VNGuard: An NFV/SDN Combination Framework for Provisioning and Managing Virtual Firewalls

Juan Deng†, Hongxin Hu†, Hongda Li†, Zhizhong Pan†,
Kuang-Ching Wang†, Gail-Joon Ahn‡, Jun Bi♮, Younghee Park♯
†Clemson University ‡Arizona State University ♮Tsinghua University ♯San Jose State University

***Abstract*—Network Function Virtualization (NFV) together with cloud technology enables users to request creating flexible virtual networks (VNs). Users also have specific security requirements to protect their VNs. Especially, due to changeable network perimeters, constant VM migrations, and user-centric security needs, VNs require new security features that traditional firewalls fail to provide, because traditional firewalls rely greatly on restricted network topology and entry points to provide effective security protection. To address this challenge, we propose VNGuard, a framework for effective provision and management of *virtual firewalls* to safeguard VNs, leveraging features provided by NFV and Software Defined Networking (SDN). VNGuard defines a high-level firewall policy language, finds optimal virtual firewall placement, and adapts virtual firewalls to VN changes. To demonstrate the feasibility of our approach, we have implemented core components of VNGuard on top of ClickOS. Our experimental results demonstrate the effectiveness and efficiency of virtual firewalls built on VNGuard.**

I. Introduction

Network function virtualization (NFV) was recently proposed to decouple network functions (NFs), such as firewall, load balancer, NAT, and web proxy, from dedicated hardware and implement them as pure software instances on industrial standard high-volume servers, networking and storage. NFV calls for a carrier-grade cloud platform, leveraging de-facto industry cloud management and standards [1]. NFV together with cloud platforms allows a user to build flexible *virtual networks* (VNs).

Users also have security requirements to protect their VNs, more specifically, the applications or services running on the VNs. Cloud providers, such as Google Cloud Platform [2], have already allowed users to request protecting their services. Unfortunately, traditional firewall technique lacks the flexibility and adaptivity to meet the new security needs arising with VNs.[1] First of all, a traditional firewall depends on restricted network topology and entry points to provide effective security protection. However, due to the dispersion of virtual machines (VMs) in VNs across racks and data centers, and VM migrations for the purpose of resource management and optimisation, the perimeter of VNs become blurred and fluid [1]. Second, a user may have specific security requirements for VNs, which require dedicated small firewalls tailored to accommodate their requirements, while traditional firewalls are usually large proprietary appliances, and provisioning customized appliances for each user is costly and impractical. Third, VNs change frequently and the changes take effect fast. Changes may come from either users or the service provider. For example, a user may issue VN changes because s/he has needs to change or update the services running within the VN, or the NFV provider decides to migrate VMs for the purpose of resource management and optimization. This calls for dynamic and fast firewall reconfiguration and replacement to adapt to VN changes, which cannot be easily achieved by traditional firewalls.

In NFV, a firewall, serving as a vital security network function, is implemented as a software instance (a.k.a virtualized firewall or *virtual firewall*). A virtual firewall offers the necessary flexibility and mobility to effectively protect VNs. With virtual firewalls, users are enabled to flexibly define and manage customized firewalls to protect their own VNs. Also, virtual firewalls break the dependency on fixed network topology and entry points, since they can be placed on any VM with great flexibility, as long as the VM can provide the required resources. In addition, virtual firewalls, being software instances, can be reconfigured and moved easily and fast to adapt to VN changes.

The flexibility and mobility of virtual firewalls must rely on the support of dynamic, fast and reliable traffic steering. As virtual firewalls are no longer required to be placed at fixed network entry points, underlying flexible traffic steering must be in place to pass network traffic to virtual firewalls. The migration of virtual firewalls also need a robust traffic steering support to ensure no security holes during and after virtual firewall migration. Software Defined Networking (SDN), recognized as complementary technology to NFV [4]–[8], is able to provide the traffic engineering needed by virtual firewalls.

To this end, we propose *VNGuard*, a comprehensive framework for *effectively provisioning and managing virtual firewalls*, leveraging both NFV and SDN techniques. *VNGuard* enables users to readily define their security policies for the VNs without concerning low-level VN deployment. *VNGuard* can also automatically translate high-level security policies into low-level firewall rules, and find an optimal virtual firewall deployment solution while still respecting resource and performance constraints. In addition, *VNGuard* enables virtual firewall reconfiguration and replacement to adapt to VN changes and ensures that the replacement remains optimal. To the best of our knowledge, *VNGuard* is the first solution for

[1]In traditional networks, firewall functions are generally implemented on vendor proprietary appliances or *middleboxes*. However, middleboxes usually lack a general programming interface, and their flexibility and versatility are also very limited [3].

