Automatic, verifiable and optimized policy-based security enforcement for SDN-aware IoT networks

Automatic, Verifiable and Optimized Policy-based Security Enforcement for SDN-aware IoT networks

Daniele Bringhenti[a,*], Jalolliddin Yusupov[a], Alejandro Molina Zarca[b], Fulvio Valenza[a], Riccardo Sisto[a], Jorge Bernal Bernabe[b], Antonio Skarmeta[b]

[a] *Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy*
[b] *Dep. Communications and Information Engineering, University of Murcia, Murcia, Spain*

Abstract

The pervasiveness of Internet of Things (IoT) has made the management of computer networks more troublesome. The softwarized control provided by Software-Defined Networking (SDN) is not sufficient to overcome the problems raising in this context. An increasing number of attacks can, in fact, occur in SDN-aware IoT networks if the security configuration enforced on the SDN switches is manually computed and not formally verified. To mitigate this problem, this paper proposes a novel methodology which leverages Maximum Satisfiability Modulo Theories (MaxSMT) to automatically compute a formally correct and optimized allocation scheme and configuration of SDN switches by refining security policies, user-defined or derived from detected attacks. This mechanism is compliant with the main characteristics of virtualized IoT-based networks, such as the simultaneous presence of numerous interconnected devices and strict latency requirements. The feasibility and the performance of the framework developed to implement this methodology have been validated in a realistic use case.

Keywords: security, IoT, SDN

1. Introduction

The *Internet of Things* (IoT) [1] is bringing countless benefits for many applications in next generation computer networks. However, several IoT characteristics, such as its pervasiveness, heterogeneity, constrained device nature, lack of security measures on cheap devices, lack of vendor support, continuous software updates, large-scale, and native complexity, are making IoT deployments subject to an increasing number of threats and attacks, e.g., *Distributed Denial of Service* (DDoS) attacks [2].

The inadequate level of security expertise in IoT device design and manufacturing in comparison with traditional cyber-device design [3] is another factor that facilitates cyber-security attacks on such systems.

Many IoT-based systems are also mission-critical or safety-critical. The interruption or alteration of their services, caused by successful cyber-attacks, may have dramatic consequences, making security risk very high for them. This trend is particularly evident in *Industrial Control Systems* (ICSs) that require *massive Machine Type Communications* (mMTC). Nonetheless, many other types of IoT-based systems share the same features.

In IoT deployments, it is common practice that security appliances needed in computer networks, including switches, routers, or firewalls, are manually configured by human beings. However, with the increasing complexity

*Corresponding author
Email addresses: daniele.bringhenti@polito.it (Daniele Bringhenti), jalolliddin.yusupov@polito.it (Jalolliddin Yusupov), alejandro.mzarca@um.es (Alejandro Molina Zarca), fulvio.valenza@polito.it (Fulvio Valenza), riccardo.sisto@polito.it (Riccardo Sisto), jorgebernal@um.es (Jorge Bernal Bernabe), skarmeta@um.es (Antonio Skarmeta)

of IoT-based systems, this manual configuration of security functionalities is progressively becoming unpractical, unbearable for security administrators, and error-prone. The huge number of devices that might be connected to the network at the same time usually requires heterogeneous security measures [4], and human beings cannot have everything under their control when the size and heterogeneity of the systems grow. Therefore, it is clear that new approaches must be pursued to overcome these problems.

Introducing automation in security configuration can solve this issue and, at the same time, enable self-healing and self-protection capabilities in the managed IoT systems. Automation can be leveraged with the support of network softwarization. The groundbreaking paradigm called *Software-Defined Networking* (SDN) now offers a softwarized and centralized network management approach that facilitates and automates network (re-)configuration by splitting control and data planes. This mechanism can be employed to automatically and dynamically change the configuration rules of the main network devices that are used in SDN networks, i.e., SDN switches [5], so dynamically changing the enforced security policies, e.g., in reaction to cyber-attacks [6].

By pairing rigorous formal methods with automation, it is also possible to provide the high correctness assurance of the security management operations related to the SDN switches, required for safety-critical or mission-critical IoT-based systems [7].

In light of these motivations and ideas, this paper proposes a verified, optimized, and automatic policy-based security enforcement mechanism for SDN-based IoT computer networks. The proposed approach aims at automatically computing the optimal allocation scheme and configuration of a distributed architecture of SDN switches within an IoT system through policy refinement, also allowing automatic reaction to cyber-attacks and optimal self-reconfiguration for the IoT network. These objectives are achieved by formulating the automatic configuration problem as a *Maximum Satisfiability Modulo Theories* (MaxSMT) problem, which provides at the same time a cost-effective formal approach and optimization. This proposal represents a progress in literature, as past research works [8] on SDN-based security management of IoT networks do not provide automated solutions to compute the allocation scheme of SDN switches (beyond the calculation of the shortest path), and the SDN rules in a provably correct and optimal way.

Our solution stems from a recent proposal of using MaxSMT for automatic configuration of packet filters in NFV-based service graphs [9]. The approach proposed in [9] can automatically configure traditional firewalls in NFV networks, but it does not capture the expressiveness of SDN switches. Moreover, the only optimization goal of that approach is to minimize the number of firewalls and of configuration rules. With respect to [9], this paper makes some steps ahead, by providing the following new contributions:

- The formal models proposed in [9] have been extended to adhere to the higher complexity of virtualized IoT networks. Examples of new features that have been specifically designed for this purpose are the models for the rules of the SDN switches and for the network security policies in the IoT context. They model, for example, the possibility to refer to multiple protocols.
- New optimization goals tailored to the specific needs of IoT networks have been defined. Examples of these objectives are achieving the best performance of the filtering operations, minimization of attack impact, and bandwidth optimization (e.g., by placing SDN switch rules as close as possible to the source of the traffic to be blocked). The definition of these new optimization goals could not be managed just by changing the objective function, but it was necessary to reformulate the soft constraints of the MaxSMT

problem, so as to make them compliant with the new models and the new objectives.

- The approach illustrated in this paper is not limited to computing the allocation scheme and configuration of SDN switches, but it also addresses other security orchestration operations (e.g., detection and mitigation mechanisms). By adding these new features, we defined a single integrated process capable of managing the life-cycle of security services based on SDN switches in an automatic way, without requiring the aid of other external tools or human intervention. This integrated process is thus perfectly suitable for modern SDN-aware networks, where high agility and dynamism are key characteristics that should be managed by avoiding operational delays as much as possible.
- A new framework has been developed to fully implement and test the proposed approach. This framework leverages the SDN orchestrator ONOS for processing the computed SDN configuration and it can interact with monitoring agents for getting the feedback required by the integrated mitigation mechanism.
- The solution and its implementation have been extensively validated, in terms of both effectiveness and scalability. The framework has been tested in an experimental set-up representing a realistic SDN-aware IoT scenario, and the performance of the different operations (e.g., automatic configuration, SDN enforcement) has been evaluated.

The rest of this paper is structured as follows. Section 2 describes the related work. Section 3 defines the overall approach. Section 4 delves into the formal model for SDN switch configuration. Implementation and evaluation of the proposal are described in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

One main contribution of this paper is the definition of an automatic mechanism for the configuration of SDN switches in IoT networks. The proposed approach combines three features –automation, formal verification, and optimization–, so that the automatically computed security configuration is provably correct and optimized. In Subsection 2.1, a comparison with the most related papers, representing the state of the art for automatic security configuration, is presented. The aim is to underline how the proposed approach is the first one that combines all the three mentioned features jointly for the management of SDN switches in IoT-based networks.

Another main contribution is the integration of this automatic configuration mechanism within traditional SDN-based detection and reaction processes of SDN orchestration. Therefore, Subsection 2.2 discusses papers focusing on SDN-based approaches for strengthening security in IoT networks, in order to underline how this integration represents an added value with respect to current strategies adopted in SDN environments.

2.1. Automatic security configuration

Most of the papers about automation for network security configuration focus on firewalls, as they represent the most common filtering function. The first works that introduced automation in firewall configuration are [10], [11] and [12]. They are, nevertheless, limited with respect to this paper: their methodologies are designed to exclusively work in traditional networks, based on hardware middleboxes, and they lack both formal verification and optimization. Later, formal verification has been introduced in the techniques described in [13] and [14], but their outcome is not optimal. Instead, optimization is pursued in the approaches presented in [15], [16] and [17], but they can only fix an already configured firewall rule set, instead of automatically creating it. Finally, the technique presented in [9] can automatically compute firewall configurations in

an optimal and provably correct way, through a MaxSMT formulation of the problem. However, it can only work in environments based on the *Network Functions Virtualization* (NFV) principle and it is not adequate to work with SDN switches in an industrial SDN-based IoT environment. Two other relevant works [18][19] broaden the scope to other types of network security functions, including other filtering functions. On one side, [18] proposes an automatic mechanism for the configuration of security functions; yet, that mechanism lacks a formal guarantee that the computed configuration is compliant with the security policies to be enforced in the network. On the other side, the technique illustrated in [19] can establish how the functions should be allocated in a virtual topology to enforce the security policies, and it can guarantee the correctness of this decision because it is based on the definition of formal models. However, the effective rule sets of the allocated functions are not automatically computed. Moreover, the resulting allocation scheme may be sub-optimal since heuristics are employed.

