Adaptive Network Security Service Orchestration Based on SDN/NFV

Priyatham Ganta[1], Kicho Yu[2,3], Dharma Dheeraj Chintala[1], and Younghee Park[1,3(✉)]

[1] Computer Engineering Department, San Jose State University, San Jose, USA
younghee.park@sjsu.edu
[2] Computer Science Department, Northeastern University, Shenyang, China
[3] Silicon Valley Cybersecurity Institute (SVCSI), San Jose, USA

Abstract. The integration of Software-Defined Network (SDN) and Network Function Virtualization (NFV) is an innovative network architecture that abstracts lower-level functionalities through the separation of the control plane from the data plane and enhances the management of network behavior and network services in real time. It provides unprecedented programmability, automation, and control for network dynamics. In this paper, we propose a flexible and elastic network security service management system for timely reacting to abnormal network behavior by orchestrating network security functions based on the technology of SDN/NFV. In designing the system, we address key challenges associated with scalability, responsiveness, and adversary resilience. The proposed system provides a real time and lightweight monitoring and response function by integrating security functions in the SDN/NFV domain. The SDN automatically learns the network conditions to orchestrate security functions for effective monitoring against attacks. The system is implemented based on an open-source SDN controller, RYU, and consists of three main agents; network monitoring, orchestration agents, and response agents. Experimental results have shown that our approach achieved low network latency with small memory usages for virtual intrusion detection systems.

1 Introduction

Software-defined networking (SDN) and Network Function Virtualization (NFV) provide new ways to enable the introduction of sophisticated network control for security and dependability. The advent of SDN/NFV has shifted the traditional perspective of the network from ossified hardware-based networks to programmable software-based networks [10]. It introduces significant granularity, visibility, flexibility, and elasticity to networking by decoupling the control plane in network devices from the data plane and through virtualization. A control program can automatically react to dynamic changes of the network state and thus maintain the high-level policies in place. The centralization of the control logic in a controller with global knowledge of the network state simplifies the

H. Kim (Ed.): WISA 2021, LNCS 13009, pp. 231–242, 2021.
https://doi.org/10.1007/978-3-030-89432-0_19

development of more sophisticated network functions. This ability to program the network and to control the underlying data plane is therefore the crucial value proposition of SDN/NFV and opens new opportunities to implement responsive actions to networks under attack in a timely manner.

SDN/NFV shift traditional perspectives on defense methods in a new direction against attacks in real time. It provides programmability, flexibility, and scalability not possible in traditional computer networks, which facilitates logically centralized control and automation [10]. These capabilities provide insight into network behavior that can be used to automatically detect malicious network attacks. However, previous work has addressed specific attacks in the different layers of SDN instead of using both SDN and NFV to develop defense methods [11,17,19]. A few of the previous works have focused on the integrated security services between SDN and NFV. Furthermore, most previous works have studied the internal security problems of SDN or NFV in terms of software vulnerabilities. Now, it is important to develop a general centralized security service system that automatically reacts to unpredictable changes in the network states and initiates actions to deploy countermeasures against any attack in the intrusion detection and response systems. However, it has been passive to deploy hardware or software IDS in fixed locations that cause many disadvantages, such as low attack detection rates, low resource utilization, and little flexibility against zero-day attacks. For an effective defense system, a timely response action should be implemented and function properly in countering various attacks, especially on a real-time basis.

This paper proposes an adaptive network security orchestration system to dynamically and effectively deploy security functions and detect various attacks in real time. It is an intelligent and automatic framework integrated with SDN/NFV to be reactive to real time network behavior. This framework orchestrates virtual security functions into the network while monitoring real-time traffic to detect network attacks. The southbound interfaces that communicate with the switches deploy these security functions with the help of SDN. The proposed framework consists of three agents: a network monitoring agent, an orchestration agent, and a response agent. The network monitoring agent collects network statistical data to continuously monitor network states and evaluates the best routing path to monitor and detect network abnormalities in real time. The orchestration agent retrieves state information to dynamically deploy security functions for intrusion detection. The response agent dynamically changes network states to make the network safe while initiating an appropriate action into the network based on the reactive routing in SDN. The three agents keep interacting with the SDN controller for centralized monitoring and controlling of the entire network space. Our experimental results demonstrate the effectiveness of this framework to distribute virtual security functions with a reasonable network and system overhead. The results give us the insight to help the SDN controller with NFV properly react to known or unknown attacks by orchestrating a set of light security functions in real time.