978-1-4673-6884-1/15/$31.00 ©2015 IEEE

systematic design and management of virtual firewalls based on NFV and SDN. We summarize our contributions as follows:

- We introduce a high-level service-oriented policy language in *VNGuard* that allows users to specify high-level security policies without considering low-level concrete VN deployment.
- We propose an approach based on Integer Programming to find an optimal virtual firewall placement, which fulfils resource constraints.
- We provide a mechanism for dynamic virtual firewall adaption to protect the VNs, which can be frequently changed and migrated.
- We implement core components of VNGuard on top of ClickOS [23]. Our experiment results show that our approach can efficiently and effectively provision and manage virtual firewalls.

The rest of the paper is organized as follows. Section II presents an overview of *VNGuard* framework. Section III describes our high-level service-oriented policy language. Section IV presents our solution for finding optimal virtual firewall placement. Section V introduces our approach for dynamic virtual firewall adaption. Section VI presents our *VNGuard* implementation and the evaluation of *VNGuard* efficiency. We overview related work in Section VII. Section VIII discusses several important issues. Conclusion and future work are addressed in Section IX.

II. *VNGuard* Framework

In designing *VNGuard*, we achieve the following design goals:

- **High-Level Service-Oriented Policy Language.** A user has specific security requirements to protect the services running on the VNs, but s/he may lack the expertise and experience to be able to write low-level firewall rules that are associated with VN deployment. Besides, VN deployment information may not be readily available to users, because the service provider who provisions VNs may intentionally hide the deployment information to protect the infrastructure. Therefore, a new firewall policy language is desirable and should be *high-level, service-oriented*, and *user-centric* for a user to easily specify his/her security requirements.
- **Optimal Virtual Firewall Placement.** A user's security requirements need to be translated to firewall rules that are associated with VN deployment. There may exist thousands of firewall rules for a user. Placing all rules in one virtual firewall instance may not be applicable, considering that rule overload causes firewall performance degradation, as suggested in Figure 5. Obviously, more resources can help achieve higher performance. However, resources in cloud are valuable, and must be utilized optimally. Given a set of firewall rules, finding a placement solution that meets the optimization goal while respecting to both performance and resource constraints is a bin packing problem, which is typically combinatorial NP-hard.

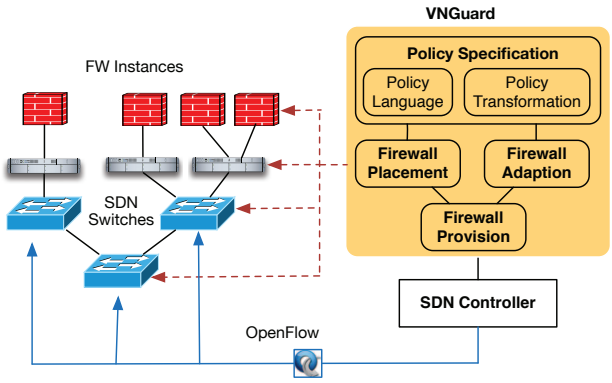

Fig. 1. VNGuard framework.

- **Dynamical Virtual Firewall Adaption.** VNs are more flexible than traditional networks and face changes more frequently. VN changes require virtual firewalls to be properly reconfigured and replaced, so that VNs receive the same protection during and after changes. A mechanism should be design to dynamically adapt virtual firewalls to VN changes.

We design *VNGuard* as shown in Figure 1, which is an SDN application [6]. *VNGuard* has four major components: *Policy Specification*, *Firewall Placement*, *Firewall Adaption*, and *Firewall Provision*. *Policy Specification* defines a high-level service-oriented *Policy Language* for a user to specify security policies. *Policy Transformation* transfers high-level security policies to low-level firewall rules. These rules are then processed by *Firewall Placement* to find an optimal virtual firewall placement. *Firewall Adaption* is responsible for virtual firewall adaption with respect to VN changes and migrations. *Firewall Provision* is responsible for provisioning virtual firewalls and also maintains a database on the deployment information of virtual firewalls.

III. High-Level Service-Oriented Policy Language

Our purpose to design a *high-level*, *service-oriented*, and *user-centric* policy specification language is to enable users to easily specify their security policies without the knowledge of low-level VN deployment information.

Our policy language defines several basic components as shown in Table I. $V = \{v_1, v_2, ...\}$ is a set of VNs that a user has. Each VN hosts multiple services. A *service* is defined as a combination of VNs that host the service, and the protocol and the port number that the service uses. For example, an *http* service running in VN v_1 is described as

$$\langle v_1, 80, tcp \rangle$$

Using such a definition, a user can define different security policies for the same service in different VNs. For example, a user may allow incoming SSH connections to one VN, but ban them to another.