In literature, only a limited number of papers deal with automation specifically for the configuration of SDN switches. The main proposals in this research area are CloudWatcher [20], Procera [21], Fresco [22] and OpenSec [23]. The techniques presented in these papers automatically enforce user-specified policies, expressed with user-friendly languages such as Ethane [24], Frenetic [25] or PolicyCop [26], into the configuration of SDN switches. However, these methodologies are not as feature complete as the one that is proposed in this paper. They cannot automatically establish the optimal and correct allocation scheme of SDN switches, and they cannot automatically synthesize the SDN rules in a provably correct and optimal way.

Other two relevant papers about optimal rule placement in SDN networks are [27] and [28]. The first one ([27]) deals with the design of the first *Software-Defined Internet Exchange Points* (SDX) controller that can work with networks of the same scale as the largest *Internet Exchange Points* (IXPs). The role of such an industrial-scale SDX controller is to distribute flow rules in the switches. However, this is achieved without focusing on security issues and without the aid of formal verification techniques to prove the validity of the controller's work. Instead, in the second paper [28], an SDX controller is used to provide IP-spoofing protection, through the automated synthesis of the optimal low-level flow rules that must be inserted in the SDN switches. With respect to our contribution, however, in this work the policies specified by the user are simply translated into the corresponding flow rules by means of a compiler. Instead, in our approach a policy refinement mechanism is used, so that the rules generated from the user-specified policies are optimized for the switches they will be installed on.

Other approaches proposed in literature [29] [30] aim at solving the traditional service function chaining problem, i.e., identifying a correct order of virtual functions to fulfill security requirements. However, these approaches do not challenge the auto-configuration problem, and do not leverage formal methods. Besides, virtualized SDN switches are only a type of function, among all the ones that are considered in these works, and therefore they do not represent their real focus.

Finally, as mentioned so far, the number of papers addressing the needs of formal verification for the configuration of firewalls and, more specifically, of SDN switches is small. One may argue that in literature formal methods have been extensively adopted to formally verify network security policies in computer networks [31, 32, 33, 34, 35, 36, 37]. A large variety of different formal methods have been employed in state-of-the-art verification approaches: header space analysis [31], symbolic execution [34], incremental algorithms [35], *Satisfiability* (SAT) solvers [32], and SMT solvers [33, 36, 37]. However, all these approaches lay their foundations on an a-posteriori verification, i.e., the consistency and correctness of the secu-

rity configuration is formally verified only after its generation. Therefore, in case the computed configuration is not correct, another generation is required, and the overall process may become time-consuming and may not be able anymore to address the dynamism required by SDN-aware IoT networks. Instead, this paper pursues an approach based on the correctness-by-construction principle characterizing MaxSMT formulations. As such, the automatically computed solution is formally guaranteed to be correct, and there is no need to apply a-posteriori formal verification techniques anymore.

2.2. Strengthen security in IoT networks with SDN

Surveys [4] and [38] analyze the main benefits brought by SDN to strengthen security in IoT networks. In this sense, Rawat et al. [39] analyze diverse security threats and attacks and how they can be mitigated using proper countermeasures based on SDN.

Among the papers described in these surveys, [40] and [41] are two of the most relevant works proposing SDN-based detection and mitigation mechanisms against DDoS attacks. In particular, [40] presents the design of Bohatei, a flexible and elastic DDoS defense system, which steers suspicious traffic while minimizing user-perceived latency and network congestion. Instead, [41] describes a lightweight method for DDoS attack detection, where the most important features of malicious traffic flows are extracted and analyzed with a very low overhead compared to traditional approaches. However, in both these approaches, DDoS is the only attack kind that is addressed. Besides, their main objectives concern the optimization of network parameters, such as latency, instead of security concerns such as isolation from a malicious host.

Other more recent works address this topic in a broader manner. In [8], the authors propose a policy-driven holistic cyber-security management framework that relies on SDN and NFV to reinforce IoT security. The framework can dynamically orchestrate security in IoT, detecting cyber-security incidents and reacting according to the context by enforcing security policies dynamically. Likewise, in [42], the authors rely on that framework and SDN/NFV to detect and counter cyberattacks in IoT through virtual honeynets as *Virtual Network Functions* (VNFs), that simulate real IoT networks deployments, so that attackers can be distracted from the real target. However, in the above papers the security policy enforcement process is not provably correct and it does not provide optimal solutions as we achieve in this paper.

Finally, [43] and [44] specifically focus on SDN-based policy enforcement for securing IoT networks. In both papers, the authors describe innovative strategies to detect volumetric attacks in SDN networks, exploiting *Manufacturer Usage Description* (MUD) policies. These works, however, do not address automatic responses to the identified attacks, and simply represent a proposal of how policy-based management can be used in SDN-based IoT networks.

3. Approach

This section gives a high-level overview of the proposed approach to automatically allocate and configure SDN switches in virtualized IoT-aware networks, to satisfy user-specified network security policies, or in reaction to an attack detection. The only interaction between a network administrator and the automated process is in the policy specification phase, which occurs whenever policies are modified. Full automation is thus effectively achieved, alongside with all the benefits which it can carry over, such as avoidance of human errors and faster completion of the configuration operations. Figure 1 illustrates the main phases of the automated process (policy specification, configuration generation, configuration enforcement, inspection and detection, mitigation). The description of each phase follows, specifying the role of each component of the global architecture.

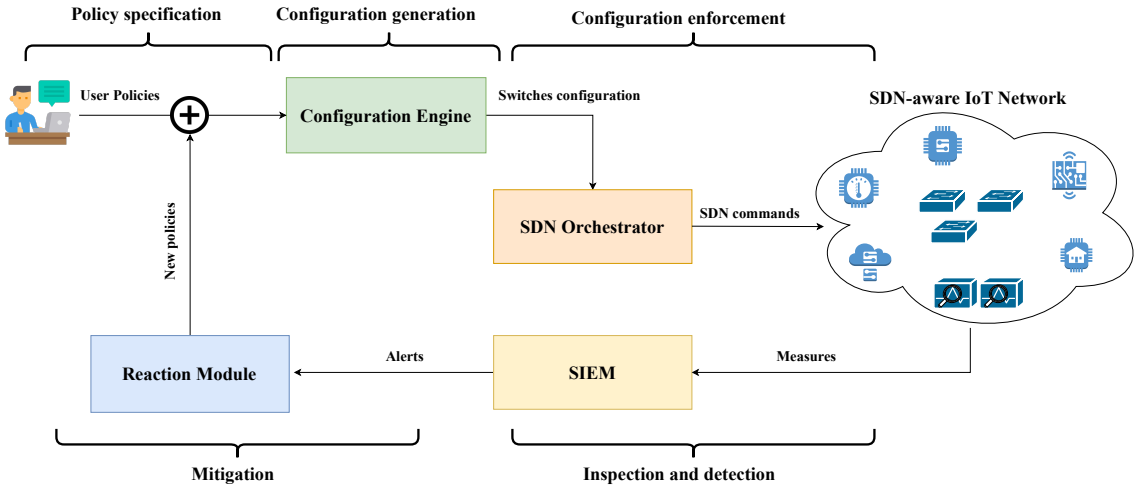


Figure 1: Workflow of the approach

Policy specification: At the beginning, when the SDN switches have not yet been allocated or configured in the IoT environment, the network administrator specifies a set of initial *network security policies*, each one describing a security requirement which must be fulfilled in the network – e.g., isolation between an IoT Application server and a group of IoT devices after it has been victim of an attack –. Policies can be defined in a high-level language, which is user-friendly, independent from the specific commands required for the set-up of the switches, and close to natural human language. This high-level language follows the traditional "subject-action-object-attributes" paradigm, also known as "target-effect-condition" paradigm [18]. Examples of security policies are "allow traffic from host A to server B only if it uses TCP" and "block the traffic generated by the subnetwork of the corporate section of company C and headed towards company D". The high-level representation of each policy can be easily translated, without requiring policy refinement, into statements written in a medium-level language, that provides all the required information for deciding how SDN switches should be configured but abstracts the specific configuration settings of their different implementations. This medium-level language is later formalized in Subsection 4.2. If the security administrator has a good expertise, she can even formulate the policy set directly in medium-level language. In both cases, the specified policy set may be not optimized (e.g., some policy rules could be redundant) or not correct (e.g., some conflicts may be present).

