International Journal of Computer Engineering and Information Technology

VOL. 10, NO. 6, June 2018, 85–91
Available online at: www.ijceit.org
E-ISSN 2412-8856 (Online)

OpenFlow Switches with Integrated Tiny NIDS to Mitigate Network Attacks

Binh Tran-Thanh[1], Cuong Pham-Quoc[2] and Tran Ngoc Thinh[3]

Faculty of Computer Science and Engineering, Ho Chi Minh city University of Technology, Vietnam

{[1]thanhbinh, [2]cuongpham, [3]tnthinh}@hcmut.edu.vn

ABSTRACT

In the IoT era, billions of devices connect to the Internet and generate a vast amount of data across the network. However, the traditional networks cannot handle those volumes of data as well as connections because of its limitations. To deal with that issue, Software Defined Networking (SDN) has been introduced as an emerging solution. Similar traditional networks, SDN suffers from challenges. In this paper, we propose an approach to mitigate the network attacking by integrating tiny Snort-based Network Intrusion Detection Systems (NIDS) into OpenFlow switches, instances of SDN. Our proposed architecture is able route packet according to Open Network Protocols (ONP) to detect instruction to protect the network from attacking. To prove the feasibility of the proposed approach, we use Verilog Hardware description language (HDL) to implement the system on a NetFPGA 10G board. In addition, we design two testing scenarios for the throughput and accuracy experiments. The experimental results show that our system runs at up to 105.854 MHz, handles the traffic rate up to 9.809 Gigabit per second (Gbps), and detects up to 62% of network threats.

Keywords: *Network Intrusion Detection Systems, Software-Defined Networking, OpenFlow Protocol, Snort, FPGA.*

1. INTRODUCTION

According to the report of Cisco Internet Business Solutions Group [1], there will be 50 billion devices connected to the Internet by the year of 2020. In other words, a huge number of connections along with big data traveling on the Internet for every second in the future. With current traditional network infrastructures, where are complex to change the routing flows, hard to expand the network, and inconsistence between devices in the network, it is difficult to handle the vast volumes of data and connections. To overcome those issues, the SDN has been introduced as an emerging approach. Unlike traditional networks, which are decentralized control, SDN is centralized control [2]. Basically, SDN divides network become three planes: applications, centralized controller, and data planes. With centralized control, it makes the network more flexible to modify flows and connections in data plane depending on the status of the current traffic. Besides, it is easier to expand, apply the policy, and keep the consistency the network.

Like traditional network, SDN suffers from the several challenges which may affect systems. One of the key challenges is network attacking, especially DDoS. On the one hand, the centralized control brings a lot of benefits to the network. On the other hand, it is also the weakest point of SDN. When the SDN controller is down, the operation of a network may crash the system completely or slow down performance. Furthermore, network attackers launch attacks on SDN with different perspectives [3]. In application and controller layers, the attackers can directly attack controller, applications, and the interface between controller and applications to manage and control the network. In term of infrastructure (data plane), attackers exploit either Switch Flow tables or the OpenFlow protocol (connection between controller and OpenFlow switches) to attack the system by sending a huge amount of strange traffic to OpenFlow switches. Since OpenFlow switches receive those packets, it causes exhausting the switch flow tables or sends all those packets to controller then cause network attacking.

To the best of our knowledge, the combination of SDN and NIDS has not been well discussed in the previous works. In this paper, we propose an approach to mitigate network attacks, which try to attack to centralized control plane through data plane, by embedding tiny NIDS inside OpenFlow switches. Along with, the functionalities of OpenFlow switch, we develop an architecture which includes Decode module, Scan (Header and Payload) engines, Management and Report module inside OpenFlow switches to detect network attacking and threats. In the proposed architecture, the Decode module parses incoming packets (TCP and UDP types) into signature features based-on ISO model. The Scan (Header

and Payload) engines are key cores of the NIDS. They scan the decoded features of the incoming packets base on Snort rules, then issuing decisions depended on the scanned results. The Management and Report modules take over for reporting malicious packets to network administrators whenever the Scan engines detect the threat. To implement the system, we use Verilog HDL and a NetFPGA 10G board. In order to prove the throughput and accuracy of the system, we provide two testing scenarios for experiments. After that, we collect and analyze the results. The experimental results show that the system runs up to 105.854 MHz, handles the traffic rate up to 9.809 Gb/s, and detects up to 62% of network threats.

