should be reconfigured. In this case all the nodes crossed by the flow should be selected for reconfiguration since the algorithm could not decide a priori which is the specific node among the many possible ones which is the optimal choice for blocking the traffic. Considering a new Isolation requirement $r \in added$ and a given allocation graph A_G, the procedure to compute the network elements to be reconfigured is presented in algorithm 2. The starting point is the set of Atomic Flows F_r correlated to r. We recall some functions which are useful: $\pi(f)$ returns the nodes belonging to the given flow (excluding the source), $allocated(n)$ returns true if there is a firewall allocated in the given node, $deny_n(t)$ returns true if the node n is configured to block traffic t, and finally $\tau(f, n)$ returns the traffic in input to node n for flow f.

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

Input: an isolation requirement r, and an AG G_A
Output: nodes to be reconfigured $N_{reconfigure}$

```
1:  for f ∈ F_r do
2:      found ← False
3:      for n_i ∈ π(f) = [n_1, n_2, ..., n_d] do
4:          if allocated(n_i) & deny_{n_i}(τ(f, n_i)) then
5:              found ← True
6:              break
7:          end if
8:      end for
9:      if found == False then
10:         N_reconfigure ← π(f)        ▷ All nodes in the path should be reconfigured
11:     end if
12: end for
13: return N_reconfigure
```

The algorithm would consider for each added Isolation requirement all the related flows, then checks (lines 3-7) if there is a node belonging to the flow's path which is blocking the traffic received in input. If there is no such node for a flow, then all the nodes belonging to the path are selected as to be reconfigured since the algorithm can not decide which among the many nodes would be the best selection

for blocking the traffic. The decision is taken later by the solver. Note that only the nodes which have already a configuration in the provided graph are added to the set of nodes to be reconfigured $N_{reconfigure}$, in particular only firewalls and forwarders since they the two possible outcome of an used allocation point (i.e., an allocation point crossed by a traffic flows could become a firewall if the solver decides it is an optimal position, or a forwarder otherwise).

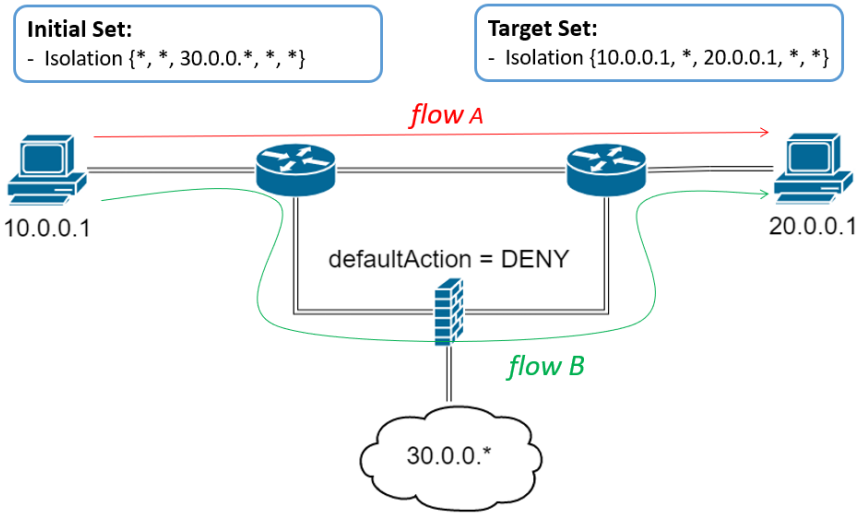

Figure 5.1: Example of addition of an Isolation requirement

Considering as example the figure 5.1, the given inputs are the two sets of Initial and Target requirements and the already configured network, which is composed of a firewall and two forwarders. As depicted, the new requirement that should be added is the Isolation from the client 10.0.0.1 to the server 20.0.0.1, with any ports and protocol. The associated paths are just two and each one has an Atomic Flow, called respectively A and B. The algorithm would check if there is a node blocking the traffic for each one of the flows. In this case flow B crosses a firewall which blocks all traffic as default action and so the condition is satisfied, instead flow A does not include any network function which is blocking the traffic and so the configured nodes belonging to its path must added to the set $N_{reconfigure}$ (i.e., the two forwarders in this case).

5.4.2 Reachability Requirement

The other kind of connectivity requirement is Reachability. Considering a reachability requirement belonging to the set of added NSRs, for it to be satisfied there must be at least one associated traffic flow which is not blocked from source to destination. If such Atomic Flow is not found, the algorithm considers for all the correlated flows all the nodes that are blocking the traffic in the provided configuration, and adds them to the set of nodes which are possibly reconfigured, $N_{reconfigure}$, for satisfying the requirement. Even in this case the algorithm could not select which is the optimal Atomic Flow selected for the satisfaction of the requirements, it is something under the responsibility of the solver. Considering a

new Reachability requirement $r \in added$ and a given allocation graph A_G, the procedure for selecting the network elements to reconfigure is presented in algorithm 3. In this case the formulation is different because the condition to be verified is the opposite of the previous one, and because we could possibly stop the algorithm before having checked all flows if it finds one which is uninterrupted from source to destination and that satisfies the requirement. This possibility is handled using a temporary structure called $tmpReconfigure$ which keeps track of all nodes that could be possibly reconfigured if no such flow is found. In fact, if a flow that satisfy this condition is found the structure is emptied (lines 9-10) and no nodes is selected for reconfiguration, otherwise it is returned as the set of nodes to be reconfigured $N_{reconfigured}$ (line 16).

Algorithm 3 Algorithm for selecting network area to reconfigure for each added Reachability requirement

Input: a reachability requirement r, and an AG G_A
Output: nodes to be reconfigured $N_{reconfigure}$

1: $tmpReconfigured \leftarrow \emptyset$
2: **for** $f \in F_r$ **do**
3: $\quad tmpFlow \leftarrow \emptyset$
4: $\quad$ **for** $n_i \in \pi(f) = [n_1, n_2, ..., n_d]$ **do**
5: $\quad\quad$ **if** $allocated(n_i)$ & $deny_{n_i}(\tau(f, n_i))$ **then**
6: $\quad\quad\quad tmpFlow \leftarrow n_i$
7: $\quad\quad$ **end if**
8: $\quad$ **end for**
9: $\quad$ **if** $tmpFlow.isEmpty()$ **then** ▷ Found a flow satisfying the requirement
10: $\quad\quad tmpReconfigured \leftarrow \emptyset$
11: $\quad\quad$ break
12: $\quad$ **else**
13: $\quad\quad tmpReconfigured \leftarrow tmpFlow$
14: $\quad$ **end if**
15: **end for**
16: **return** $N_{reconfigured} \leftarrow tmpReconfigured$

Considering the example in picture 5.2, we have a similar situation as before but with different inputs. In this case the network has two firewalls configured in whitelisting mode and a forwarder. The new requirement that should be added is the Reachability from node 10.0.0.1 to node 20.0.0.1, with any possible source and destination ports and any protocol type. The requirement has two paths, each with one associated flow, namely A and B. In this case the algorithm should checks if there is at least one of them which does not block the traffic, and if this is not the case, it should reconfigure, for each flow, the nodes that are blocking the traffic. In this case both flows have a firewall that is blocking them, so the algorithm select the nodes blocking both of them since the solution would be to reconfigure either $FW1$ or $FW2$ in order to have at least one Atomic Flow which reaches the destination.

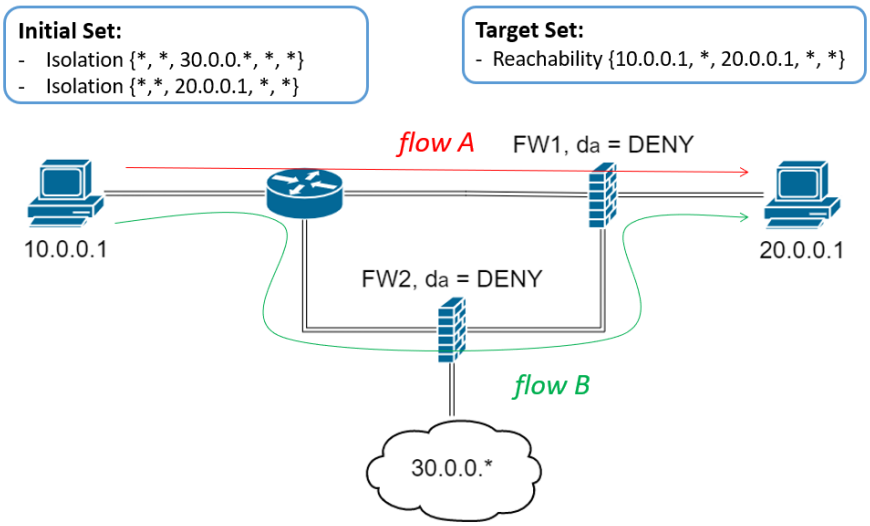

Figure 5.2: Example of addition of a Reachability requirement

Special Scenario with NATs

During the testing phase, using some synthetically generated networks, a critical scenario which was not supported by the original version of the algorithm was detected. This very specific error case would result in an unsat answer by the solver because the algorithm was selecting a set of reconfigured nodes which was not large enough for being compatible with a possible solution. This scenario required a patch to the algorithm in order for it to be supported. The particular situation could manifest if there are NATs in the network, in particular when working with an added reachability requirement which source node is effected by the shadowing operation of a nat. In this situation there could be cases of conflict between the aforementioned reachability requirement and an isolation requirement already enforced by the configuration and which is also subject to the shadowed operation by the same nat. This special conflict is caused by the action of the nat which converts the two initially different source traffics into a single unified traffic if the sources are both in the set of shadowed addresses. Note that the set of requirements causing this problem is still conflict-free as supposed in the hypothesis. This scenario is represented in minimal terms by the Allocation Graph in picture 5.3.

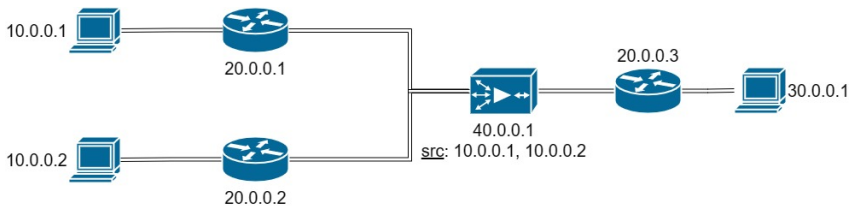

Figure 5.3: Example of Allocation Graph for special case

This situation is unsat using the previous algorithm because the the only node selected for reconfiguration would be the one placed after the nat. In this case the two opposite requirements can not be enforced in the same reconfigured node because they require an opposite action to be applied on the same Atomic Predicate,

since both requirements reach the node with the same input traffic. This situation is explained more clearly in the reconfiguration example of picture 5.4. In the example, the initial configuration of the network for satisfying the Initial Set or Network Security Requirement is composed of: three endpoints (10.0.0.1, 10.0.0.2, 30.0.0.1), a nat (40.0.0.1) having as sources two of the endpoints, two forwarders (20.0.0.1, 20.0.0.2) and a firewall (20.0.0.3) configured with whitelisting behavior. The Target set would requires the modification of one Isolation into a Reachability requirement, in particular the one from node 10.0.0.2 to node 30.0.0.1.

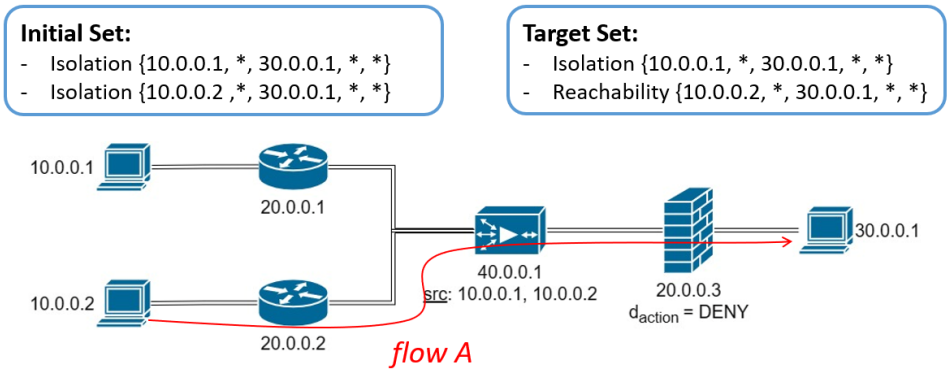

Figure 5.4: Example of special reconfiguration case

This requirement has only one associated flow which is represented in the picture, flow A. Considering the previous algorithm 3, the only selected node to be reconfigured would be the firewall with address 20.0.0.3, being the only one blocking the flow associated with the reachability requirement. However node 20.0.0.3 has no possible configuration which satisfy both of the requirements in the Target set, because they require two opposite actions over the same traffic in input, i.e., both the isolation and reachability requirements reach the node 20.0.0.3 with Atomic Predicate $\{40.0.0.1, *, 30.0.0.1, *, *\}$ because of the nat. The only correct configuration could be achieved by placing a firewall before the nat on the path of the isolation requirement.