Configuration generation: The configuration engine represents the core element of the optimization process. It requests a policy set and the description of an SDN-based IoT network as inputs. Through policy refinement, it then automatically establishes the optimal allocation scheme of the SDN switches in the network and generates the optimal rule sets which must be enforced on them to fulfill all the policies. This result is reached by solving a MaxSMT problem, which formally encodes the orchestration and configuration problem. The problem is defined following an approach similar to [9], but specifically designed to tailor the requirements of allocating and configuring SDN switches in IoT networks. Some elements of the SDN switch model (e.g., the capability of working on multiple protocols at the same time, and the limits on the maximum number of rules that can be installed) have been introduced to deal with specific features of devices for IoT networks. Also, the engine can pursue IoT-related optimization objectives, such as attack impact and bandwidth minimization, when a large number of IoT devices are simultaneously connected. Additionally, the generated rules

are compliant with the format that is requested by the SDN orchestrator for their enforcement, differently from the input policy set.

In this context, it is worth underlining the different meaning of the term "placement" with respect to "allocation" for virtualized functions. "Placement" corresponds to the deployment of the virtual machines on the physical infrastructure, i.e., deciding on which general-purpose servers of the physical real network they must be embedded. Allocation means deciding where the functions must be put in the logical topology representing the control plane view of a virtualized network (e.g., all the virtual functions could be placed on the same server, but their allocation scheme in the virtual topology might be complex and ramified). In light of this clarification, the goal of the configuration engine is to compute the allocation scheme, not to design the placement plan.

Configuration enforcement: Once the configuration (made by allocation scheme and SDN rules) has been generated, the SDN orchestrator gathers this information and builds an enforcement requests message by using the orchestration driver which corresponds with the current SDN Controller deployment. By using different orchestration drivers, it can be implemented in an extensible and reusable way, allowing the communication between the SDN orchestrator and different implementations of SDN Controllers (e.g., ONOS, ODL and Ryu). In general, this communication is based on a Northbound/Southbound API approach. In that sense, firstly the orchestrator driver sends the configuration to the SDN Controller Northbound API. Then, the SDN Controller processes the SDN configuration, and it sends the required SDN commands to the specific switches by using an SDN protocol (e.g., OpenFlow FlowMod command) through the Southbound API. These commands will define the SDN switches behavior in order to comply with the defined security policies. In this context, the SDN Controller is also the component that is in charge of the placement of virtual SDN switches, which is not directly addressed by the auto-configuration methodology proposed in this paper.

Inspection and detection: Since IoT networks have more restrictions than traditional computer networks, they are suitable targets for attackers, especially in terms of DDoS attacks where IoT devices can be the victims or they can be infected in order to contribute in a botnet. Apart from these kinds of attacks, IoT infrastructures must be prepared to prevent and react to different kind of threats, like those related to authentication or authorisation, malware infection, traffic manipulation or even zero-day vulnerabilities. In the workflow of the proposed approach, by providing and deploying different types of monitoring and analysis tools, the infrastructure is able to detect a big amount of different kinds of attacks as well as notify the system for strange behaviors. For this detection phase, the proposal can rely on different existing monitoring agents and tools [8][45], such as *Montimage Monitoring Tool* (MMT)-probe [46], IoT brokers, *Intrusion Detection System* (IDS) instances (e.g., Snort), multiple incident detectors (e.g., MMT-security), event-based security tools such as XL-SIEM or Apache Storm, and AI-based attack detection modules such as [45].

In particular, the attacks that can be identified for this approach are the attacks that are carried out throughout the network. For instance, replays attacks, aimed to intercept, repeat or delay network data transmissions, can be combined with masquerade/spoofing attacks and man-in-the-middle attacks, whereby malicious users aim to receive or access traffic from the victim(s). In these cases, once an IDS detects the attempt, our proposed system can mitigate the attack by automatically isolating the traffic from/towards the attacker (e.g., spoofed IP) in the SDN switch, by adding a new filtering policy. Additionally, DDoS attacks, launched for instance by IoT botnets, can be countered efficiently and automatically by our system, at the edge of the network, close to the source (the attacker), i.e., re-configuring the policies in SDN switches

nearest to the bots. This prevents the attack to be expanded in the network affecting a high number of network elements. Other kinds of application-level attacks, such as malicious code injection in services or in memory SQL Injection (SQLi), unexpected access to the network and queries to data-bases, can be detected by *Deep Packet Inspection* (DPI) tools and by the *Security Information and Event Management* (SIEM). In this case, our system can isolate the attacker automatically through our filtering policies enforced in the SDN switches.

In addition to the detection of these types of threats, the *Authentication, Authorisation and Accounting* (AAA) infrastructure can notify the system for each authentication or authorisation issue. IoT brokers can notify the system if they are not receiving properly the data from IoT devices. In order to perform traffic inspection for detecting abnormal behaviors, on the one hand, rule-based IDSs and *Intrusion Prevention Systems* (IPSs), like Snort or Surikata, are deployed at different points of the network, generating events or alerts according to the configuration rules. It is important to highlight that monitoring tools can be deployed and configured in a proactive and reactive way, depending on monitoring security policies. For instance the security administrator can enforce network traffic analysis policies which contain *MonitoringConfigurationConditions* (e.g., set of signatures or specific matches per fields) as well as different *MonitoringActions* (e.g., reports or alerts). These security policies are then translated into specific IDS/IPS configurations (e.g., Snort or MMT rules) which can be enforced in new instances or in already existing ones. These IDS/IPSs are deployed in wired and wireless parts of the SDN network, and tools like MMT-Probe are able to analyze traffic directly in 6lowPAN networks. On the other hand, SDN Controllers provide important information regarding network status like traffic volume for specific device ports, links bandwidth or flow statistics. These different information sources are then analyzed and correlated by using SIEM tools. Throughout this analysis, the SIEM determines if the system is under attack, or if the behaviour could be part of a well known attack. In this case it identifies the issue and it notifies to a reaction module. For instance, it can determine if the amount of traffic for a specific port of an SDN switch is out of the common analyzed behaviour and to correlate this information with other monitoring sources.

Mitigation: When an attack carried out throughout the network is identified, the reaction module must define a new set of policies. The main purpose is to change the behavior of the IoT network, so that the security functions can enforce an adequate protection against the attack for which the previous defense barriers were not enough. Reactive SDN configurations are, in fact, able to provide powerful countermeasures against different kinds of attack. In the generation of this new policy set, it is however fundamental to consider not only the policies strictly related to the detected attack, but also the previous set which was used for the automatic computation of the SDN switches' configuration. The main reason is that it is possible that some of the original policies are not valid anymore because the network nodes which they were dealing with have been victim of a cyberattack. After the new policy set is computed, it is exploited by the configuration engine for the generation of the new allocation scheme and forwarding rules of the SDN switches, thus closing a full action-reaction loop. In this reconfiguration of the network, it is possible that the number of switches and the rules in switches would change. However, such reconfiguration would not bring service outage. The reason is that, in a virtualized environment, the SDN orchestrator can set-up new virtual functions or add new rules to a switch while the others are still working. Similarly, the SDN orchestrator can release resources, when they are not consistent with the newly computed allocation scheme and configuration anymore. Finally, this cycle can potentially be repeated every time an attack is detected or a new user-specified policy must be considered for the IoT network's

behavior.

The inspection, detection and mitigation mechanisms that have been described do not deeply differ from the common practices that have been already employed in SDN-aware networks for years. Therefore, these mechanisms do not represent the central novelty of this paper by themselves. Instead, a main novelty is how different and separated operations for security orchestration in SDN-aware IoT networks have been combined into a single optimal and provably correct technique. The traditional SDN techniques for detection and mitigation are thus joined with an innovative automatic, provably correct and optimized reconfiguration mechanism based on policy refinement, so that the resulting methodology can manage all the different orchestration phases for SDN switches.

4. Formal model for SDN switch configuration

A MaxSMT problem is a *Constraint Satisfaction Problem* (CSP). Differently from SAT problems, that consist of checking the satisfiability of a set of Boolean constraints (i.e., constraints involving only Boolean variables), a MaxSMT problem accepts constraints involving variables of different types (e.g., integers, strings, bit-arrays), so simplifying the formulation of problems involving such data types. Moreover, MaxSMT is characterized by two kinds of constraints. Hard constraints represent requirements that must be always fulfilled to get a correct solution; if at least a hard constraint cannot be satisfied, then a solution does not exist. The soft constraints, instead, do not necessarily require to be satisfied; since each soft constraint is given a weight, the goal of a MaxSMT solver is to maximize the sum of the weights assigned to the satisfied soft constraints.

The proposed methodology is based on solving a MaxSMT problem, which is formulated with two main objectives. The first one is the allocation of the minimum number of virtual SDN switches that are needed to enforce a set of network security policies, in the logical topology of the IoT network, so as to limit the number of corresponding *Virtual Machines* (VMs) that will be deployed in the physical infrastructure. The second objective is the automatic configuration of the minimum number of filtering rules in each allocated SDN switch. On one side, this leads to the minimization of resource consumption, since redundant rules are not enforced on the switches. On the other side, the performance of the filtering operations is maximized, because each SDN switch analyzes a smaller set of rules each time the next hop of a received packet must be decided.