The main contributions of this paper are listed as below:

- A proposed architecture to mitigate network threats on SDN by integrating the Snort-based NIDS into OpenFlow switches using reconfiguration hardware.
- Our system within tiny NIDS processes large volume of traffic at speed of 9.809 Gb/s. Furthermore, our system detects up to 62% out of the predefined rules of the network attacks and threats.

The rest of this paper is organized as follows: The next section presents the background and published literature related to defense mechanisms against network attacks. Our proposed architecture is illustrated in section 3, afterward the implementation and experimental results are discussed in Section 4. Finally, Section 5 concludes our studies and introduces future works.

2. BACKGROUND AND RELATED WORK

In this section, we present NIDS background, introduce Snort, then discuss the advantaged and disadvantaged of the previous researches in network securities.

2.1 Background

The main functionalities of NIDS [4], [5] are to monitor network or system and report to network operators whenever the malicious actions and violations found. There are several approaches for packets classification. The two most popular ones are signature-based and anomaly-based detections. This paper focuses on signature-based detection.

Snort [6] is an open-source signature-based intrusion detection/prevention system (ID/PS). Moreover, Snort is a powerful tool with accurately threats detection at high speed of traffic, rapid response by updating rules frequently, and high adaptability by adding new modules to applications. In addition, with over 5 million downloads and over 600,000 registered users, it is the most widely deployed intrusion detection/prevention system in the world. Snort is rule-based, and each Snort rule contains two parts: rule header and rule options. An example of rule header is alert *"tcp $EXTERNAL_NET $HTTP PORTS → $HOME_NET any"*. Rule header comprises seven fields: actions, which tell Snort what to do when packets match the rules, protocol, Source address(es), source port(s), direction (ingress or egress), Destination address(es), and destination port(s), respectively. An example of rule options is *(msg:"PROTOCOLFTP ADMw0rm ftp login attempt"; flow:to server,established; content:"USER"; nocase; content:"w0rm"; distance:1; nocase; pcre:"/U^ SERns+w0rm/smi"; metadata:ruleset community, service ftp; classtype:suspicious-login; sid:144; rev:16;)*. Where *msg* is meaningful message to output to network user/operator; *flow* indicates the direction of connections; *content* is specific packet payload which uses as pattern; *PCRE* (PERL compatible regular expressions) is also the pattern to describe packet payload; metadata is source of snort rules; *classtype* tells operator what happened to system after attacks launched successful; *SID* represents for Snort ID; and *ver* is the version of Snort rules. Once rule options have more than two related contents, then distance/offset and within/depth keywords are used to constrain the relationship between the contents.

2.2 Related work

This section introduces several SDN security threats and its countermeasures.

The survey in [3] has shown that there were many network attacks can be launched on SDN with different methods such as spoofing, tampering, repudiation, information disclosure, Denial of Service (DoS), and Elevation of Privilege. All of them have been well addressed in that survey. Each attacking mechanism has several countermeasures to deal with. This paper presents some countermeasures for spoofing and DoS/DDoS. For other countermeasures, read in [3]. For ARP spoofing, which injects fake MAC address into legitimate packets, Matias et al [7] proposed an Address Resolution Mapping (ARM) module for tracking MAC address of authorized users or hosts in the network by the controller. If ARP responses are not verified, then they are discards from the system.

For IP spoofing, which generates packets with fake/unauthorized source addresses, IP address validation methods are one of the solutions. Internet Engineering Task Force (IETF) formalized a standard for Source Address Validation Improvement (SAVI). SAVI checks addresses of packets based on a binding validation. Yao et al [8] proposed OpenFlow Virtual source Address Validation Edge (VAVE) expanding from SAVI to figure out the address resolution problem. Integrate VAVE inside a controller to verify packets, which are non-existed in OpenFlow tables.

For DDoS attacks, which are the most serious challenge, Braga et al [9] addressed a lightweight method to distinguish normal packets from DDoS flooding packets in SDN using Self Organizing Maps (SOM). Braga reused packet features in [10] by Feng et al, including Average Number of Packets in Per Flow (ANPPF), Average of Bytes per Flow (ABF), Average of Duration per Flow (ADF), Percentage of Pair-Flows (PPF) and Growth of Single-Flows (GSF). Those parameters are continuously measured to detect potential DDoS.