An *object* is a communication end point. It is defined as a combination of hosts and ports. Hosts are uniquely represented

TABLE I
BASIC COMPONENTS OF OUR HIGH-LEVEL POLICY LANGUAGE

$P = \{p_1, p_2, ...\}$	P is a finite set of ports that services use.
$T = \{tcp, udp, icmp, ...\}$	T is a finite set of protocols that services use.
$V = \{v_1, v_2, ...\}$	V is a finite set of VNs that a user has.
$s ::= \langle \bar{V}, p, t \rangle$	A service s has three members with $\bar{V}$ being a subset of a user's VNs that run s (that is, $\bar{V} \subseteq V$), $p \in P$ being the port number used by s, and $t \in T$ being the protocol used by s. s can call its members by $s.1 = \bar{V}, s.2 = p$ and, $s.3 = t$.
$Z = \{s_1, s_2, ..\}$, and for all $s_{(i)}.1 = v$	A security zone Z is a finite set of services $\{s_1, s_2, ...\}$ in the same virtual network.
$a ::= \langle accept \mid deny \mid delete \rangle$	An action a is one of $accept$, $deny$ and $delete$. $delete$ is used in the virtual firewall adaption.
$o ::= \langle IP \mid domain\_name \mid VN\_name, p \rangle$	An object o has two members with the first being either an IP address, a domain name, or the name of a VN, and the second element $p \in P$ being a port number.
$policy ::= \langle s, a, o \rangle$	A policy defines the access right of object o on service s. A policy can call its members by using $policy.1 = service, policy.2 = r$, and $policy.3 = o$. We also allow consecutive member calling. For example $policy.1.1 = \bar{V}, and policy.3.2 = p$.

by either IP addresses, a domain name, or the name of a VN. Hosts may exist inside or outside of a user's VNs. Hosts outside of a user's VNs are identified by either IP addresses or a domain name. Hosts inside a user's VNs are usually identified by either IP addresses or the name of a VN.

Our language can supports a wide card representation of policies. For example,

$$\langle *, 80, tcp \rangle$$

describes an $http$ service exiting in all virtual networks of a user.

A *policy* is formalized as

$$\langle s, a, o \rangle$$

that defines access right of object o on service s. For example,

$$\langle \langle v_1, 80, tcp \rangle, deny, \langle v_2, * \rangle \rangle$$

states that traffic from a virtual network v_2 is denied to access the *http* service in a virtual network v_1.

Our policy language can easily support the definitions of *global policies*, *group policies*, and *local policies* by simply manipulating member $\bar{V}$ in the service definition $\langle \bar{V}, p, t \rangle$. A global policy is defined as

$$\langle \langle *, p, t \rangle, a, o \rangle$$

that states that the $service$ defined by port p and protocol t in ALL of a user's VNs *deny/accept* traffic from object o. A group policy is defined as

$$\langle \langle \bar{V}, p, t \rangle, a, o \rangle \rangle \text{ where } (\bar{V} \subset V \ )$$

that states that the $service$ defined by port p and protocol t in the subset of a user's VNs, $\bar{V}$, *deny/accept* traffic from object o. Naturally, a local policy is defined as

$$\langle \langle v, p, t \rangle, a, o \rangle$$

that states that the $service$ defined by port p and protocol t in a user's VN, v, *deny/accept* traffic from object o.

With our policy language, users are able to write high-level policies easily without knowing low-level VN deployment. *Policy Transformation* (Figure 1) translates the high-level polices to low-level firewall rules that are associated with the low-level VN deployment. To do so, *Policy Translation* needs to retrieve VN deployment information, which is maintained by *Firewall Provision*. For example, for a policy

$$\langle \langle v_1, 80, tcp \rangle, deny, \langle v_2, * \rangle \rangle$$

Policy Transformation retrieves the deployment information of v_1 and v_1, and finds the IP address (10.10.1.2) of the VM in v_1 that hosts the $http$ service and IP addresses (130.127.24.*) of v_2, and generates a firewall rule as [2]

$$\langle 130.127.24.*, 10.10.1.2, 80, tcp, deny \rangle$$

Note that *Policy Transformation* in *VNGuard* can also detect and resolve rule conflicts by using the conflict detection and resolution mechanism introduced in our previous work [18].

IV. OPTIMAL VIRTUAL FIREWALL PLACEMENT

Figure 2 shows the process of virtual firewall placement in *VNGuard*. User-specified security policies are processed by *Policy Transformation* and transformed to low-level firewall rules. *Rule Database* keeps a copy of the rules for each user, which will be used later for virtual firewall adaption. The rules are then sent to *Firewall Placement*. Firewall placement subjects to resource constraints, which are stored in *Resource*

[2] A firewall rule is normally specified as a 6-tuple of ⟨*source address, source port, destination address, destination port, protocol, action*⟩.

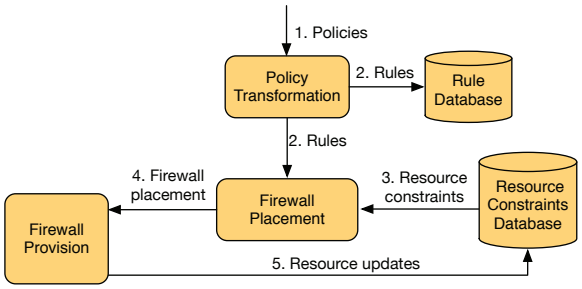

Fig. 2. Firewall Placement Digram.