The problem formulation as MaxSMT provides formal correctness-by-construction and optimization of the solution. Formal assurance is achieved by modeling the SDN filtering rules as open variables, so that their values are established by the problem solver respecting the constraints that express input requirements. This approach avoids the application of a-posteriori formal verification techniques and improves efficiency. Optimization is instead reached introducing soft constraints with weights balanced depending on the priority of the objectives.

As already mentioned, formal techniques for automatic configuration of firewalls already exist ([13, 14, 9]). Specifically, MaxSMT-based automatic configuration of firewalls has already been proposed in [9]. As this approach was proved successful for firewalls in NFV environments, it is adopted here as a starting point to develop a similar methodology for SDN-aware IoT networks. In order to do so, we need to adapt it to the specific features of these systems. In an IoT environment, the number of devices (e.g., sensors) which could connect to the network is potentially very high. This feature has several implications. One is that a limit on the maximum number of rules that can be configured in each virtual switch has to be set, depending on the resources (RAM, disk and CPU) available in its VM. By simply adopting the formulation of [9], which does not include this constraint, solutions that respect the security policies but are practically unfeasible could be established. Another implication is that, because

of the large number of devices, in IoT systems it is common to use IPv6 addresses, which were not considered in [9]. Finally, an additional requirement to be considered in these systems is to block the packets that are required to be denied as close as possible to their source. If this requirement is ignored, we could find solutions characterized by poor bandwidth management, because useless packets are forwarded to part of the system.

The constraints of the MaxSMT problem and the related definitions are presented in the reminder of this section. For the sake of clearness, they are divided into four groups, each one representing a different set of model elements. Nevertheless, all of them must be included in the MaxSMT formulation to get the correct and optimized solution. In greater detail, the hard constraints modeling the network topology are presented in Subsection 4.1, whereas the hard constraints modeling the network security policies in Subsection 4.2. Then, the soft constraints related to the optimal allocation of SDN switches are discussed in Subsection 4.3, and those related to the optimal generation of their rule sets in Subsection 4.4. Additionally, each subsection focuses on the specific aspects, concerning the management of SDN-aware IoT networks, that are addressed in this paper and that are built upon the approach from which the current proposal stems [9].

4.1. Allocation Graph model

The logical topology where the virtual SDN switches can be allocated is called *Allocation Graph* (AG). An example is graphically shown in Figure 2, where a legend explains the network component that each icon represents. In particular, the circles denoted with the a letter are called *Allocation Places* (APs). They are the logical possible positions where a switch can be allocated.

Following the same approach used in [9], an AG is formally represented as a directed graph:

$$G_A = (N_A, L_A) \tag{1}$$

where N_A is the vertex set and L_A is the set of directed links

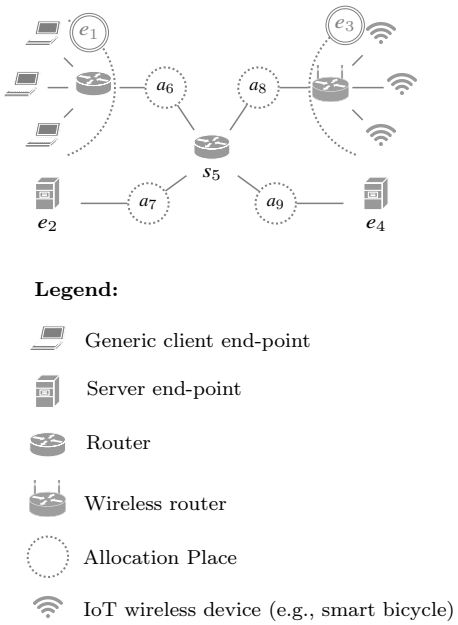


Figure 2: Allocation Graph example

(note that in the graphical example depicted in this paper each link without direction represents the adjacency of two directed links in opposite directions). In greater detail, the vertex set is composed by three non-overlapping subsets:

$$N_A = E_A \cup S_A \cup A_A \tag{2}$$

E_A is the set of end points, i.e., network nodes which can be the source or the destination of network packets. As in the example depicted in Figure 2, each $e \in E_A$ can be a single network node (e.g., the SDN controller e_2 and the server e_4) or a subnetwork (e.g., the office e_1 or the IoT network e_3). This formalization enables the formal verification of reachability or isolation for communications opened by devices belonging to the same group $e \in E_A$. The ability of representing groups of hosts as single nodes is fundamental in order to keep the size of the model limited for IoT environments, usually characterized by a huge number of possible IoT devices.

S_A is the set of service functions (e.g., s_5). They provide functionalities like routing, address translation, load balancing, caching. Finally, A_A is the set of the APs.

Each $n \in N_A$ is assigned with a single IP address, an IP address range or a set of IP addresses. The addresses can be IPv4 for traditional network elements, or IPv6 for

IoT devices (e.g., sensors connected to a wireless gateway). The addrIP4(n) function, when applied to node n, returns the set of IPv4 addresses assigned to n, while addrIP6(n) returns its IPv6 addresses. Finally, addrMac(n) returns the MAC address of n. Instead, each $a \in A_A$ is associated with the characteristics of the corresponding VM implementing the virtual switch that can be potentially allocated on that AP, formalized as a set of 3 elements:

$$V_a = \{r_a, m_a, c_a\} \tag{3}$$

where r_a represents the RAM (in GB) assigned to the corresponding VM, m_a the storage (in GB), and c_a the CPU usage (in MHz). This part of the model enables the expression of constraints (explained in Subsection 4.4), related to the maximum number of rules that can be installed and fulfilling stricter requirements in terms of RAM, storage and CPU usage, as it is typical of IoT networks.

Additionally, for each $n \in N_A$, a formal model of its forwarding behavior is defined. It is worth underlining that modeling the forwarding behavior is sufficient for the network functions in the AG, since it is the only relavant aspect for the definition of the SDN switch placement and configuration. In particular, only the *possibility* of the forwarding operations is considered, rather than representing the full decision algorithm. Given these considerations, the formal models of the forwarding behaviors of the network functions are based on two predicates, that represent the possibility that each packet is received or forwarded by any node. These predicates are defined for each $n_i, n_j \in N_A$ and $x \in X$, where X is the set of packet classes, identified by the possible combinations of true/false values for the predicates used in the formalization of the network security policies (see Subsection 4.2), i.e., all packet classes that are of interest for enforcing the satisfiability of the network security policies: (i) $recv(n_i, n_j, x)$ which is true if node n_j can receive a packet x from node n_i; (ii) $send(n_i, n_j, x)$ which is true if node n_i can send a packet x to node n_j.

4.2. Network security policy model

Network security policies are modeled in medium-level language and are formalized as

$$p = (C, \text{type}) \tag{4}$$

where C is the condition set, which is composed of predicates establishing which packets match the policy:

$$C = (\text{MacSrc, MacDst, IP4Src, IP4Dst, IP6Src, IP6Dst, pSrc, pDst, ICMPType, MPLSLabel, ...}) \tag{5}$$

Each one of these predicates expresses a condition on one of the packet fields that can be used in the rules of SDN switches, and the packets that match the rule are all those that satisfy all predicates in C. For example, the MacSrc/MacDst predicates express conditions on the source/ destination MAC address, IP4Src/ IP4Dst/IP6Src/IP6Dst on the source/ destination IPv4/ IPv6 address, pSrc/pDst on the source/destination port, tProto on the transport-level protocol, ICMPType on the type of ICMP message, and MPLSLabel on the number an MPLS label may have[1]. Each predicate can be used to specify not only a single admitted value for the corresponding field, but also a range or set of values (e.g., the IP4Src predicate may specify that any packet having an IPv4 address in 123.51.2.0/24 satisfies that condition). The special wildcard symbol $*$ stands for the predicate that is true for all values of the field, so applying the policy to packets having any possible value of that field (i.e., when that field is not of interest for the enforcement of that specific policy). In this model, predicates related to the traffic source (e.g., MacSrc, IP4Src, pSrc) refer to the characteristics of the packets generated by the source, whereas predicates related to the traffic destination (e.g., MacDst, IP4Dst, pDst) refer to the characteristics of the packets received by the destination.

For a pair of predicates c_1 and c_2 expressing a condition on the same packet field, the $\subseteq$ operator is employed

[1]The list of predicates that have been reported is not exhaustive. As the model of this medium-level policy language is fairly general, predicates related to any other fields that may be used in SDN switches can be included in the C set, without any impact on the formulation and resolution of the MaxSMT problem.

to state that a predicate expresses a condition that is included in the other, i.e., $c_1 \subseteq c_2$ means $c_2 \implies c_1$. For example, considering a policy $p \in P$ and a packet class $x \in X$, x.IP4Src $\subseteq$ p.IP4Src means p.IP4Src $\implies$ x.IP4Src.

Instead, *type* is the policy type, which can be isolation or reachability. In the former case, the packets satisfying the policy conditions C must be blocked before they reach their destination. In the latter, the packets satisfying C must be allowed to reach their destination.

