Detection of DDoS in SDN Environment Using Entropy-based Detection

Tamer Omar, Anthony Ho, Brian Urbina,
Department of Electrical and Computer Engineering, California State Polytechnic University, Pomona
Email:tromar@cpp.edu

***Abstract*—Software-defined networking shifts the current paradigm of network infrastructures by providing a central control layer, improves network management, and implements programmability for flexibility. However, recent studies have shown the vulnerabilities that emerge within this architecture that can prove detrimental to the overall network infrastructure. In this work we analyze the effects of Distributed Denial of Service attacks on a software-defined networking environment and proposes an entropy-based approach to detect these attacks. The study uses the flexibility of OpenFlow protocol, and an OpenFlow controller (POX) to mitigate the attacks upon detection. Initially, through simulation the results of the detection algorithm was observed, and then implemented into a small-scaled network test bed, and finally the results of the proposed algorithm were presented and analyzed.**

***Index Terms*—SDN, OpenFlow, POX, Entropy, DDOS.**

I. Introduction

As technology matures, traditional networking is slowly transitioning to Software-Defined Networking, or SDN. It is found that SDN has a lot of benefits because of it provides programmability in networks, which helps contain operational costs and enable business growth. However, security is one of the limiting factor that is preventing real world SDN deployment. Although SDN is adopted by large web scale providers like Google, Amazon, and ATT, it is not adopted on a large scale by enterprises due to the lack of security solutions, standardization, and low level of maturity of SDN. Using the granular control provided by SDN, the security solutions need to be developed as to encourage the adoption and use of SDN.

The current networking paradigm involves switches, routers and gateways where these networking devices constitute both logical thinking as well as routing of packets. Traditionally the network administrator is responsible for configuring and managing these devices manually and at all times, which makes it a tedious task. Although these traditional networks are widely-used and popular, they have several drawbacks. Traditional networks are not programmable, which means they are static and inflexible network. They possess little agility and flexibility during deployment. SDN can remove the drawbacks of traditional networking. SDN is all about bringing programmability, automation, and superior control in the network to increase scalability and flexibility. Unlike traditional networks, the processing of the packets is not done by the switches. The architecture of SDN decouples the network control plane from the data or forwarding plane which consists of network devices forwarding traffic based on the control plane policy as shown in Fig.1.

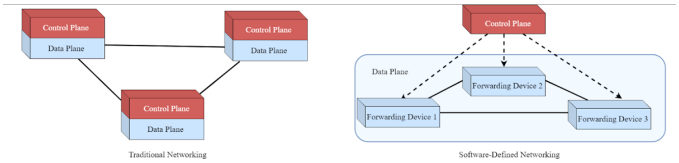


Figure 1: Traditional vs Software-Defined Networking

For the control plane and the data plane to communicate, OF is one of the protocol used to establish the communication between the separated data plane and control plan in an SDN environment. The controller is a software running on a server which acts as the network operating system of the SDN. The devices in the data use secure transport layer to communicate securely with the controller using the OF protocol. When a packet arrives at a switch, the header of the packet is checked with the fields in the flow entries. If a match is found, then the corresponding action associated with the flow entry is executed. Otherwise, the packet is forward to the controller to make the next decision and processing. The controller will determine if the packet will be forwarded by the switch, or it will be dropped. With this in mind, the controller plays a big role in the SDN.

The logically centralized controller can lead to many security challenges. Without the controller, the whole SDN architecture is lost. One of the main reason for a controller to be become unavailable is due to cybersecurity attacks such as DDoS. If a DDoS attack is launched, every incoming packets will be sent to the controller for processing which will exhaust the computing resources of the controller. Thus, the controller can become unavailable for processing of legitimate packets. To address the vulnerabilities in the SDN controller, the goal of this project is study and utilize a statistical method, Entropy, to address and recognized the difference between a normal and malicious network traffic. The effectiveness of attack detection solution will be studied and realized. The best solution will be proposed and implemented in our SDN architecture. A server will be running the POX controller, one client as the malicious user who be responsible of sending the DDoS attacks, and another client will be the victim of the attacks.