Constraints Database. *Firewall Placement* retrieves the constraints and works out an optimal placement solution. The solution is sent to the *Firewall Provision* for creating and running virtual firewall instances. We consider two resource constraints on virtual firewall rule placement:

- NFV service provider limits the number of virtual firewall instances that a user can have.
- NFV service provider limits the resource that each virtual firewall instance can have. Hence, there is a cap on the number of rules that a virtual firewall instance can hold.

To achieve the optimization goal of virtual firewall placement, *Firewall Placement* in *VNGuard* uses an Integer Programming based solution to find the optimal placement.

Integer Programming Based Solution Integer Programming is a method for the optimization of a linear objective function, subjecting to linear constraints. We model the resource constraints as

- Let $F = \{f_1, ..., f_N\}$ be a set of virtual firewall instances that a user can have.
- Let $C = \{c_1, ..., c_N\}$ with c_i be the number of firewall rules that an instance f_i can hold.

These constraints are stored in *Resource Constraints Database*.

Let $R = \{r_1, r_2, ..., r_M\}$ be the set of rules to be placed. Each rule is to be placed on only one instance, because virtual firewall performance degrades as the number of rules loaded on it increases (Figure 5). Let $v_{ij} \in \{0, 1\}$ be an indicator of placing rule r_i on instance f_j. If $v_{ij} = 1$, it indicates that rule r_i is placed on instance f_j. Otherwise, the rule is placed on another instance.

Definition: A placement $V = \{v_{11}, ..., v_{MN}\}$ is feasible if it satisfies the following conditions:

$$\textit{condition(1): } \sum_i v_{ij} \leq c_j \textit{ for each } j$$
$$\textit{condition(2): } \sum_j v_{ij} = 1 \textit{ for each } i$$
$$\textit{condition(3): } v_{i,j} \in \{0, 1\} \textit{ for each } i, j$$

Condition (1) states that the number of firewall rules placed on each instance f_j must not exceed its capacity. Condition (2) guarantees that a firewall rule is placed on only one instance.

We denote the objective function that a service provider wants to optimize as $G(V)$, a function of placement V. Our Integer Programming formulization for virtual firewall placement problem is:

$$min\text{: } G(V)$$

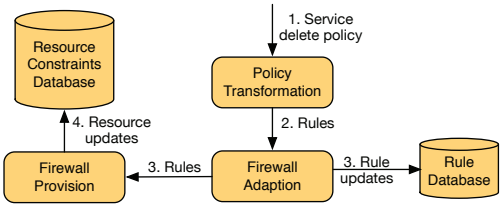

Fig. 3. Firewall Adaption to Service Deletion.

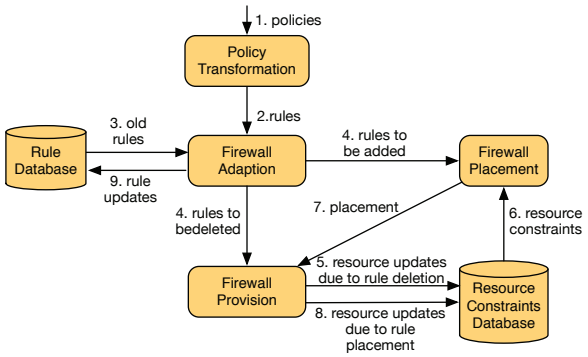

Fig. 4. Firewall Adaption to Policy Update.

$$s.t.\ condition(1), condition(2), condition(3)$$

Solving the above problem gives a feasible placement V. Integer Programming deals with linear objective functions and constraints. For the firewall placement formulization, the constraints are always linear, but the objective functions may be nonlinear. For example, if we consider the objective function to be the number of firewall instances, then $G(V) = \sum_j min(1, \sum_i v_{ij})$. In this case, non-linear Integer Programming [19] should be used.

Resource Updates *Firewall Provision* creates the firewall instances according to the optimal solution provided by *Firewall Placement*. Then it updates *Resource Constraints Database*. For the instances in F that have rules placed, decrease their corresponding c values by the number of rules placed on them. The updated c values reflect the number of new rules that can be placed on these instances in the future. The update of *Resource Constraints Database* is crucial for the virtual firewall adaption.

V. Dynamic Virtual Firewall Adaption

VNs are relatively easy to be changed. VN changes may be initiated by either a user or an NFV service provider. In this paper, we consider three types of VN changes.

- Type I: A user adds or deletes services in a VN.
- Type II: A user updates security policies for a VN.
- Type III: VM migration and scaling out/in.

The changes of a VN require reconfiguring and replacing virtual firewalls to ensure that the VN receives the same protection during and after changes. We describe below how *VNGuard* adapts virtual firewalls to VN changes.