In addition to a set of policies, a guideline has to be specified, in order to decide how to handle the cases not explicitly covered by the policy rules. We adopt the same guidelines already considered in [9]: 1) whitelisting, which means blocking all the packets for which no explicit policy is defined; 2) blacklisting, which means allowing all the packets for which no explicit policy is defined. In addition to them, a new guideline specifically designed for SDN-aware IoT networks is introduced: 3) bandwidth-oriented, which means deciding the behavior not explicitly defined by the policies so as to block packets as close as possible to the source. This profile is motivated by the consideration that, in IoT networks, the high number of connected devices may have a serious impact on network bandwidth.

Given a policy $p \in P$, all the pairs of end points $e_i, e_j \in E_A$ related to p, i.e., such that constraint (6) is satisfied, are identified.

$$\begin{aligned} &\text{addrMac}(e_i) \subseteq p.\text{MacSrc} \wedge \text{addrIP4}(e_i) \subseteq p.\text{IP4Src} \wedge \\ &\text{addrIP6}(e_i) \subseteq p.\text{IP6Src} \wedge \text{addrMac}(e_j) \subseteq p.\text{MacDst} \wedge \\ &\text{addrIP4}(e_j) \subseteq p.\text{IP4Dst} \wedge \text{addrIP6}(e_j) \subseteq p.\text{IP6Dst} \end{aligned} \tag{6}$$

Then, the set of all the possible paths between any pair of such identified end points is extracted from the AG. A function π is introduced to represent the mapping from a policy to its set of paths.

$$\pi\colon P \to \mathcal{P}((N_A)^*) \tag{7}$$

Consequently, each path returned by this function for a policy p is a list of nodes of N_A, where the first node is the nearest to the source of the policy, while the last one is the nearest to its destination. This function will be used later to support the bandwidth-oriented guideline and also to take decisions about the cardinality of the rule sets that must be automatically generated for the SDN switches.

For each policy $p \in P$, it is necessary to impose some hard constraints, to guarantee that the solution of the MaxSMT problem satisfies the policy. Of course, such constraints depend not only on the policy, but also on the allocation scheme and configuration of the SDN switches in the AG. These hard constraints are the same as those presented in [9]. They are consequently not reported here since no significant change has been brought in.

4.3. SDN switch allocation model

The first optimization objective is the minimization of the allocated SDN switches. The "allocates" predicate is used to represent on which AP an SDN switch is allocated:

$$\text{allocates} : A_A \to \mathbb{B} \tag{8}$$

where $\mathbb{B}$ is the set of Boolean values, i.e., $\mathbb{B} = \{\text{true}, \text{false}\}$.

The soft constraint (9) is thus defined for each $a \in A_A$ to require that, when possible, a virtual switch is not allocated on each $a \in A_A$. In (9), the Soft(c, w) notation is used to represent a soft clause: c represents the constraint, while w is the assigned weight. It is also worth underlining the negative sign on the weight of the soft clause: due to that sign, the solver will try not to satisfy this constraint, compatibly with the fulfillment of all the hard constraints.

$$\text{Soft}(\text{allocates}(a), -c_a) \tag{9}$$

If the user requires a bandwidth-oriented profile for the enforcement of the security policies in an SDN-aware IoT network, the weight c_a is different for each $a \in A_A$ and it is computed so as to require that each SDN switch is allocated (if needed) as nearest as possible to the source of the packets which it must block. For this purpose, let us introduce the "index" function:

$$\text{index} : N_A \times (N_A)^* \to \mathbb{N} \tag{10}$$

Given a node and a list of nodes as input (e.g., a path between a pair of end points related to a policy, as beforehand described), it retrieves the numerical position of the

node, assuming that the first node in the list is in position 1. If the node is absent from the list, the function conventionally returns 0. For example, if the index function receives a_8 and $[a_1, s_5, a_8, s_{11}]$ as input, the output is 3.

Having introduced this element, the weight c_a is computed as the sum of the values produced as output by the index function, when it is applied on the AP $a \in A_A$ and on the paths (i.e., list of nodes) extracted for any network security policy. This is computed as:

$$c_a = \sum_{p \in P} \left(\sum_{l \in \pi(p)} \text{index}(a, l) \right) \tag{11}$$

4.4. SDN switch configuration model

The second optimization objective is the minimization of the filtering rules in each allocated SDN switch. The soft constraints for this objective depend on other open variables, whose value will be established by the solver, representing the rules to be configured in each allocated SDN switch. For this purpose, for each $a \in A_A$, a $P_a \subseteq P$ set is defined as the smallest set that satisfies (12).

$$\forall p \in P.\ ((\exists\, l \in \pi(p).\ (a \in l)) \implies p \in P_a) \tag{12}$$

For each $p \in P_a$, a corresponding *placeholder rule* is created. The set of all the placeholder rules of $a \in A_A$ is identified as R_a and its cardinality is the same as the P_a cardinality, i.e., $|R_a| = |P_a|$. Each $r \in R_a$ represents a possible rule that the switch needs to enforce some requirements. It is formally modeled as:

$$r = (C, act) \tag{13}$$

C is the condition set, act is the rule action. C is modeled as the C set of a policy p. However, in this case each field is modeled with open variables, so that their effective values are established by the MaxSMT solver at run time. act is instead set to allow the matching packets, since in SDN the default behavior is whitelisting.

The "configures" predicate is introduced to let the solver decide if each placeholder rule must be configured to enforce some security policies or if it must not be deployed on the switch:

$$\text{configures} : R_a \to \mathbb{B} \tag{14}$$

This predicate returns true if the input placeholder rule has not been configured by the MaxSMT solver, i.e., the input filtering rule is not useful in the optimal solution. Note that a single rule could be actually sufficient to enforce multiple policies. For instance, if the user required isolation from each host in the network 20.2.12.0/24, then the solver will configure a single rule to enforce all those policies. This optimization objective is formally modeled with a soft constraint, shown in (15), defined for each $r \in R_a$.

$$\text{Soft}(\text{configures}(r), -c_r) \tag{15}$$

However, the allocation objective has a higher priority than the configuration objective, since the setup of a new virtual switch is more time and memory consuming than the enforcement of additional rules. This priority order is established by imposing the following relationship for the weights of the soft constraints:

$$\sum_{r \in R_a} (c_r) < c_a \tag{16}$$

The hard constraints modeling the forwarding behavior of the SDN switches are the same used for firewalls in [9]. However, an important novelty has been introduced when extending the methodology for IoT SDN-aware networks. In each virtual switch a maximum number of placeholder rules can be effectively configured, depending on the characteristics of the corresponding VM in terms of RAM, disk memory and CPU usage. In the IoT context, this constraint is more important, considering the high number of devises composing the networks. The maximum number of configurable rules for each $a \in A_A$ is computed as shown in (17), where K_1, K_2 and K_3 are constants whose value can be decided by the user.

$$\text{maxR} = K_1 \cdot r_a + K_2 \cdot m_a + K_3 \cdot c_a \tag{17}$$

This maximum number of configurable rules must be put in relationship with the configured predicate by means of a hard constraint. However, if maxR is an integer value, configured is a Boolean function. For this reason, the "bool_to_int" function is introduced.

$$\text{bool\_to\_int} : \mathbb{B} \to \{0, 1\} \tag{18}$$

The behavior of this function is shown in (19): it returns 0 if the input value is false, 1 otherwise.

$$\text{bool\_to\_int}(b) = \begin{cases} 0 & \text{if } b = \text{false} \\ 1 & \text{if } b = \text{true} \end{cases} \tag{19}$$

Having introduced this notation, the required constraint is finally represented in (20). Note that the outcome of the configures predicate counts in the summation only when the placeholder rule on which it is applied is effectively configured.

$$\sum_{r \in R_a} \text{bool\_to\_int}(\text{configures}(r)) \leq \text{maxR} \tag{20}$$

5. Implementation and validation

This section first describes the implementation of the proposed methodology in Subsection 5.1. Then, it illustrates the results of the validation on a real use case, with scalability tests in Subsection 5.2, and it compares these results with the performance of state-of-the-art solutions in Subsection 5.3.

5.1. Implementation

The configuration engine has been implemented as a Java framework. It offers REST-based APIs for the interaction with either the network administrators or the other components of the architecture. This module exploits z3 [47], an efficient MaxSMT solver provided by Microsoft Research. The z3 release 4.8.5 has been adopted for the implementation.

Policy translator, plugins and Security Orchestrator have been developed in python 3.7. Both provide REST HTTP APIs for policy translation and policy enforcement respectively. For the SDN Controller, ONOS version 4.2.6 has been deployed and all networks, machines, and IoT devices have been virtualized by using mininet-wifi version 2.3. Specifically, we have virtualized a router which connects 11 Open Virtual Switches (OVS), one per department office. Whereas desktop and server machines use IPv4 and they are connected to the switches directly through Ethernet, IoT devices use IPv6 and they are connected through 6LowPAN Gateways. Since we follow a *deny by default* approach, all traffic must be denied from the beginning so ONOS traffic forwarding application is disabled. Only ARP and NDP messages are sent to the controller in order to allow the SDN controller to discover and manage the network topology. Regarding the policies enforcement, ONOS provides a REST HTTP Northbound API which enables different ways to apply network configuration, like per-flows or per-intents. While with per-flow configuration the whole path must be provided in order to allow connectivity between two (or more) devices, with intents only source/s and destination/s are required by ONOS, which automatically calculates the path and configures all the involved SDN switches. In this regard, different plugins have been developed for policy translation, in order to translate security policies both into ONOS flows and ONOS intents. However the current implementation applies the configurations per-flow since the configuration engine takes into account not only the available network information provided by ONOS but also the current security policies status on the system as well as its constraints.