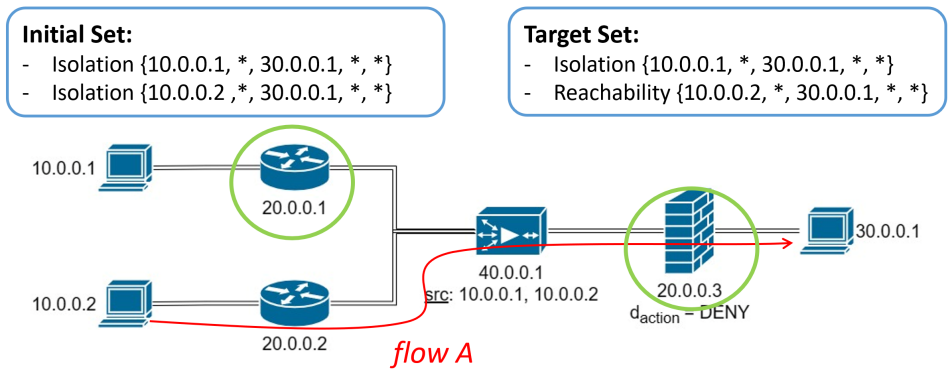

Figure 5.5: Example of special reconfiguration case with new algorithm

Therefore, the algorithm has been slightly modified for supporting this edge case. Note that only reachability requirements require this additional check, since

Algorithm 4 Modified algorithm for selecting network area to reconfigure for each added Reachability requirement

Input: a reachability requirement r, and an AG G_A
Output: nodes to be reconfigured $N_{reconfigure}$

```
1: tmpReconfigured ← ∅
2: for f ∈ F_r do
3:     tmpFlow ← ∅
4:     NAT ← False
5:     for n_i ∈ π(f) = [n_1, n_2, ..., n_d] do
6:         if !NAT & n_i.getFunctionalType().equals(NAT) then
7:             NAT ← True
8:         end if
9:         if allocated(n_i) & deny_{n_i}(τ(f, n_i)) then
10:            tmpFlow ← n_i
11:            if NAT then
12:                for f_p ∈ n_i.getCrossingFlows() do
13:                    if f_p.property == isolation & τ(f, n_i) == τ(f_p, n_i) then
14:                        for n_j ∈ π(f_p) do
15:                            if n_j.getFunctionalType().equals(NAT) then
16:                                tmpFlow ← n_{j-1}
17:                            else if n_j == n_i then
18:                                break
19:                            end if
20:                        end for
21:                    end if
22:                end for
23:            end if
24:        end if
25:    end for
26:    if tmpFlow.isEmpty() then      ▷ Found a flow satisfying the requirement
27:        tmpReconfigured ← ∅
28:        break
29:    else
30:        tmpReconfigured ← tmpFlow
31:    end if
32: end for
33: return N_reconfigured ← tmpReconfigured
```

they are the only ones which could cause this kind of conflicts. For each added requirement we must consider an extra flag which is set as soon as a nat is encountered in each traffic flow. If this flag is set and the algorithm finds a configured firewall blocking the input traffic, an additional check must be performed to account for the special conflicting scenario. This control consists in scanning all the traffic flows crossing the firewall and search for another traffic flow in this .set corresponding to an Isolation requirement and reaching the firewall with the same input traffic.

When such a condition is found, the algorithm starts from the source of the selected traffic flow for the found isolation requirement and marks as to be reconfigured all the nodes which are placed before each encountered nat until the firewall, crossed by the initial added reachability requirement, is reached. This is expressed in the modified algorithm 4. Also, picture 5.5 is showing which are the nodes that would be selected for reconfiguration with this updated version of the algorithm for the same example seen before in picture 5.4. Note that using this new version of the algorithm the reconfigured nodes are the firewall, as before, but also the forwarder in node 20.0.0.1.

5.5 Deleted Network Security Requirements

The previous reasoning was all about the *added* NSRs, the other set which has been considered is the set of *deleted* NSRs. This one includes all the requirements which are belonging to the Initial set but that are no more present in the Target set, in other word those requirements which are already configured in the initial network but whose satisfaction is no more required in the reconfigured network. The idea for this set of requirements is to find which are the parts of the configuration that are currently guaranteeing their satisfaction and possibly reconfigure them, allowing then the solver to choose to not allocate anymore a Network Security Function or remove a configured rule. As in the previous case, the algorithm will be presented separately for the two kinds of connectivity requirements.

Notably, one interesting aspect to consider is the cost of the pre-processing task of generating the set of Atomic Predicates to be used for the generation of Atomic Flows. These predicates are computed starting from the set of interesting predicates of the network, which includes the source and destination for each NSRs, and the transformation behavior of all the configured transformers, in our case firewalls and nats. The reconfiguration approach would have in general an higher cost for this phase. The first element which is responsible for the increase in the computation time is the presence of Network Security Functions already configured in the provided network, which models are considered in the reconfiguration scenario but not in a configuration starting from zero. The other interesting aspect is the set of *deleted* NSRs, which represents a more relevant contribution in the increase of the computation time. When we need to process the deleted requirements in the algorithm, we must generate the set of Atomic Flows representative for them and also the corresponding set of Atomic Predicates. The explicit delete operation requires to consider also the interesting predicates generated for these deleted requirements, instead of considering only the added or updated ones, causing an increased number of Atomic Predicates and Flows. For this reason, the algorithm has been designed in a parameterized manner for supporting different reconfiguration profiles:

- **NoDelete**. In this case the approach produce a faster but still formally correct reconfiguration by avoiding the explicit deletion of the requirements which are not needed but present in the initial configuration. In this way the user could achieve a better computation time trading it for a possible worst optimality, since the solution would not remove all the configurations

which are not needed if this operation is not necessary for satisfying the Target set of requirements. We could say that the approach in this profile is "don't care", the user is not interested in how the framework configures all the communications that are not covered by an explicit requirement. For example, if the user remove an isolation constraint from 10.0.0.1 to 20.0.0.1 and uses this profile, the framework implies that he does not necessarily wants to remove the parts of the configuration enforcing the isolation and thus having a reachability condition for the same traffic. If the user wants to remove a possible configuration that cause the traffic to be blocked, he should explicitly insert a new requirement from 10.0.0.1 to 20.0.0.1 expressing the opposite action.

- **Delete**. In this case the framework will apply the reconfiguration algorithm for all the sets of requirements, considering also the deleted ones. This will cause a degradation of the performance (caused by the pre-processing task and possibly by a larger reconfigured area resulting in a more complex model of the problem) but possibly reaching a better optimality of the final configuration. This profile could be further characterized using the profiles already present in VEREFOO (i.e., whitelisting, blacklisting, security-oriented specific, rule-oriented specific) for achieving other secondary goals, such as preferring a solution that could be more security cautious blocking all communications if it is not specified otherwise (i.e., the removal of a reachability requirement would imply the enforcing of an isolation) or a solution that is prioritize network reachability allowing all communications for which there is no specific requirement (i.e., the deletion of an isolation requirement would imply the enforcing of a reachability).

This thesis propose the idea and implementation of both profile but only the *NoDelete* one has been tested more extensively, being the most interesting one in terms of achieved performance compared to the complete reconfiguration. This is also motivate by the results presented in chapter 6 which compares the scalability of the pre-processing task for generating the Atomic Flows in the two different profiles.

5.5.1 Isolation Requirement

An isolation requirement is satisfied if there is an allocated firewall for each Atomic Flow which is blocking the associated Atomic Predicate in input. In this case the idea is to find all the nodes that are satisfying the condition of the requirement and reconfigure them. This implies considering all the correlated traffic flows of each deleted isolation requirement and selecting all the nodes that are blocking the traffic in input as to be reconfigured. This is what is expressed in algorithm 5. The starting point is always the associated set of Atomic Flows F_r. For each Atomic Flow $f \in F_r$ the algorithm search for all the configured nodes that are blocking the traffic in input for the considered flow f and adds them to the set of reconfigured nodes (lines 3-5).

We also see the application of the algorithm in the example represented by picture 5.6. In this case, the combination of Initial and Target set would let only

Algorithm 5 Algorithm for selecting network area to reconfigure for each deleted Isolation requirement

Input: an isolation requirement r, and an AG G_A
Output: nodes to be reconfigured $N_{reconfigure}$

1: **for** $f \in F_r$ **do**
2: **for** $n_i \in \pi(f) = [n_1, n_2, ..., n_d]$ **do**
3: **if** $allocated(n_i)$ & $deny_{n_i}(\tau(f, n_i))$ **then**
4: $N_{reconfigure} \leftarrow n_i$
5: **end if**
6: **end for**
7: **end for**
8: **return** $N_{reconfigure}$

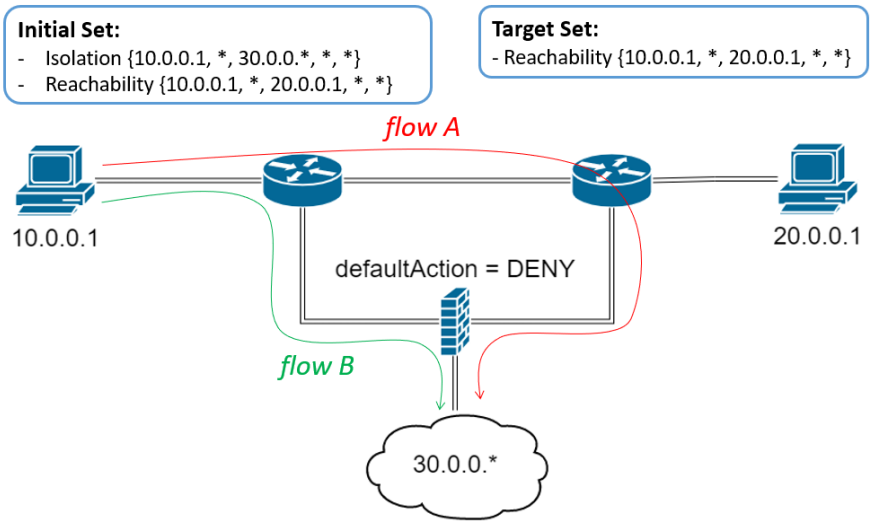

Figure 5.6: Example of deletion of an Isolation requirement

one requirement in the *deleted* set, which is the Isolation from node 10.0.0.1 to the subnetwork 30.0.0.0/24 (represented as 30.0.0.* using the wildcard symbol). The generated Atomic Flows are two, A and B, one for each possible path. For both flows there is only one node that could be possibly selected as to be reconfigured, which is the firewall configured in whitelisting, i.e., the default action is to deny all traffics. In this case the solver could then decide to not allocate the firewall in that position if there is no requirement in the Target set needing it, like in this situation.

5.5.2 Reachability Requirement

The other kind of connectivity requirement is reachability. A reachability requirement is satisfied if there is at least one associated traffic flow which is not blocked from source to destination. The algorithm, for each reachability requirement r in the set of *deleted* NSRs, searches among all the associated Atomic Flows the ones

(they could be more than one) which are not blocked by any node up to the destination. This is expressed in algorithm 6. The adopted approach is to check, for a given flow $f \in F_r$, if there is a node blocking the traffic received in input for f (line 4) and if so a boolean variable *found* is set to false. Whenever the variable remains true after scanning all the nodes this implies that an Atomic Flow satisfying the reachability condition has been found. When such a flow is encountered, all its nodes are added to the set of reconfigured ones $N_{reconfigure}$ (lines 9-11) since all of them could be potentially updated since the reachability requirement is no more present in the Target set.

Algorithm 6 Algorithm for selecting network area to reconfigure for each deleted Reachability requirement

Input: a requirement r, and an AG G_A
Output: nodes to be reconfigured $N_{reconfigure}$

```
1:  for f ∈ F_r do
2:      found ← True
3:      for n_i ∈ π(f) = [n_1, n_2, ..., n_d] do
4:          if allocated(n_i) & deny_{n_i}τ(f, n_i)) then
5:              found ← False
6:              break
7:          end if
8:      end for
9:      if found == True then          ▷ Found a flow which satisfy the condition
10:         N_reconfigure ← π(f)
11:     end if
12: end for
13: return N_reconfigure
```

The figure 5.7 represents an example for the case of deletion of a reachability requirement. In this case, the requirement which is no more needed in the Target set is the reachability from node 20.0.0.1 to node 10.0.0.1, which was configured in the network with a rule in the firewall to allows the corresponding traffic. In this case, the generated flows A and B are both uninterrupted from source to destination and consequently all nodes should be considered as to be reconfigured. The solver could then reconsider the configuration of the firewall and remove the rule which is not needed anymore.

5.6 Updated Network Security Requirements

This set is a subcategory of the *added* set. In fact it includes all those requirements that were already present in the Initial set but have been changed partially in the Target set. For this reason a requirement in this set could be seen as a combination of the two previous actions, the removal of the old configured requirement that belongs to the *deleted* set and the addition of the modified requirement belonging to the *added* set. This can be seen in picture 5.8 with an example. For this reason

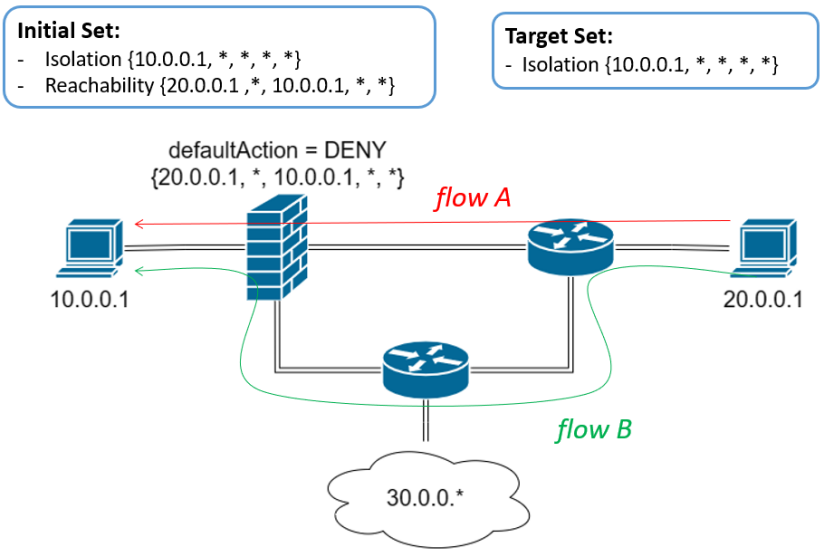

Figure 5.7: Example of deletion of a Reachability requirement

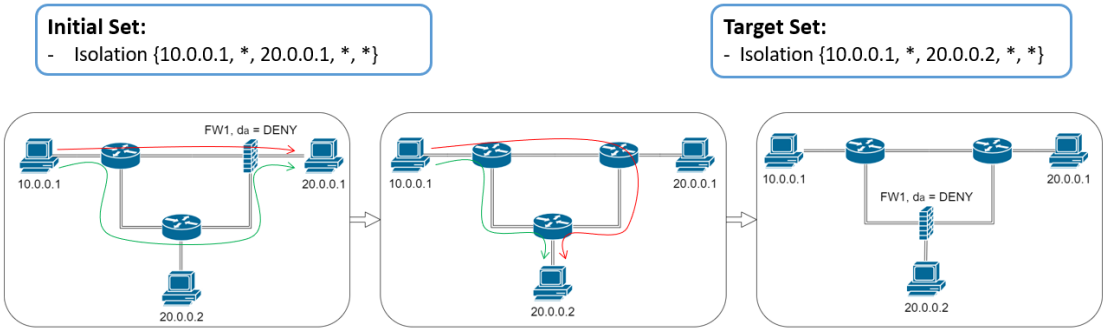

Figure 5.8: Example of an updated requirement

this is considered just as a theoretical set of requirements but in practice it is treated exactly as described for the other two sets.

