P4Guard: Designing P4 based Firewall

Rakesh Datta
Computer Engineering
San Jose State University
San Jose, CA USA
rakesh.datta@sjsu.edu

Sean Choi
Electrical Engineering
Stanford University
San Jose, CA USA
yo2seol@stanford.edu

Anurag Chowdhary
Computer Engineering
San Jose State University
San Jose, CA USA
anurag.chowdhary@sjsu.edu

Younghee Park
Computer Engineering
San Jose State University
San Jose, CA USA
younghee.park@sjsu.edu

***Abstract*— A virtual firewall based on Network Function Virtualization (NFV) with Software Defined Networking (SDN) provides high scalability and flexibility for low-cost monitoring of legacy networks by dynamically deploying virtual network appliances rather than traditional hardware-based appliances. However, full utilization of virtual firewalls requires efficient management of computer virtualization resources and on-demand placement of virtual firewalls by steering traffic to the correct routing path using an SDN controller. In this paper, we design P4Guard, a software-based configurable firewall based on a high-level domain-specific language to specify packet processing logic using P4. P4Guard is a protocol-independent and platform-agnostic software-based firewall that can be incorporated into software switches that is highly usable and deployable. We evaluate the efficiency of P4Guard in processing traffic, compared to our previous virtual firewall in NFV.**

Keywords—P4, Firewall, SDN/NFV, Security

I. INTRODUCTION

The advent of Network Function Virtualization (NFV) and Software Defined Networking (SDN) has significantly reduced operation costs and capital costs by virtualizing hardware network appliances, such as firewalls, NAT, load balancers, and BRAS. Combining virtualization with SDN offers the many advantages of programmability, scalability, and flexibility [1, 2, 3]. At the same time, hardware firewalls are rapidly being replaced with virtual firewalls at entry-points in enterprise networks and the cloud [18]. For example, VMware has introduced VMware NSX [17] to protect enterprise networks with virtual firewalls, dramatically increasing enterprise revenues through realization of huge hardware maintenance cost savings.

The benefits of NFV/SDN have been offset by certain limitations in the design of NFV/SDN. Many NFV/SDN devices require the network to operate under the constraints of specific packet headers and actions (e.g,. firewall rules) that adhere to OpenFlow protocols [2, 3]. In addition, OpenFlow is limited in its ability to support customized protocols. However, a new packet processing language called P4, coupled with P4 supported hardware and software, enables network operators to specify their own packet headers and packet processing logic, which in turn, allows the data plane to understand and process customized packet protocols within a network [4, 5, 6].

In this paper, we propose P4Guard, a software firewall specified in P4 language that programs packet-forwarding functions in the data plane. Our software provides a target-oblivious and protocol-independent firewall that can be used with any P4-supported programmable switches. P4Guard has a controller to dynamically initiate and destroy software firewalls across networks. Instead of hardware-based firewalls, P4Guard can be easily configurable to update functionalities through compiling the data plane and to install dynamic firewall rules on the fly. The software firewall functions are defined in the data plane of P4-enabled switches. We define packet headers, packet parsers, and packet processing behaviors for the proposed software firewall. We implement a *counter table* to record the statistical flow information to control bandwidth [19, 20] when the controller detects various flooding attacks based on packet counters [21]. Our contributions are summarized as follows. First, we propose an open-source implementation of P4Guard, a software firewall without virtualization, by using the high-level domain specific language (DSL) called P4. Second, we introduce a P4Guard controller to manage the software firewalls in a centralized fashion. It accepts the high-level security policy language and sends the RPC message (i.e. security rules) to the P4-enabled switches. Third, we implement the structures of the firewall rule tables into the flexible and reprogrammable forwarding tables in the P4 data plane. Lastly, we discuss and analyze the proposed P4Guard with VNGuard [1], a virtual firewall in NFV to evaluate a packet-processing capability with different settings.

The rest of the paper is organized as follows. We discuss related work in Section II and present a software firewall with P4 in Section III. In Section IV, we compare our new firewall with our previous virtual firewall while analyzing the two firewall approaches in Section V. Finally, we present our conclusions in Section VI.

II. RELATED WORK

Enabling programmability in the data planes has allowed us to design customizable protocols and semantics for packet forwarding [4, 5, 6, 7, 8, 9]. In particular, P4 program language allows us to program the forwarding planes of networking devices, shifting the paradigm of switches from a vendor lock-in architecture to one that provides user-

978-1-5386-7185-6/18/$31.00 ©2018 IEEE

customizable programmable data planes [4, 5, 6]. Regardless of the underlying protocols, packet headers, packet parsers, and packet processing behaviors can be defined in P4. Then, P4-enabled devices process packets according to the specified P4 program.

