Introducing programmability and automation in the synthesis of virtual firewall rules

# Introducing programmability and automation in the synthesis of virtual firewall rules

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

***Abstract*—The rise of new forms of cyber-threats is mostly due to the extensive use of virtualization paradigms and the increasing adoption of automation in the software life-cycle. To address these challenges we propose an innovative framework that leverages the intrinsic programmability of the cloud and software-defined infrastructures to improve the effectiveness and efficiency of reaction mechanisms. In this paper, we present our contributions with a demonstrative use case in the context of Kubernetes. By means of this framework, developers of cybersecurity appliances will not have any more to care about how to react to events or to struggle to define any possible security tasks at design time. In addition, automatic firewall ruleset generation provided by our framework will mostly avoid human intervention, hence decreasing the time to carry out them and the likelihood of errors. We focus our discussions on technical challenges: definition of common actions at the policy level and their translation into configurations for the heterogeneous set of security functions by means of a use case.**

***Index Terms*—network functions virtualization, firewall, automatic programmability, cloud networking, formal methods**

## I. Introduction

The networking field is currently facing a deep revolution based on virtualization. In the decade which has just ended, innovative paradigms shook the traditional vision of networks as a mesh of heterogeneous functions providing different services. The first time when networking embraced virtualization is represented by the born of *Software-Defined Networking* (SDN) [1], [2]. The main pillars of this technology are the decoupling between data plane and control plane, centralization of all the control plane functions in a software module that is referred to as SDN controller, abstraction between the specificity of user applications and the generality of controller interfaces. More recently, *Network Functions Virtualization* (NFV) [3], [4] introduced the possibility to create network functions as software programs and to make them run as traditional virtual machines or containerized applications, supervised by a software *MANagement and Orchestration* (MANO) module [5]. Physical middleboxes have been thus progressively replaced by general-purpose servers where the programs implementing the network functions can run.

Among the contributions brought over, automatic (re)programmability of the network functions are nowadays becoming feasible, with respect to the traditional troubles coming from a manual function configuration [6]. On one side, a fundamental novelty provided by SDN has been the reactive generation and deployment of forwarding rules by the controller onto the data plane switches. Whenever a packet which does not match any switch rule is received, it is forwarded to the controller, so that it can take the best decision according to the internal logic and consequently generates rules for all the network switches which would have to manage packets with the same characteristics in the future. On the other side, if the network functions are implemented in the NFV fashion, MANO can automatically manage their life-cycle and deployment, so as to optimize either resource consumption for the underlying physical infrastructure or availability of the provided services.

Although many organizations are migrating virtual machine (VM)-based applications to containers, virtualization is still present in data centers and public clouds. We are also seeing new ways of integrating virtualization with containers and Kubernetes (K8s) to provide innovative solutions to new problems. In other words, virtual machines are also becoming part of the cloud-native architecture – this concept is called container-native virtualization. Kubernetes is an example of the fulfillment of SDN and NFV paradigms, which is an open-source system for automating deployment, scaling, and management of virtualized applications. It significantly simplifies the works of network and service administrators.

However, in an environment like Kubernetes, where multiple software processes run in parallel, the correct global management becomes more difficult than what traditionally was with single hardware devices. The increase of complexity, unfortunately, contributed to raising the number of cyberattacks, which became more variable by having the possibility to exploit new kinds of breaches. In particular, misconfiguration of network functions has become more critical, because more variable factors must be considered when enforcing a correct security defence against both external and internal attacks. This statement is confirmed by recent surveys, such as Verizon's most recent study [7]. In this report, the misconfigurations have been identified as the third most critical threat in cloud environments, that can lead to catastrophic breaches.

In light of these observations, the challenge we propose to face is to effectively exploit the benefits provided by the virtual networking paradigms, minimizing the impact of their beforehand illustrated drawbacks. With this aim, we designed a framework based on the innovative methodology presented in [8], based on *Maximum Satisfiability Modulo Theories* (MaxSMT), and we integrated it in the context of Kubernetes. The proposed approach automatically configures virtual

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

firewalls, where a consistent number of configuration errors are traditionally performed. Moreover, we will particularly describe how this methodology is effectively introduced in the framework architecture of ASTRID (AddreSsing ThReats for virtualIseD services), which is an EU H2020 Project [9].