5.7 Optimality of the Reconfiguration

It has been already discussed in chapter 3 that VEREFOO has one main goal, which is to generate a formally correct allocation and configuration of the necessary Network Security Functions provided a set of high-level security policies, but it has also some secondary goals which are related to the optimality of the computed configuration. In particular, the solution which is automatically computed by the framework should be the optimal one in terms of minimization of the allocated firewalls and minimization of the configured rules. The approach proposed in this thesis instead could produce a solution which is optimal only with respect to the network areas which are identified for reconfiguration, so as possibly modifiable by VEREFOO, and not an optimal solution in a global sense. Using the algorithm that has been presented in this chapter, only some locations are considered as possible candidates for the placement of new Network Security Functions, and

the tool could then analyze only a subset of all the possible solutions since it can not modify the other nodes which are not selected for reconfiguration. We could say that we are achieving a sub-optimal result using the reconfiguration approach compared to a complete reconfiguration starting from zero. Nevertheless, this aspect is compensated by the improved computation time which represent a more relevant parameter for the proposed scenario of a cybersecurity attack. This aspect will be further developed in the testing phase, trying to estimate the entity of this sub-optimality.

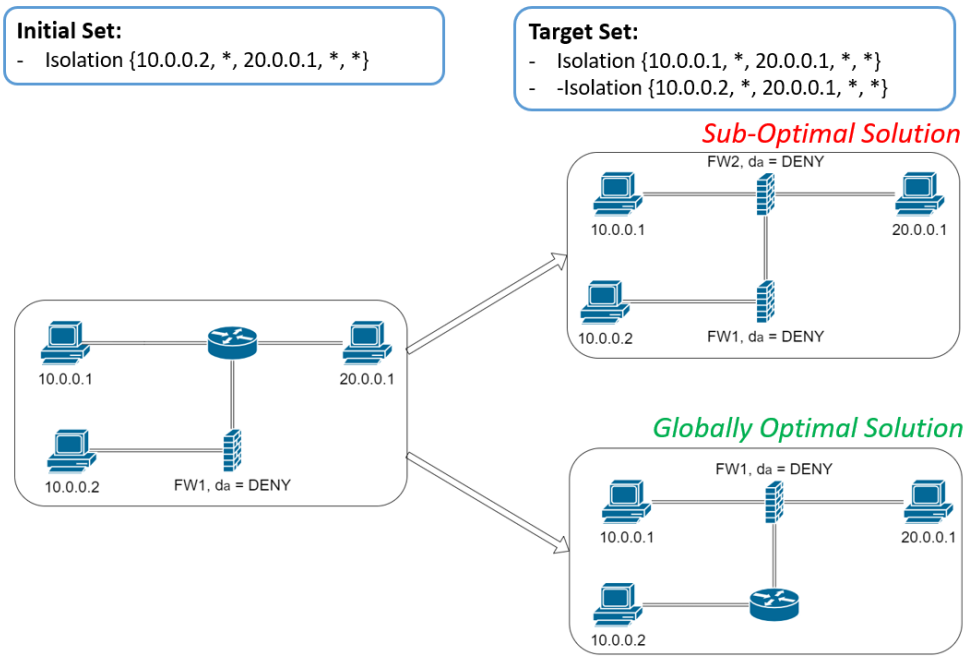

Figure 5.9: Example of the sub-optimality of the proposed reconfiguration approach

Considering the picture 5.9, we can see an example representative of the sub-optimality achieved with this approach compared to the globally optimal one. In the presented scenario the initial configuration should be updated to include the additional Isolation requirement from source node 10.0.0.1 to source node 20.0.0.1. For this situation the algorithm would select as node to be reconfigured only the forwarder, maintaining unchanged the allocation and configuration of the firewall $FW1$ since it is not crossed by any traffic flow correlated with the additional requirement. The solver would then reconfigure just the node of the forwarder to satisfy the requirement and so it will decides to allocate another firewall $FW2$ to block the traffic. However, the computed solution is sub-optimal in a global sense because the Target set of requirements could be satisfied with just only one firewall, which would be the solution achieved if the configuration is recomputed from zero. This globally optimal solution was discarded in the reconfiguration approach since $FW1$ was considered as immutable, thus it was not one of possible solutions.

5.8 Soft Constraints

With the proposed algorithm for detecting the network areas to be reconfigured the space of the solutions for the solver has already been reduced, since only some nodes are considered as mutable and which configuration has to be determined by the solver, while others are fixed and their configuration must be maintained unchanged. This approach by itself could be helpful in reducing the time for the resolution of the MaxSMT problem and achieving a faster computation time for the overall reconfiguration process, however this thesis contributed also to a redesign of the soft constraints used for the problem definition in order to make the solution even more effective. In particular, the idea which has been introduced in the problem is that the current state of the network, and its configuration, should be preferably maintained. The current formulation of the MaxSMT problem has been modified according to this idea, in particular the soft constraints have been changed so that the optimal solution would be still the one which minimize the number of firewalls and rules (respecting in this way the optimality goals of VEREFOO) but also the one which produces the smallest possible number of changes to the initial configuration. The better formulate the reasoning behind this solution, the idea is that an already allocated firewall should be preferred instead of a newly allocated one, and similarly an already configured rule should be preferred instead of a new one. In this way the resolution of the MaxSMT problem is simplified, since the solver will search first for a solution which reuses as much as possible the initial configuration. The solver is reaching the possible optimal solution in a shorter time with respect to considering all configuration soft constraints with an equal weight.

This approach has been adopted mainly for the aforementioned reason of improving the performance of the solver but it also has a cybersecurity related motivation. Indeed, another important problem in the area of automated network security is the preservation of the security policies during the transient from the initial configuration to the final one with the applied changes. By the distributed nature of the security functions, the process of modifying the configuration could present different insecure temporary states where the required protection is not guaranteed. Considering the context of this thesis, so the configuration of distributed firewalls systems, the security policies are enforced by multiple security functions and a change in the configuration requires the update of one or more elements in the network. The approach should aim to a protocol for the application of the updates which guarantees the satisfaction of the largest number of network security policies in each transient state as described in [29]. For this reason, the proposed change to the soft constraints also acts towards this principle, preferring a solution which maintains the initial configuration and consequently requires less temporary and transient insecure states.

The following soft constraint regulates the allocation of a firewall for each node in the set of all allocation places A, it is formulated as:

$$\forall a_i \in A. Soft(allocated(a_i) = false, c_k)$$

This instructs the solver to prefer a solution that does not allocate any firewall for each allocation place. Recall that in the MaxSMT problem the optimality of

the solution is reached when all the hard constraints are satisfied and the sum of the weights of all satisfied soft constraints reach its maximum value. In this case, the soft clause is assigning a weight to the non allocation of each firewall since the optimal solution will be the one allocating the minimum number of them. Instead, the soft constraint regulating the configuration of a possible firewall rule is formulated as:

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

For each firewall which is automatically configured by VEREFOO, the framework calculates the set of rules that could be possibly needed for the specific firewall considering the input Network Security Requirements and their associated traffic flows. This set is P_k and contains all the placeholder rules for node k, representing the rules which could be needed in the firewall configuration. The soft constraint states that each one of the placeholder rules should be not configured. Of course this clause has far less importance than the non-allocation of the firewall itself, for this reason the weight associated to the non-allocation of a firewall is much greater than the non-configuration of any of its rules:

$$\sum_{i:p_i \in P_k} (c_{ki}) < c_k$$

This combination of soft constraints has been modified in this work for all the firewalls which are selected as to be reconfigured. Indeed, once a firewall is marked as to be reconfigured, it will be processed in a different way by the solver and the weights associated to its soft constraints are modified. For each reconfigured firewall the soft constraints are used to tell the solver to prefer it with respect to a new one. The weight associated to the first soft constraints regulating the allocation should be c_{kR}, and it must be less than the weight associated to the allocation of an empty allocation place, so $c_{kR} < c_k$. Note that the weight is associated to the non-allocation, which means that a soft constraint for the non-allocation of an allocation point is preferred to the non-allocation of a reconfigured firewall. The same principle is applied for the configuration of each placeholder rule. If a reconfigured firewall has a rule p_i which was already configured in the initial configuration, this rule has a corresponding soft constraints with an associated weight c_{kiR} which is less than the weight associated to a non configured placeholder rule. As before these weights are assigned to the non-configuration, which means that maintaining a rule already present in the initial configuration should be preferred with respect to configuring a new one. If p_i is an already configured placeholder rule for a firewall in node n_k and p_j is a placeholder rule for an allocation point n_p, then the corresponding soft constraints would be $Soft(\neg configured(p_i), c_{kiR})$ and $Soft(\neg configured(p_j), c_{pj})$, with $c_{kiR} < c_{pj}$

5.9 Allocation Graph generator

Another contribution that has been done to VEREFOO for supporting the reconfiguration process is the new generation of the Allocation Graph starting from the Service Graph. As introduced in chapter 3, these two logical representations are

very similar since they share most of the nodes and connection, but the Allocation Graph has an additional kind of node, the Allocation Place. This new type is used to represent an empty spot in the network that could be used for allocating a new Network Security Function (i.e., firewall) if the solver decides it is the optimal place, or can be left unused and a forwarder will be placed in its place if it is crossed by an Atomic Flow associated to one of the requirements. In the original version the allocation places are inserted automatically on any link of the service graph, with the added options for the service designer to forbid the generation of on some links or to force the allocation of a Network Security Function on certain positions.

First we gives a more formal view of the process for the automatic generation of the Allocation Graph starting from the Service Graph. Recalling the models of both network representations, a Service Graph could be seen as $G_S = (N_S, L_S)$:

- N_S is the set of all network nodes for the service graph, could be further described as $N_S = E_S \cup S_S$. Meaning that is the union of the set of network endpoints (E_S) and the set of service functions (S_S).
- L_S is the set of links interconnecting pairs of elements in N_S, in particular $l_{ij} \in L_S,\ i \neq j$ implies that $n_i \in N_S$ is directly connected with $n_j \in N_S$.

Instead, the Allocation Graph is $G_A = (N_A, L_A)$ and its characterized by the two sets:

- N_A is the set of all nodes in the Allocation Graph, this set is composed as $N_A = E_A \cup S_A \cup P_A$. The sets of endpoints and service functions are the same as those in the Service graph, i.e. $E_A = E_S$ and $S_A = S_S$, but additionally there is the set of all the Allocation Places.
- L_A is the set of links interconnecting a pair of elements in N_A, this sets will have more connections than L_S since there are additional nodes.

For every $l_{ij} \in L_S,\ i \neq j$, the service designer could perform two different actions:

- $forbidden(l_{ij})$ forbids the creation of an Allocation Place on the correspondent link
- $forced(l_{ij})$ forces the allocation of a firewall in the correspondent link, a new allocation place will be generated and it will be obligatory to allocate a firewall in its place

According to these two possible choices, the rules adopted for generating the new allocation places are the following:

$$\begin{aligned} &\forall l_{ij} \in L_S. \neg forbidden(l_{ij}) \wedge \neg forced(l_{ij}) \\ &\quad \implies p_{ij} \in P_A \wedge l_{ip_{ij}} \in L_A \wedge l_{p_{ij}j} \in L_A \end{aligned} \tag{5.1}$$

$$\forall l_{ij} \in L_S. forbidden(l_{ij}) \implies l_{ij} \in L_A \tag{5.2}$$

$$\forall l_{ij} \in L_S.forced(l_{ij}) \implies p_{ij} \in P_A \wedge l_{ip_{ij}} \in L_A \\ \wedge l_{p_{ij}j} \in L_A \wedge allocated(p_{ij}) = true \tag{5.3}$$

The three formulas 5.1, 5.2 and 5.3 are mutually exclusive, only one of them can be used according to the choice made by the service designer. The typical scenario, with no constraints imposed by the user, is the one of 5.1. Considering this formula, for each link $l_{ij} \in L_S$, $i \neq j$, a new Allocation Place is created adding it to the set P_A and also the links are created, connecting the new node with the nodes n_i and n_j.

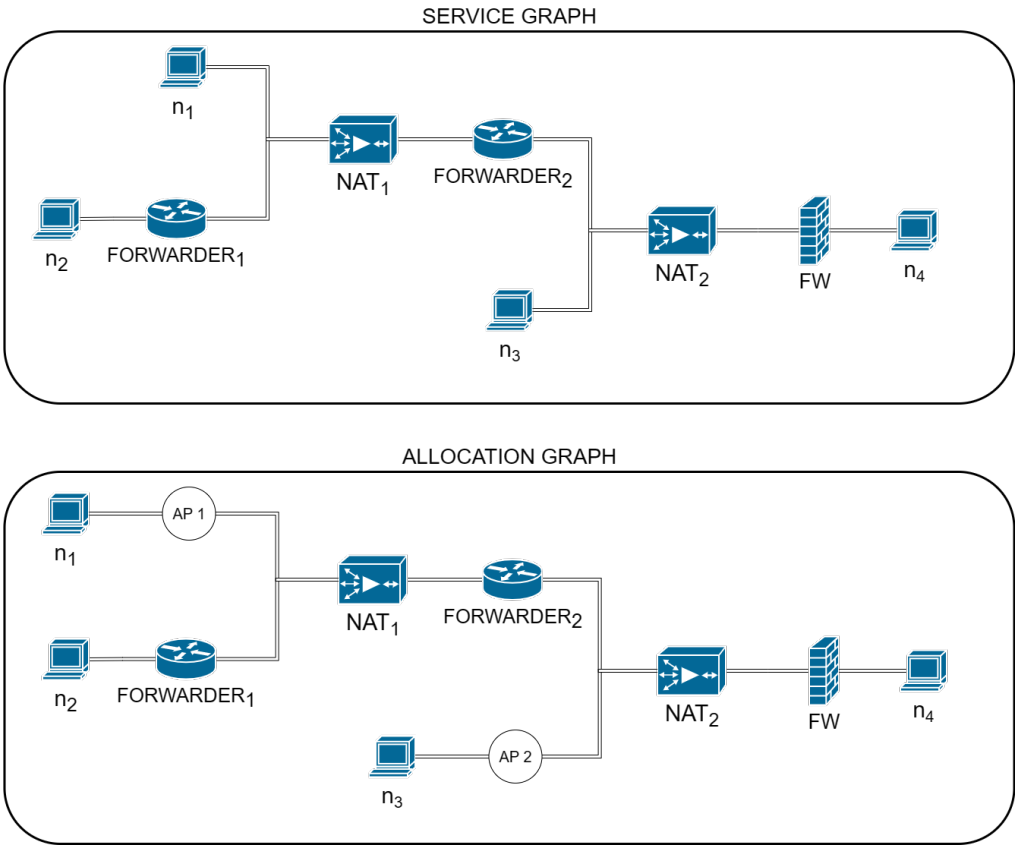

Figure 5.10: Example of the new generator of an Allocation Graph starting from a Service Graph

This procedure has been modified in the approach proposed by this thesis. In the situation of a reconfiguration, the provided service graph already contains several forwarders and firewalls which were produced during a previous configuration through the configuration of some inserted allocation places. The idea is then to reuse them for implementing the updated set of requirements instead of inserting new allocation places. Since the approach is to reconfigure the network and possibly reuse the already present Network Security Functions, it would not be correct to allocate others allocation places near them. On the other hand, the allocation places which are not crossed by any requirement are possibly removed and so it is necessary to re-allocate them where it is needed. For this reason the generator responsible of producing the Allocation Graph has been modified so that it avoids placing a new allocation place near a firewall or a forwarder, which are the two outcome of an allocation place which is crossed by an Atomic Flow. Note that in this

work we consider the forwarder as the function placed on an allocation place which has not been selected as the position of a firewall. This is not a limitation because the user could use others service function with the same behavior for representing an explicit node which works as a forwarder but could not be reconfigured as if it was an allocation place (i.e., router, hub, etc.).