II. Related Work

As mentioned previously, the most vital component in SDN is the controller, which resides in the control plane of the SDN architecture. The separation of the control plane and

data plane in the SDN architecture allows the application plane to focus on developing network service applications that can utilize network resources provided by the control plane. Communication between the controller and forwarding devices in the data plane is provided by a control-data plane interface such as OpenFlow(OF), a growing standard within the SDN community. In order to communicate, both the controller and forwarding devices (i.e. switches) must support OF. OF switches contain and utilize what are known as flow tables for packet lookup and forwarding. The use of flow tables is the fundamental backbones between the OF switch and controller relationship. These flow tables contain flow entries which consists of rules, statistics and actions regarding the packet in question [1]. Likewise, OF enabled controllers include their own flow table and flow entries regarding the current state of the network which OF switches rely on for decision making events and updating their flow tables based on the out message sent by the controller.

Typically, an OF switch within a SDN environment will determine the next path and action for any incoming packets based on the flow entries within its flow table or wait for the controller to transmit an output message that updates the flow table of the switch [2]. This allows network administrators and developers to construct policies that can be driven by the controller and implemented through the switches. This is useful for structures that require a more dynamic network where mobility and scalability is a core value [3]. However, the rise of 5G networks shows promise for SDN as mobile broadband networks have begun to implement SDN within their 5G framework to aid in scalability and flexibility [4].

Nevertheless, the improvements made by SDN also make way for new security threats. With the separation of the control and data plane, SDN switches rely on the controller for forwarding with no additional intelligence. This means attackers can flood the switches with hundreds of requests and overload the flow tables. Flow tables within OF switches are of fixed length and therefore, each flow entry has its own timeout length which can negatively affect the growth rate of the flow table, result in frequent overflows [5]. As a result, the SDN model begins to reach its limitations as network applications designate stateful processing intelligence to the controller. Flow states that have real time requirements will experience latency due to the flow entry timeouts [6].

On the contrary, it is possible to predefine the flow entry timeout to be short which would be ideal to minimize latency however, as presented in [5] short flow entry timeouts can cause flow tables to be congested more rapidly and render the controller and switch unresponsive. This can be due to massive influx of request constructed by attackers to overload the network such as the a DDoS attack, which is primary focus of this project.

A DDoS attack is a type of cyber-attack that causes a bandwidth overload using the communication traffic within the network and can be used to temporarily disable the network services. Several types of DDoS attacks exist. Direct attacks which utilize one host within the network generate random traffic while changing the originating IP address. Reflection attacks use more than one host that have been infected with malware programs to be controlled in an attempt to form a strategic attack on a particular target. Even though DDoS attacks have become avoidable in traditional network infrastructure the emergence of software defined networking has proven vulnerable to this form of attack.

The increase of IoT devices within the past several years has multiplied and has now become a primary focus for attack groups [7]. Therefore, the importance of security within SDN has become a focal point in network communication research. Currently, there are several schemes to detect and defend SDN infrastructures. Most effective techniques thus far are by utilizing the programmability of SDN and implementing a detection algorithm within initialization of the controller. Using a statistical approach for the basis of the algorithm provided researches with a method of detecting certain anomalies within a cluster of traffic within a network [8].

III. System Architecture

A. Network Topology

The network topology is shown in Fig.1 . During a normal traffic between clients in an SDN network, the first packet transmitted from one client to another. Initially, the switch will not have any entry in its flow table as to where it should forward the incoming packet. The default route will be forwarding the packet it to the controller. The controller will dictate the switch to forward the packet to respective destination port. The controller will also provide an entry rule in the flow table in the switch for that particular source and destination. In the future, the packet with the same source and destination will be automatically forwarded without the intervention of the controller.

1) Controller Modules : The detection program will run on POX as stand-alone modules. Multiple modules will be used to allow the controller to perform different functionalities such as installing flows, forwarding packets, and validating the DDoS attacks.

2) L3 Learning Module : This module, L3 learning, is the most important module in the POX controller. It is a simple layer 3 learning module that provides connectivity between the nodes in the network. L3 learning module handles the incoming packet and maintains a list of bindings ports OVS the MAC addresses of the connected clients. It utilizes this information to install the rule that replaces a destination MAC while forwarding a packet to the destination port. If no binding is found, the module instantiates ARP requests. Along with the connectivity, it is integrated with the detection algorithm to detect a malicious traffic flow.