PISCES is a P4 supported software switch that allows users to define custom protocol specifications using P4 without modifying the internal switch code [5]. It is a modified version of Open vSwitch (OVS) coupled with P4 compiler to support packet processing logic in the software switch. Hyper4 virtualized the data plane based on the P4 programming language to support parallel executions of multiple programs in the switch [4]. H. Song introduced protocol-oblivious forwarding to remove protocol dependency on the forwarding elements for SDN [9]. To accelerate the use of P4, industry has utilized this programmable data plane on various hardware devices, using P4 software such as Xilinx SDNet [10], P4FPGA [14], and Netronome SDK [11].

VNGuard [1] is a virtual firewall based on ClickOS [12] using click modular routers. It virtualizes the firewall on ClickOS while a controller manages dynamic firewall rules and firewall placement according to high-level policy rules. The controller creates forwarding rules into the underlying Open vSwitch. The virtual firewall could be dynamically created, destroyed, and updated in new states. In this paper, we developed P4Guard using a model similar to our previous VNGuard, replacing VNGuard's virtualization, SDN, OpenFlow and Open vSwitch with the P4 programming language and P4-enabled switches.

III. System Design of P4Guard

A. Overview

Figure 1 shows the overall system architecture of P4Guard. P4Guard has two parts: a P4 control plane and a P4 data plane. The control plane of P4Guard dynamically provides firewall services and monitors a network based on statistical information. In other words, the controller plays the important role of managing firewall rules and software firewall placement across networks for service provisions, and also collects statistical information from the P4 switches for network monitoring. The software firewall in the P4 data plane is designed to process the packets according to the firewall rules via three main components: a packet parser, a packet forwarder, and a packet generator. The firewall in the P4 switch also embraces a statistical subsystem to record the number of packets. Therefore, the controller manages the software firewalls with the firewall rules in a centralized location, and the data plane processes packets according to packet forwarding policies in the P4 data plane.

Figure 2 shows a brief layout of the data plane of P4Guard inside the P4-enabled switch. The P4 switch has an Apache Thrift RPC server to receive messages from the Thrift client with a P4 Runtime CLI interface, as pictured in Figure 1. In Figure 2, the two virtual interfaces (veth 0 and veth1) are connected to the end hosts. The base P4 switch has an ingress buffer and an egress buffer to process incoming/outgoing packets with a basic parser and deparser.

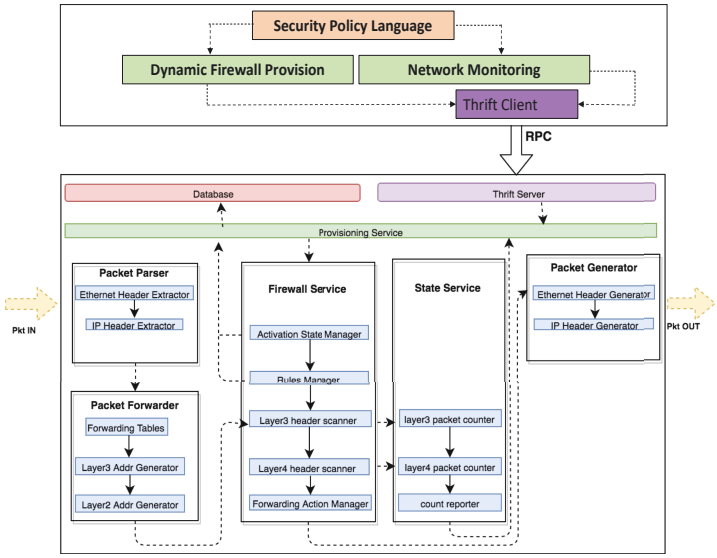

Fig. 1. P4Guard System Architecture

We redefine the software firewall with the new packet processing logic in the data plane of the switch. The dotted boxes in Figure 2 indicate the new table structures for the firewall data plane used to create the flow-table and flow-rule structures by using the syntax of P4 language. The target program is compiled to generate a JSON file including the parser, five new tables, and the deparser along with actions for each table and controlling blocks, which define packet-forwarding logic for each table. We upgrade the ingress pipelines in order to forward packets by applying firewall rules, and the egress pipelines send a frame to a destination through an egress port.