To prevent DoS attacks, The Avant-Guard system proposed by Shin et al [11] as an enhancement to OpenFlow, showed that it is possible to handle DoS attacks and eliminate their negative impact on the network. All above approaches embed the security mechanism on control plane, hence the controller degrade performance while it is already overwhelmed with heavy tasks. Moreover, the mentioned approaches try to protect the network threats, which directly attack control plane. Nevertheless, fewer solutions protect attacks to control plane through data plane. In additions, almost implementations of solutions are software-based. It means the countermeasure handles traffic at high speed hardly because of software-based. In this paper, we address a novel architecture for OpenFlow switches to detect network attacks in advance using reconfiguration hardware.

3. SYSTEM ARCHITECTURE

This section presents the deploying of the system in industrial environments, and then illustrates the details architecture Fig. 1. Deploying system in the industry to protect the organizations from the network threats and attacks.

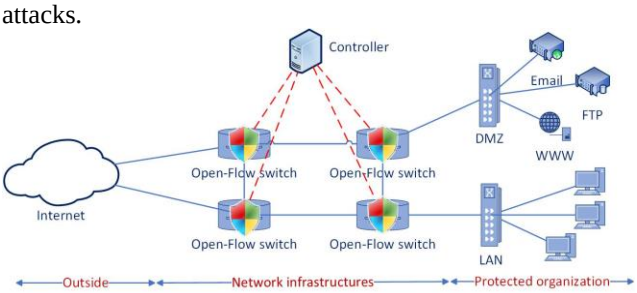

Fig. 1. Deploying system in the industry

As mentioned in section 2, when network attacks come to the targets (control planes in this case) via data plane, there are huge amount of new packets/flows sent to control plane and cause exhausting OpenFlow tables or DoS. To overcome those situations, the tiny Snort-based NIDS is embedded inside each OpenFlow switch to detect/prevent in advance the network attacks. Figure. 1 demonstrates the deploying of the system a real network. The organizations are protected from attacks by integrating the Snort-based NIDS inside each OpenFlow switch. The next paragraphs describe the architecture in detail. The proposed architecture consists of three main components: Decoder, NIDS core, and OpenFlow tables as shown in Figure. 2.

3.1 Decoder

This component collects all incoming packets to the system from the Internet. After that, the packets are decoded into predefined fields based-on Snort rules. The decoded fields then are forwarded to the NIDS core to detect harmful traffics, meanwhile the original incoming packets are stored into buffers to forward to OpenFlow tables.

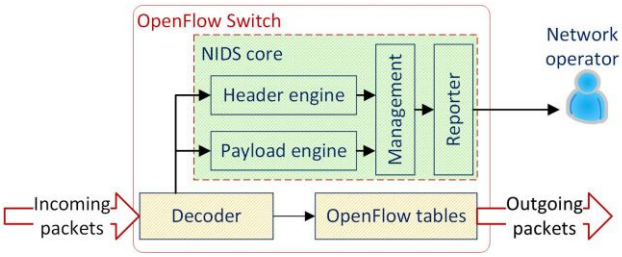

Fig. 2. The block diagram of the system inside OpenFlow Switch

3.2 NIDS core

The functionality of this component is to classify the malicious packets from incoming traffics. The scan (Header and Payload) engines are signature-based, therefore the input of NIDS cores are the features of packets such as source/destination address/port, protocol, and payload of packets. After receiving the decoded fields, the Header engine checks input fields and figures out the header rule ID while the Payload engine looks inside the payload of packets to detect payload rule ID. The Rule IDs issuing from the Header and Payload engines are forwarded to the Management module to synthesize the final rule IDs. If the header rule ID matches Payload rule IDs, then the final Rule IDs are asserted and sent to the Report module, since the system marks those packets as malicious. On the contrary, the system considers those packets as normal ones.