Our first contribution is to propose an intelligent and autonomous orchestration system that can effectively react to network threats by utilizing SDN/NFV. In addition, we develop a lightweight orchestration agent by using containerization technology in order to deploy various security functions while monitoring network states. Finally, based on the programmability of SDN/NFV, we develop a resource-efficient security service system for real-time network intrusion detection and response system. Lastly, this paper implements the proposed system with open sources and evaluates the proposed framework in a real cloud environment.

The rest of the paper is organized as follows. Section 2 presents our new proposed framework. In Sect. 3, we evaluate our framework and show experimental results with the implementation details. In the rest of the sections, we discuss related work and state our conclusions.

2 System Architecture

This section presents our proposed framework to respond to network threats by using the orchestration service while leveraging the programmability and automation offered in SDN/NFV. The system consists of three main agents: a network monitoring agent, an orchestration agent, and a response agent, as shown in Fig. 1. First, the network monitoring agent has a traffic analyzer and a network measurement feature to keep track of real-time network status and conditions. Second, the orchestration agent determines the most appropriate reaction from real-time network states to monitor network behavior for intrusion detection. Lastly, the response agent takes reactive action to defend against network abnormalities for intrusion detection. A detailed explanation follows.

2.1 A Network Monitoring Agent

The network monitoring agent records statistical information and measures network usage in order to identify the current network state through the SDN controller. It analyzes traffic by byte rates and the number of packets according to each port and each protocol. It determines the network effective bandwidth to decide the number of security service functions per node while avoiding network congestion points. Network monitoring is an essential building block to initially understand the network situation and deploy a set of security functions to detect network abnormalities.

The SDN controller periodically polls statistical information from switches. The data includes topology information, switch information, and statistics about flow rules and packets. We compute the overall usage of the links in a network by using the number of bytes and the number of packets both for each port and each protocol to determine the routing path for incoming traffic. Also, the traffic classification module captures protocols that contribute the most traffic to the currently utilized bandwidth. Having identified these protocols, the corresponding flow rules are selected and pushed to the switch under consideration. As