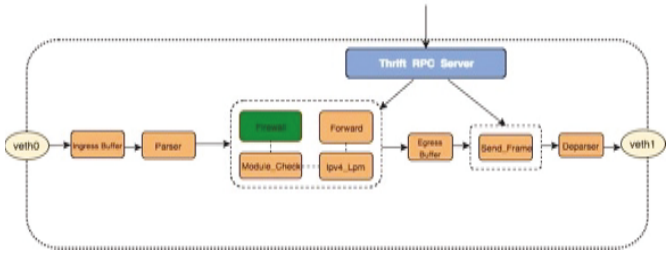

Fig. 2. *The data plane for P4Guard*

B. The control plane of P4Guard

The P4 controller is intended for dynamic firewall placement and networking monitoring. The controller communicates with the P4 switches through an Apache Thrift-based remote procedure call (RPC) channel. In other words, if the controller sends a high-level security policy language to the P4 switch, the policy language is translated into low-level firewall packet

forwarding rules for the P4 data plane. The controller is implemented using P4-Runtime CLI with various python scripts. The controller activates or deactivates the firewall remotely and provide a network monitoring function based on a counter to keep the statistical information in the data plane. Thus, the controller can detect various flooding attacks by counting the number of packets with a predefined threshold at ingress and egress points according to different protocols, which it transmits to the controller.

We define a high-level security policy language to specify each firewall rule without regard to the underlying infrastructure. Table 1 shows basic elements to be used for the security policy. In Table 1, a security policy is defined as < *ID*, T_a, T_{name}, *S*, *action*>. Note that *S* is a service with <IP_{src}, IP_{dst}, *P*, *p*>. T_a is an action to add/modify/delete an entry in the table in the data plane. There are many different tables (T_{name}) that we define in the data plane for the packet processing logic. *P* is a protocol type, such as IP, TCP, UDP or ARP. *IP* is a source or a destination IP address. *p* is a port number for a particular service. For example, *<1, table_add, firewall, 10.0.0.1, 10.0.0.2, TCP, 80, drop>* means that switch ID 1 needs to add a new firewall rule in the *firewall* table in the data plane. The specific firewall rule is to drop packets from the source address (10.0.0.1) to the destination address (10.0.0.2) with the port number 80. Based on the policy language, a user can define different security policy rules in a network and the controller transfers the policy to the switches to follow or to reconfigure the forwarding logic in the data plane.

The P4-controller is embedded with a Thrift software plugin. The controller converts the high-level security policy language into a P4 Runtime CLI by sending the RPC messages to the Thrift server, a remote P4 switch to push these specific security policy rules in the underlying P4 switch. The P4 switch continuously processes incoming messages and incorporates the firewall rule as well as flow rules into the forwarding data plane.

TABLE I. TABLE TYPE STYLES

	Definition	Details
T_a	Table actions	add/modify/delete
T_{name}	Table names	The tables in the P4 data plane
P	Protocols	IP/TCP/UDP/ARP
IP	IP address	A source or destination IP address
p	Port numbers	A set of ports in use.
action	Policy action	<allow | drop>
ID	Switch Data Plane ID	A set of the data plane ID

C. The data plane of P4Guard

We design a data plane for firewall services by using P4, called a software firewall. The software firewall consists of a packet parser, a packet forwarder, and a packet generator for packet processing logic and forwarding table structures, as in Figure 1. When a packet arrives in the data plane of P4Guard, the packet is parsed and matched with the existing flow rules. The packet is then either forwarded or dropped.

Packet Parser: The parser generally decodes the incoming packets in the switch ports. It defines the Ethernet and IP structures constructed from the corresponding packet-header values of the incoming packets. When packets come into the switch port, it first classifies packets into an Ethernet or IP header, which can be processed in the next step based on a finite state machine generated from the P4 program. The Ethernet packet header includes the Ethernet protocol type, layer2 source, and destination addresses. The IP packet header includes the IP version (note that we only support version 4 now), length, TTL, IP protocol type, and source and destination addresses.

Packet Forwarder for Firewall Service: The packet forwarder determines the next hop IPv4 and layer2 address based on the user-define forwarding-rules in the switch tables: *ipv4_lpm* and *forward*. The *ipv4_lpm* table consists of the switch port interconnection rules that mean to set IP address for next hop and its egress port. It checks the destination of the IPv4 address and sets the next-hop IPv4 address as well as the outgoing switch port as per the user-defined forwarding rules. The *forward* table includes the host interconnection rules to set a next hop of destination MAC addresses. It checks the next-hop IPv4 address for the packet, and sets the layer2 destination address (MAC address) for the next-hop.