B. Non-Functional Requirements

1) Interface Requirements : 1- OF Protocol: Provides a medium for the SDN controller to direct traffic along the switches within the network. 2- Northbound API's:Allow the SDN controller to communicate between services and applications running over the network owing to programmatic nature of SDN. 3- Southbound API's: Allow the SDN controller to communicate between the network devices within the network. 4- Python 2.7: Programs and modules created for the POX controller will require Python 2.7 since that is the fundamental

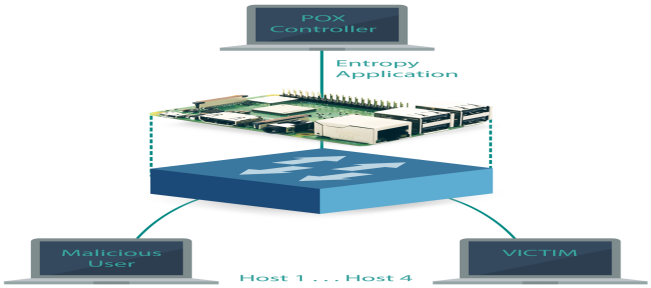


Figure 2: Network Topology

2) Software Quality Attributes : The correctness of this study depend upon the accuracy of our Entropy based application that will detect a DDoS attack. For this purpose, the Entropy based application will be trained in the SDN environment, so that we came find necessary constant value for our Entropy's threshold. The second attribute is availability, ,s long as the network is up and running, this service would be also be available continuously monitoring the network for any anomalies. The last attribute is usability, when a DDoS attack is detected, a simple notification will appear within the server that is responsible for running the POX controller. It will not involve using complex application for monitoring purposes. No specific training of the user is needed for using this system.

C. Software Requirement Specifications

The system uses a Raspberry Pi as OVS which has at least two host connected to it. The following software are required:

POX Software Defined Network Controller: The core of the SDN architecture is the controller. POX controller will be used to setup our SDN. It is one of the most widely used SDN for early-stage development across various platforms.

Mininet: Mininet is a network emulator that will create a virtual SDN network, running real kernel, and switch on a single machine within a matter of seconds. Custom topologies can be written in Python. One of the main feature is that it utilizes SDN. With OF protocol, the controller can be programmed to simulate any kind of network scenario.

Open vSwitch (OVS): This is a multilayer virtual switch that leverages OF and OVS database management protocol. Using OVS for virtual networking is considered the core element of any SDN deployment. In addition, Mininet uses OvS to forward packets across the interfaces and operates using the OF protocol.

Wireshark: Wireshark is a packet analyzer used to capture network packets and extract as details regarding incoming and outgoing traffic within a network. Just as multimeters are used to understand troubleshoot circuits, Wireshark is used to troubleshoot network and security problems, debug protocol implementation and understand how network protocols work.

Ubuntu: Ubuntu is an open-source Linux-based operating system. It will be used to run our simulator network, Mininet.

Python 2.7: Python is a high-level dynamic programming language that allow us to deploy custom typologies on Mininet and develop Entropy based detection module for our POX controller.

Python Scapy: Scapy allows users to create a custom packet with Python. Users can send a packet within a specific structure or manipulation so that it can be tested on how a particular machine or network will respond to that particular type of packet.

IV. System Design

A. System Architecture

During a normal traffic between clients in a SDN network, the first packet transmitted from one client to another. Initially, the switch will not have any entry in its flow table as to where it should forward the incoming packet. The default route will be forwarding the packet to the controller. The controller will dictate the switch to forward the packet to respective destination port. The controller will also provide an entry rule in the flow table in the switch for that particular source and destination. In the future, the packet with the same source and destination will be automatically forwarded without the intervention of the controller.

B. Controller Modules

The detection program will run on POX as stand-alone modules. Multiple modules will be used to allow the controller to perform different functionalities such as installing flows, forwarding packets, and validating the DDoS attacks.