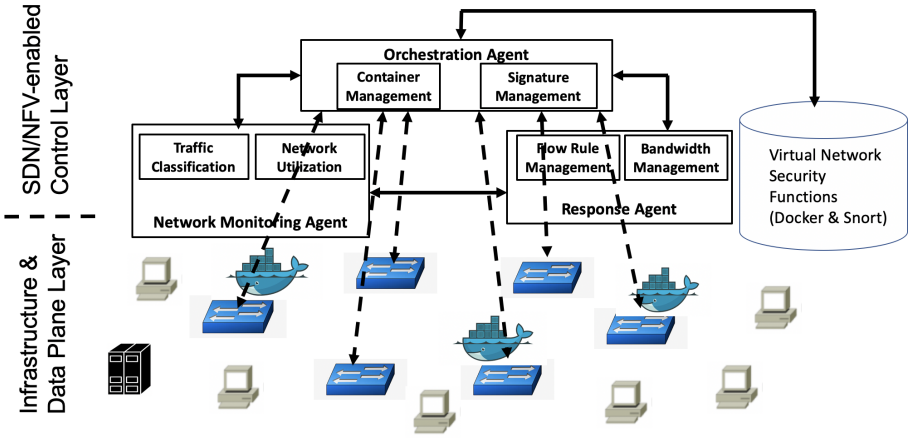

Fig. 1. System architecture

mentioned earlier, traffic classification is carried out in a fine-grained manner, that is, in a per-switch per-port manner, and this granularity makes it possible to deploy flow rules only on the switches that require them. Matching flow rule actions drop the selected packets and allow the same when the traffic rate for the particular protocol reduces. If the protocol-specific usage is relatively high, and the control is passed over to the flow-rule manager for threat responses, the drop action would be selected for the protocol with the highest protocol-specific usage. The system then observes the protocol-specific usage for subsequent iterations and then issues an allow flow rule on the switch to return it to normal behavior. It is important to understand that the flow rules are pushed on individual switches and not all of them, and hence even when a drop action is selected for a particular protocol on some switch, the rest of the network functions unchanged. Also, there are multiple routes present in the network for all the hosts, so the packets can still be routed using alternative paths. Protocol-specific network utilization is defined as how much bandwidth is occupied by a certain protocol (e.g. TCP, UDP, ICMP, HTTP, and DNS) out of the total current used bandwidth. This provides a method to identify specific network congestion locations for efficient orchestration services.

2.2 An Orchestration Agent

The orchestration agent aims to orchestrate virtual network security functions to edge devices dynamically based on the network traffic to that device. As shown in Fig. 1, the orchestration agent has two main components: container management and signature management. The container management monitors system resources and network utilization to determine the number of containers for orchestration services. The signature management decides a set of the Snort rules based on the input from the network monitoring agent. Finally, the

agent can orchestrate a set of virtual security functions in SDN to monitoring real-time traffic and detect network attacks. The SDN controller keeps monitoring the network flow to orchestrate the virtual security functions on remote devices for intrusion detection. As multiple functions need to be executed on the same edge devices, it is important to design a lightweight orchestration service like container orchestration. The controller receives the network status information from the network monitoring agent and the orchestration agent deploys a container with a software-based IDS. To achieve this goal, it is significant to develop a lightweight orchestration service since the existing orchestration services require a lot of system resources. For example, Kubernetes and OpenShift have a minimum resource requirement of 8 Gigabytes of physical memory on the master and slave nodes. It is impossible to multiple security services on the same edge devices. To overcome resource limitations, we develop a new orchestration security service system by using Docker with Snort since the Docker needs only minimal physical memory as 256 megabytes. The Docker can be used to integrate and communicate with docker daemons running in remote hosts. To implement virtual security functions, this system utilizes Snort on the Docker. However, instead of including all the signature files in Snort, each signature file is dynamically loading into Snort according to protocol types. For example, under HTTP traffic, the security function includes only signature files related to TCP and HTTP by excluding other protocol-related signatures, such as DNS, UDP, or ICMPE. By dynamically controlling each security function, we can optimize the signature matching process in parallel by orchestrating these security services on the designated path in the network.

2.3 A Response Agent

The response agent is a reactive action to control network behavior when each security functions report alerts by using the reactive routing function of SDN. It has two reaction actions: bandwidth management and flow rule management. First, the Bandwidth Manager can be thought of as an interface for a southbound API to modify the bandwidth of the switch. The value of the bandwidth is determined by the orchestration agent, making it possible for the system to scale up and scale down the bandwidth as needed. The agent can select the best bandwidth based on the input of the network monitoring agent. There are many network attacks that can be countered by just modifying the network bandwidth to some extent. According to attack detection and congestion points, the SDN controller can control bandwidth according to the results of each security function. Second, the flow rule manager is another interface for a southbound API to update flow rules on the switch. This API sends an OpenFlow message to the particular switch to indicate the update in the flow table. With the help of this API, the framework can add, modify, and remove rules corresponding to various packets. When each security function detects attacks, the SDN controller can update flow rules in real time. For example, under the HTTP flooding attacks, the reactive action could be to limit or stop the incoming HTTP traffic to a particular switch or host in the network according to the report of the security functions.

3 Evaluation

This section evaluates our proposed orchestration system in terms of network efficiency and overhead in CloudLab, an open cloud infrastructure supported by NSF (National Science Foundation). We will discuss our implementation, experimental setup, and experimental results in the following sections.