Picture 5.10 shows this approach applied to a possible network. In this situation only two new allocation places are inserted in the graph, because only these two positions are coherent with the proposed new allocation rule. Representing with FW_S and F_S, respectively the sets of firewalls and forwarders configured in the Service Graph, the only formula which is modified 5.1 that is transformed into 5.4

$$\begin{aligned} &\forall l_{ij} \in L_S . \neg forbidden(l_{ij}) \wedge \neg forced(l_{ij}). \\ &n_i \notin FW_S \wedge n_j \notin FW_S \wedge n_i \notin F_S \wedge n_j \notin F_S \\ &\implies p_{ij} \in P_A \wedge l_{ip_{ij}} \in L_A \wedge l_{p_{ij}j} \in L_A \end{aligned} \tag{5.4}$$

5.10 Clarification example about reconfiguration of firewalls

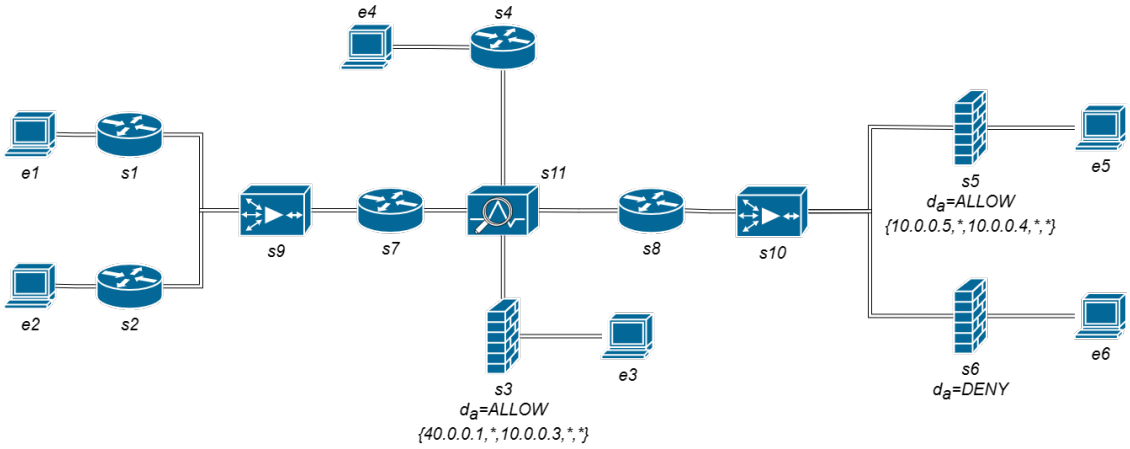

Figure 5.11: Input Allocation Graph with configured Initial Set of NSRs

The complete approach can be clarified by an example which is now proposed. This is also useful to highlight the advantages of this methodology compared to the reconfiguration from zero adopted by the previous versions of VEREFOO. Let us consider the scenario of an already configured network with a satisfied set of security policies which must be updated. The inputs given to the framework are the allocation graph in picture 5.11 and the two sets of requirements, Initial and Target. The inputs are given to VEREFOO using the XML representation, the one for this example is in listing 5.1. The Initial Set includes all the requirements which are configured in the provided network and these are listed in table 5.3. The service designer should provide also the Target Set, which includes the requirements that must be satisfied in the reconfigured network and these are listed in table 5.4. Note that the IP addresses and the function type of each node in the graph are showed in table 5.2. Moreover, the configuration of the firewalls already allocated

is shown in table 5.1, other than being included in the picture. This shows that the Initial set of Network Security Requirements requires three allocated firewalls to be satisfied, one on node $s3$ with blacklisting policy (i.e., the default action is allow) and one configured rule for blocking the predicate $\{40.0.0.1, *, 10.0.0.3, *, *\}$, another on node $s5$ with same blacklisting policy and a rule to block the predicate $\{10.0.0.5, *, 10.0.0.4, *, *\}$, and finally a last firewall on node $s6$ with whitelisting policy (i.e., default action is deny) and no configured rules.

Firewall *s3*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Deny	40.0.0.1	*	10.0.0.3	*	*
2	Allow	*	*	*	*	*

Firewall *s5*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Deny	10.0.0.5	*	10.0.0.4	*	*
2	Allow	*	*	*	*	*

Firewall *s6*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Deny	*	*	*	*	*

Table 5.1: Filtering Policy rules for the allocated firewalls

Identifier	IP address	Function type
e1	10.0.0.1	Endpoint
e2	10.0.0.2	Endpoint
e3	10.0.0.3	Endpoint
e4	10.0.0.4	Endpoint
e5	10.0.0.5	Endpoint
e6	10.0.0.6	Endpoint
s1	20.0.0.1	Forwarder
s2	20.0.0.2	Forwarder
s3	20.0.0.3	Firewall
s4	20.0.0.4	Forwarder
s5	20.0.0.5	Firewall
s6	20.0.0.6	Firewall
s7	20.0.0.7	Forwarder
s8	20.0.0.8	Forwarder
s9	40.0.0.1	NAT
s10	40.0.0.2	NAT
s11	50.0.0.1	Traffic Monitor

Table 5.2: IP addresses and function types

The first necessary step for the proposed approach is a pre-processing task for computing the sets of *added*, *deleted* and *kept* Network Security Requirements from the given input sets, Initial and Target. For the following parts, the requirements are referred with the integer identifier associated to each one of them in the tables 5.3 and 5.4.

ID	Type	IPSrc	pSrc	IPDst	pDst	tProto
1	Isol	10.0.0.1	*	10.0.0.3	*	*
2	Isol	10.0.0.2	*	10.0.0.3	*	*
3	Isol	10.0.0.6	*	10.0.0.4	*	*
4	Isol	10.0.0.5	*	10.0.0.4	*	*
5	Isol	10.0.0.6	*	10.0.0.3	*	*
6	Reach	10.0.0.5	*	10.0.0.3	*	*
7	Reach	10.0.0.1	*	10.0.0.4	*	*
8	Reach	10.0.0.3	*	10.0.0.2	*	*
9	Reach	10.0.0.5	*	10.0.0.1	*	*
10	Reach	10.0.0.3	*	10.0.0.4	*	*

Table 5.3: Initial Set of Network Security Requirements

ID	Type	IPSrc	pSrc	IPDst	pDst	tProto
1	Isol	10.0.0.1	*	10.0.0.3	*	*
2	Isol	10.0.0.2	*	10.0.0.3	*	*
3	Isol	10.0.0.6	*	10.0.0.4	*	*
4	Isol	10.0.0.5	*	10.0.0.4	*	*
5	Isol	10.0.0.6	*	10.0.0.3	*	*
11	Isol	10.0.0.5	*	10.0.0.3	*	*
12	Isol	10.0.0.4	*	10.0.0.3	*	*
7	Reach	10.0.0.1	*	10.0.0.4	*	*
8	Reach	10.0.0.3	*	10.0.0.2	*	*
13	Reach	10.0.0.5	*	10.0.0.2	*	*
10	Reach	10.0.0.3	*	10.0.0.4	*	*

Table 5.4: Target Set of Network Security Requirements

According to these identifiers, the sets provided in input to VEREFOO are:

$$Initial\ Set = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$Target\ Set = \{1, 2, 3, 4, 5, 11, 12, 7, 8, 13, 10\}$$

Using the algorithm 1 for computing the intersection of the previous two sets, the identified groups are:

$$added = T \setminus I = \{11, 12, 13\}$$

$$deleted = I \setminus T = \{6, 9\}$$

$$kept = I \cap T = \{1, 2, 3, 4, 5, 7, 8, 10\}$$

The proposed example follows the "*NoDelete*" profile, therefore only the added and kept requirements are considered in the reconfiguration process. Nevertheless also a brief reasoning for the "*Delete*" version of the solution will be detailed to highlight the differences and the additional costs of this profile compared to the first one. These two differs the most in the set of requirement considered in the algorithm for computing the network areas to be reconfigured. Including the explicit

reconfiguration of the deleted requirements it produces an higher computation time as it was explained before. This increase has two main contributions, the first and most obvious is the expansion of the network area which is reconfigured. Having an increased set of requirements the algorithm processes more Atomic Flows and ultimately marks more nodes to be reconfigured. The second contribution to the higher computation time is instead associated to the pre-processing step to compute the set of Atomic Predicates, which is mandatory if using Atomic Flows. As described in chapter 2, to compute the set of Atomic Predicates we must start from the set of "interesting" predicates, which are the source and destination predicates for each Network Security Requirement and the transformation function and forwarding behavior of each transformer node. Consequently, the number of interesting predicates depend on the set of requirements and considering also the deleted set would increase this value. Note that the reconfiguration approach has in general an higher number of interesting predicates compared to the complete configuration from zero, because other than the source and destination predicates for the requirements, which are in the same number for both approaches, in the reconfiguration we consider a partially configured graph with some allocated Network Security Functions that contributes for some additional interesting predicates. Beware that only after having generated this set of predicates we can compute the set of Atomic Predicates. Notably, the "NoDelete" profile applied to this example generates 54 Atomic Predicates, whereas in the "Delete" version there are 63 Atomic Predicates. After this necessary pre-computation, the next step is the computation of the Atomic Flows associated with the added and kept NSRs (i.e., the Target set).

The core process is the identification of the nodes which must be reconfigured to achieve the satisfaction of the added requirements. In this case the additional requirements are **11**, **12** and **13**. The first two are of type isolation whereas the last one is of type reachability. The algorithm for the added isolation requirements 2 consist in scanning all the correlated flows and ensure if all of them are blocked by at least one of the encountered nodes. If an Atomic Flow which is not satisfying this property is found, all the nodes in its path are selected for reconfiguration, in particular the algorithm select only firewalls or forwarders being the two possible outcome of an allocation place after a configuration, thus representing a possible location which could be configured by the framework. The requirement **11** has one associated Atomic Flow which is $[e5, 0, s5, 0, s10, 7, s8, 7, s11, 7, s3, 7, e3]$, with the included Atomic Predicates $0 = \{10.0.0.5, *, 10.0.0.3, *, *\}$ and $7 = \{40.0.0.2, *, 10.0.0.3, *, *\}$. Since this is not blocked by any of the nodes, for this requirement the nodes to be reconfigured are $\{s5, s8, s3\}$. The second isolation requirement, **12**, has too only one associated Atomic Flow which is $[e4, 1, s4, 1, s11, 1, s3, 1, e3]$, with the single predicate $1 = \{10.0.0.4, *, 10.0.0.3, *, *\}$. Also in this case there is no node blocking the traffic, consequently also in this case all nodes are reconfigured, specifically $\{s4, s3\}$. Finally, the requirement **13** is of type reachability, in this case the algorithm 4 scans all the correlated flows searching if at least one of them is not interrupted from source to destination, and if such flow is not found all the nodes that are blocking the flows are selected for reconfiguration. The added requirement has two associated flow which are:

$$af_1 = [e5, 9, s5, 9, s10, 24, s8, 24, s11, 24, s7, 24, s9, 24, s2, 24, e2]$$

$$af_2 = [e5, 11, s5, 11, s10, 46, s8, 46, s11, 46, s7, 46, s9, 24, s2, 24, e2]$$

and the included predicates are $9 = \{10.0.0.5, *, 10.0.0.2, *, *\}$, $11 = \{10.0.0.5, *, 40.0.0.1, *, *\}$, $24 = \{40.0.0.2, *, 10.0.0.2, *, *\}$, and $46 = \{40.0.0.2, *, 40.0.0.1, *, *\}$. The only allocated Network Security Function for both flows is the firewall $s5$, but it does not block any of the received traffics for the associated flows. Consequently this requirement is already satisfied by the current configuration and does not produce any additional nodes to be reconfigured. In total the identified nodes are four $\{s3, s4, s5, s8\}$ in this case, if instead we would have used the "NoDelete" profile the set of reconfigured nodes would include 6 of them $\{s1, s3, s4, s5, s7, s8\}$, the additional one being necessary for the deletion of requirement **9**. This clearly shows the higher computation time that could be cause by choosing this profile.

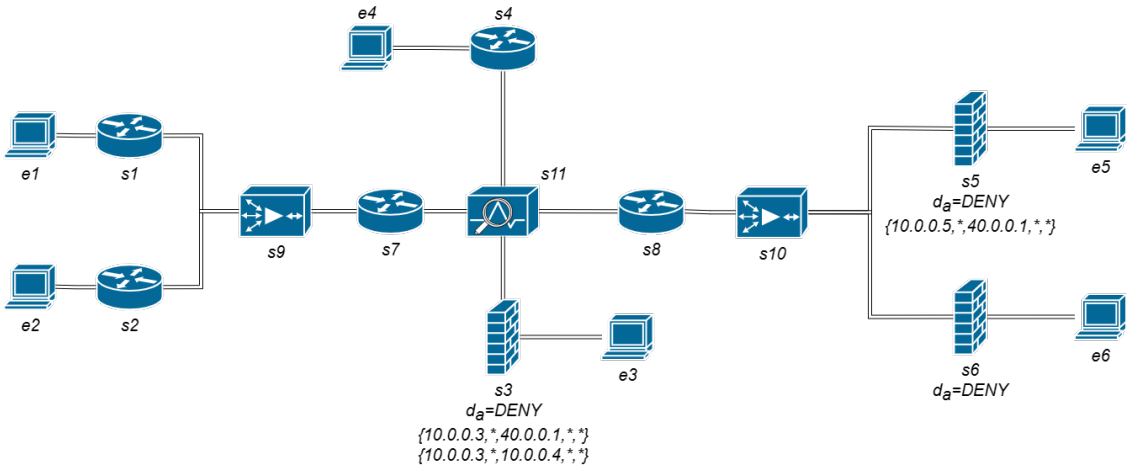

Figure 5.12: Resulting Reconfigured Network

Firewall *s3*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Allow	10.0.0.3	*	40.0.0.1	*	*
2	Allow	10.0.0.3	*	10.0.0.4	*	*
3	Deny	*	*	*	*	*
Firewall *s5*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Allow	10.0.0.5	*	40.0.0.1	*	*
2	Deny	*	*	*	*	*
Firewall *s6*						
#	Action	IPSrc	pSrc	IPDst	pDst	tProto
1	Deny	*	*	*	*	*

Table 5.5: Filtering Policy rules for the Reconfigured firewalls

The successive step is the actual reconfiguration of these nodes. The firewalls are modified such that their allocation and configuration would be completely recomputed by the solver (i.e., flags *autoallocated* and *autoconfigured* are set to true), instead the forwarders are converted into allocation places (i.e., the functional type is set to null). Finally, the model for the MaxSMT problem is generated as described briefly in chapter 3, with the only change being in the definition of the

soft constraints. In particular the reconfigured nodes have a set of soft constraints with a different associated weights, so that the allocation and configuration of these nodes are possibly preferred with respect to new ones. The outputs of VEREFOO are the allocation scheme of the distributed firewall instances in the service graph 5.12 and the filtering policy for each allocated firewall 5.5.