1) L3 Learning Module : This module, L3 learning, is the most important module in the POX controller. It is a simple layer 3 learning module that provides connectivity between the nodes in the network. L3 learning module handles the incoming packet and maintains a list of bindings between the OVS ports and the MAC addresses of the connected clients. It utilizes this information to install the rule that replaces a destination MAC while forwarding a packet to the destination port. If no binding is found, the module instantiates ARP requests. Along with the connectivity, it is integrated with the detection algorithm to detect a malicious traffic flow.

2) DDoS Detection Algorithm: The design of the detection algorithm is based on three main inputs which are flow entries, packet counts, and Entropy calculations. This design is written in Python (2.7) and will be implemented within the POX controller as an additional module. It will also feature an alerting mechanism. If an attack is detected, a SMS notification will be sent to an administrator indicating that the SDN has been breach.

3) Flow Classification: Flows are sequence of packets that share similar characteristics. The characteristics are source IP address, destination IP address, source port number, destination port number,and protocol type. This flow classification information can be extracted from the header of each packet. Each incoming flow based on TCP and UDP protocol usually contain these information. In a SDN architecture, the OVS has a flow table that contains multiple flow entries. Each entry has its own rule, this rule allows the switch to know how to handle each incoming packet. An example would be a client trying to communicate with other clients, and to communicate a packet is sent to the switch. The incoming packets are grouped together to form flows. The incoming flows will navigate

through the OVSs to find the rule associated with the entry flow. If there is no match, the switch will generate and send a packet to the controller to acquire a new flow rule. Then, the controller will implement a new flow rule in the flow table so that the OVS can handle any new flows.

With that in mind, attacks or malicious users can exploit the flow rule by sending a large amount flows that are not presented in the OVSs flow table. These new flows will consume the OVS because the controller have to handle and establish new rules for these incoming flows. This will overload the controller and possibly disrupt the entire network. Such attack is known as DDoS. Understanding the flow classification allows us to design an algorithm to identify DDoS attacks from normal traffic through classifying these malicious flows and normal flows. Let's consider O, an observation sequence of different flows F that injects to an interface i of the OVS. Assume X is a malicious flow if the total number of packets within this flow is lower than a certain threshold. The threshold is predefined through data mining of flow classification. If X is a normal flow, then the total number of packets within this flow is equal to or greater than the threshold. X is defined as follows:

$$X = F_o^i = \begin{cases} 1 & \#\ of\ packets \geq Threshold \\ 0 & \#\ of\ packets > Threshold \end{cases} \quad (1)$$

C. DDoS Detection with Entropy-based Algorithm

Shannon Entropy [9] is an important concept for understanding the design of entropy-based detection algorithm. Entropy measures uncertainty or randomness associate with a random variable [10]. In our case, the random variable is the destination address. The entropy-based detection is similar to algorithm used in [11]. To collect packets for entropy analysis, a fixed window size is defined. Incoming packets need to be divided into two small groups (windows) based on the elapsed-time or the number of incoming packets. For each window, the packets are classified into groups that based on their destination IP address. All of these incoming packets will be coming from different source addresses. The destination IP addresses will be monitored for every new flow. These monitored flows are grouped into windows. Each window contains a hash table or a dictionary of two columns. The first column will state all the IP addresses, and the second column will show the number of times it has occurs. The window equation can be calculated as shown in equation 2.

$$W_{window} = (X_1, Y_1), (X_2, Y_2), (X_3, Y_3), ..., (X_i, Y_i) \quad (2)$$

Where i denotes each unique element in the window and n is the number of all IP destination addresses for each window. Then, the probability of each unique IP destination address can be calculated with Equation 3.

$$P_i = Y_i/n \quad (3)$$

Each window is calculated as shown in equation 4. The entropy will compare to the predefined threshold. If the entropy value is larger than the predefined threshold, this will indicate that there is no attack. Any value lower than the predefined threshold will indicate that there is a possibility of an attack. If an attack is not detected, the entropy threshold will be updated to current calculated entropy to prevent further false analysis. This allows the detection algorithm to dynamically adjust to the nature of traffic flow.

$$\varepsilon = \sum_{i=0}^{n} -P_i log P_i \quad (4)$$

The entropy-based algorithm is designed to detect an internal attack. The attacker and the victim are located within the internal SDN. The attacker has the options of attacking the controller or clients within the network. In either scenario, the attack is most likely to have a spoofed source IP addresses. Therefore, the incoming packets will not match the flow table so it will be forward to the controller. With this algorithm implemented, the controller can determine if the network is under attack through the drop in the entropy value due to the large number of packets containing the same destination address. The perfect example of such attack is UDP flooding.