3.1 Implementation

We implement the proposed system (i.e. application) based on RYU [1], an open-source SDN controller with Docker and Snort. RYU is an event-driven SDN controller developed in the Python programming language. It is modular and relatively simple to understand. The event-driven nature of the controller essentially boils down to the concept of event handlers and event listeners. The implementation of the framework spans across a library and an internal component integrated with the existing RYU code with Open vSwitch (OVS) and OVSDB. Docker is an application containerization technology that isolates the operating system from the application. This isolation helps us to run multiple applications of Snort, in parallel and each having its own secure environment called a container. There are three main components of the Docker Engine: Docker daemon, REST API, and client. In the current architecture, the proposed system is a client application running alongside the controller which is connected to docker daemons on all switches in the topology. To ensure a secure connection, a TLS handshake is enabled between the daemon and the client. Snort architecture consists of multiple modules. For Snort to read a packet on an interface to flag an alert or to allow the packet, all these modules should perform their operations consistently. For experimentation purposes, Snort will be packaged into a docker image with necessary configurations. It includes all the snort rules, community, and registered, categorized into 34 different sets based on their application layer protocol. A configuration file is also included for each set to specify rules. The container orchestration service is a python flask web service listening locally for any incoming requests from the controller process. It provides two services to set up the container and the bridge and then to start Snort. The docker API client will be used to trigger any deployment instructions to the docker daemon. Whenever a new packet arrives at the switch, an *ofp_event.EventOFPPacketIn* event is generated and a packet in handler will be triggered on this event. Details of the packet information like a switch IP address, port numbers, source, and destination addresses are extracted in the network monitoring agent. This extracted information is used to request the orchestration service to start Snort on the specified switch. Based on the information received, a specific Snort configuration file will be chosen according to protocol types. The docker API client will be used to get a container from the available list and execute necessary commands to run Snort with a selected configuration file in Inline mode. Snort is configured to report the alerts to a Unix socket. Simultaneously, a script is started to relay these alerts to the remote Ryu controller. As the existing container is used, a new thread is started to set up the container and bridge to allocate future requests.

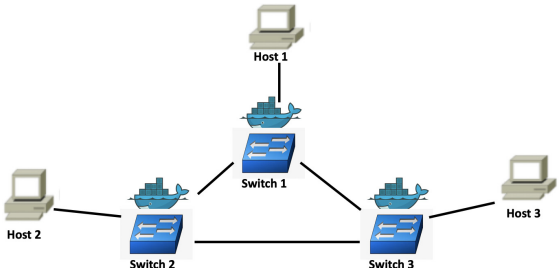

Fig. 2. Experimental setup in CloudLab

In the flow rule management, any packet entering the switch without corresponding flow rules in the switch will trigger the *packet_in_handler* code part in RYU. Before starting the container, default flow rules should be added in order to take care of switches where snort inside the container will not be started. This situation might arise if there is already a container deployed in one of the switches for this kind of flow. Once the generic flow rules are added, the request to start snort inside a container is sent to the orchestration service along with the flow parameters. If the service returns a positive response, the generic flow rules will be deleted and the flow rules to forward every packet to the container will be added onto the switch. If the response from the container is negative, no operation will be done on the generic flow rules.

3.2 Environmental Setup

The proposed system is evaluated with a generic network topology with three hosts and three switches on a cloud platform (called CloudLab) as shown in Fig. 2. Cloudlabs is a cloud testbed built for experimenting with real-time cloud, network, and orchestration scenarios and provides necessary transparency and control over the nodes deployed in the environment. The switches used are virtual machines with UBUNTU-18 installed and run OVS-Switch as a daemon. The Ryu controller, with version 4.34, is run remotely from the topology. The docker orchestration service, with docker server version 18.09.7 and API version 1.39, is run alongside the Ryu controller in the same node. The system has a Ubuntu machine running Ubuntu 16.04.1 LTS version, Intel(R) Xeon(R) CPU E5 with 2.4 GHz. Each switch has 16 GB of RAM with a quad-core processor and we used Snort 2.9.7.o GRE with Ubuntu 18.04.3 LTS on Docker.