From the monitoring and reaction parts, several Snort version 2.9.7.0 GRE instances have been deployed both in wired and wireless network segments. These IDS instances have been configured in different ways depending on the network allocation, e.g., Snort instances in the wireless segments have been configured in order to alert the system if they detect any traffic but CoAP messages.

Table 1: Results for the use case

	Use case in Figure 3
Number of reachability policies	18
Number of isolation policies	18
Number of allocated switches	11
Number of hard constraints	630
Number of soft constraints	384
Computation time (s)	2.7

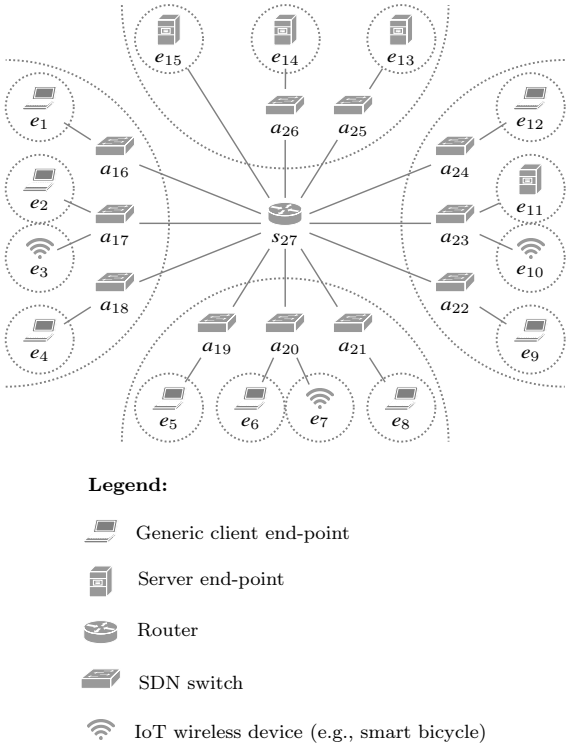


Figure 3: Use case exploited for validation

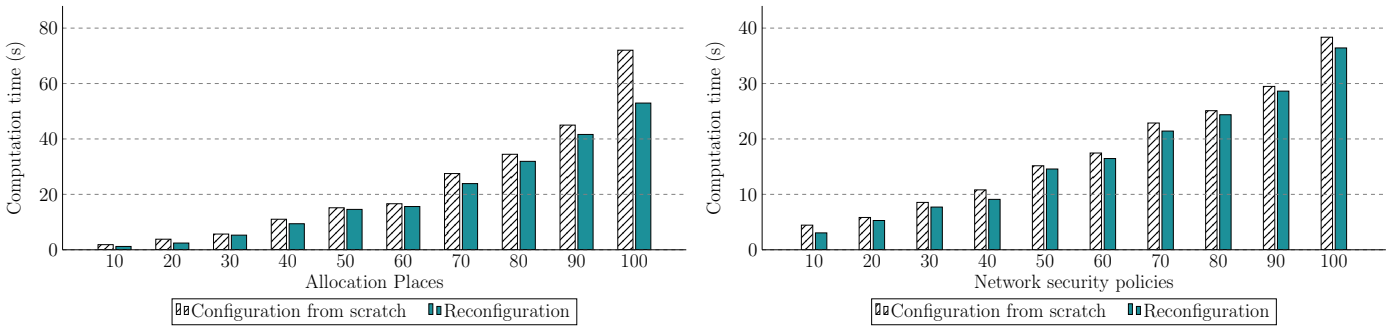
5.2. Validation

The performance and correctness of the framework have been studied in an experimental set-up similar to a real scenario, where the aim is to automatically compute the virtual topology shown in Figure 3. This figure includes a legend explaining the role of each icon used in the picture. It represents the expected outcome of the framework, i.e., the logical topology of a virtual network where SDN switches have been allocated. The physical network is instead only composed of a few general-purpose servers, on which all the virtual functions of the topology shown in Figure 3 are deployed. Additionally, Table 1 reports the main characteristics of the use case with respect to the policies and the IoT network topology, the number of hard and soft constraints of the MaxSMT problem, the number of allocated switches after the computation of the solution, and the required computation time. Among these characteristics, the numbers of the hard and soft constraints are the ones that derive from the formalization of the different model elements in the MaxSMT problem (i.e., network topology, allocation places, network function models, network security policies) and of the optimization goals. These numbers, therefore, specify how many constraints are exactly created by the tool and passed on to the solver. As it can be seen from the table, the optimal allocation scheme and configuration of the SDN switches is computed in less than 3 seconds. This result demonstrates the feasibility of the methodology for topologies similar to the network used in the example.

Then, the scalability of the configuration engine has been validated by means of some tests, carried out on a machine characterized by an Intel i7-6700 CPU running at 3.40 GHz and 32GB of RAM. Figure 4 presents the results of these tests. The two metrics of interest have been the number of APs and the number of network security policies. For each test case characterized by a given number of APs and policies, 50 runs have been executed and the average value is graphically represented. The reason is that the computation time of the z3 engine varies in relation to the integer numbers that are present in the IP addresses. The topologies where the tests have been conducted are synthetically generated extensions of the network in Figure 3. Besides, for the results shown in Figure 4a the number of NSRs is fixed to 50, whereas for the results of Figure 4b the number of APs is fixed to 50.

For each test case, two different scenarios have been considered. On one side, the computation time that is required for a complete configuration from scratch of the allocation scheme and configuration of all the SDN switches have been evaluated. On the other side, the reconfiguration of a network as reaction to attacks have been investigated. For each test case regarding this second scenario, in 70% of the APs SDN switches with an existing configuration are already installed.

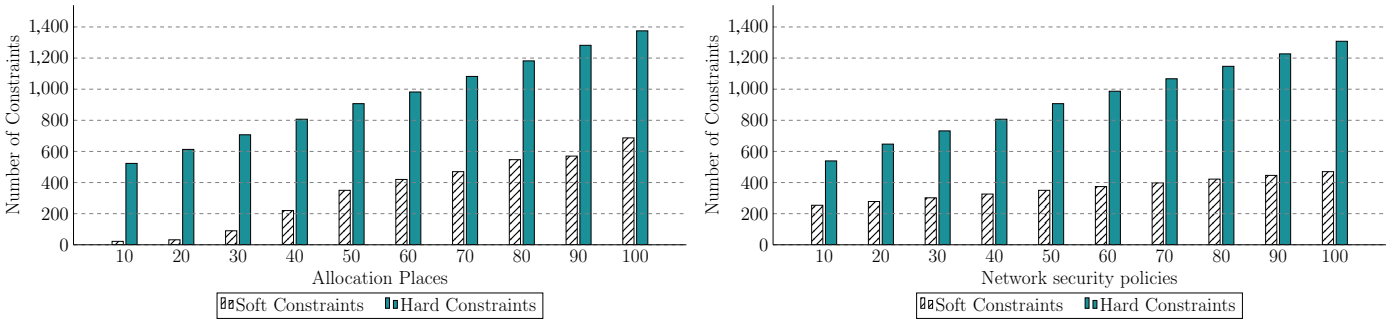
A first consideration that can be extracted from the charts is that the computation time does not increase exponentially with respect to the two analyzed metrics. Even though MaxSMT is theoretically an NP-complete problem,



(a) Scalability for increasing number of APs

(b) Scalability for increasing number of policies

Figure 4: Scalability evaluation of the SDN switch allocation and configuration



(a) Number of constraints for increasing number of APs

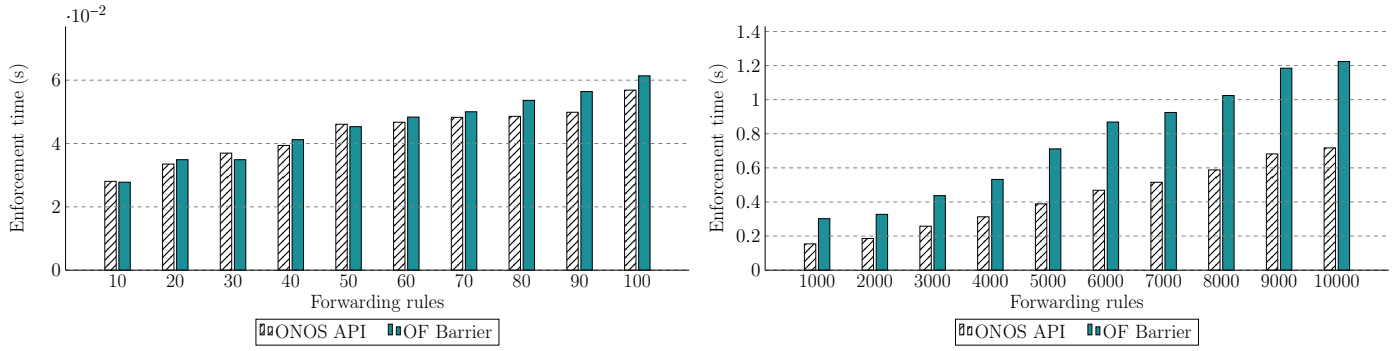
(b) Number of constraints for increasing number of policies