For match-action tables for firewall services, the *firewall* table has multiple packet-forwarding tables with the match-action format. (a) The *firewall* table contains the firewall rules, which decide a particular action (*allow* or *drop*) for packets. (b) The *module_check* table in the activation manager continuously monitors firewall status to dynamically activate or deactivate the software firewall remotely. (c) The rule manager applies the packet-forwarding tables on incoming packets for layer 3 and layer 4 action tables. (d) The forwarding action manager determines the final destination of the packets based on the *send_frame* table for the packet generator. The *send_frame* table consists of host interconnection rules (a relationship of an output port with its source MAC address) to determine outgoing switch ports with the layer 2 address.

Packet Generator: The packet generator receives packets with metadata (i.e. standard metadata and routing metadata) for packet generation for egress pipeline. The packet generator decides the next-hop IPv4 address, and determine whether packets allow or drop. According to the results, the packet generator assembles the packet with a modified header, and sends out the determined output port in the switch. Lastly, the deparser sends the packets to the output port based on the current state of the parsed object.

Beside the basic packet processing logic for the firewall services, we create a *count* table for statistical information in

the data plane to keep recording the number of packets per rule or per flow. The packet counter is also associated with the forwarding rules in the firewall to control a rate limit. It keeps track of packets hitting a specific flow rule for each source-destination IPv4 address pair as well as each layer4 protocol type, such as ICMP, UDP and TCP. The counting information is used for network monitoring purposes to detect various flooding attacks by fetching it periodically from the switch to the controller.

IV. Analysis of P4Guard

We discuss the advantages of P4Guard while presenting the differences of VNGuard and P4Guard.

Full Programmability in the data plane: Both P4Guard and VNGuard implement a firewall, independent of hardware for high scalability. However, P4Guard is implemented by the high-level domain specific language (DSL) called P4. It consists of the controller for dynamic management and network monitoring and the firewall forwarding data plane for defining the full forwarding behaviors in the data plane without virtualization techniques. On the other hand, VNGuard is implemented by ClickOS to virtualize the firewall functions. Instead of defining the firewall behaviors in the data plane, VNGuard is an application to perform a firewall functions with the help of software switches like Open vSwitch. It just utilizes OpenFlow in Open vSwitch which might be efficiently defined in the kernel space for the forwarding behavior (i.e. firewall rules) in the data plane, which is not really related to firewall functions.

Configurability and Flexibility: P4Guard is a configurable protocol-independent software firewall which is inherited by P4. After defining the packet processing logic in the P4 data plane, any packets regardless of protocols or platforms are processed by the specific packet forwarding logic in the data plane. The logic can be updated with P4 compiler whenever we need to upgrade firewall functionalities. However, VNGuard handles packets according to the user-defined firewall rules which are interpreted into specific packet headers and actions of OpenFlow protocols.

P4Guard is a part of the data plane in the P4 switches that specifies the firewall rules into flexible and reprogrammable forwarding tables. It interprets high-level firewall policies into low-level forwarding rules in the data plane, which are then implemented according to the tables into the P4-enabled switches. By contrast, our previous VNGuard is executed in a guest domain in a virtualized domain as an application. A virtual firewall function runs in virtualization without involving the data plane. To use this virtual firewall, a SDN controller with Open vSwitch needs to push the firewall rules by using OpenFlow.

Easy Dynamic Placement: P4Guard easily and remotely activates or deactivates the software firewall with only one bit flag (on- or off- flag) in the *module_check* table of the activation manager in the control plane. The P4Guard control plane can remotely manage the dynamic deployment of the firewalls by issuing one command through the P4 runtime CLI. By contrast, VNGuard requires an efficient virtual resource management to spawn a set of new instances and to destroy them. For example, Xen hypervisor in ClickOS need to initiate VNGuard, and then the Xen hypervisor needs to manage computing resources to spawn the virtual firewall and return the used resource when the virtual firewall is destroyed. Thus, the efficiency of VNGuard depends on the virtualization techniques in the host domains managing virtual guest domains. Thus, P4Guard requires almost zero cost to activate or deactivate as many as firewalls, but VNGaurd needs intensive resource management cost to handle many virtual firewall instances.