Listing 5.1: XML representation of VEREFOO input: AG and NSRs

```
<NFV>
    <graphs>
        <graph id="0" deleteOldProperties="false" serviceGraph="false">
            <node name="10.0.0.1" functional_type="WEBCLIENT">
                <neighbour name="20.0.0.1"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webclient nameWebServer="10.0.0.3"/>
                </configuration>
            </node>
            <node name="10.0.0.2" functional_type="WEBCLIENT">
                <neighbour name="20.0.0.2"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webclient nameWebServer="10.0.0.3"/>
                </configuration>
            </node>
            <node name="10.0.0.3" functional_type="WEBSERVER">
                <neighbour name="20.0.0.3"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webserver>
                        <name>10.0.0.3</name>
                    </webserver>
                </configuration>
            </node>
            <node name="10.0.0.4" functional_type="WEBCLIENT">
                <neighbour name="20.0.0.4"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webclient nameWebServer="10.0.0.3"/>
                </configuration>
            </node>
            <node name="10.0.0.5" functional_type="WEBCLIENT">
                <neighbour name="20.0.0.5"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webclient nameWebServer="10.0.0.3"/>
                </configuration>
            </node>
            <node name="10.0.0.6" functional_type="WEBCLIENT">
                <neighbour name="20.0.0.6"/>
                <configuration name="confA" description="A␣simple␣description">
                    <webclient nameWebServer="10.0.0.3"/>
                </configuration>
            </node>
            <node name="40.0.0.1" functional_type="NAT">
                <neighbour name="20.0.0.1"/>
                <neighbour name="20.0.0.2"/>
                <neighbour name="20.0.0.7"/>
                <configuration name="confAutoGen">
                    <nat>
                        <source>10.0.0.1</source>
                        <source>10.0.0.2</source>
                    </nat>
```

```
        </configuration>
    </node>
    <node name="40.0.0.2" functional_type="NAT">
        <neighbour name="20.0.0.5"/>
        <neighbour name="20.0.0.6"/>
        <neighbour name="20.0.0.8"/>
        <configuration name="confAutoGen">
            <nat>
                <source>10.0.0.5</source>
                <source>10.0.0.6</source>
            </nat>
        </configuration>
    </node>
    <node name="50.0.0.1" functional_type="TRAFFIC_MONITOR">
        <neighbour name="20.0.0.3"/>
        <neighbour name="20.0.0.4"/>
        <neighbour name="20.0.0.7"/>
        <neighbour name="20.0.0.8"/>
        <configuration name="50.0.0.1">
            <forwarder>
                <name>Traffic Monitor</name>
            </forwarder>
        </configuration>
    </node>
    <node name="20.0.0.1" functional_type="FORWARDER">
        <neighbour name="40.0.0.1"/>
        <neighbour name="10.0.0.1"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="20.0.0.2" functional_type="FORWARDER">
        <neighbour name="40.0.0.1"/>
        <neighbour name="10.0.0.2"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="20.0.0.3" functional_type="FIREWALL">
        <neighbour name="50.0.0.1"/>
        <neighbour name="10.0.0.3"/>
        <configuration name="AutoConf">
            <firewall defaultAction="ALLOW">
                <elements>
                    <action>DENY</action>
                    <source>40.0.0.1</source>
                    <destination>10.0.0.3</destination>
                    <protocol>ANY</protocol>
                    <src_port>*</src_port>
                    <dst_port>*</dst_port>
                </elements>
            </firewall>
        </configuration>
```

```
            </node>
            <node name="20.0.0.4" functional_type="FORWARDER">
                <neighbour name="50.0.0.1"/>
                <neighbour name="10.0.0.4"/>
                <configuration name="ForwardConf">
                    <forwarder>
                        <name>Forwarder</name>
                    </forwarder>
                </configuration>
            </node>
            <node name="20.0.0.5" functional_type="FIREWALL">
                <neighbour name="40.0.0.2"/>
                <neighbour name="10.0.0.5"/>
                <configuration name="AutoConf">
                    <firewall defaultAction="ALLOW">
                        <elements>
                            <action>DENY</action>
                            <source>10.0.0.5</source>
                            <destination>10.0.0.4</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
            <node name="20.0.0.6" functional_type="FIREWALL">
                <neighbour name="40.0.0.2"/>
                <neighbour name="10.0.0.6"/>
                <configuration name="AutoConf">
                    <firewall defaultAction="DENY"/>
                </configuration>
            </node>
            <node name="20.0.0.7" functional_type="FORWARDER">
                <neighbour name="50.0.0.1"/>
                <neighbour name="40.0.0.1"/>
                <configuration name="ForwardConf">
                    <forwarder>
                        <name>Forwarder</name>
                    </forwarder>
                </configuration>
            </node>
            <node name="20.0.0.8" functional_type="FORWARDER">
                <neighbour name="50.0.0.1"/>
                <neighbour name="40.0.0.2"/>
                <configuration name="ForwardConf">
                    <forwarder>
                        <name>Forwarder</name>
                    </forwarder>
                </configuration>
            </node>
        </graph>
    </graphs>
    <Constraints>
        <NodeConstraints />
        <LinkConstraints />
    </Constraints>
```

```
    <PropertyDefinition>
        <Property graph="0" name="IsolationProperty" src="10.0.0.1"
    dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.2"
            dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.6"
            dst="10.0.0.4" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.5"
            dst="10.0.0.4" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.6"
            dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.5"
            dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.4"
            dst="10.0.0.3" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.1"
            dst="10.0.0.4" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.3"
            dst="10.0.0.2" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.3"
            dst="10.0.0.4" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.5"
            dst="10.0.0.2" />
    </PropertyDefinition>
    <InitialProperty>
        <Property graph="0" name="IsolationProperty" src="10.0.0.1"
    dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.2"
            dst="10.0.0.3" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.6"
            dst="10.0.0.4" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.5"
            dst="10.0.0.4" />
        <Property graph="0" name="IsolationProperty" src="10.0.0.6"
            dst="10.0.0.3" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.5"
            dst="10.0.0.3" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.1"
            dst="10.0.0.4" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.3"
            dst="10.0.0.2" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.5"
            dst="10.0.0.1" />
        <Property graph="0" name="ReachabilityProperty" src="10.0.0.3"
            dst="10.0.0.4" />
    </InitialProperty>
</NFV>
```

Chapter 6

Implementation and Validation

This chapter presents the tests which have been conducted on the implementation of the proposed approach, to show which goals have been achieved and to understand the limitations which should be overcome in the future. As described in chapter 2, the work done by VEREFOO can be divided into two phases, the first one consisting in the computation of all the traffic flows related to the input requirements, and the second one consisting in the resolution of the formulated MaxSMT problem in order to find the optimal allocation and configuration of the security functions which are needed to satisfy the requirements. Having described in chapter 5 the two possible profiles for the proposed reconfiguration approach, "*Delete*" and "*NoDelete*", a first test which has been done is an analysis of the impact of these possible profiles on the first phase of VEREFOO, the traffic flows computation. This allowed to quantify with empirical results the advantage that a less-complete but equally correct approach, the "NoDelete" one, has compared to the first profile which considers also the removal of the old configuration components. Moreover, this test phase is using larger networks and this allowed to put to stress the proposed algorithm, evaluating which is the impact of this additional step over the total computation time. Then a second and more complete testing phase has been conducted on the complete resolution of the Refinement problem, so the whole process executed by VEREFOO. The main goal is to compare the proposed approach with the previous implementation of the framework, which does not include a specific resolution for a reconfiguration scenario but would perform a reconfiguration starting from zero. This validation process has been useful for determining the scalability of the new implementation with respect to the previous one, comparing the total computation time but also the optimality of the obtained configuration. Indeed, another interesting aspect which has been evaluated is the impact of the reconfiguration approach on the optimization goals of VEREFOO, the minimization of allocated firewalls and the minimization of the configured rules for each firewall. Finally, the implementation has been briefly tested with even larger networks, with an higher number of endpoints and requirements, to see its scalability potential.

6.1 Design of the synthetic network generators

All the tests were executed using some synthetically generated networks, but the adopted generators are different on the base of the test scenario. In particular, the first phase is concerning the pre-processing task for computing the set of traffic flows and the comparison of the two different reconfiguration profiles. In this case the generated networks are more complex and large, including a multitude of nats and firewalls which are pre-configured with some random filtering policies. In this way the number of generated traffic flows and Atomic Predicates become significant and this would allow to better compare the two different profiles and the performance of the implemented algorithm. For further stressing the algorithm used to compute the network area to reconfigure, an additional parameter is adopted in this generator to allow the possibility to create a synthetic network capable of generating an higher number of flows which, along with the different middleboxes, must be considered while processing the requirements in the algorithm. Instead, the second part of the validation process has been conducted over the complete execution of VEREFOO, so including the resolution of the MaxSMT problem after the generation of the traffic flows for the input Network Security Requirements. This second phase required a new generator for the synthetic networks and sets of requirements, as well as a redesign of the test execution. In particular the comparison must be done on the reconfiguration, which implies that the input should be a network that has been previously configured by the framework. Furthermore, for highlighting the improvement over the previous implementation, it has been important to modify the test so that it would be possible to compare the reconfiguration approach with a reconfiguration starting from zero over the same network and input sets of requirements. For this reasons the second generator does not allocate nor configure any firewall, since these will be allocated and configured automatically by a preliminary run of VEREFOO only in the optimal locations. Notably, there is also a third generator which has been adopted for the final test on the scalability of the algorithm usibng an increasing number of requirements and endpoints. In this case the generated networks do not include firewalls or nats because the resolution of the refinement problem gets more complex with their inclusion and the scalability of the framework is limited for these larger networks.

6.2 Test parameters

The parameters that can be configured in the tests are slightly different between the several generators which have been used, however there are some common ones which are recurrent, such as the number of Network Security Requirements (**REQ**) which have to be enforced on the network, the number of clients (**WC**) and the number of servers (**WS**) which represent the set of endpoints in the network. Then, there are other parameters which are relevant to the middleboxes that can be included in the networks, such as the number of nats (**NATs**), the number of firewalls (**FWs**), the number of sources present in each nat (**NATSrcs**) and the number of rules configured in each firewall (**FWRules**). An additional parameter important for the reconfiguration is the percentage of Network Security Requirements which

are belonging to the *kept* group as described in chapter 4. This is representing the extend of overlapping between the Target set of requirements and the Initial set, it is defined using the parameter **percReqKept**. In the reconfiguration problem, one aspect which has a big impact on the computation time is the number of *added* and *deleted* requirements because these must be processed and enforced in the network, whereas the *kept* ones are possibly already satisfied by the initial configuration. For this reason an increase in the percentage of requirements kept cause a decrease in the number of added and deleted requirements and a lower computation time on average. The first test includes also some additional specific parameters. The first and most important one being an additional variable for the specification of the profile to be used for the parameterized version of the implementation, this variable can take only values "**Delete/NoDelete**". Then, the second parameter allows to modify the way in which the rules for each firewall are generated. This is possible through a second flag (**FWRulesRandom/FWRulesFromReq**) that specify whether the rules that are configured for the firewalls are generated using the sources and destinations of the requirements or randomly from the set of endpoints in the network. In the first case there are only rules on firewalls which affect nodes on which there is at least one requirement, whereas in the second case this is not necessary. This second flag would vary in a significant way the number of "interesting" predicates and consequently the number of generated Atomic Predicates. Because any configured rules for which source or destination are not taken from the set of requirements would constitute a new "interesting" predicate of the network. Conversely, if the source or destination are belonging to the set of requirements, these rules would not create any additional "interesting" predicate.

6.3 Test on Atomic Predicate computation

This section will describe the results obtained with the first phase of the validation process, the one regarding the comparison of the Atomic Predicate and traffic flow computation time in the two different profiles of the parameterized algorithm, the one considering the deleted requirements and the one ignoring them. The generated networks are belonging to five different classes, each with a progressive increase of all parameters at the same time. Note that the magnitude of the values for the parameters is very large compared to the other tests that will be presented afterwards because in this phase only the pre-processing phase is executed, not the complete process with the resolution of the MaxSMT problem that is not feasible for these complex networks. In particular the considered test cases are the following ones

- **Case A**: 100 REQ, 200 WS, 200 WC, 50 NAT, 50 FW, 20 NATSrc, 20 FWRules
- **Case B**: 150 REQ, 300 WS, 300 WC, 75 NAT, 75 FW, 30 NATSrc, 30 FWRules
- **Case C**: 200 REQ, 400 WS, 400 WC, 100 NAT, 100 FW, 40 NATSrc, 40 FWRules

- **Case D**: 250 REQ, 500 WS, 500 WC, 125 NAT, 125 FW, 50 NATSrc, 50 FWRules
- **Case E**: 300 REQ, 600 WS, 600 WC, 150 NAT, 150 FW, 60 NATSrc, 60 FWRules

The conducted test considered a complete combination for the other parameters. The **PercReqKept**, which is the parameter characteristic of the reconfiguration process, was used with the values 10%, 50% and 90% to cover a wide range of scenarios. Instead, the binary parameters for the selection of the profile and the generation of the rules for each firewall have also been used and all their combinations have been tested. In total six different situations have been analyzed, using the three different percentages of kept requirements for each one of the binary parameter for firewall rules generation and for each of these situations the "Delete" and "NoDelete" profiles are compared.