Type I. When a service is deleted, the corresponding firewall rules protecting that service should be located and safely

deleted, to save resource. We are aware that one firewall rule may be applied to multiple services. Analysis should be conducted so that the rule deletion will not affect services other than the deleted one. Figure 3 shows the process of firewall adaption to service deletion. A user specifies the services to be deleted using our policy language, with $action$ setting to $delete$:

$$\langle s, delete, o \rangle$$

Policy Transformation translates the service deletion policies to firewall rules with $action$ setting to $delete$, and sends them to *Firewall Adaption*. *Firewall Adaption* sends the rules to *Firewall Provision*, and deletes them in *Rule Database*. *Firewall Provision* locates the firewall instances that have these rules and then deletes them. After rule deletion, *Firewall Provision* updates *Resource Constraints Database*. For the instances in F who have rules deleted, their c values will be added by the number of deleted rules, because after the rule deletion, the instances release room for new rules in the future.

When a user adds a new service, s/he also specifies the security policies for that service. The policies will be transformed to firewall rules and placed as we have discussed in Section IV.

Type II. A user may update security policies for a VN. Policy update may result in some old firewall rules to be deleted, and/or some new rules to be added. Figure 4 shows the process of firewall adaption to security policy updates. *Policy Transformation* generates the new rules for the updated security policies, and sends them to *Firewall Adaption*. *Firewall Adaption* retrieves from *Rule Database* the old rules, compares new rules with old ones, and finds (1) old rules to be deleted and (2) new rules to be added. Rules to be deleted are sent to *Firewall Provision*. Rules to be added are sent to *Firewall Placement*, which works out a placement solution and then sends the solution to *Firewall Provision*. *Firewall Provision* will perform rule deletion and placement operations. At last, *Firewall Provision* will update *Resource Constraints Database*.

Type III. Changes occur to VMs in a VN often. Common changes include VM migration and scaling out/in [22]. Virtual firewalls protecting the VN must adapt to these VM changes. A VM may be migrated from one rack to another, due to resource maintenance. In this case, the old firewall placement may no longer be optimal and the placement must be conducted again. In scaling out, new copies of existing VMs are created. New firewall rules must be added so as to protect these new copies. In scaling in, a VM is deleted. Then, the firewall rules that are associated with the VM must be located and deleted. We further discuss some safety issues caused by Type III changes in Section VIII and would explore solutions in our future work.

VI. Implementation and Evaluation

We implemented the core components of *VNGuard* on top of ClickOS [23] for provisioning and managing virtual firewalls. ClickOS is a Xen-based software platform optimized

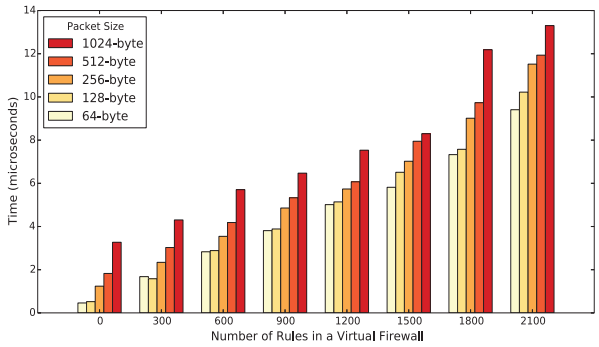

Fig. 5. The average processing time per packet of the virtual firewalls with different numbers of rules. The incoming traffic rate is set to 90Mbps and packet size varies from 64-byte to 1024-byte.

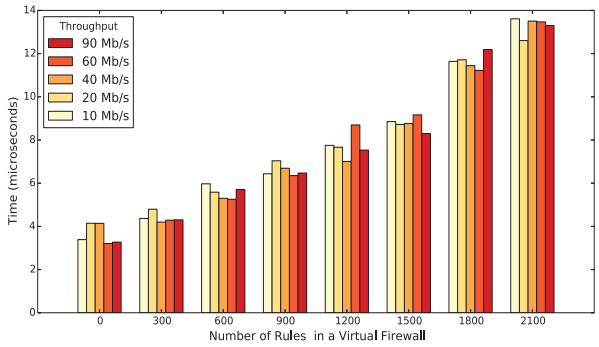

Fig. 6. The average processing time per packet of the virtual firewall with different number of rules. Packet size is set to1024-byte and throughput varies from 10Mbps to 90Mbps.

for fast provision of virtual network functions at large scale. A virtual network function instance created in ClickOS can be as small as 5MB, and can be booted within 30 milliseconds. It only takes around 220 milliseconds to create and boot 400 instances. ClickOS adopts Click [24] for the virtual network function development. Click provides large numbers of simple, well-known networking processing elements for building virtual network functions. However, ClickOS does not offer the necessary features to support virtual firewall adaption. In particular, ClickOS does not allow firewall rules on a virtual firewall to be updated without rebooting the firewall. To solve this problem, we developed three new Click elements: 1) FirewallTable; 2) FirewallMatch; and 3) FirewallManager. The FirewallTable element acts as the basic storage of firewall rules. The FirewallMatch element receives and forwards packets according to the firewall rules in FirewallTable. The FirewallManager element is designed to send messages to update firewall rules in FirewallTable. Firewall administrators can specify a network interface in a virtual machine and send messages encapsulated by IP packets via that network interface. For the virtual firewall placement, *VNGuard* makes use of a Matlab Integer Programming solver.