3.3 OpenFlow tables

This component contains OpenFlow entries to forward packets to destination. After receiving original packets from the Decoder component, the system extracts information from packets and looks up information in OpenFlow tables. If the packets information match to matched fields of the OpenFlow entries, the packets are processed by set of instructions of those entries such as forward, modify, etc. Otherwise, the system sends those packets to controller to process depended on the policy of the organization.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we describe the implementation of our system. After that, in order to prove the accuracy and throughput of the system, we provide two testing models for experiments.

4.1 Implementation

There was a project [12] which implemented OpenFlow switch on NetFPGA 10G board. Therefore, we decide to inherit that project to develop the tiny NIDS inside OpenFlow switches. The NetFPGA 10G board [13] contains four SFP+ ports, Xilinx Virtex-5 TX240T, 27 Mbytes Quad Data Rate Static Random-Access Memory (QDRII SRAM), and 288 Mbytes Reduced Latency Random Access Memory (RLDRAM). Four SFP+ ports are suitable to build network applications. Besides, Xilinx Virtex-5 TX240T provides powerful hardware to handle huge amount of traffic in the Internet. Moreover, with 27 Mbytes QDRII SRAM and 288 Mbytes RLDRAM, the system creates space for packet storages. Figure. 3 describes the details architecture of the system using NetFPGA 10G board. The following paragraphs present the implementation of each module with Verilog language:

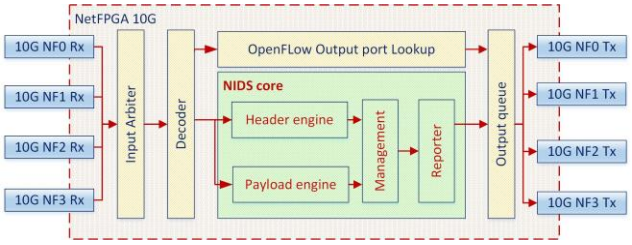

Fig. 3. The Architecture of the system

Input Arbiter module: This module is provided in [12], a NetFPGA project by Stanford University. It is responsible for collecting all incoming packets to the system via AXI interface from four SFP+ ports, and then forwarding all packets to Decoder module.

Decoder module: The main functionalities of this module are to parses the packet into the signature features based-on the Snort rule fields and to deliver the decoded features to the NIDS core. First, Decoder module checks the type of incoming packets. If the packets are neither UDP nor TCP, the Decode module ignores those packets. Otherwise, this module extracts packets information based-on the OSI network model. In term of packet headers, the Decoder module extracts source/destination IP/port and protocol from the packets then stores all decoded fields into Header FIFO, which is provided by Xilinx. In the same way, this module decouples payload from the packets and storing in Payload FIFO. Afterward, the Header and Payload FIFOs send all decoded fields to Header and Payload modules accordingly. when decoding packet, there is a queue to store all the original packets to forward to OpenFlow Output Port Lookup module.

NIDS core: This module contains 4 main submodules: Header engine, Payload engine, Management, and Report. It receives signature features and payloads of packets as inputs, then scans them based-on Snort rules to figure out the rule IDs. After that, this module reports the rule IDs to network operator via a SFP port. Due to hardware resources challenges, we do not implement all Snort rules in a single NetFPGA 10G board. Therefore, we select 1000 Snort rules to integrate inside OpenFlow switches, the remaining rules are installed in other devices (i.e. power PC) for later securities.

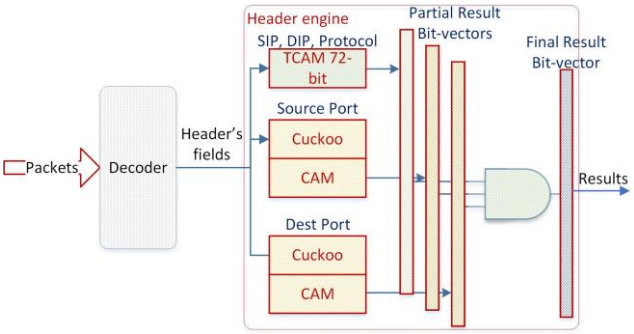

Fig. 4. The detailed Header engine blocks diagram

Header engine submodule: As discussed in Section 2.1, the Snort rules consist of 2 parts: rule header and rule options. This module charges for rule header. Inputs of the Header module are five header fields (source/destination address/port and protocol) and output is the 256-bit vector which represents for header rule ID. After receiving the header features from the Decoder module, this module separates them into 3 submodules as depicted in Figure 4. Source/destination IP and protocol are look up in TCAM 72-bit to produce partial result Bit vector. In term of source ports, the Cuckoo hashing [14] functions take source ports as keys, and then query the hashed results in CAM to get bit vector results for source ports. In like manner, destination has same mechanism, but CAM stores the destination port data. The final header bit vector is combination of the results of three above parts.