In a normal network, traffic is expected to spread out to every client in the network. During a DDoS attack, the number of packets destined for a targeted host will rise immediately and the entropy value decrease. A decrease in the entropy is an indication that the network may be under attack. It is vital to have a fast and responsive detection because attackers could severely disrupt network service and possibility loss of data. To determine suiting window size , Oshima et al. [11] proposal is used based on entropy computation through calculating entropy in small size windows. Their study proposed a window size ranging from 50 to 5000 and concluded that window width of 50 and 500 successfully detect DoS/DDoS attacks. As proposed by Oshima et al., a window size of 50 is used for this research. The detection algorithm includes a function that collects incoming flow from the OVS. The flows will be stored in a file which will calculate the total number of packets for every destination IP in the current interval.

D. Entropy Based Detection Algorithm

Algorithm 1 shows the Entropy Based Detection Algorithm. The input, new packet in, correspond to a new packet that has arrive with a new source address. The destination IP address is also examined to see if it has an existing instance in the window. If it does exist, the count for that IP address will incremented. If the window gets full, the entropy is computed and then compared with the threshold. If the computed value for an entry is higher than the threshold for five consecutive counts, it will be classified as an attack.

E. Port Mitigation Technique

In order to prevent an attack after it has been detected we used the flexibility of the OpenFlow Protocol to mitigate the attack by modifying the flow-table of our OVS switch to output the incoming traffic from the attacker to a non-existing port. With the Pox controller, all incoming and outgoing traffic is analyzed using the flow table of the controller which keeps

Algorithm 1 Entropy Based Detection Algorithm

1: Function ENTROPY
2: Collect flow from switch and store in a file.
3: Calculate total number of packets for every destination ip in current interval.
4: Declare global variables
5: End Function
6: Function COLLECT_STATISTICS
7: Calculate the probability using, $P_i = X_i / \sum_{i=0}^{N} X_i$ where, P_i =Probability of i^{th} destination ip, X_i =Packet count on ith destination ip, N =Total number of destination ip
8: Calculate entropy of network using, $H(S_j) = \sum_{i=1}^{n} -P_i log P_i$ where, $H(S_j)$ is the Entropy of j_{th} switch
9: Calculate the difference between above calculated & normal entropy value
10: End Function
11: Function ENTROPY_DETECTION
12: Compare the above calculated $diff$ with predefined threshold value.
13: if $diff > threshold$ then
    14: Increment DDoS detected count
15: if DDoS detected count > min DDoS detected in particular window then
    16: Generate alert of DDoS attack
18: else
    19: Increment program counter.
20: end if
21: end function

Algorithm 2 Port Mitigation

1: Function MITIGATION
2: Collect Port and DPID of current flow
3: Confirm Attack is occurring
4: Collect Port and DPID of Attack
5: if $current - low = attack - flow$ then
    6: Create new flow entry with attack parameters
    7: Update new flow entry output_action where, $output - action = port_0$
    10: Send new flow entry to OVS switch
11: else
    12: Do nothing
13: end if
14: end function

track of the ingress ports and switch ID or DPID. These two parameters can also be used to modify any flow entry and implement an action for the flow entry. Algorithm 2 shows the port mitigation technique used to defend the network against attacks.

After detection of an attack the detection algorithm will collect the DPID and Port number associate with the attack and store it in a list. Then, the mitigation algorithm will analyze current traffic flow and compare its associated port number and DPID with the collected attack parameters. If a match occurs a new flow entry will be created to install a new action for that associated attack location. This action will drop any traffic incoming or outgoing to this particular attack location.