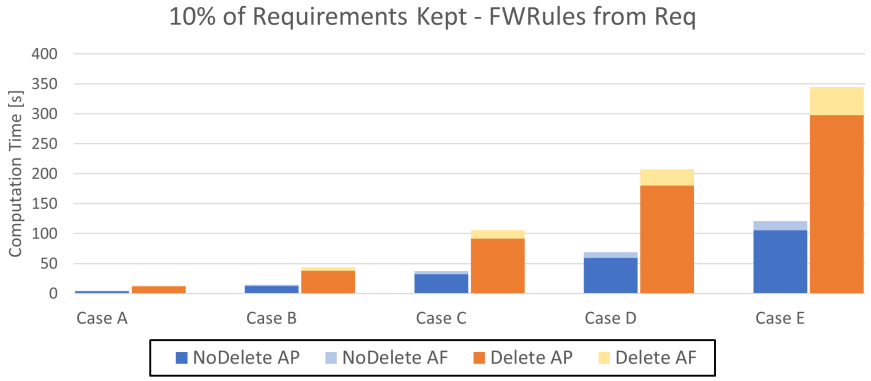

Figure 6.1: AFs computation with 10% PercReqKept and FWRulesFromReq

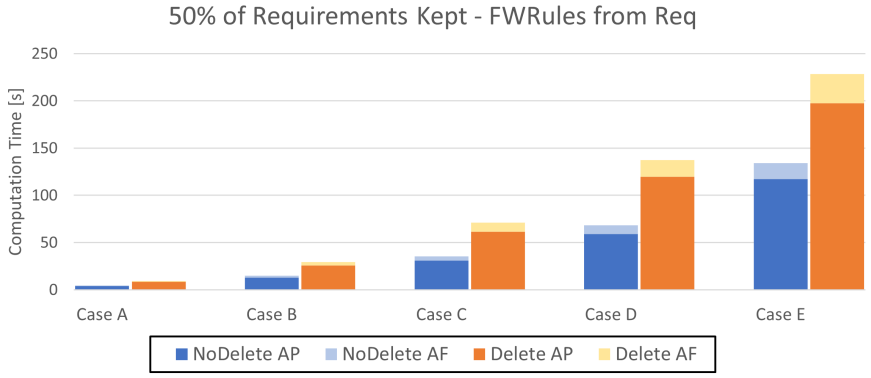

Figure 6.2: AFs computation with 50% PercReqKept and FWRulesFromReq

We start seeing the three graphs produced with the **FWRulesFromReq** parameter. These are distinguished on the base of the **PercReqKept**, in particular figure 6.1 is for the value 10%, figure 6.2 is for the value 50%, and finally 6.3 is for the value 90%. Note that these graphs are grouping the results for each network class and they are comparing the two profiles concerning the computation time, considered as the sum of the Atomic Predicate computation time and the Atomic

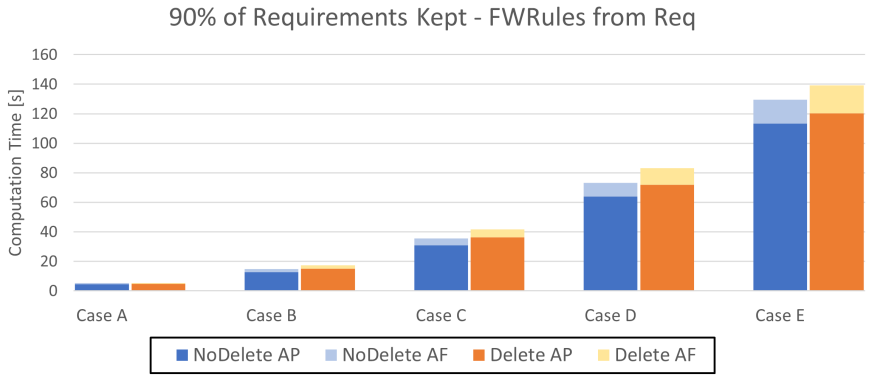

Figure 6.3: AFs computation with 90% PercReqKept and FWRulesFromReq

Flow computation time. As we can see, the experimental results confirm what was expected from the theory. In particular, the smaller is the percentage of overlapping between the Initial and Target sets and the higher is the difference in time between the "Delete" and "NoDelete" profiles. This is evident from the reasoning already done in chapter 5, because the number of deleted requirements, which is increased as the percentage of kept requirements is decreased, has an impact over the number of interesting predicates for the network and consequently over the number of generated Atomic Predicates. For this reason the "Delete" profile has an increased cost which gets more significant as the overlapping between the sets decreases. As said in chapter 2, the majority of the pre-computation time is spent for the generation of the Atomic Predicates while the Atomic Flows generation requires a much smaller time. However, even if with a different contribution, both of these steps have an increased time for the "Delete" profile compared to the "NoDelete" one.

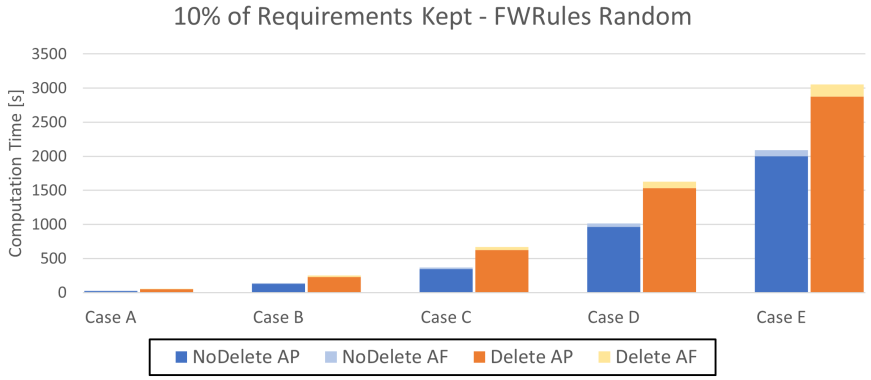

Figure 6.4: AFs computation with 10% PercReqKept and FWRulesRandom

Then, also the situation with the **FWRulesRandom** parameter has been extensively tested. In particular the same combination of **PercReqKept** has been considered with the same three produced graphs, these are showed in picture 6.4, 6.5 and 6.6, and respectively correspond to the values of 10%, 50% and 90% for the percentages of overlapping between Initial and Target sets. As it was explained in the introduction of the chapter, using random endpoints for the source and destination of each configured firewall's rule is causing a steep increase in the total computation time. This is correlated to the increase in the number of interesting

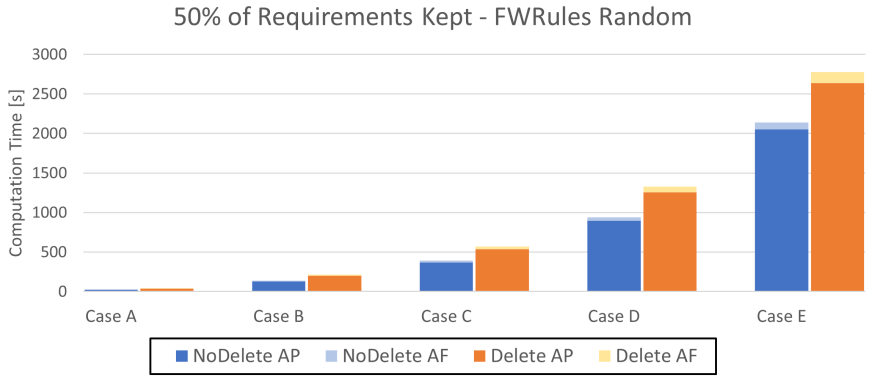

Figure 6.5: AFs computation with 50% PercReqKept and FWRulesRandom

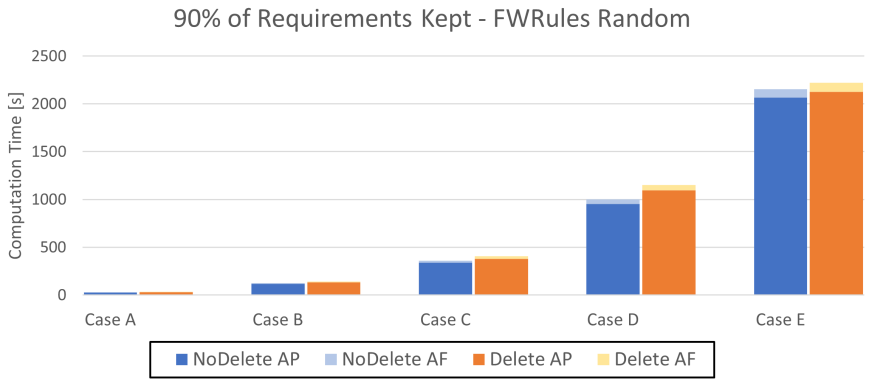

Figure 6.6: AFs computation with 90% PercReqKept and FWRulesRandom

predicates for the network because the rules configured for each firewall are possibly not including nodes already present in the interesting predicates computed for the requirements and so must be added. This second part of the test has been interesting to further stress this pre-computation step and the behavior of the implemented algorithm. First, we can conclude that there is a strong correlation between the parameter **perReqKept** and the observed results, the difference between the two profiles of the algorithms gets more relevant if the overlapping between the two sets decreases because the deleted requirements contribute to the increase in the number of new Atomic Predicates and Atomic Flows. Lastly, this has been fundamental for validating the algorithm for the detection of network areas to reconfigure in a worst case scenario. It was found that the impact of the algorithm could be considered irrelevant when compared to the overall computation time since its contribution always ranged between 0 to 100 ms, with most of the runs being under 10 ms.

6.4 Comparison with previous implementation

The second phase for the validation process has been conducted in a more extensive way and wanted to highlight the advantages obtained with this approach compared to the previous one. The two main parameters which are considered and evaluated in this case are the total computation time, considered as the combination of the

pre-processing time for Atomic Flows generation and the time needed to solve the MaxSMT problem, and the optimality of the solution, in particular how the new approach behave with respect to the optimality goals of VEREFOO for minimization of used resources, goals which have been relaxed in this scenario. This validation phase includes an extra parameter that is correlated to the weight difference assigned to the updated soft constraint for maintaining the configured rules for each firewall present in the initial configuration. This has been useful for choosing the correct weight of this clause and to show with experimental results the effect that the changes on the MaxSMT formulation have on the overall computation time.

Note that the previous approach was not suitable for testing the complete implementation of the framework for many reasons, in particular the firewall placement and their configuration are chosen randomly and not coherent with the network an the requirements. Moreover, also the objective of comparing the reconfiguration approach and the complete configuration from zero required the design of a new process for testing. The proposed solution uses a new test case generator that does not place any firewalls and the idea is that they are allocated directly by the framework only when needed. Then, the adopted process uses three steps: the generation of a new service graph G_1 and a first run of the framework performing the first configuration which produces the graph G_2, then a new set of Network Security Requirement is generated and added to G_2 as the Target set for executing a second run which is in this case a reconfiguration, and finally the same set of new requirements is used with G_1 in order to simulate the reconfiguration from zero that would have been done by the previous implementation of VEREFOO.

Also in this case we are using a progressive enlargement of the network, however the magnitude of the parameters is not the same as the previous tests but include smaller values. The largest tested network includes a little over one hundred endpoints and tens of transformers because the resolution of the MaxSMT problem gets increasingly more complicated as the network complexity increases and we have limited our analysis to situations which resolution required a reasonable time to be completed. The used test cases are the following one:

- **Case A**: 10 REQ, 30 WS, 30 WC, 5 NAT, 5 NATSrc
- **Case B**: 15 REQ, 40 WS, 40 WC, 10 NAT, 10 NATSrc
- **Case C**: 20 REQ, 50 WS, 50 WC, 15 NAT, 15 NATSrc
- **Case D**: 25 REQ, 60 WS, 60 WC, 20 NAT, 20 NATSrc
- **Case E**: 30 REQ, 70 WS, 70 WC, 25 NAT, 25 NATSrc

Regarding the other parameters instead, all tests have been conducted considering only the "Delete" profile since it appears to be the most promising one in terms of achieving the greatest performance advantage for the reconfiguration scenario. Then, different values for **perReqKept** have been considered, in particular the values 10%, 50%, 70% and 90%, and for the soft constraint about the maintaining of the current configured rules three different scenarios have been tested:

1. a difference in weight between an already configured rule and a new one equals to a ratio of 2 (**2X**), i.e., the soft constraint for not using a new rule has a weight k and the one for not using an already configured rule has weight $k/2$.

2. a difference in weight between an already configured rule and a new one equals to a ratio of 10 (**10X**), i.e., the soft constraint for not using a new rule has a weight k and the one for not using an already configured rule has weight $k/10$.

3. not using the modified soft constraint (**NoSoft**), i.e. the constraint for not using a new rule and the one for not using an already configured rule have the same weight

6.4.1 Computation Time

The first results which have been found are considering the total computation time, comparing the reconfiguration approach proposed in this thesis with the *Complete Reconfiguration*, which is the one adopted in the previous implementation of VEREFOO. The presented results are divided by the chosen ratio between the soft constraints for the configuration of firewall's rules, namely the **NoSoft** is represented in picture 6.7, then the **2X** is in picture 6.8, and finally picture 6.9 is for the **10x** case. The experimental results confirmed what was the theoretical hypothesis, in particular that the implementation of the modified soft constraint has an optimization effect on the formulation of the MaxSMT problem resulting in a faster resolution by the solver. Moreover, the higher is the ratio between the weight associated to the new rules compared to the already configured ones and the better is the performance improvement (i.e., **10x** produces better results than **2X**, which is respectively better than **NoSoft**). This because the solver is instructed to prefer a solution which maintains most of the initial configuration, in this way the optimization phase of the problem is simplified and the approach could achieve a better computation time. The results show an advantage overall for the reconfiguration approach with respect to the complete reconfiguration, which is the one performed in the previous version of the framework. This is particularly evident for the larger networks, whereas for case A and case B the results are computed very quickly and the improvement could not be easily evaluated. Another interesting correlation is between the computation time and the parameter of **PercReqKept**, the lower is the percentage of requirements kept and the better is the improvement in performance because it would imply a smaller set of new Network Security Requirements which must be added in the new configuration, a smaller set of nodes which must be reconfigured, and consequently a faster resolution of the problem. The general trend is a faster average computation time if the overlapping of the Initial and Target sets increases, as well as a faster average computation time if the difference in weight for the soft constraint increases. Notably the gap between 6.7 and 6.8 is not as evident as the one with 6.9, however we can see that the **2x** case still shows a performance improvement which becomes more evident as the complexity of the network increases, and consequently the complexity of the MaxSMT problem, such as for networks in case D and case E. This is explained by the newly introduced

optimizations to the soft constraints which impact gets relevant only as the complexity of the problem increases, instead it is limited for smaller network with lower complexity. Beware that case **10x** seems to have the best performance improvement but this is correlated to an high degradation for the optimization goals of minimizing the number of rules and firewalls, which is described later.