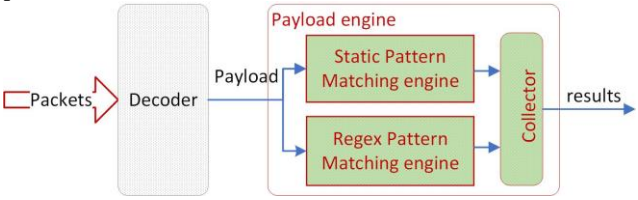

Fig. 5. The detailed Payload engine blocks diagram

Payload engine submodule: While Header engine submodule scans header fields to figure out rule header ID, this module charges for rule options (representing for packet payload). Input of this submodule is payload in byte and output is 10- bit payload rule IDs. As previously mentioned in section 2.1, rule options contain "*contents*" and/or "PCRE" keywords. Therefore, the Payload engine submodule splits input payload into two flows as shown in Figure 5. The Static Pattern Matching Engine (SPME) detects the static payload while the Perl Compatible Regular Expressions (PCRE) is scanned by Regex Pattern Matching Engine (RPME). The Collector submodule receives both results from SPME and RPME, then makes a final payload rule IDs.

Management submodule: The main function of this submodule is that it synthesizes the final rules IDs from header and payload engine submodules. After receiving header bit vector and payload rule IDs, this submodule compares payload rules ID to converted header rule ID. If the converted header rule ID and Payload rule IDs match, it means the packets are malicious actions or violations, then Management sends a signal and its final rule IDs to Report module to trigger the alerts. If the converted header rule ID does not match any payload rules IDs, the system considers those incoming packets as normal traffic and warming signal is not sent to the Report module.

Report submodule: This submodule uses for warning network operators whenever the system detected malicious actions or violations. After receiving the rules ID and warning trigger from the Management, this submodule encapsulates the specific packet, which is combination of the incoming packets information and rule IDs, then sends that packet to network operator for logging or preventing network attacks. OpenFlow Output port Lookup module: This module is part of OpenFlow switch projects [12] which is provided by Stanford University. Packets after passing Decoder module, are forked into two flows. The first one is decoded to get signature features and driven to NIDS core to detect threats or harmful traffic. The second one is forwarded to this module for switching to destinations based-on the OpenFlow rules.

Output Queue module: Opposite to Input Arbiter module, this module drives the outgoing packets to SFP+ ports to travel to destinations.

Table 1: The device utilization summary (estimated value) of the system

Logic Utilization	Used	Available	Utilization
Number of Slide Register	68,550	149,760	45%
Number of Slide LUTs	61,515	149,760	41%
Number of fully used LUTT-FF pairs	42,912	87,153	49%
Number of bonded IOBs	67	680	9%
Number of Block RAM/FIFO	283	324	87%
Number of BUFG/BUFGCTRLs	18	32	56%
Clock speed	105.854 MHz		
Power	10.015W		

We use Xilinx ISE design suite version 13.4 to synthesized and generate the bit-file for our implementation. Table 1 shows hardware resources usage of the system. It is observed that our system runs at 105.854 MHz and takes 10.015 Watts of power, utilizes 45% Slides Registers, 41% Slide LUTs, and 87% Block-RAMs/FIFOs.

4.2 Experimental environments

To measure the throughput and accuracy of the system, we provide 2 testing models as shown in Figure 6, 7, respectively. In throughput scenario, we use three NetFPGA 10G boards and Open Source Network Tester (OSNT - a software tool for networking). OSNT [15] is fully open source packet generator and monitor. The OSNT inside the NetFPGA 10G board can generate/monitor packets up to speed of 40 Gbps for all packet sizes. The first board is our system which implement tiny NIDS inside OpenFlow switches. The second NetFPGA board uses for both packets generator, which generates packets at high speed, and packets monitor, which captures the throughput of the system by using OSNT. The NetFPGA 10G (2) board connects to our system via four SFP+ ports. In term of accuracy scenario, Figure. 7 describes the resource use for testing. It includes a NetFPGA 10G board, a router, four computers, and a server. Our system is inside NetFPGA 10G board while four computers, where three of them act as network attacks and the remaining one represents for normal user, connect to the system through the router. The server connects to the system through port SFP+ 1 to receive the report of network attacks.