We designed experiments to evaluate three key aspects

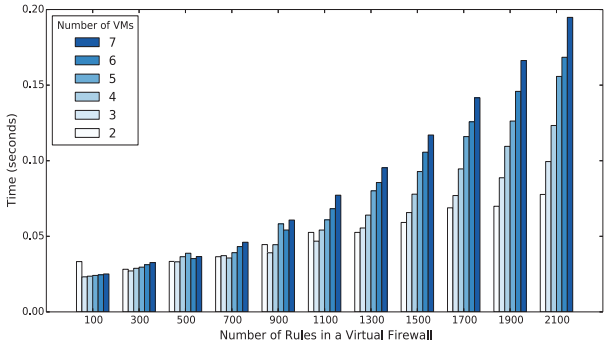

Fig. 7. Firewall Placement Performance.

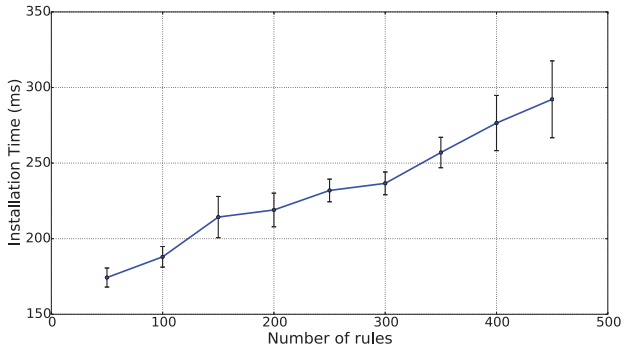

Fig. 8. The average time of firewall rule addition.

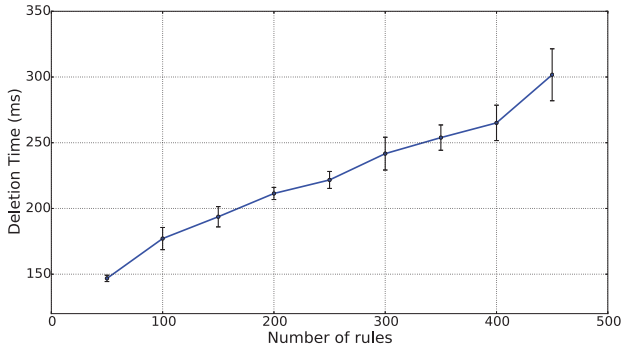

Fig. 9. The average time of firewall rule deletion.

of *VNGuard*: (1) the packet processing performance of the virtual firewalls provisioned by *VNGuard*; (2) the performance of virtual firewall placement; and (3) the performance of virtual firewall adaption. Our experiments are conducted using CloudLab [25], a platform providing computing infrastructure that enables experimenters to run cloud software stacks such as OpenStack and CloudStack.

Experiment 1. The packet processing performance of a virtual firewall is influenced by (1) the number of firewall rules, (2) the packet size, and (3) the rate of incoming packets. To test the performance, we set up a client generating traffic in varied sizes and rates to a server, and the traffic was passed to and processed by a virtual firewall provisioned by our *VNGuard*. We measured the processing time per packet of the virtual firewall. The client and server are two VMs on CloudLab, and each with two Intel E5-2660 V2 10-core CPUs at 2.20GHz,16x 16GB DDR4 RAM and Dual-port Intel 10be NIC. The virtual firewall created by *VNGuard* had 1 CPU and 40 MB memory. Figure 5 shows the average processing time per packet with 90Mbps incoming packet rate. The observation indicates an approximately line-rate increase of the average processing time per packet as we added up to 2100 firewall rules. With the same number of rules, the processing time appears to increase as the packet size grows. Figure 6 shows the average processing time of each packet, with the packets size setting to 1024 bytes, and the throughput varing from 10Mbps to 90Mbps. As the number of rules increases, we can see a steady, line-rate increase of the average processing time per packet. However, with the same number of firewall rules, the throughput appears to have little effect on the average processing time per packet.

Experiment 2. We adopted an Integer Programming based approach for virtual firewall placement in *VNGuard*. The performance of the approach is influenced by (1) the number of rules (M) to be placed, and (2) the number of instances (N) to place the rules in. In our experiment, we measured the time to find an optimal placement solution for different M and N values. Figure 7 shows our experiment results. Larger M and N are longer it takes to find an optimal placement. According to the study conducted by [26], the average number of firewall rules in real networks is around 793. We have tested our approach with the maximum of 2100 rules in our experiments. Even though there are 7 instances, our system takes less than 0.2 second to find an optimal placement solution, which indicates that our approach is pretty efficient.