The remainder of this paper is structured as follows. In Section II, the most related works are described, so that the main differences with respect to the methodology proposed in this paper are illustrated. In Section III, first, the general architecture of the ASTRID framework is presented. Then the focus will be shifted on the methodology for the automatic firewall configuration, present inside the Security Controller, the central component of the ASTRID framework which enforces security in cloud-based networks. In Section IV, additional details about the implementation will be provided, alongside with a validation based on the framework's application in a realistic scenario. Finally, Section V briefly concludes the paper and describes the planned future works.

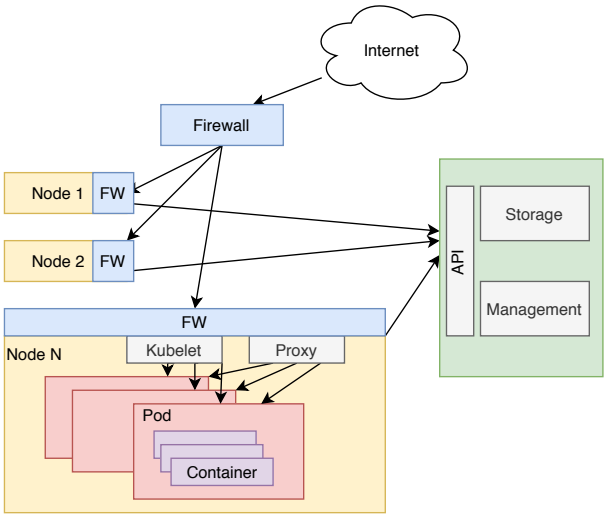


Fig. 1: Kubernetes architecture

## II. Related Work

The focus of this paper is centered on the automatic configuration of firewalling functions in Kubernetes framework. Therefore, we briefly introduce the main characteristics of the Kubernetes framework and then we report the main works related to the automatic firewalls configuration.

As shown in Fig. 1, a Kubernetes cluster is composed of multiple nodes, which can be virtual or physical. A Pod is a minimal management unit and can accommodate one or more containers. Each Pod is protected by a packet filter (i.e., FW in Fig 1). Pods are assigned with network addresses and are allocated to nodes. Containers inside a Pod share resources, such as volumes where they can write and read data. Clients contact the cluster through another firewall, which distributes requests to nodes according to load balancing rules. The proxy receives requests from this firewall and forwards them to Pods. Each node has a proxy installed. If a Pod is replicated, the proxy distributes the load among the replicas. The kubelet is a component that manages Pods, containers, images and other elements in the node. The kubelet forwards data on the monitoring of containers to the main node, which acts when necessary. In this framework, one of the main key points concerns the correct and consistent configuration of this graph of firewalls that protect the access to each container.

In literature, an automatic configuration of firewalls is a challenge where research has been partially carried out. However, most of the works describe either technique which can be only applied to traditional networks (e.g., with hardware firewalls), or mathematical models that do not have a correspondent implementation proving their feasibility and effectiveness. Moreover, only a limited subset of them enrich the computed configurations with optimality or formal correctness assurance [10].

The three papers which gave birth to this research trend have been [11], [12] and [13]. In particular, Firmato [11] represents a vital milestone, because it is the first approach based on policy refinement that is able to automatically synthesize a firewall configuration, by exploiting an entity-relationship model for the representation of the security policies and the network topology. Nevertheless, its limitations are evident:the most critical is that it has been validated on a single centralized firewall, instead of a distributed security architecture. The other two works ( [12], [13]) added the possibility to configure a distributed firewall as the main novelty. However, all these three works exclusively target traditional networks, and do not offer either optimality or formal verification.