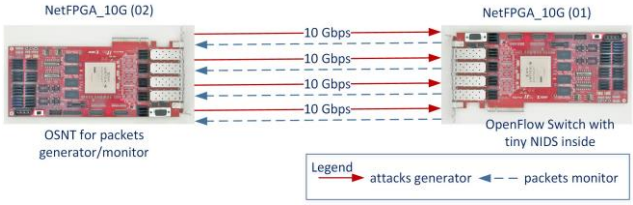

Fig. 6. The throughput testing model

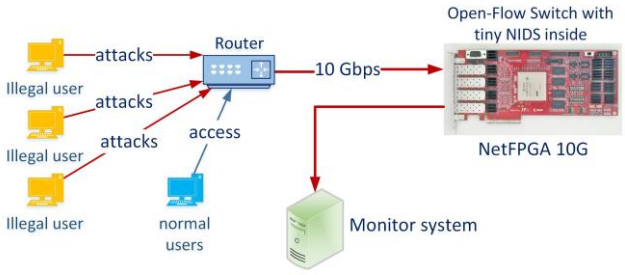

Fig. 7. The Accuracy testing model

4.3 Experimental results

For the throughput experiment, the scenario is designed to test the speed of decoding and scanning engines. According to the snort rules, we prepare the packets which contain the rules with various of packet lengths (64, 128, 256, 516, 1024, 1500 in byte). From packets generator side, NetFPGA 10G (02), OSNT generates and sends the packets to the system at the maximum speed of OSNT generator. From packets monitor side, NetFPGA 10G (02) board captures the outgoing packets from the system then displays connections statistic out to OSNT monitor user interface (UI).

Table 2: The packet classification statistic

		Desirable decision of the system	
		Positive	Negative
Our system decision	Positive	620	0
	Negative	380	1000

In the accuracy experiment, we test the precision of combination of two filters: Header and Payload engines. we use the same dataset, which uses for the throughput experiment, to test the precision of the system. First, the computers send the 2000 packets to the system, half of them is legitimate traffic and the remaining half represents for network attacks packets. The engines then scan the incoming packets, afterward reporter sends the information to server whether the engines detected the network threats. After collecting all reports from the system, we analyses accuracy, precision, reliability, and false negative rate as mentioned in [16]

In our experimentation, The Figure. 8 shows throughput values according to the packet sizes. The vertical, horizontal axial show throughput value in Gbps and in byte, respectively. As the results in Figure. 8, the throughput of the experimental system achieves 9.809 Gbps. For 64-Byte packets length test-cases, the throughput of the Decode and Scans modules reach 9.809 Gbps. However, the figures of the throughput of decoding and scanning drops sharply to 1.299, 1.289, 1.28, 1.2751, 1.266 for 128-Byte, 256-Byte, 512-Byte, 2014-Byte, 1500-Byte, respectively. There are some reasons that affect the throughput when increasing packet sizes. First, the system scans in every payload of packet. The more packet lengths the system receives the more time-consuming the system spends. Second, when the system scans packets, other packets come to the system. That requests sufficient buffer to store those packets. Due to hardware resource issues, the system cannot store all incoming packets. In other words, some packets are dropped out of queue, and some packets wait for process, it makes the throughput of the system decline.

The Table 2 shows the statistical values of accuracy test of the system. We send a total of 2000 packets to the experimental system. A half of those packets is legitimate packets and the remaining half represent for network threats. The experimental system detects 620 out of 1000 legitimate packets are legitimate, detects 1000 out of 1000 normal packets are normal, detects 380 out of 1000 legitimate packets is normal, and 0 out of 1000 normal packets is legitimate. According to results shown in Table 2, the precision is 62% while the accuracy is 81%, the reliability is 38% and 100% is the false negative rate.