V. Evaluation

A. Implementation

P4Guard is implemented with the behavioral model version 2, p4c-behavioral [13] (called bmv2) with the compiler P4_14. The P4 bmv2 simulator is installed using the publicly available P4 makefiles with all the dependencies and the necessary linux packages. We implemented a *firewall.p4* file that uses P4 code. This is compiled using the P4-compiler to obtain the *firewall.json* file, which is used by the *simple_switch* executable on startup [15]. To start the switch, the *simple_switch* executable is run on the bash and the p4-compiled JSON file is passed to it as a CLI argument. This P4 file consists of the standard switch forwarding table definition, firewall table definition, and match-action pairs of each type of table.

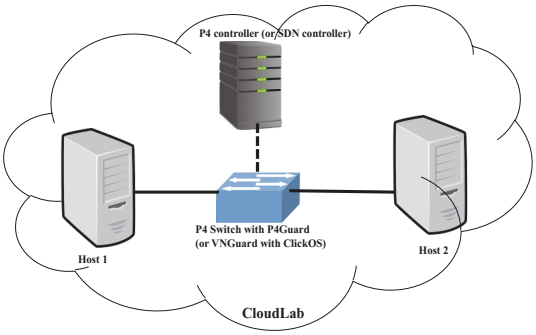

Fig. 3. *A network toplogoy in CloudLab for evaluating P4Guard*

We compare P4Guard to VNGuard [1] in CloudLab [16], a cloud-based open platform. Figure 3 shows a network topology for our experiments. We linearly connected three hosts (virtual machines) in CloudLab. The three hosts run on two Intel E5-2630 v3 8-core CPUs with 2.4GHz and 128 GB RAM and two 1.2 TB Disks. The link capacity is 10GB. The switch node is located in the middle position between two end hosts. The switch node is installed with P4Guard (P4 bmv2 simulator) or VNGuard (ClickOS). The switch node has bare linux Ubuntu 4.9.1 for P4Guard and Ubuntu 4.8.2 for VNGuard with Xen hypervisor.

B. *Experimental Results*

We evaluate P4Guard, comparing it to VNGuard in terms of packet processing time and roundtrip time using various settings in CloudLab.

Instance Initiation: We first measured the time to execute the software firewall (P4Guard) and the virtual firewall (VNGuard). Because P4Guard is a package of the P4 application after compiling, the deployment of the P4Guard consumed only 0.181 seconds. VNGurad took 0.4615 seconds to run one firewall network function on ClickOS. The time for VNGuard excluded the time it took to initiate the Xen Hypervisor execution time since the Xen Hypervisor was already installed for ClickOS. Additionally, the usage of CPU for VNGuard was 46.8 %, but P4Guard used only 0.3% of CPU for the execution of the software firewall. Therefore, P4Gurad was much more easily and quickly activated (or deactivated) than VNGaurd with very minimal overhead.

Processing Time: We evaluated the packet processing time for P4Guard and VNGuard as shown in Figures 3 and 4. In Figure 4, according the number of packets, we measured the packet processing time for P4Guard and VNGuard. The baseline indicated a regular switch function without P4Guard and VNGuard. When the number of packets was below 1,000, P4Guard had a faster packet processing time than VNGuard. However, over 1,000 packets, VNGuard had a faster processing time. Figure 5 shows the results for packet processing time at different transmission rates. Similarly, below the 10 MBps rate, P4Guard had a faster packet processing time than VNGuard. However, as the transmission rate increased, VNGuard had a faster packet processing time than P4Guard.