Formal mathematical models have been, instead, presented in [14] and [15], where formal methodologies are used to automatically compute firewall configuration. However, in both cases these techniques work only in specific cases, not related to virtualized networks: [14] follows the syntax of IPChains and Cisco's PIX, whereas [15]'s technique has been validated only with SCADA-firewall configuration. Besides, optimization is overlooked in both these two works.

A recent work which, with respect to all the others, specifically targets NFV-based networks is [16], [17]. The proposed approach is the first step toward a security policy aware NFV management, with the introduction of a specific module, called Security Awareness Manager (SAM), into frameworks which provide NFV MANO, such as OpenMANO. This module performed a complete refinement of high-level, user-specified network security policies into the low-level configuration of virtual network functions, using optimization models defined for each function type. There are limitations in this work, though: the achieved results are not formally verified and little information is provided about how firewall policies are managed, since this paper provides a comprehensive approach for multiple security function types. Anyhow, it shows how, despite its drawbacks, virtualization is altogether characterized by features which can be positively and efficiently exploited in the automatic programmability of next-generation computer networks

Finally, the proposed work integrates the automatic configu-

ration approach, presented in [8], into Kubernetes. Specifically, the solution in [8] adopts a formal approach based on the MaxSMT problem, which provides formal assurance about the correctness of the solution. More details will be provided in the next sections.

## III. Approach

This section presents the design of the ASTRID framework and presents a generic workflow to illustrate the main functionalities. Next, our proposed approach is presented as a Security Controller component that resides in the ASTRID framework.

### A. ASTRID framework

The term orchestration is commonly being used in the IT field. In the NFV and microservice system, there is Service Orchestration for Service, and in the Cloud system, there is Cloud Orchestration for cloud resource description. With the development and maturity of container technology, more and more enterprises and individuals choose to containerize traditional applications or directly develop container-based cloud-native applications and then run applications on the container platform. Faced with a complex container operating environment, the needs for container orchestration have raised. In general, container orchestration is responsible for the life-cycle scheduling of containers, and it improves container usage by managing container clusters. There are currently three major industry giants such as Kubernetes, Docker Swarm, and Apache Mesos. They belong to the category of DevOps infrastructure management tools and are called "container orchestration engines".

But when developers enter the world of orchestration, one thing that needs special attention is security. Various blogs, videos, books, and tutorials teach developers how to use these solutions, but only a few mention the need to add security controls to protect applications in the cluster. Moreover, if the underlying infrastructure of the cloud is unreliable (or configured in a vulnerable manner), for instance, there is no way to guarantee the security of a Kubernetes cluster built on this foundation.

The main goal of the AddreSsing ThReats for virtualIseD services (ASTRID) project is to address these technological gaps in the scope of cloud infrastructures. The project proposes a novel cyber-security framework to provide situational awareness for cloud applications and NFV services. The overall workflow of the framework is presented in Fig. 2. According to the workflow, the ASTRID framework allows software and service developers to provide a description of the service request, which is enriched with security policies by security provider entity. The Security Orchestrator component of the framework is in charge of reaction, creation, delivering end-to-end services.

The scope and the contribution of this work are associated with the Initialization and Reaction phase provided in Fig. 2. We develop the Security Controller that is in charge of this phase in the workflow. It is one of the most valuable parts of the run-time subsystem, conceived to automate as much as possible the behaviour of the security functions, in the control plane. In the next subsection, we describe the component in detail.