3.3 Experimental Results

We present the experimental results obtained from the evaluation of the proposed system based on the following metrics: network latency and memory usage. Network latency is an important factor in evaluating the effectiveness of the proposed framework and memory usage gives us an idea of the system resource requirement. The experimental results were obtained in the network topology shown

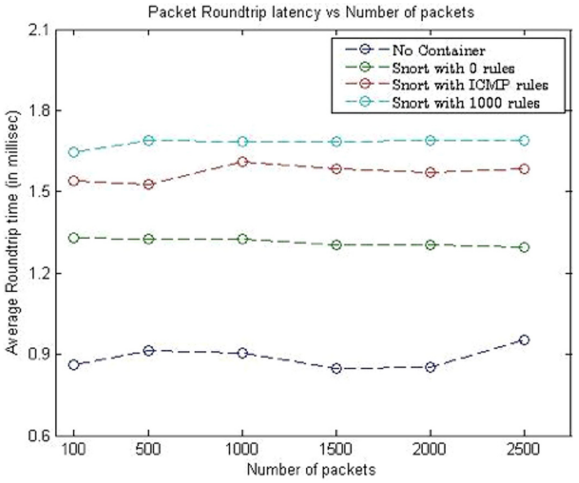

Fig. 3. Network latency in CloudLab

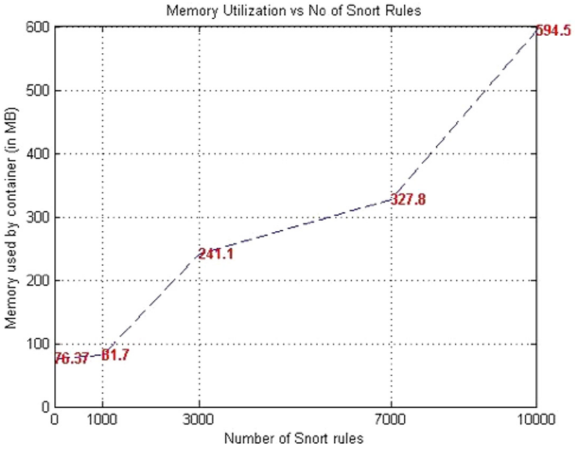

Fig. 4. Memory usage in CloudLab

in Fig. 2. Note that attack detection rates are mostly impacted by the Snort performance with the signature files, which is out of our research scope.

First, the network latency is calculated as the round-trip time (RTT) between two hosts in the network. The round-trip time accumulates the delay for each link in the path between two hosts. Network latency captures the delay in switch processing time, which includes flow rule and bandwidth update processing times. In Fig. 3, we measure network latency without containers or with containers depending on the number of Snort rules (i.e. attack signatures) to be matched with each packet. As shown in Fig. 3, the network latency increases with the orchestration service since it requires matching incoming packets with the attack

signature for intrusion detection. When we deployed only the container with Snort without any rules, it shows an average of 1.4 ms round-trip time. With the Snort rules, the network latency was slightly increased and the increase was impacted by the number and the type of the Snort rules. It includes various factors: the switch capacity, the packet header processing time, the packet payload processing time, and so on. In particular, when every packet needs to compare bytes for malware with every rule configured in Snort, the network latency significantly increases according to the number of rules. However, the orchestration services can bring more benefits over such network overhead as we discussed in the previous sections.

Second, we evaluate the CPU and memory usage. The memory usage was obtained using the *top* terminal utility command. Figure 4 shows the physical memory used by the controller during the security orchestration service with different numbers of the Snort rules. CPU and memory utilization are noted while varying the number of rules. The memory utilization gradually increased depending on the number of the Snort rules. However, the CPU usage remained the same from 0.3% to 0.5% increases depending on the packet rates and the number of the Snort rules. Therefore, the memory utilization is gradually increased by increasing the number of the rules whereas the CPU utilization of the container remained similar in all cases.

4 Related Work

A number of approaches to security in SDN address controller specific attacks [11,17,19] by deploying additional defense modules within the SDN controller to keep track of various API calls or network traffic. The AEGIS [11] framework addresses security vulnerabilities that arise from SDN applications. It monitors API calls and usage of core controller modules to avoid any misuse. AVANT-GUARD [17] and FLOODGUARD [19] focus on delivering data plane solutions to SDN security.