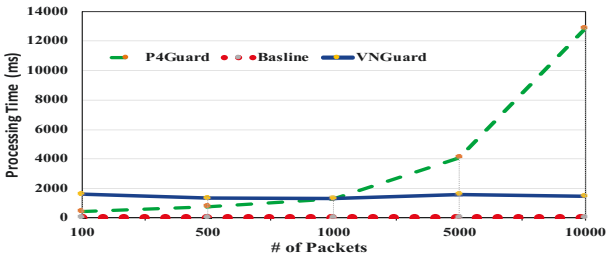

Fig. 4. *Packet processing time according to the number of packets*

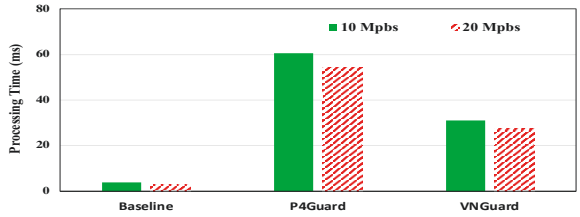

Fig. 5. *Packet processing time according to different transmission rates*

There are multiple reasons for the differences in processing time. While the p4c-behavioral model for P4Guard is not optimized yet in the packet buffering, the Open vSwtich for VNGuard is optimized in the kernel space. In other words, we ran our P4Guard on Linux, which might not have sufficient buffer space to hold many packets at one time. By contrast, VNGuard used Open vSwitch, which was already optimized in the kernel space for packet buffering and processing. If we implemented P4Guard inside the switch code instead of in an application domain, the faster processing time for P4Guard that were seen in experimental results for processing under 1000 packets would remain faster than VNGuard for higher packet numbers as well.

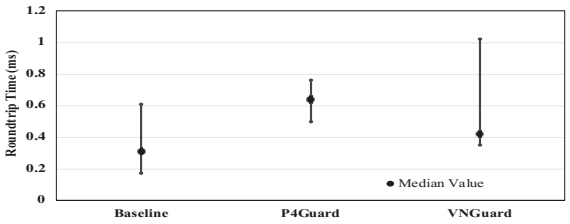

Fig. 6. *Roundtrip time with 64-byte packets*

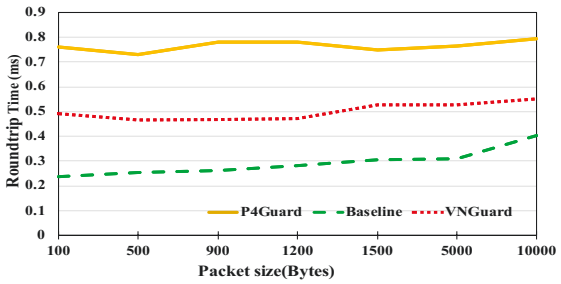

Fig. 7. *Roundtrip time with various packet sizes*

Roundtrip Time: Figure 6 shows the roundtrip time of Ping packets between the two end hosts with P4Guard or VNGuard. Each experiment was run ten times and Figure 6 shows the median value was selected instead of the average value with minimum and maximum values. The experiment also measures packet processing time for 64-byte packets. As shown in Figure 6, we can say that P4Guard has lower network latency in processing 64-byte packets. P4Guard generally showed a smaller network delay in roundtrip time than VNGuard. Figure 7 shows the roundtrip time increases as the packet size increases. The P4Guard showed less network delay than VNGuard for smaller packets and VNGuard performed better with large packet sizes. It is the same reason with the case of packet processing times as shown in Figure 4 and 5.

Rule Installation Time: We evaluated the installation time of firewall rules for P4Guard and VNGuard, as in Figure 8. The rule installation time for VNGuard was much faster than the time for P4Guard. This shows that the high-level firewall policy language in the P4 controller is not efficiently translated into the low-level policy rules in the P4-enabled switches. We need to develop a more cost-effective policy translation framework to improve the installation of firewall rules.

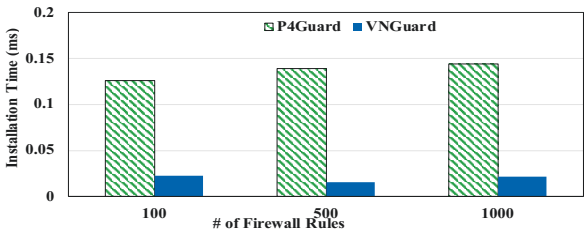

Fig. 8. *Installation time according to the number of firewall rules*

One of the reasons that P4Guard had slower processing time than VNGuard for large numbers and sizes of packets is that VNGuard handles the switching actions and the forwarding behaviors of the switch using the highly efficient Open vSwitch. On the other hand, to implement P4Guard, we used an experimental software switch (*p4c-bmv2*) with the old compiler (*P4_14*) and all processing was handled by the processor, thus the network buffers were limited. Thus, such issues might explain why P4Guard's packet processing times increased specifically with high transmission rates or large number of packets. The behavioral model (bmv2) of P4 for P4Guard is a single-threaded application, meaning it processes the ingress and egress packets using a single thread. Hence, P4 is not able to process two-way traffic in parallel for high throughput and is not able to fully utilize system resources.