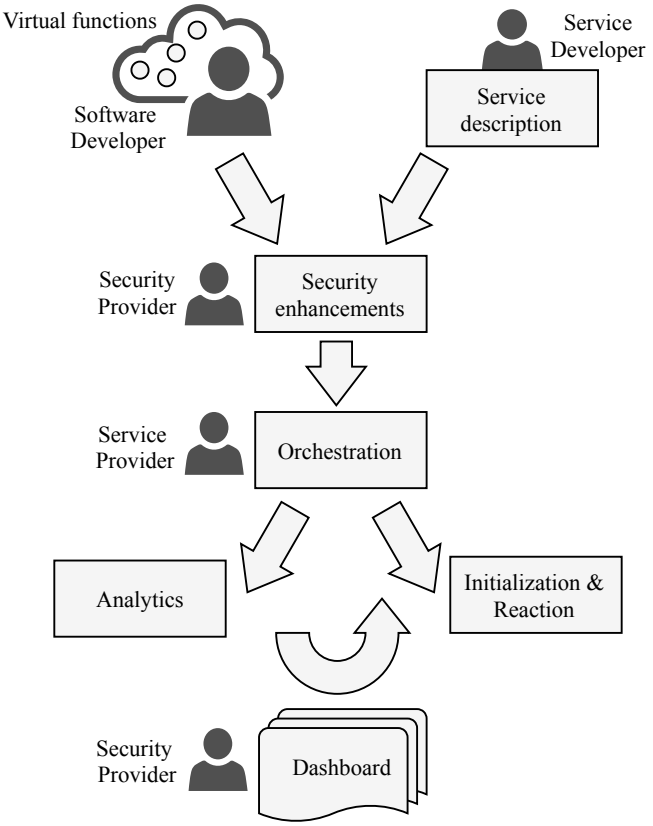


Fig. 2: Overall workflow of the ASTRID framework [18]

### B. Security Controller

The Security Controller has been developed on the basis of the methodology presented in [8]. It incorporates programmability and automation in the synthesis of virtual firewall rules from a user-provided security policy. With this respect, the Security Controller works in close coordination along with the service orchestrator. The service orchestrator is in charge of providing a description of the service graph as well as the infrastructure information. The infrastructure information includes the actual number of launched virtual network functions and parameters assigned after the enforcement process such as IP and port addresses. After receiving the required data from the orchestrator, the controller performs an automatic translation from the high-level policy to low-level configuration parameters for firewall network functions. This process of automatic configuration is formally proven to meet these security policies as a part of this analysis. The security controller formulates the problem of the automatic configuration of firewall rule tables as the Maximum Satisfiability Modulo Theories (MaxSMT) problem. It is a basic constraint optimization problem that we use to provide two main features: i) high assurance in the correctness of the

computed solutions, thanks to the intrinsic formal correctness-by-construction paradigm; ii) optimality of the solution, by minimizing the number of automatically generated firewall rules, with the purpose to improve the filtering operations.

To this day, optimization problems are modeled by Integer Programming (IP) languages. At the same time, most of them are NP-hard classes, and large-scale integer problems are difficult to solve. Moreover, none of the variations of the IP formulation are able to model the problem of automatic firewall configuration having in mind the verification of end-to-end reachability. This is due to the less expressive power of the approaches compared to the Constraint Satisfaction Problem (CSP) representations. An instance encoding of CSP, MaxSMT in our case, is defined by a set of variables, a set of possible values (or domains) for each variable, and a set of soft and hard constraints, each constraint involving one or more variables. A MaxSMT solver determines for a given instance whether it is possible to assign a value to each variable from its respective domain in order to satisfy all hard constraints and an optimal number of soft constraints simultaneously.

The Security Controller translates the input service graph into a MaxSMT instance by means of a set of First-Order Logic formulas. In a nutshell, these formulas will be converted to boolean variables in Conjunctive Normal Form, eventually. In addition to the topological definition of the service graph, each network function of the service graph will be translated into the abstract model according to the guidelines given by Verigraph [19]. This allows us to provide a higher level of assurance that the automatically generated configuration parameters of the firewall will satisfy the security policies in the presence of complex network functions. The level of abstraction of these models covers all the forwarding behavior of the network and their configuration parameters that are already defined. Instead, we model the firewall network function by introducing soft constraints over variables, which then will be decided to satisfy or not by the MaxSMT solver. These variables represent the IP and port addresses to be autoconfigured in order to satisfy the end-to-end policies. Initially, these variables are set to false, which means that a firewall does not contain any rule. If the policy requires that the firewall must block the traffic, it must falsify the soft constraint in favor of satisfying the policy requirement. Hence, the policy requirement is modeled as a hard constraint, which means it must be always satisfied. In this way, the solver tries to minimize the falsifying constraints in the formula and satisfy the hard constraints. This is the definition of the optimization problem we pursue to solve.