Experiment 3. To adapt to VN changes, two operations are essential: firewall rule addition and deletion. We tested the average time of these two operations on a virtual firewall created by our *VNGuard*. Figure 8 shows the average time used to add firewall rules to a virtual firewall. As the number of firewall rules increases, the time consumed to add the rules increases. The addition is pretty efficient, as it took less than 300ms to add as many as 450 firewall rules. Figure 9 shows the average time used to delete firewall rules from a virtual firewall, which demonstrates the efficiency of rule deletion.

VII. Related Work

ClickOS [23] has been developed as a high-performance platform to support the development of virtual network functions. Virtual machines built on ClickOS are small (5MB) and boot quickly (around 30 milliseconds). Also, ClickOS can run tens of them concurrently. Our *VNGuard* implementation leverages some features provided by ClickOS for building virtual firewalls. However, the virtual network functions created by ClickOS lack adaptivity, since they cannot be dynamically updated without rebooting the system. We have addressed this

critical problem by developing several new Click elements in *VNGuard* for more effective provision of virtual firewalls.

Hu et al. [28] proposed a comprehensive framework, called FlowGuard, for building robust SDN firewalls, which are actually SDN applications running on top of SDN controllers. Challenges in designing SDN firewalls were identified and solved in FlowGuard. However, the work in this paper attempts to investigate solutions for effectively provisioning and managing *virtual* firewalls in context of NFV.

Zhang et al. [20] demonstrated that careless policy updates may result in security volitions. They presented safe and efficient policy update algorithms for firewall policy updates. However, the proposed algorithms are only able to deal with policy updates on a *single* firewall. The policy adaption mechanism in *VNGuard* coordinates policy updates across *multiple virtual* firewalls.

Different network function control frameworks have been recently proposed [3], [7], [9]–[13]. In particular, Split/Merge [13] is a control framework to achieve virtual middlebox elasticity, providing the ability to create or delete virtual middlebox replicas on demand. Split/Merge identifies the internal and external states of a virtual middlebox, and provides APIs for copying the internal states from the virtual middlebox to its replica. OpenNF [7] is another framework for controlling the internal states of virtual middleboxes. OpenNF provides APIs for richer controls, including copy, move, and some other operations. OpenNF also solves the race condition problem when some internal state is being moved, packets might arrive at the source instance after the move starts, or at the destination instance before the state transfer completes. Both Split/Merge and OpenNF can be applied to help solve the race condition problem in virtual firewall adaption, since it also needs to move internal states (firewall rules) from one virtual firewall to another. Some other network function control frameworks, including Slick [9], FlowTags [10], Stratos [11], and SIMPLE [12], mainly provide controls over traffic steering among network functions.

VIII. Discussion

In this work, we employed an Integer Programming based approach to find optimal virtual firewall placement. In building the Integer Programming model, we have considered resource constraints. However, service level agreements (SLAs) on performance should be also satisfied in reality. Hence, performance constraints may need to be considered when determining virtual firewall placement. The performance depends on the number of rules held in a virtual firewall instance and the resources (e.g. CPU, memory, etc.) assigned to the instance. We would explore a new model with respect to various network performance factors for virtual firewall placement in the future.

We have demonstrated the efficiency of the Integer Programming based approach for virtual firewall placement in Section VI. Some other optimization algorithms could be also explored for the policy placement. For example, Moshref *et al* [15] proposed a heuristic algorithm to place network flow rules in data centers. It would be interesting to compare the Integer Programming based approach with other approaches for virtual firewall placement.

Virtual firewall scaling in/out are other important problems that should be addressed in *VNGuard*. In virtual firewall scaling out, a new instance is created, and some rules in the old firewall instance will be moved to the new instance with the network traffic being moved alongside. This situation could cause two safety implications. One is the race condition that has been discussed in [7]. The "lost-free" and "order-preserving" move algorithms have been proposed in [7] to solve the race condition problem. However, there are limitations with those algorithms, because they rely on buffering network traffic at the SDN controller side during moving network states, which significantly consumes valuable bandwidth between SDN controller and switches. Thus, a new solution should be investigated to address the virtual firewall scaling in/out problems in *VNGuard*.

IX. Conclusion and Future Work

In this paper, we have proposed *VNGuard*, an NFV/SDN combination framework for effectively provisioning and managing virtual firewalls to safeguard VNs. *VNGuard* defines a high-level firewall policy language, finds optimal virtual firewall placement, and adapts virtual firewalls to VN changes. Our experimental results have demonstrated the efficiency and effectiveness of virtual firewalls built on *VNGuard*. In the future, we will expand our *VNGuard* framework for building robust stateful virtual firewalls, especially considering the safety state migration management for virtual firewalls. We also plan to implement *VNGuard* in other popular open-source NFV platforms, such as OPNFV [27].

Acknowledgments

This work was partially supported by the grants from National Science Foundation (NSF-IIS-1527421, NSF-CNS-1537924 and NSF-CNS-1531127).