Many previous approaches have utilized SDN features to detect network intrusions in SDN. Wang, et al. [18] were the first ones to propose a network security framework that relied solely on the features of SDN. They proposed DaMask a framework that uses deep learning to determine if the network is under attack and deploys network administrator input as the reactive measure against it. Lim et al. [9] discuss an SDN-based defense against DoS and DDoS attacks by botnets. The authors send a redirection message to thwart the botnet, which is powered by OpenFlow protocol to manage DDoS attacks. Braga, Mota and Passito [2] propose a lightweight DoS detection framework using NOX [8] controller. They use the concept of IP Flows [5] to detect network attacks.

Enterprise networks are populated with a large number of proprietary and expensive hardware-based network functions (NFs), or middleboxes [16], which provide key network functionalities, such as firewall, IDS/IPS, and load balancing. Hardware-based NFs present significant drawbacks including high costs, management complexity, slow time to market, and unscalability, to name a

few [14,15]. Network Function Virtualization (NFV) was proposed as another new network paradigm to address those drawbacks by replacing hardware-based network appliances with virtualized (software) systems running on generic and inexpensive commodity hardware, and delivering NFs as network processing services. Different control frameworks for virtualized NFs have recently been proposed to address the *safe* scaling of virtual network functions [6,7,12,13]. In particular, Pico Replication [12], Split/Merg [13] and OpenNF [7] are all control frameworks over the internal state of NFs. Pico Replication provides APIs that NFs can use to create, access, and modify internal states. Split/Merg achieves load-balanced elasticity of virtual middleboxes, via splitting internal states of NFs among virtual middlebox replicas and re-route flows. However, neither Pico Replication nor Split/Merg are loss-free during NF state migration. OpenNF provides fine-grained control over movement within internal NF states and enables loss-free and order-preserving movement from one NF state to another. One significant drawback of OpenNF lies in the fact that it relies on the central controller to do heavy traffic buffering during state moves. This results in scalability and safety issues. A second drawback is the triangular routing of packets, which introduces large traffic overhead and latency. Jacobson et al. [6] proposed two enhancements to OpenNF: packet-reprocessing and peer-to-peer transfers. Although packet-reprocessing reduces the amount of traffic that needs to be buffered, it still relies on the controller to buffer in-flight traffic. Peer-to-peer transfer eliminates dependence on the controller to buffer, but still suffers from triangular routing issues. Our migration scheme neither depends on the controller to buffer, nor performs triangular routing. In addition, VNGuard [3] was recently introduced for effective provision and management of virtual firewalls based on NFV to safeguard virtualized environments. Also, NFV and SDN techniques have recently been used to overcome the inflexibility and inelasticity limitations of hardware-based DDoS defense appliances. In particular, Fayaz et al. [4] proposed Bohatei, a flexible and elastic virtual DDoS defense system, for effectively defending DDoS attacks.

5 Conclusion

The combination of SDN and NFV has changed our paradigm in networked systems with flexibility, scalability, and programmability. SDN enables us to control network behavior in real time and NFV allows us to utilize a virtual middleware for all services including security. This paper proposes an adaptive orchestration system to distribute virtual security functions in SDN for intrusion detection. The system consists of three components to manage a set of virtual security functions for intrusion detection and response. The virtual security functions are implemented with Docker and Snort. The proposed system defends network attacks autonomously and intelligently through real-time orchestration services. The goal of this framework is to protect the network from various network attacks through adaptive security orchestration services by using SDN/NFV. The SDN controller handles topology management by collecting switch statistics and by updating appropriate flow rules on the switches for intrusion response

when the security functions raise alarms. Our experimental results increase network latency and memory usage depending on the number of the Snort rules inside each security function. To be efficiently and effectively matched with the signatures, the proposed system dynamically loads a set of the relevant rules depending on each traffic type. Our proposed framework makes a significant contribution in the direction of such intelligent and autonomous defense systems by using SDN and NFV.