In order to represent the reachability policies by means of hard constraints, we introduce the concept of packet flows between endpoints. The first constraint we assert is that the network function model defined in the service graph must forward a packet flow if it receives a packet flow. This constraint must be true under the functional behavior of the network device. For instance, this is true if a firewall network function does not contain any rule that drops packets. The second constraint states that the packet flow sent from a source node must be received by the destination node. Other constraints include the forwarding path definitions and static configuration parameters of network functions.

This concludes the fact that IP formulation of the same problem would be limited to a set of constraints over binary, integer, or real variables. Instead, the approach presented in this paper allows us to model the problem and using very expressive constraints. These constraints include configuration parameters of network functions, forwarding behavior of the service graph, and complex security policies, in addition to the automatic configuration constraints of the problem. Therefore, existing IP algorithms are not comparable to our algorithm for that class of problems. In the next section, we demonstrate our approach by means of a representative scenario.

## IV. Use case scenario

This section presents our framework in greater detail with a practical use case and motivates our design decisions. For the sake of simplicity, we focus our attention on a specific component of the ASTRID framework, the Security Controller, and emphasize the fact that the interaction between other components is performed by means of a REST API. We expose a number of resource endpoints to the Security Orchestrator, which will use to deliver the service graph and infrastructure information and to retrieve the automatically generated firewall rules. We underline the fact this methodology can be extended to more general scenarios than the ASTRID framework. In fact, the Security Controller is a standalone web service application, which makes it possible to be easily incorporated into existing cloud platforms and orchestrators.

We consider the scenario where an administrator predefines the logical service graph presented in Fig. 3a and feeds it to the dashboard of the ASTRID framework. This service graph represents a realistic scenario where the *nginx* web server is made public to the Internet and functions as a reverse proxy to fetch dynamic data from multiple instances of *nodejs* and *apache* servers. In this case, both servers can acquire data from a *mysql* database. As we can see from the figure, reachability policies required by the use case are rather obvious (i.e., highlighted with arrows). Instead, the isolation property required by the service graph is not evident. For instance, all the communications, which are not highlighted with arrows must be isolated. Considering the fact that each service in the graph is associated with a firewall, firewalls are preconfigured with *deny-all* rules, in order to satisfy this policy. This ensures that all other interactions within the service graph must be isolated, except the ones predefined by the user (i.e., arrows).

A Service Orchestrator of the ASTRID framework is in charge of deploying the service graph onto the infrastructure and generating the enriched service graph shown in Fig. 3b. During this enforcement phase, all the services are assigned with corresponding IP addresses and ports where these services can be reached. It is important to highlight that the multiple instances of the services are deployed in separate Pods and each will have its own IP addresses. In this scenario,