Table I: Traffic Flow

Simulated Traffic	Normal	Multiple Victims @ 25% attack rate	Multiple Victims @ 50% attack rate
Packet Type	UDP		
Traffic Interval	0.5 sec	0.125 sec	0.03 sec
Traffic Rate	2 packets/sec	8 packets/sec	33.3 packets/sec

V. Simulation and Test Results

Mininet is used to emulate the network with POX as the controller platform. The detection program will be a stand-alone module integrated within the L3 Learning module. This will allow the controller to perform additional functionalities such as installing flows and validating DDoS attacks. In this testbed, OF-switch is used to simulate the behavior of an edge switch in a SDN network. The Mininet network comprises a victim node that is the destination for both normal traffic and attack traffic generated from a client within the SDN network. By using Mininet, we can attack a virtual host and analyze the results of our detection algorithm. The client's IP addresses are assigned incrementally from 10.0.0.1.

For this research, three hosts are implemented, one OVS, and controller in the physical testbed. Typical networks consist of thousands of devices; however due to resources limitation the previous testbed is build to emulate the similar scenarios in large networks. To develop and validate the effectiveness of our solution in a real time environment, we created a virtual network with these exact constraints to better represent our testbed. Scapy is used to generate UDP packets with varying payloads and traffic interval. To fulfill our test case scenario, the time interval for the simulated traffic is defined. If an attack traffic has 25% rate, then the traffic interval of 0.025 seconds. For a normal traffic, it will have an interval of 0.1 seconds.

The simulations and evaluations were performed on multiple platforms. The main platform used to test our detection algorithm is iOS environment set up on a 1.4 GHz Intel Core i5 processor and a memory of 4GB RAM. Normal Traffic Simulation is used to determine the average entropy for a normal traffic, we launched three different traffic patterns.

Normal traffic is defined as traffic with long duration flows as compared to attack traffic which has short duration's flows with small numbers of packets. The goal is to determine the average normal entropy value for a network consisting of four nodes. A traffic was launched on the network with a traffic interval of 0.5 seconds. The traffic rate is defined by (1/0.5 = 2 packets/sec). Table 1 summarizes the specifications of all simulated traffic scenarios. In Figure 3, the chart describes how the entropy varies during the traffic. The lowest threshold captured during this traffic flow is 0.884 with the highest point captured at 0.900. The average entropy value of this small network is 0.892.

For attack traffic simulation with multiple victims, the topology and parameters remain the same as the one use to simulate the normal traffic. The only difference is the traffic interval and rate. For the multiple victim attack, we tested a 25% attack rate. The attack traffic interval is defined by

Figure 3: Entropy: Normal Traffic

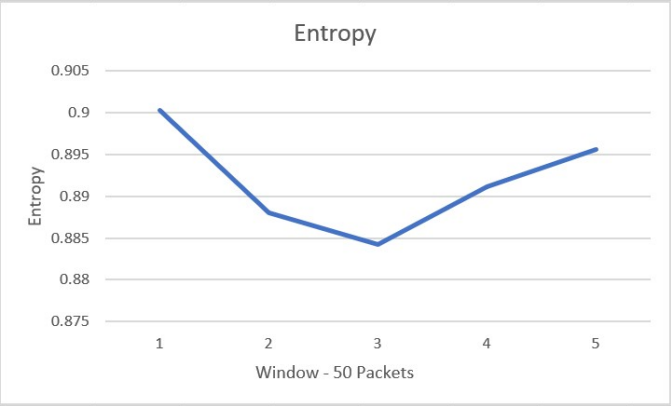
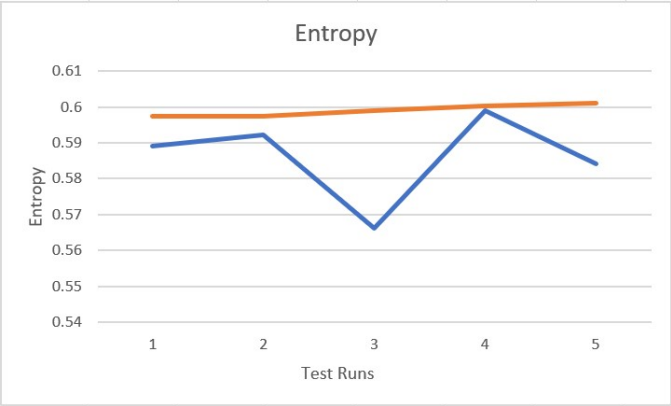


Figure 4: Entropy: Multiple Victims Attack Traffic



(Normal Traffic x 25% = 0.125 sec) with a traffic rate at 8 packets/sec. In this multiple victim attack scenario, the traffic interval of 0.125 sec will generate 8 packets a second per house along with 2 packets/sec being generate for legitimate traffic. As expected, the attack was detected. For further analysis, we increased the attack to 50% to see how fast our detection algorithm is able detect the attack.