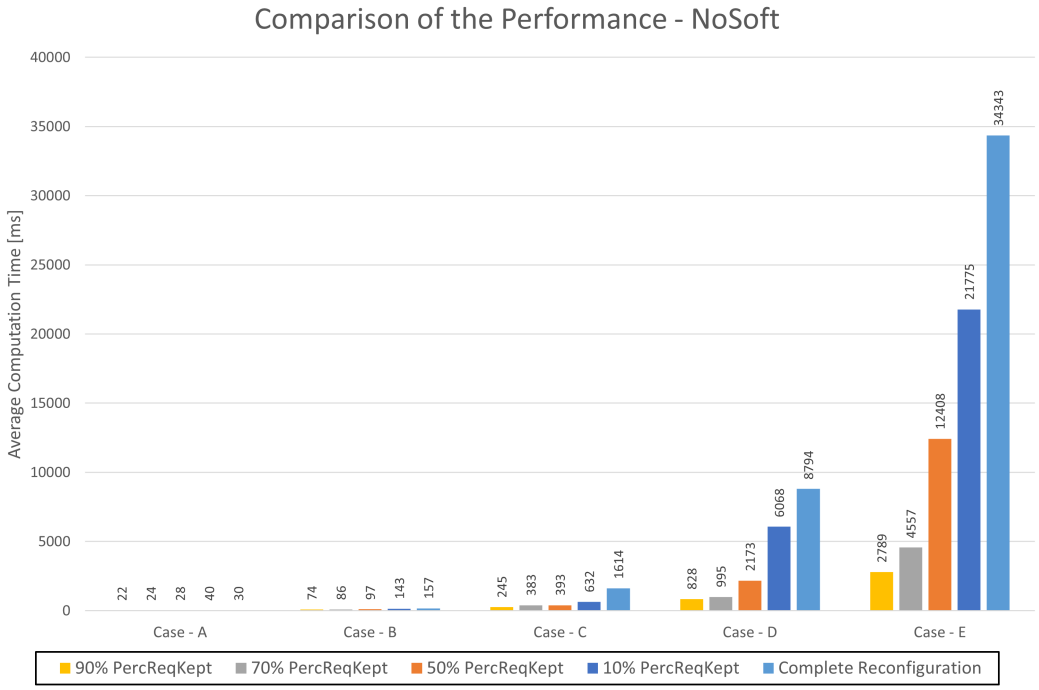

Figure 6.7: Comparison of computation Time for NoSoft scenario

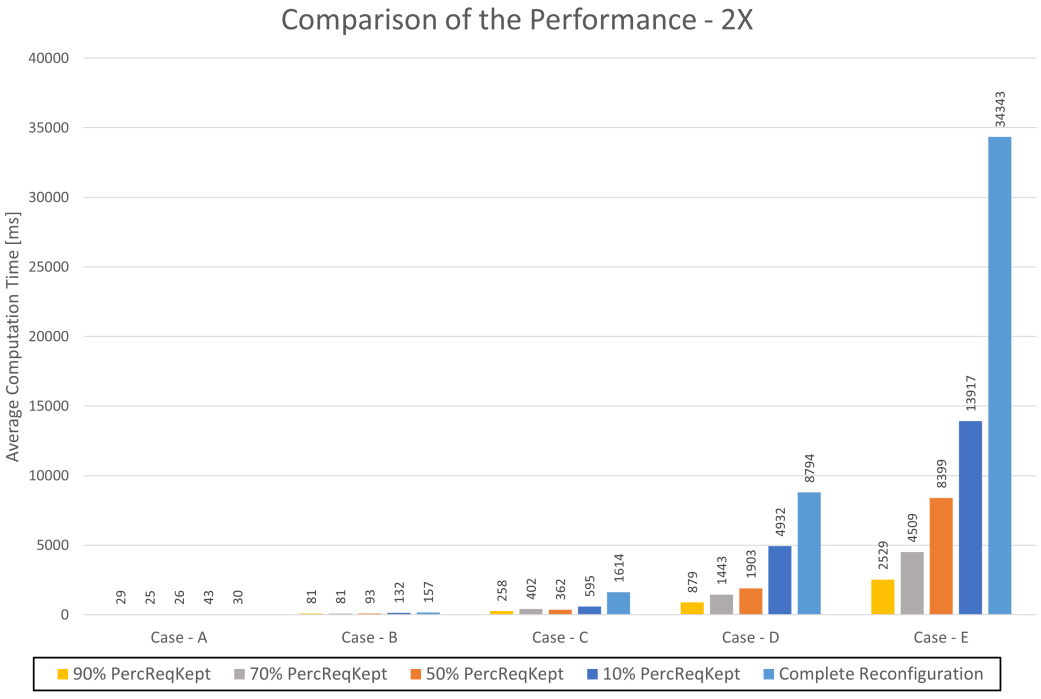

Figure 6.8: Comparison of computation Time for 2X scenario

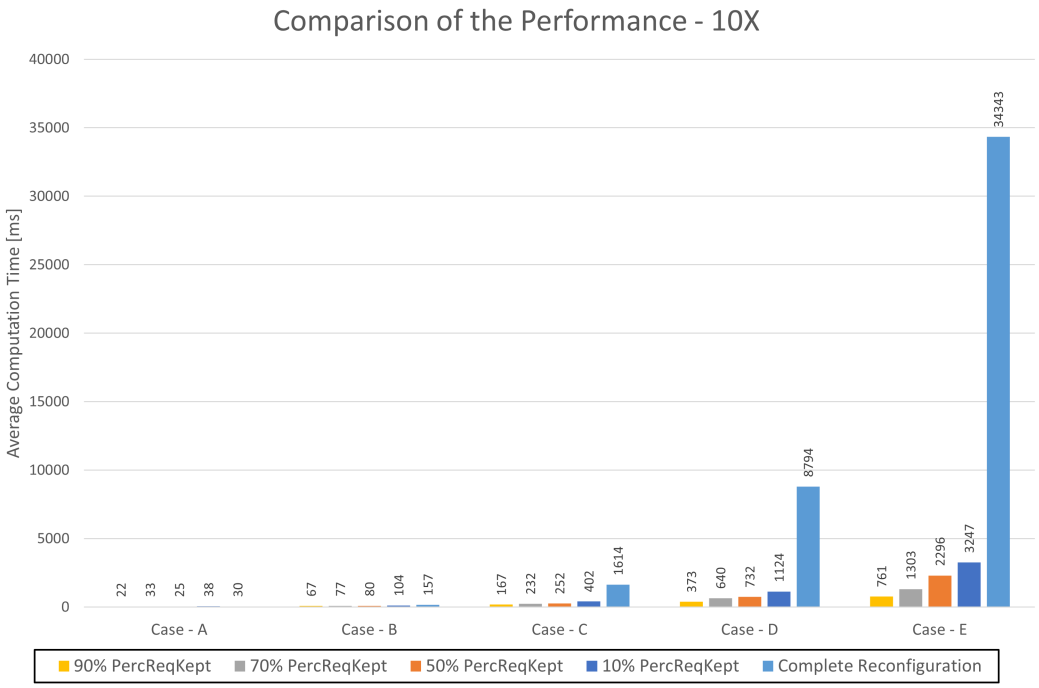

Figure 6.9: Comparison of computation Time for 10X scenario

One aspect that should be noted is that all the above graphs are representing the average values over more than 50 runs, however the set of results presents a large variance which should be taken into consideration. This is showed with another representation limited to the network class "E" and the attribute **2X** in picture 6.10. As we can see the results are sparse over a large interval but the majority is closer to the mean value and the median value is always below it. For this reason we could say that the performance of the reconfiguration approach are reasonably represented by the mean value, and most of the runs are even lower than that (since there are some outliers which are influencing its value).

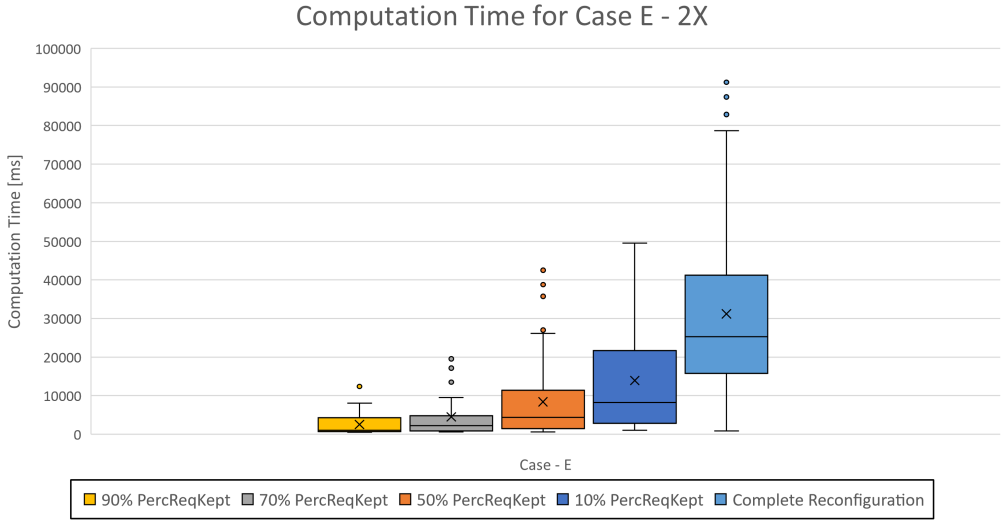

Figure 6.10: Variance of the Computation time

6.4.2 Optimality

The second aspect which has been analyzed in this second phase of the testing process is the optimization results obtained with the reconfiguration approach compared to the ones obtained by the previous implementation. The expected result is that the implementation proposed in this thesis would produce a worst result in terms of minimization of the number of allocated firewalls and configured rules for each one of them. As it was explained in chapter 5, with this solution the set of nodes which configuration is determined by the solver is limited with respect to a configuration from zero because the idea is to maintain some parts of the configuration as they are in order to gain an improvement in the computation time, consequently the solution would be optimal only with respect to the subseto of reconfigured nodes. The presented results are for the same test scenarios used before (i.e., three different cases for the ratio used for the soft constraint and for each one four different values for PercReqKept) but are comparing the average number of configured firewalls and firewalls' rules.

Number of allocated Firewalls

Concerning the number of firewalls, the proposed solution modified the soft constraint associated to their non-allocation so that an already configured one would be preferred with respect to a new one. However the difference in weight associated to the clause has not been modified during the tests, consequently the same results have been obtained for the three cases and are all represented by the graph 6.11. With the proposed approach there is a slight increase of the average number of allocated firewalls which gets more important as the percentage of kept requirements is decreased and consequently the number of added requirements is increased. This is explained by both the modified soft constraint to maintain the state and so prefer the already allocated firewalls with respect to new ones, and also to the suboptimality of the proposed approach which includes only a subset of the possible allocation places in the reconfiguration problem, as it was explained in 5.

Number of configured rules for each firewall

The second analysis has been conducted on the total number of configured rules in each allocated firewall. In this case the results are different according to the adopted ratio for the weight of the soft constraints used for reconfigured firewalls and for new ones. The graph 6.13 compares the average total number of configured firewall's rules with the **10X** scenario and comparing the previous approach (i.e., Complete Reconfiguration) with the new one with different percentages of kept requirements. Instead, graphs 6.13 and 6.14 are similarly representing the same situation but for the **2X** and **NoSoft** scenarios. As it was briefly introduced before, in the **10X** case we are using a more extreme ratio for the soft constraints and strongly preferring the configuration of an already allocated rule with respect to a new one. The result is that the computation time shows a great performance improvement but, as we see in 6.12, there is an high price to pay in the optimality of the configuration. This scenario results in a significant increase in the number of configured rules

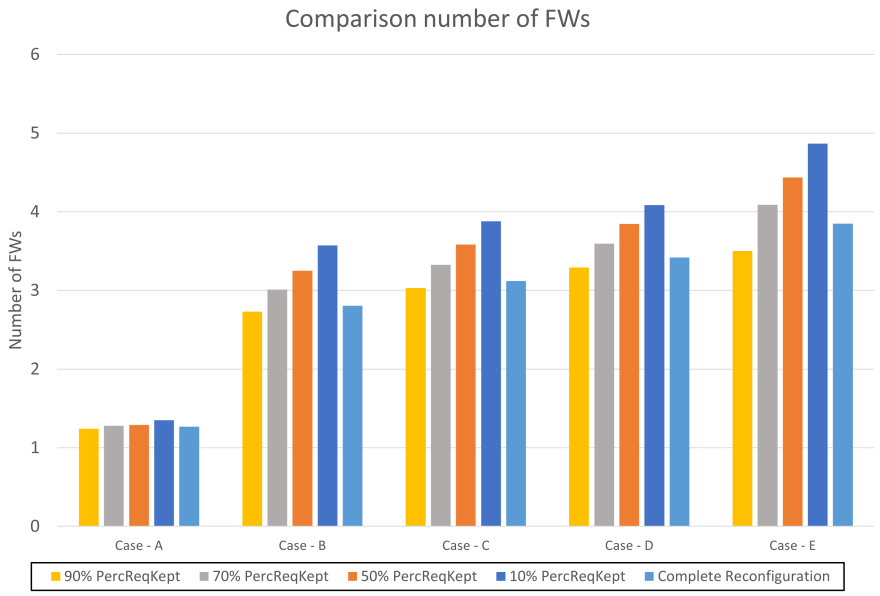

Figure 6.11: Comparison of the number of allocated firewalls

for each firewall and the difference with the previous implementation has been evaluated up to almost one order of magnitude. A more significant result has been obtained by reducing this ratio in the case **2X**. In this scenario the average number of configured rules is almost identical to the scenario where this modification has not been used, i.e., **NoSoft**. This is important because shows that using this ratio we have almost no degradation in terms of optimality but we can still gain some relevant performance advantages during the resolution of the MaxSMT problem.