Figure 5: Number of constraints for the MaxSMT formulation of the SDN switch allocation and configuration problem

many instances of the problem can be solved in polynomial time [48]. Moreover, z3, like other state-of-the-art solvers, uses internal strategies and heuristics to solve MaxSMT instances more quickly. Other key factors in enabling these results are pre-pruning strategies, developed to reduce the solution space without losing correctness and optimality, and a correct tuning of the optimization parameters (e.g., the weights of the soft constraints). We made also some experiments to study how the computation time depends on the number of hard and soft constraints defining the MaxSMT problem. In fact, the higher these numbers are, the more complex the MaxSMT problem is, and, therefore, the higher the computation time is. With the aim to confirm this statement, Figure 5a and Figure 5b plot the number of soft and hard constraints characterizing the MaxSMT problems formulated for the scenario where a configuration from scratch is performed under the same conditions respectively explained for Figure 4a and Figure 4b.

A second consideration is that the computation time required for the scenario where some APs are already filled with configured SDN switches is slightly inferior to the scenario where a configuration from scratch is required. This result is explained by the fact that a lower number of soft constraints characterizes the corresponding MaxSMT problem, since they are formulated when the engine must take optimized decisions. At the same time, the number of hard constraints increases: the rule set of the already allocated switches must be in fact modeled and it can impact on the satisfiability of the network security policies.

Some scalability limitations are still present. In particular, the number of network security policies might be expected to be bigger than 100 in networks having thousands of IoT devices connected at the same time. As such, it represents a parameter for which the scalability of the current implementation should be improved. However, as shown in the comparison with state-of-the-art automatic solutions, the current results already represent an impor-



(a) Scalability for increasing number of filtering rules

(b) Scalability for a high-range number of filtering rules

Figure 6: Scalability evaluation of the SDN enforcement using ONOS

tant step in improving the state-of-the-art.

An additional experimental validation has been carried out to evaluate the performance of the SDN rules enforcement, when different amounts of forwarding rules were enforced. First, we repeated 10 different enforcements with different settings which contain from 10 to 100 IPv6 forwarding rules. These forwarding rules were previously translated into ONOS IPv6 SDN rules and they were enforced by using the ONOS Northbound REST API. Figure 6 shows two different times for different numbers of rules. On the one hand, it shows the time taken since the orchestrator requests the SDN rules enforcement until it receives the API result (ONOS API). On the other hand, it shows the time taken since the orchestrator requests the SDN rules enforcement until the SDN switch notifies the enforcement (OF Barrier). For the latter, we captured the Openflow traffic and we compared the time elapsed between the send of the enforcement request by the orchestrator to the SDN Controller, and the receive of the Openflow OFPT _BARRIER _REPLY message by the SDN Controller. This kind of reply is an answer for the OFPT _BARRIER _REQUEST message which ensures that the switch will complete the processing of all previous messages before processing any messages sent after the Barrier Request message. Each experiment was repeated 30 times. It is interesting to highlight that, when ONOS receives the enforcement request, it updates the internal data structures as well as it requests in parallel the new rules enforcement, but it does not wait for the answer from the SDN switches before sending the enforcement reply to the orchestrator. In fact, Figure 6a shows that the enforcement results are quite similar until we enforce more than 80 rules. The reason is that in this range the network configuration finishes almost at the same time ONOS sends the enforcement reply. In order to verify this behavior we performed the same kind of experiment but increasing significantly the number of rules. Figure 6b shows the results of repeating 10 different enforcements with different settings which contain from 1000 to 10000 IPv6 forwarding rules. According to the trend of the results, they follow a linear progression, taking into account that the more SDN rules the more enforcement time and the more the difference between the ONOS internal management and the SDN final enforcement.

In addition to the evaluation of the performance and scalability of the approach, the actual achievement of an optimal and correct solution, provided by the MaxSMT solver, has been checked for the use case in the experimental setup. With respect to the optimization feature, the optimal distribution of the policies on the allocated switches has been checked on the experimental test bed, by enumerating all the possible solutions and checking that the enforced one is the best one, i.e the solution that minimizes the number of allocated switches and configured rules. Instead, with respect to the formal correctness of the solution, some tests were carried out in the same test

bed, to check that the automatically computed configuration for the SDN switches of the validation scenario really fulfills the required network security policies.

Specifically, after security policies have been enforced, an automatic process verifies that the SDN network behavior works as expected, according to the policy requirements. The process analyzes each enforced security policy, mapping policy information like capability, action and condition with available testing tools for different network levels. This means that different network tests will be executed depending on the parameters of the network policy. For instance, on one hand a *traffic divert* capability with a *forward* action between two IP addresses (L3) executes a simple testing tool for verifying the reachability between them. Namely, it executes automatically different ICMP Echo Request and Response operations and verifies that there is connectivity between the IP addresses defined in the security policy. On the other hand, a *filtering L3* capability with a *drop* action also uses the same tool, but in this case it verifies that there is no connectivity between the devices based on the ICMP error status. When the security policy contains information of L4 network layer, (e.g., transport protocol, source or destination ports), other kinds of tools like nc and nmap have been used to check the desired behaviour in the same way as in the previous case, but now also targeting the specified ports and transport protocol (TCP/UDP).

5.3. Comparison with state-of-the-art approaches

The performance results achieved for the allocation and configuration process not only represent a significant improvement with respect to traditional manual approaches, but they also improve the state-of-the-art of the automated methodologies for allocation and configuration of security functions. In fact, the performance of such methodologies is not significantly better than the one of our approach, despite they do not provide all the features of our solution. It is also important to note that they do not address SDN switches for IoT-based networks specifically.

Although none of the state-of-the-art solutions addresses SDN switches for IoT-based networks specifically, in this section we compare our approach to the existing state-of-the-art tools.

The papers [10], [11], and [13] propose techniques that can be applied only in simplified scenarios like centralized traditional firewalls, and they do not provide any information about their performance, so preventing performance comparisons.

Other approaches [12, 19, 18, 14] that are reported to scale up to small networks, perform worse than our approach, considering the more limited tasks they complete. For example, the approach described in [19] requires around 80s to compute the allocation scheme of 20 NSFs, while our framework takes less than 80s to allocate **and** configure up to 100 NSFs. Similar considerations apply to the approach proposed in [18]. It can configure 20 NSFs in 4s, which is the same time taken by our approach for the same number of NSFs. However, it does not solve the allocation problem, but only the configuration one, and the solution is nor optimized neither formally verified. Moreover, the approach is not reported to scale to bigger networks. Other methodologies that are more specific to SDN-aware networks, such as CloudWatcher [20], Procera [21], Fresco [22] and OpenSec [23]) have not been tested on networks as large as the ones we have used for our tests.

Finally, we compare the methodology for automatic allocation and configuration of SDN switches presented here with the approach from which it stems [9]. The approach in [9] can manage a network of around 100 APs taking a time that is slightly inferior to, but in the same magnitude order of, the one required by our technique. However, our method is characterized by more complex formal models and it integrates additional optimization objectives (e.g., attack impact and bandwidth optimization). These targets have been chosen because they are the most relevant ones for the IoT context, and as a trade-off between per-

formance and optimization. In fact, if other optimization targets were added, our methodology would have worse performance, because it should analyze a bigger solution space, and the benefits of having those objectives would not be substantial, because they are less significant for the IoT context.

6. Conclusions

This paper presented a novel methodology for a policy-based security enforcement in industrial SDN-aware IoT networks. This methodology can automatically establish the optimal and formally correct allocation scheme and configuration of virtual SDN switches, with the aim to satisfy security policies defined by an administrator or derived from the detection of an attack. In the proposed approach, automation, optimization, and a-priori formal verification are leveraged through the formulation of a MaxSMT problem. The implementation of this automated mechanism has been experimentally validated in a real use case and with some scalability tests.

Some future works are planned in order to overcome the existing limitations of the proposed approach. First of all, alternative formulations of the hard and soft constraints for the MaxSMT problem, in particular in relation to the network security policies, will be investigated, with the aim to further improve the performance scalability of this automated methodology. In doing so, the introduction of some heuristics will be evaluated. Moreover, another future work is the definition of a formal model addressing the variety and complexity of all the different kinds of IoT end-points which can be connected to an SDN-aware IoT network, since currently the models for the end-points have not distinguishing features. Then, alternative approaches could be evaluated to better balance the trade-off between performance and new optimization targets. Finally, the formal model for the policy-based security enforcement could be extended to support other complex virtualized network security functions, in addition to virtual firewalls and SDN switches.