References

1. Ryu SDN framework [software] (2014). https://osrg.github.io/ryu/
2. Braga, R., Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: IEEE Local Computer Network Conference, Denver, CO, pp. 408–415, October 2010. https://doi.org/10.1109/LCN.2010.5735752. https://dx.doi.org/10.1109/LCN.2010.5735752
3. Deng, J., et al.: VNGuard: an NFV/SDN combination framework for provisioning and managing virtual firewalls. In: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN 2015). IEEE (2015)
4. Fayaz, S.K., Tobioka, Y., Sekar, V., Bailey, M.: Bohatei: flexible and elastic DDoS defense. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 817–832 (2015)
5. Feng, Y., Guo, R., Wang, D., Zhang, B.: Research on the active DDoS filtering algorithm based on IP flow. In: Fifth International Conference on Natural Computation, Tianjin, China, vol. 4, pp. 628–632, August 2009. https://doi.org/10.1109/ICNC.2009.550
6. Gember-Jacobson, A., Akella, A.: Improving the safety, scalability, and efficiency of network function state transfers. In: Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, pp. 43–48. ACM (2015)
7. Gember-Jacobson, A., et al.: OpenNF: enabling innovation in network function control. In: Proceedings of the 2014 ACM Conference on SIGCOMM, pp. 163–174. ACM (2014)
8. Gude, N., et al.: NOX: towards an operating system for networks. SIGCOMM Comput. Commun. Rev. **38**(3), 105–110 (2008). https://doi.org/10.1145/1384609.1384625. https://doi.acm.org/10.1145/1384609.1384625
9. Lim, S., Ha, J., Kim, H., Kim, Y., Yang, S.: A SDN-oriented DDoS blocking scheme for botnet-based attacks. In: Sixth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 63–68, July 2014. https://doi.org/10.1109/ICUFN.2014.6876752
10. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008). https://doi.org/10.1145/1355734.1355746. https://doi.acm.org/10.1145/1355734.1355746
11. Padekar, H., Park, Y., Hu, H., Chang, S.Y.: Enabling dynamic access control for controller applications in software-defined networks. In: Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, SACMAT 2016, pp. 51–61. ACM, New York (2016). https://doi.org/10.1145/2914642.2914647. https://doi.acm.org/10.1145/2914642.2914647

12. Rajagopalan, S., Williams, D., Jamjoom, H.: Pico replication: a high availability framework for middleboxes. In: Proceedings of the 4th Annual Symposium on Cloud Computing, p. 1. ACM (2013)
13. Rajagopalan, S., Williams, D., Jamjoom, H., Warfield, A.: Split/merge: system support for elastic execution in virtual middleboxes. In: NSDI, pp. 227–240 (2013)
14. Sekar, V., Egi, N., Ratnasamy, S., Reiter, M.K., Shi, G.: Design and implementation of a consolidated middlebox architecture. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 24. USENIX Association (2012)
15. Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., Sekar, V.: Making middleboxes someone else's problem: network processing as a cloud service. ACM SIGCOMM Comput. Commun. Rev. **42**(4), 13–24 (2012)
16. Sherry, J., Ratnasamy, S., At, J.S.: A survey of enterprise middlebox deployments (2012)
17. Shin, S., Yegneswaran, V., Porras, P., Gu, G.: Avant-guard: scalable and vigilant switch flow management in software-defined networks. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & #38; Communications Security, CCS 2013, pp. 413–424. ACM, New York (2013). https://doi.org/10.1145/2508859.2516684. https://doi.acm.org/10.1145/2508859.2516684
18. Wang, B., Zheng, Y., Lou, W., Hou, Y.T.: DDoS attack protection in the era of cloud computing and software-defined networking. Comput. Netw. **81**, 308–319 (2015). http://dx.doi.org/10.1016/j.comnet.2015.02.026. http://www.sciencedirect.com/science/article/pii/S1389128615000742
19. Wang, H., Xu, L., Gu, G.: FloodGuard: a DoS attack prevention extension in software-defined networks. In: 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 239–250, June 2015. https://doi.org/10.1109/DSN.2015.27