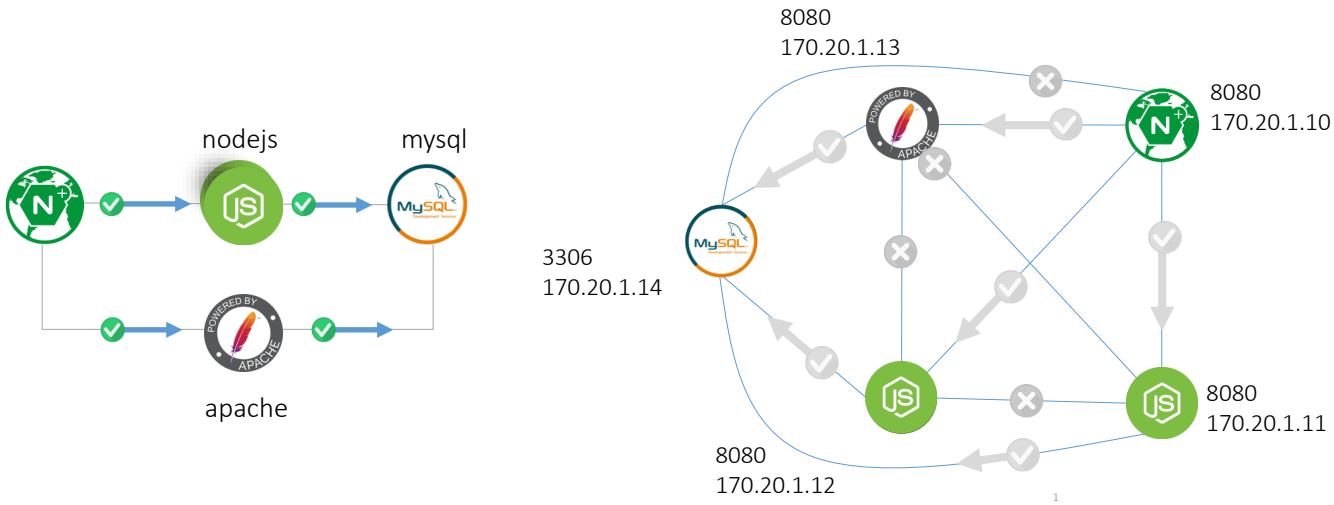


Fig. 3: a) a logical service graph b) enriched service graph after the deployment

the user specified to have two instances of the *nodejs* server to handle the load. To illustrate the complexity introduced by this simple use case, we included all the links connecting each service in the infrastructure in Fig. 3b. Taking into account the *deny-all* rules of each firewall of the service, we can assure that there is no reachability between the Pods in this phase. Although, we have specified the user policy that needs to be satisfied by means of the arrows in the figure. As an example, *apache* server needs to be configured to allow traffic from itself to a *mysql* database and allow communication from *nodejs*. However, it needs to be isolated from each instance of the *nodejs* servers.

Without the Security Controller, an administrator of the infrastructure must manually configure each firewall. This process of manual configuration of each firewall is error-prone and time-consuming. This scenario motivates the use of the Security Controller presented in this paper, in order to automatically generate firewall configurations for each service and provide formal assurance that the network policy defined by the user is satisfied. To obtain the low-level configuration of each firewall component, the Security Controller accepts as an input the infrastructure information and logical service graph as described in Section III. Infrastructure information contains the IP and port addresses of each service that is shown in Fig. 3b. This information is required to define the firewall rules, which allows to block specific packet flows involving specific Pods. In the next step, the Security Controller automatically generates an output with a low-level configuration of each firewall component. As an example, we present the partial output format and the actual configuration parameters generated by the Security Controller in Listing 1. In this prototype evaluation experiment, we use a machine with 3.40 GHz Intel i7-6700 CPU and 32GB of RAM. The average time needed for the overall procedure is less than a second. We need to emphasize the fact that for most service requests, the time required to schedule VMs to be several orders of magnitude larger than the this computation time.

Listing 1 shows the configuration parameters generated for the firewall component of the *mysql* service. It includes all the neighbors of the firewall in the infrastructure network and firewall rule entries. According to the output, we need to configure the firewall with 3 entries.

Listing 1: Automatic Configuration Output for mysql

```
<node name="172.20.1.34" functional_type="FIREWALL">
<neighbour name="172.20.1.14"/>
<neighbour name="172.20.1.30"/>
<neighbour name="172.20.1.31"/>
<neighbour name="172.20.1.32"/>
<neighbour name="172.20.1.33"/>
<configuration name="mysql" description="172.20.1.14">
  <firewall defaultAction="DENY">
    <elements>
      <action>ALLOW</action>
      <source>172.20.1.13</source>
      <destination>172.20.1.14</destination>
      <protocol>ANY</protocol>
      <src_port>*</src_port>
      <dst_port>*</dst_port>
    </elements>
    <elements>
      <action>ALLOW</action>
      <source>172.20.1.11</source>
      <destination>172.20.1.14</destination>
      <protocol>ANY</protocol>
      <src_port>*</src_port>
      <dst_port>*</dst_port>
    </elements>
    <elements>
      <action>ALLOW</action>
      <source>172.20.1.12</source>
      <destination>172.20.1.14</destination>
      <protocol>ANY</protocol>
      <src_port>*</src_port>
      <dst_port>*</dst_port>
    </elements>
  </firewall>
</configuration>
</node>
```