References

[1] P. Busschbach, "Network functions virtualisation - challenges and solutions," Alcatel-Lucent Corp., France, Strategic White Paper, 2013. [Online]. Available: http://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2013/9377-network-functions-virtualization-challenges-solutions.pdf

[2] *Google Cloud Platform.* [Online]. Available: https://cloud.google.com/compute/docs/networking#firewalls

[3] S. Rajagopalan, D. Williams, and H. Jamjoon, "Pico replication: A high availability framework for middleboxes," in *SoCC'13,* Santa Clara, CA, 2013.

[4] J. Machi. (2013, December 17). *NFV said to SDN: I'll be there for you.* [Online]. Available: https://www.sdncentral.com/market/nfv-said-sdn-ill/2013/12/

[5] S. K. N. Rao, "SDN and its use-cases-NV and NFV," NEC Technologies India Limited, White Paper, 2014. [Online]. Available: http://www.nectechnologies.in/en_TI/pdf/NTI_whitepaper_SDN_NFV.pdf

[6] Open Networking Foundation (ONF), "OpenFlow-enabled SDN and network function virtualisation," Tech. Rep., Feb., 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-sdn-nvf-solution.pdf

[7] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: enabling innovation in networking function control," in *SIGCOMM'14,* Chicago, IL: ACM, 2014, pp. 163-174.

[8] F. Paganelli, M. Ulema, and B. Martini, "Context-aware service composition and delivery in NGSONs over SDN," in *IEEE Communications Magazine,* vol. 52, issue 8, 2014, pp. 97-105.

[9] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A slick control plane for network middleboxes," in *HotSDN'13*, HongKong, China: ACM, 2013, pp. 147-148.

[10] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "FlowTags: Enforcing network-wide policies in the presence of dynamic middlebox actions using flow tags," in *HotSDN'13*, HongKong, China: ACM, 2013, pp. 19-24.

[11] A. Gember, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds," in *arXiv preprint arXiv:1305.0209*, 2013.

[12] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu , "SIMPLE_fying middlebox enforcement using SDN," in *SIGCOMM'13*, HongKong, China: ACM 2013, pp. 27-38.

[13] S. Rajagopalan, D. Williams, H. Jamjoon, and A. Warfield, "Split/merg: system support for elastic execution in virtual middleboxes," in *NSDI'13,* Lombard, IL: ACM, 2013, pp. 227-240.

[14] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *arXiv preprint arXiv:1406.1058*, 2014.

[15] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centres," in *NSDI'13*, Lombard, IL: ACM, 2013, pp. 157-170.

[16] S. Zhang, F. Ivancic, C. Lumezanu, Y. Yuan, A. Gupta, and S. Malik, "An adaptable rule placement for software defined networks," in *2014 44th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Atlanta, GA, 2014, pp. 88-99.

[17] European Telecommunications Standards Institute (ETSI) GS, "Network functions virtualisation (NFV); Infrastructure Overview," Group Specification, 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf

[18] H. Hu, G. Ahn, and K. Kulkarni, "Detecting and Resolving Firewall Policy Anomalies," in *IEEE Transactions on dependable and secure computing*, vol. 9, no. 3, 2012, pp. 318-331.

[19] M. Jünger, Th.M. Libelling, D. Naddef, G.L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and A. Wolsey, "Nonlinear integer programming," in "50 Years of Integer Programming 1958-2008: The Early Years and State-of-the-Art Surveys", New York, Spring-Verlag, 2009.

[20] C. C. Zhang, M. Winslett, and C. A. Gunter, "On the safety and efficiency of firewall policy deployment," in *IEEE Symposium on Security and Privacy,* Berkeley, CA, 2007, pp. 33-50.

[21] Y. Gao, X. Chen, N. Pan, and Z. Morley Mao, "On the safety of enterprise policy deployment," in *NDSS 2010*, San Diego, CA, USA, 2010.

[22] European Telecommunications Standards Institute (ETSI) GS, "Network functions virtualisation (NFV); Terminology for main concepts in NFV," Group Specification, 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.01.01_60/gs_nfv003v010101p.pdf

[23] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualisation," in *11th USENIX Symposium on Networked System Design and Implementation,* 2011, pp. 459-473.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Frans Kaashoek, "The click modular router," in *ACM Transactions on Computer Systems,* vol. 18, issue 3, 2000, pp. 263-297.

[25] University of Utah, Cloudlab Technology. Available: http://www.cloudlab.us/technology.php

[26] M. Chapple, "Firewall rules are meant to be managed, not broken," 2012. Available: http://www.biztechmagazine.com/article/2012/08/firewall-rule-management-key-network-security.

[27] OPNFV. Available: https://www.opnfv.org/

[28] H. Hu, W. Han, G. Ahn, and Z. Zhao, "FlowGuard: building robust firewalls for software-defined network," in *HotSDN'14*, Chicago, IL, USA, 2014.