We observed that P4Guard is faster in processing packets when there is a relatively small number of packets and small sizes of packets. Therefore, we need to optimize our implementation to maximize the performance of P4Guard for all cases. First, we can optimize P4Gaurd with the new release compiler (p4c-bm2-ss with P4_16). We tested the performance of the previous and updated compilers with the *simple_router.p4* example in the GitHub [15]. Following the experimental design shown in Figure 3, we verified that the new compiler performed almost 30% better than the old compiler in processing packets at a speed of 10 Mbps. Additionally, P4Gaurd has many advantages discussed in Section IV.

VI. Conclusion

This paper proposes a new software firewall without virtualization. In P4Guard, we defined the forwarding behavior for the firewall in the data plane to parse and deparse packets according to header information and predefined policies. To implement the software firewall, we defined various tables in the data plane to process packets according to the firewall rules. We compared the designed new software firewall to VNGuard. The experimental results demonstrated that P4Guard showed faster packet processing time and lower network latency when the packet size was small. As future work, we will modify the software firewall to consistently yield high-speed processing time by developing various optimization techniques for the design of our P4 software-based firewall.

Acknowledgment

This work was supported by NSF SaTC #1723804. Dr. Park is a corresponding author.

References

[1] Deng et al., "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," In Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015.

[2] D. Hu et al., "Design and Demonstration of SDN-Based Flexible Flow Converging with Protocol-Oblivious Forwarding (POF)," In Proceedings of IEEE Global Communications Conference (GLOBECOM), San Diego, CA, 2015.

[3] Hamid Farhadi, Ping Du, and Akihiro Nakao, "User-defined actions for SDN," In Proceedings of The Ninth International Conference on Future Internet Technologies (CFI '14). ACM, New York, NY, USA 2014.

[4] Sándor Laki, Dániel Horpácsi, Péter Vörös, Róbert Kitlei, Dániel Leskó, and Máté Tejfel, "High speed packet forwarding compiled from protocol independent data plane specifications," In Proceedings of the ACM SIGCOMM Conference (SIGCOMM '16). ACM, New York, 2016.

[5] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford, "PISCES: A Programmable, Protocol-Independent Software Switch," In Proceedings of the ACM SIGCOMM Conference (SIGCOMM '16). ACM, New York, NY, USA 2016.

[6] Barefoot Networks. https://www.barefootnetworks.com/resources/

[7] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker, "P4: programming protocol-independent packet processors," SIGCOMM Comput. Commun. Revview, 2014.

[8] David Hancock and Jacobus van der Merwe, "HyPer4: Using P4 to Virtualize the Programmable Data Plane," In Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies (CoNEXT '16). ACM, New York, NY, USA 2016.

[9] Haoyu Song, "Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane," In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13). ACM, New York, NY, USA 2013.

[10] G. Brebner. "Programmable Target Architectures for P4," In the 2nd P4 Workshop by Stanford/ONRC, 2015.

[11] J. T¨onsing, "P4/PIF + C Programmable Intelligent NICs: Requirements and Implementation Notes," In the 2nd P4 Workshop by Stanford/ONRC, 2015.

[12] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici, "ClickOS and the art of network function virtualization," In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14). USENIX Association, Berkeley, CA, USA 2014.

[13] P4 Language Consortium. Behavioral Model (bmv2) https://github.com/p4lang/behavioral-model.

[14] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav,Nate Foster, and Hakim Weatherspoon, "P4FPGA: A Rapid Prototyping Framework for P4," In Proceedings of the Symposium on SDN Research (SOSR '17).ACM, New York, NY, USA 2017.

[15] Simple_router.p4. https://github.com/p4lang/tutorials/blob/master/SIGCOMM_2015.

[16] CloudLab. https://www.cloudlab.us

[17] Network Virualization and Security Platform. https://www.vmware.com/products/nsx.html

[18] T. Benson, A. Akella, A. et. al., "CloudNaaS: Networking Platform for Enterprise Applications," In Proceedings ACM SOCC, June 2011.

[19] H. Ballani, P. Costa, et. al., "Towards Predictable Datacenter Networks," in Proceedings ACM SIGCOMM, August 2011.

[20] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architectur with Bandwidth Guarantees," in Proceedings of the ACM CoNEXT, December 2010.

[21] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam, "Revisiting traffic anomaly detection using software defined networking," In Proceedings of international conference on Recent Advances in Intrusion Detection (RAID), Springer-Verlag, Berlin, Heidelberg 2011.