The first rule states that the packets arriving from the Pod with an IP address 172.20.1.13 need to be allowed. The rest

of the rules are associated with the two instances of the *nodejs* server of the service graph. Due to the default action set by the firewall in line 8, Listing 1, all the other packets arriving from the network is dropped. For instance, intruders from the Internet are not able to access the *mysql* database in accordance with these rules. This, in fact, ensures the satisfiability of the initial service graph policy defined by the user. Eventually, the output file generated by the Security Controller is sent back to the Context Broker, which is in charge of translating the low-level configuration of each firewall into a vendor-specific format of the firewall.

An important feature of the Security Controller is in the possibility to have firewalls without any configuration as in the use case or with partial configuration, giving to the tool itself the task of providing the missing configurations as an output. The tool generates the configuration with the objective of satisfying all the requested policies while minimizing the number of generated rules in order to achieve it. In the case of partial configuration, a firewall may include static rule entries that will not be changed in the output. This is useful when the service graph is updated according to an event when a Pod is terminated or a new Pod has been created to handle the overhead to the service. In this scenario, in order not to recompute the configuration parameters of all the other services, we can provide their rules in a static manner, meaning that they can be left unchanged. This process not only generates a set of configuration parameters but also provides an optimal set of rules to satisfy the user policy. Optimality is achieved by minimizing the number of rules inside each firewall to improve the performance of the virtual network functions.

## V. Conclusion and Future Works

In this paper, we illustrated the benefits which the introduction of automatic programmability would bring for the synthesis of firewall rule sets in virtual networks, in the respect of NFV and cloud infrastructures with special emphasis on Kubernetes. In particular, the role of the presented automated methodology in the ASTRID framework architecture has been described, with an emphasis on the contributions provided to the Security Controller. We formulate the problem of automatic firewall configuration as a MaxSMT instance and solve it to provide reachability assurance between endpoints.

As possible future works, we are currently planning to introduce programmability for other kinds of network security functions, such as intrusion detection systems and security devices for channel protection (e.g., VPN gateways). Moreover, we plan to provide automatic configuration settings in the presence of minor changes in the initial service graph without solving the problem from scratch. As the initial results show promises in smaller instances, we plan to evaluate the model in larger scale scenarios.

## Acknowledgment

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

## References

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

[2] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, and C. Zunino, "Leveraging sdn to improve security in industrial networks," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, May 2017, pp. 1–7.

[3] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] W. John, G. Marchetto, F. Nemeth, P. Skoldstrom, R. Steinert, C. Meirosu, I. Papafili, and K. Pentikousis, "Service provider devops," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 204–211, 2017.

[5] European Telecommunications Standards Institute, "Network functions virtualisation (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

[6] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

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

[8] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020*, apr 2020, to appear.

[9] "ASTRID EU H2020 Project," https://www.astrid-project.eu/index.php, accessed: 2020-02-12.

[10] P. B. Copet, G. Marchetto, R. Sisto, and L. Costa, "Formal verification of lte-umts and lte–lte handover procedures," *Computer Standards and Interfaces*, vol. 50, pp. 92 – 106, 2017.

[11] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, Nov. 2004.

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

[13] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, 1997, pp. 120–129.

[14] J. Govaerts, A. K. Bandara, and K. Curran, "A formal logic approach to firewall packet filtering analysis and generation," *Artif. Intell. Rev.*, vol. 29, no. 3-4, pp. 223–248, 2008.

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

[16] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 707–720, 2019.

[17] C. Basile, D. Canavese, C. Pitscheider, A. Lioy, and F. Valenza, "Assessing network authorization policies via reachability analysis," *Computer and Electrical Engineering*, vol. 64, no. C, pp. 110–131, Nov. 2017.

[18] ASTRID an EU H2020 Project , "D1.2 - astrid architecture," January 2020. [Online]. Available: https://private.astrid-project.eu/Documents/PublicDownload/31

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