As expected, the algorithm detects the attack under 10 seconds. The 25% attack rate is detected at 9.71 seconds. With a higher traffic rate of 33 packets/sec, the attack is detected at 5.14 seconds. The entropy value for both test scenarios are approximately the same which can be noticed on Figure 4. The 25% attack rate has a slightly higher consist entropy levels ranging from 0.601 to 0.597. As for the 50% rate, the lowest threshold captured is at 0.566.

For attack traffic with single victim, the traffic interval for 50% attack rate on multiple victims is set at 0.03 seconds. To generate the same amount of attack traffic for the single victim scenario, the interval is at 0.08 seconds. In this attack scenario, the specification is shown in Table 1. In this single victim attack scenario, the attack is detected within 5.55 seconds while 2 packets/sec were generating for legitimate traffic.

The results confirm the effectiveness of the detection algorithm. The average detection rate of a DDoS attack is determined within three to ten seconds. It is safe to say that the algorithm performs very well in distinguishing the difference between a normal and attack traffic. It is important to remember that this simulation is specifically designed for the used test-bed. In a real scenario, the network and its traffic will grow. If so, the threshold must be adjusted accordingly to avoid false positive detection.

VI. Conclusion

Using the Open vSwitch software developed for OF protocol proved simplicity in programming and integrating Linux-based hardware into an OF switch. Additionally, the POX controller proved to be simple and effective in establishing a test bed for the Entropy-based detection and mitigation algorithm. During integration, testing, and hardware deployment phase, the accuracy of the proposed algorithm was presented in detecting a generated DDoS attack on single and multiple victims. The attack was detected and mitigated within 3 to 10 seconds of the entropy value reaching 0. Once mitigated traffic continued to flow normally without disturbance from the attacker(s). The flexibility that SDN provides to the overall realm of networking has been proven to aid in programmability and management for network administrators. Even though this new paradigm leaves room for vulnerabilities, the flexibility of SDN provides network administrators with new techniques to protect and expand current and future networks.

References

[1] S. Azodolmolky, *Software Defined Networking with OpenFlow*, 2012.

[2] R. Yang, Jun Bi, and Guofei Gu, "Guest editorial: Software defined networking," *China Communications*, vol. 11, no. 2, pp. i–ii, Feb 2014.

[3] A. Y. Ding, J. Crowcroft, S. Tarkoma, and H. Flinck, "Software defined networking for security enhancement in wireless mobile networks," *Computer Networks*, vol. 66, pp. 94 – 101, 2014, leonard Kleinrock Tribute Issue: A Collection of Papers by his Students. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614001133

[4] X. Liang and X. Qiu, "A software defined security architecture for sdn-based 5g network," in *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, Sep. 2016, pp. 17–21.

[5] Y. Qian, P. Bhattacharya, W. You, and K. Qian, "Security threat analysis of sdn switch flow table," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, July 2018, pp. 1–2.

[6] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1701–1725, thirdquarter 2017.

[7] Y.-J. Lee, N.-K. Baik, C. Kim, and C.-N. Yang, "Study of detection method for spoofed ip against ddos attacks," *Personal and Ubiquitous Computing*, vol. 22, no. 1, pp. 35–44, Feb 2018. [Online]. Available: https://doi.org/10.1007/s00779-017-1097-y

[8] P. M. Ombase, N. P. Kulkarni, S. T. Bagade, and A. V. Mhaisgawali, "Dos attack mitigation using rule based and anomaly based techniques in software defined networking," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*, Nov 2017, pp. 469–475.

[9] L. Li, J. Zhou, and N. Xiao, "Ddos attack detection algorithms based on entropy computing," in *Information and Communications Security*, S. Qing, H. Imai, and G. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 452–466.

[10] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2015, pp. 77–81.

[11] S. Oshima, T. Nakashima, and T. Sueyoshi, "Early dos/ddos detection method using short-term statistics," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, Feb 2010, pp. 168–173.