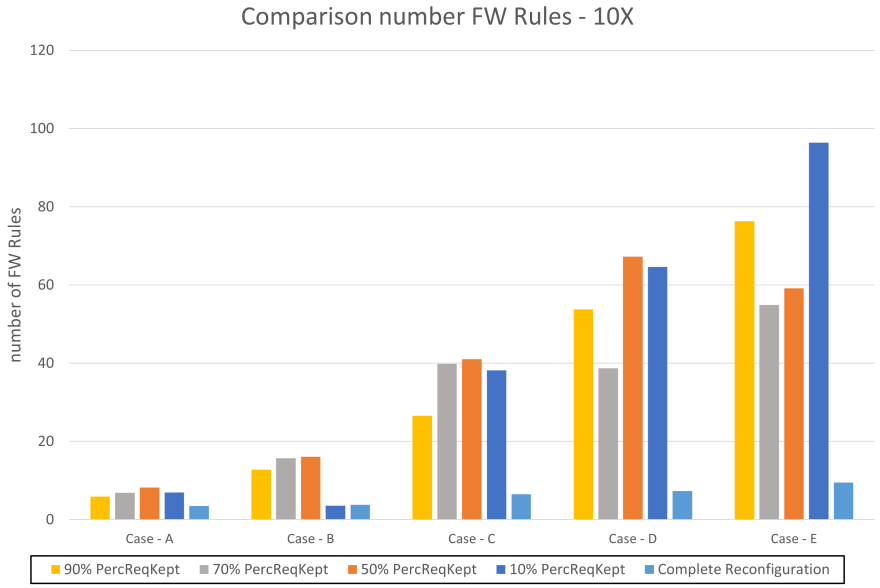

Figure 6.12: Comparison of the number of configured FW Rules for 10X scenario

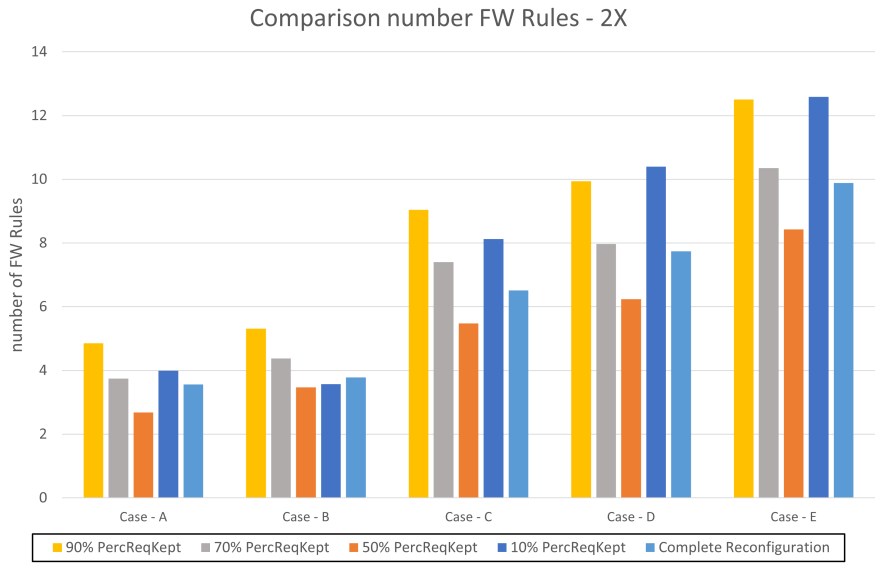

Figure 6.13: Comparison of the number of configured FW Rules for 2X scenario

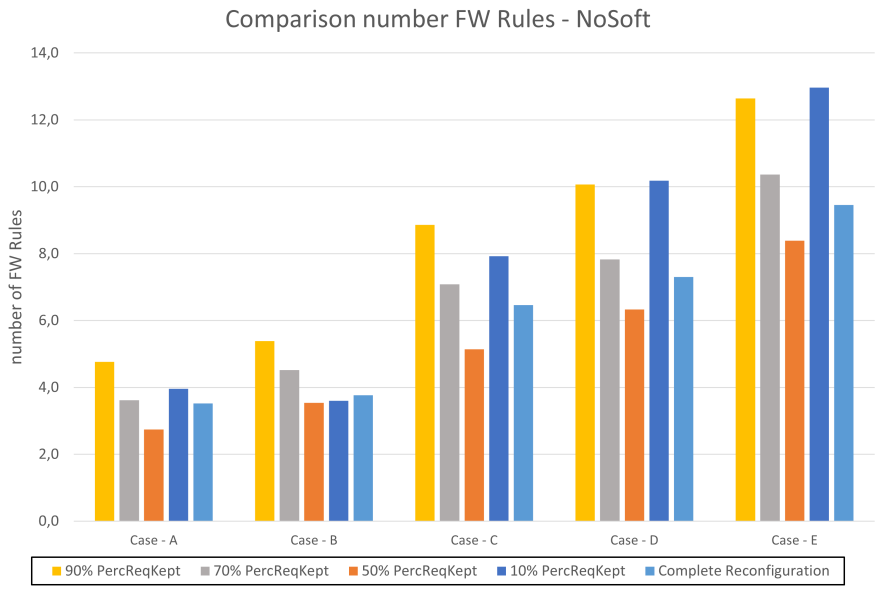

Figure 6.14: Comparison of the number of configured FW Rules for NoSoft scenario

6.5 Stressing the Reconfiguration Approach

A final analysis which has been conducted has evaluated the feasibility and scalability of the proposed approach with an higher number of endpoints and requirements in order to further stress the framework. For this phase, only one combination of parameters has been considered and chosen as the most plausible for a reconfiguration: a value of 70% for **PerReqKept**, the ratio **2X** for the rules, and finally the **NoDelete** profile. Also the synthetic model for the networks has been updated by excluding the allocation of nats and utilizing a chain topology. Specifically, the tests have been executed on the following classes of networks:

- **Case A**: 200 REQ, 20 WS, 20 WC

- **Case B**: 300 REQ, 30 WS, 30 WC
- **Case C**: 400 REQ, 40 WS, 40 WC
- **Case D**: 500 REQ, 50 WS, 50 WC

The graph 6.15 represents the resulting comparison of the achieved compuation times. This graph shows that, even considering the high variability of the values, the reconfiguration approach performs very well when compared to the previous one also in terms of scalability.

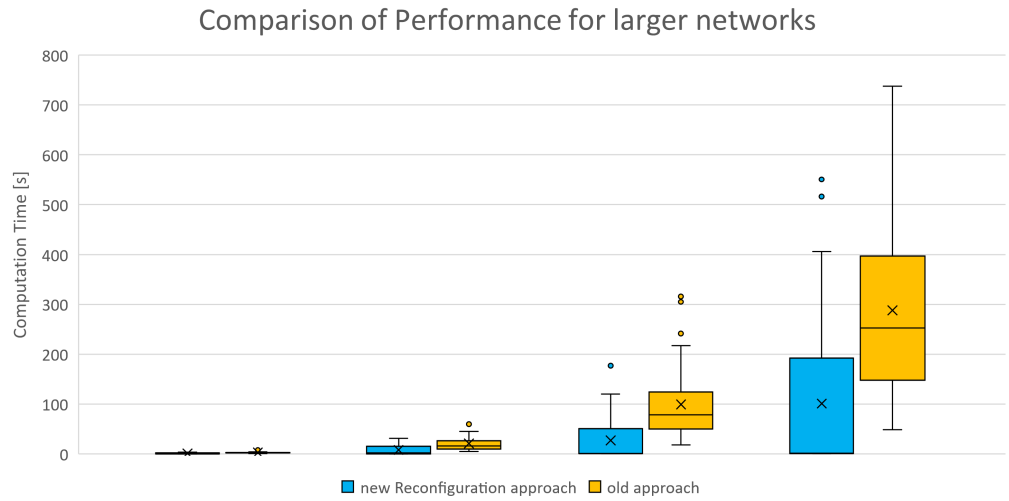

Figure 6.15: Scalability test

Chapter 7

Conclusions

In this thesis work, a novel approach to the reconfiguration problem has been designed and implemented as an extension of the VEREFOO framework, which was lacking in the previous version of an optimized resolution of this scenario. The old approach when confronted with the reconfiguration of a network would not consider the pre-existent elements and just discard them recalculating a new allocation scheme without considering where the previous functions were placed. The goal of this thesis is to propose a more efficient process that reuse the elements of the starting configuration and obtain a new and formally correct solution with a faster computation time, detecting which network components are effected by the changes and modifying only those.

Firstly, an analysis of the state-of-the art in traffic flows and network functions modelling has been conducted, with special emphasis to the two solutions already implemented in the framework, Atomic Flows and Maximal Flows. Then, also a brief overview of VEREFOO has been performed in order to understand its processing flow and for having a general understanding of its architecture. This part was needed in order to understand which components should be modified or are missing, to then introduce the novelties proposed by this thesis.

The reconfiguration problem has been abstracted as the interaction between two different sets of Network Security Requirements, namely the Initial set containing those which are already enforced in the given network and the Target set including those which must be enforced on the final configuration. This model allowed to distinguish the reconfiguration operations into the addition and deletion of requirements, operations which have been described extensively for both type of connectivity requirements, isolation and reachability. The central work has been the design of a new algorithm which permits to select the minimum set of network components that must be reconfigured because they are in conflict with the changes wanted by the user. The algorithm design has been presented in a separate way for each type of requirement and each type of action, the addition or deletion. In this phase two different profiles for the reconfiguration have been studied, one more focused on the performance and the second one more focused on the optimization.

Moreover, after having defined the algorithm and the model of the problem, also the way in which the Allocation graph was automatically generated starting from the Service graph has been modified. In particular the allocation graph generator

has been changed to take into account the reconfiguration scenario and avoid placing new Allocation Places near the nodes which could be reconfigured by the solver, with the idea that these nodes should be used instead of placing additional ones.

Finally, this thesis also updated the formulation of the constraints for the MaxSMT problem to optimize the performance of the solver for the considered reconfiguration scenario. In particular the soft constraints have been modified so that the state of the provided configuration is preferably maintained, which means that the allocation of an already allocated firewall should be preferred with respect to allocating a firewall in an empty Allocation Place, and the same for the configured rules in each firewall.

After all these works have been concluded, the new approach has been implemented and validated with a series of performance tests to compare it with the previous version of the framework. The scope was to understand the differences both in terms of computation time, and consequently the scalability of the approach, but also considering the optimality of the achieved solution. It has been found that the reconfiguration solution proposed in this thesis achieves a relevant improvement in the computation time in different scenarios with a very slight degradation of the optimality goals, solution which is more than acceptable considering the proposed application as a reaction to a cybersecurity attack. Future works may extend the same approach to include a larger set of network functions and new types of Network Security Requirements which could cover additional security features. A longer-term possibility is also to exploit this solution as a starting point for the design of a parallelized approach for the resolution of the problem, dividing the network into independent areas that could be configured separately (and in parallel), and consequently breaking one single and complex problem into multiple smaller ones, each requiring a shorter computation time.

Bibliography

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

[2] R. Chayapathi, S. F. Hassan, and P. Shah, *Network Functions Virtualization (NFV) with a Touch of SDN*. Addison-Wesley Professional, 2016.

[3] E. T. S. Institute, "Network functions virtualization (nfv); architectural framework," December 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

[4] D. Bringhenti, J. Yusupov, A. M. Zarca, F. Valenza, R. Sisto, J. B. Bernabe, and A. Skarmeta, "Automatic, verifiable and optimized policy-based security enforcement for sdn-aware iot networks," *Computer Networks*, vol. 213, p. 109123, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622002468

[5] D. Bringhenti, F. Valenza, and C. Basile, "Toward cybersecurity personalization in smart homes," *IEEE Security & Privacy*, vol. 20, no. 1, pp. 45–53, 2022. [Online]. Available: https://doi.org/10.1109/MSEC.2021.3117471

[6] G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, and A. Ksentini, "A formal approach to verify connectivity and optimize vnf placement in industrial networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 1515–1525, 2021. [Online]. Available: https://doi.org/10.1109/TII.2020.3002816

[7] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "A novel approach for security function graph configuration and deployment," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 457–463. [Online]. Available: https://doi.org/10.1109/NetSoft51509.2021.9492654

[8] I. Pedone, A. Lioy, and F. Valenza, "Towards an efficient management and orchestration framework for virtual network security functions," *Security and Communication Networks*, vol. 2019, 2019. [Online]. Available: https://doi.org/10.1155/2019/2425983

[9] H. Tabrizchi and M. Kuchaki Rafsanjani, "A survey on security challenges in cloud computing: Issues, threats, and solutions," *J. Supercomput.*, vol. 76, no. 12, p. 9493–9532, dec 2020. [Online]. Available: https://doi.org/10.1007/s11227-020-03213-1

[10] Verizon, "Data breach investigations report," 2022. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[11] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Introducing programmability and automation in the synthesis of virtual firewall rules," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 473–478. [Online]. Available: https://doi.org/10.1109/NetSoft48620.2020.9165434

[12] S. Bussa, R. Sisto, and F. Valenza, "Security automation using traffic flow modeling," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 486–491. [Online]. Available: https://doi.org/10.1109/NetSoft54395.2022.9844025

[13] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "*Firmato*: A novel firewall management toolkit," *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, 2004. [Online]. Available: https://doi.org/10.1145/1035582.1035583

[14] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, "Improving the formal verification of reachability policies in virtualized networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 713–728, 2021. [Online]. Available: https://doi.org/10.1109/TNSM.2020.3045781

[15] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2929325

[16] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *Proceedings of the 2nd Workshop on Cyber-Security Arms Race*, ser. CYSARM'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: https://doi.org/10.1145/3411505.3418439

[17] E. Wong, "Validating network security policies via static analysis of router acl configuration," 2006.

[18] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. USA: USENIX Association, 2012, p. 9.

[19] A. R. Khakpour and A. X. Liu, "Quantifying and querying network reachability," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, 2010, pp. 817–826.

[20] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," pp. 1–11, 2013. [Online]. Available: https://doi.org/10.1109/ICNP.2013.6733614

[21] B. E. Carpenter and S. W. Brim, "Middleboxes: Taxonomy and issues," *RFC*, vol. 3234, pp. 1–27, 2002. [Online]. Available: https://doi.org/10.17487/RFC3234

[22] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3160293

[23] ——, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020 - IEEE/IFIP Network Operations and*

Management Symposium, Budapest, Hungary, April 20-24, 2020. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/NOMS47738.2020.9110402

[24] J. M. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," *RFC*, vol. 7665, pp. 1–32, 2015. [Online]. Available: https://doi.org/10.17487/RFC7665

[25] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. Pastor Perales, "Adding support for automatic enforcement of security policies in nfv networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 707–720, 2019. [Online]. Available: https://doi.org/10.1109/TNET.2019.2895278

[26] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, p. 2601–2614, oct 2017. [Online]. Available: https://doi.org/10.1109/TNET.2017.2708096

[27] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005.

[28] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems - 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008, Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3

[29] D. Bringhenti and F. Valenza, "Optimizing distributed firewall reconfiguration transients," *Computer Networks*, vol. 215, p. 109183, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912862200281X