ACKNOWLEDGMENT

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

Bibliography

[1] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks (2010) 2787 – 2805.

[2] C. Kolias, G. Kambourakis, A. Stavrou, J. M. Voas, Ddos in the iot: Mirai and other botnets, IEEE Computer (2017) 80–84.

[3] K. Sha, W. Wei, T. A. Yang, Z. Wang, W. Shi, On security challenges and open issues in internet of things, Future Gener. Comput. Syst. 83 (2018) 326–337.

[4] F. A. Alaba, M. Othman, I. A. T. Hashem, F. Alotaibi, Internet of things security: A survey, J. Netw. Comput. Appl. 88 (2017) 10–28.

[5] W. John, G. Marchetto, F. Németh, P. Sköldström, R. Steinert, C. Meirosu, I. Papafili, K. Pentikousis, Service provider devops, IEEE Communications Magazine 55 (1) (2017) 204–211.

[6] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, C. Zunino, Leveraging SDN to improve security in industrial networks, in: IEEE 13th International Workshop on Factory Communication Systems, WFCS 2017, Trondheim, Norway, May 31 - June 2, 2017, IEEE, 2017, pp. 1–7.

[7] K. Hofer-Schmitz, B. Stojanovic, Towards formal verification of iot protocols: A review, Comput. Networks 174 (2020) 107233.

[8] A. Molina Zarca, J. B. Bernabe, R. Trapero, D. Rivera, J. Villalobos, A. Skarmeta, S. Bianchi, A. Zafeiropoulos, P. Gouvas, Security management architecture for nfv/sdn-aware iot systems, IEEE Internet Things J. 6 (5) (2019) 8005–8020.

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

[10] Y. Bartal, A. Mayer, K. Nissim, A. Wool, Firmato: A novel firewall management toolkit, ACM Trans. Comput. Syst. (2004) 381–420.

[11] P. Verma, A. Prakash, FACE: A firewall analysis and configuration engine, in: IEEE/IPSJ SAINT, 2005, pp. 74–81.

[12] J. D. Guttman, Filtering postures: Local enforcement for global policies, in: IEEE Symposium on Security and Privacy, 1997, pp. 120–129.

[13] J. Govaerts, A. K. Bandara, K. Curran, A formal logic approach to firewall packet filtering analysis and generation, Artif. Intell. Rev. (2008) 223–248.

[14] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, The mathematical foundations for mapping policies to network devices (technical report), arXiv.

[15] N. B. Youssef, A. Bouhoula, A fully automatic approach for fixing firewall misconfigurations, in: 11th IEEE International Conference on Computer and Information Technology, 2011, pp. 461–466.

[16] K. Adi, L. Hamza, L. Pene, Automatic security policy enforcement in computer systems, Computers & Security (2018) 156–171.

[17] A. Gember-Jacobson, A. Akella, R. Mahajan, H. H. Liu, Automatically repairing network control planes using an abstract representation, in: Proc. Symp. on Operating Systems Principles, 2017, pp. 359–373.

[18] C. Basile, F. Valenza, A. Lioy, D. R. López, A. P. Perales, Adding support for automatic enforcement of security policies in NFV networks, IEEE/ACM Trans. Netw. 27 (2) (2019) 707–720.

[19] M. A. Rahman, E. Al-Shaer, Automated synthesis of distributed network access controls: A formal framework with refinement, IEEE Trans. Parallel Distrib. Syst. 28 (2) (2017) 416–430.

[20] S. Shin, G. Gu, Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?), in: IEEE International Conference on Network Protocols, 2012, pp. 1–6.

[21] A. Voellmy, H. Kim, N. Feamster, Procera: a language for high-level reactive network control, in: Proc. HotSDN@SIGCOMM, 2012, pp. 43–48.

[22] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, M. Tyson, FRESCO: modular composable security services for software-defined networks, in: 20th Annual Network and Distributed System Security Symposium, 2013.

[23] A. Lara, B. Ramamurthy, Opensec: Policy-based security using software-defined networking, IEEE Trans. Netw. Service Manag. (2016) 30–42.

[24] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking control of the enterprise, in: Proc. SIGCOMM, 2007, pp. 1–12.

[25] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: A network programming language, in: Proc. ACM SIGPLAN International Conference on Functional Programming, 2011, pp. 279–291.

[26] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, Policycop: An autonomic qos policy enforcement framework for software defined networks, in: IEEE SDN for Future Networks and Services, 2013, pp. 1–7.

[27] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, L. Vanbever, An industrial-scale software defined internet exchange point, in: K. J. Argyraki, R. Isaacs (Eds.), 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016, USENIX Association, 2016, pp. 1–14.

[28] H. Kumar, H. H. Gharakheili, C. Russell, V. Sivaraman, Enhancing security management at software-defined exchange points, IEEE Trans. Network and Service Management 16 (4) (2019) 1479–1492.

[29] D. Bhamare, R. Jain, M. Samaka, A. Erbad, A survey on service function chaining, J. Netw. Comput. Appl. 75 (2016) 138–155.

[30] A. S. Sendi, Y. Jarraya, M. Pourzandi, M. Cheriet, Efficient provisioning of security service function chaining using network security defense patterns, IEEE Trans. Serv. Comput. 12 (4) (2019) 534–549.

[31] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: Static checking for networks, in: Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 113–126.

[32] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, S. T. King, Debugging the data plane with anteater, in: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, ACM, New York, NY, USA, 2011, pp. 290–301.

[33] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, G. Varghese, Checking beliefs in dynamic networks, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), USENIX Association, Oakland, CA, 2015, pp. 499–512.

[34] R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu, Symnet: Scalable symbolic execution for modern networks, in: Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 314–327.

[35] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P. B. Godfrey, Veriflow: Verifying network-wide invariants in real time, in: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013, USENIX Association, 2013, pp. 15–27.

[36] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, R. Sisto, Formal verification of virtual network function graphs in an sp-devops context, in: Service Oriented and Cloud Computing - 4th European Conference, ESOCC 2015, Taormina, Italy, September 15-17, 2015. Proceedings, Vol. 9306 of Lecture

Notes in Computer Science, Springer, 2015, pp. 253–262.

[37] A. Panda, O. Lahav, K. J. Argyraki, M. Sagiv, S. Shenker, Verifying reachability in networks with mutable datapaths, in: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017, USENIX Association, 2017, pp. 699–718.

[38] I. Farris, T. Taleb, Y. Khettab, J. Song, A survey on emerging sdn and nfv security mechanisms for iot systems, IEEE Commun. Surveys Tuts. (2019) 812–837.

[39] D. B. Rawat, S. R. Reddy, Software defined networking architecture, security and energy efficiency: A survey, IEEE Commun. Surveys Tuts. (2017) 325–346.

[40] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, Bohatei: Flexible and elastic ddos defense, in: J. Jung, T. Holz (Eds.), 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015, USENIX Association, 2015, pp. 817–832.

[41] R. Braga, E. de Souza Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: The 35th Annual IEEE Conference on Local Computer Networks, LCN 2010, 10-14 October 2010, Denver, Colorado, USA, Proceedings, IEEE Computer Society, 2010, pp. 408–415.

[42] A. M. Zarca, J. B. Bernabe, A. Skarmeta, J. M. A. Calero, Virtual iot honeynets to mitigate cyberattacks in sdn/nfv-enabled iot networks, IEEE J. Sel. Areas Commun. (2020) 1–1.

[43] A. Hamza, H. H. Gharakheili, V. Sivaraman, Combining MUD policies with SDN for iot intrusion detection, in: Proceedings of the 2018 Workshop on IoT Security and Privacy, IoT S&P@SIGCOMM 2018, Budapest, Hungary, August 20, 2018, ACM, 2018, pp. 1–7.

[44] A. Hamza, H. H. Gharakheili, T. A. Benson, V. Sivaraman, Detecting volumetric attacks on lot devices via sdn-based monitoring of MUD activity, in: Proceedings of the 2019 ACM Symposium on SDN Research, SOSR 2019, San Jose, CA, USA, April 3-4, 2019, ACM, 2019, pp. 36–48.

[45] N. Garcia, T. Alcaniz, A. GonzÃąlez-Vidal, J. B. Bernabe, D. Rivera, A. Skarmeta, Distributed real-time slowdos attacks detection over encrypted traffic using artificial intelligence, Journal of Network and Computer Applications 173 (2021) 102871.

[46] B. Wehbi, E. Montes de Oca, M. Bourdelles, Events-based security monitoring using mmt tool, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 860–863.

[47] L. M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: TACAS, 2008, pp. 337–340.

[48] R. Robere, A. Kolokolova, V. Ganesh, The proof complexity of smt solvers, in: Computer Aided Verification, Springer International Publishing, 2018.