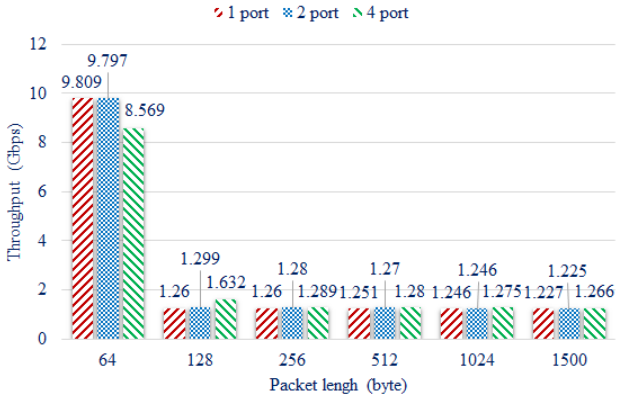

Fig. 8. The throughput testing results with different packets sizes

5. CONCLUSIONS

In this paper, we propose an architecture for mitigating network threats on SDN by integrating Snort-based NIDS into OpenFlow switches. Our implementation in NetFPGA 10G board runs at 105.854 MHz and handles the traffic at high speed and detects up to 62% of network threats. However, because of hardware resources issues, so the number of the rule in the system is small, and the complexity of the embedded rules is simple. In addition, the throughput of the system needs to improve because of serially data payload processing. In future, we consider integrating more network threats countermeasure such as TCP SYN defense, anomaly detection, etc. to improve the range of protection.

6. ACKNOWLEDGMENT

This project is funded by Vietnam National University Ho Chi Minh city under grant number B2016-20-02

REFERENCES

[1] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” vol. 1, pp. 1–11, 01 2011.

[2] O. N. Foundation. (June, 2014.) Sdn architecture. [Online]. Available: https://www.opennetworking.org/images/stories/downlo

ads/sdnresources/technical-reports/TRSDN-ARCH-Overview-1.1-11112014.02.pdf

[3] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," Computers and Security, vol. 53, pp. 79 – 108, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S01674 0481500070X

[4] A. Sundaram, "An introduction to intrusion detection," Crossroads, vol. 2, no. 4, pp. 3–7, Apr. 1996. [Online]. Available: http://doi.acm.org/10.1145/332159.332161

[5] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," IEEE Communications Surveys Tutorials, vol. 16, no. 1, pp. 303–336, First 2014.

[6] Snort. (2018) Snort. [Online]. Available: https://www.snort.org/

[7] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo, "Implementing layer 2 network virtualization using openflow: Challenges andsolutions," in 2012 European Workshop on Software Defined Networking, Oct 2012, pp. 30–35.

[8] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with openflow/nox architecture," in 2011 19th IEEE International Conference on Network Protocols, Oct 2011, pp. 7–12.

[9] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito, "Lightweight ddos flooding attack detection using nox/openflow," in Proceedings ofthe 2010 IEEE 35th Conference on Local Computer Networks, ser. LCN'10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 408–415. [Online]. Available: http://dx.doi.org/10.1109/LCN.2010.5735752

[10] Y. Feng, R. Guo, D. Wang, and B. Zhang, "Research on the active ddos filtering algorithm based on ip flow," in 2009 Fifth International Conference on Natural Computation, vol. 4, Aug 2009, pp. 628–632.

[11] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for softwaredefined networks," in In Proceedings of Network and Distributed Security Symposium, 2013.

[12] T. Y. Jad Naous, David Erickson. (2013) Open flow project. [Online]. Available: https://github.com/NetFPGA/netfpga/wiki/OpenFlowNetFPGA100

[13] Xilinx. (Jan 2018) Netfpga 10g. [Online]. Available: http://netfpga.org/site/#/systems/3netfpga-10g/details/

[14] R. Pagh and F. F. Rodler, "Cuckoo hashing," J. Algorithms, vol. 51, no. 2, pp. 122–144, May 2004. [Online]. Available: http://dx.doi.org/10.1016/j.jalgor.2003.12.002

[15] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Mckeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski, "Osnt: open source network tester," IEEE Network, vol. 28, no. 5, pp. 6–12, September 2014.

[16] S. T. Zargar, J. B. D. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," IEEE Communications Surveys and Tutorials, vol. 15, pp. 2046–2069, 2013.