computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

ADAPTIVE ENTROPY-BASED DETECTION AND MITIGATION OF DDOS ATTACKS IN SOFTWARE DEFINED NETWORKS

Jawad Dalou' [1], Basheer Al-Duwairi [2], Mohammad Al-Jarrah [1]

[1] Computer Engineering Department, Yarmouk University, Irbid 21163, Jordan,
e-mail: jadalou@just.edu.jo, jarrah@yu.edu.jo
[2] Network Engineering & Security Department, Jordan University of Science & Technology, Irbid 22110, Jordan
e-mail: basheer@just.edu.jo

Paper history:
Received 22 November 2019
Received in revised form 06 June 2020
Accepted 08 July 2020
Available online 27 September 2020

Keywords:
SDN;
DDoS;
Entropy.

Abstract: Software Defined Networking (SDN) has emerged as a new networking paradigm that is based on the decoupling between data plane and control plane providing several benefits that include flexible, manageable, and centrally controlled networks. From a security point of view, SDNs suffer from several vulnerabilities that are associated with the nature of communication between control plane and data plane. In this context, software defined networks are vulnerable to distributed denial of service attacks. In particular, the centralization of the SDN controller makes it an attractive target for these attacks because overloading the controller with huge packet volume would result in bringing the whole network down or degrade its performance. Moreover, DDoS attacks may have the objective of flooding a network segment with huge traffic volume targeting single or multiple end systems. In this paper, we propose an entropy-based mechanism for Distributed Denial of Service (DDoS) attack detection and mitigation in SDN networks. The proposed mechanism is based on the entropy values of source and destination IP addresses of flows observed by the SDN controller which are compared to a preset entropy threshold values that change in adaptive manner based on network dynamics. The proposed mechanism has been evaluated through extensive simulation experiments.

1. INTRODUCTION

The growth of networking and Internet has reached high rates in last years. This growth requires several changes in the networking industry. However, a major problem with traditional networking paradigm is that networking companies have almost full control over the hardware, firmware and software of their devices [1]. This implies that there is no practical way for developing new network protocols because of hardware dependence. To overcome this issue, a joint effort between Stanford University, the University of California at Berkeley, and several other universities had resulted in creating the Global Environment for Network Innovation (GENI) program in 2000 [2]. An important outcome of the GENI program is the Software Defined Networks (SDN) in 2006 [3].

Software defined networking (SDN) has emerged as a new networking paradigm that is based on the decoupling between data plane and control plane [4]. Therefore, it makes it possible to address many of the challenges and limitations of traditional computer networks and provide fixable and efficient management of networking resources. The control plane is abstracted by the SDN controller which is a logically centralized entity that oversees the whole networking components and orchestrates their operation. The controller has communication interfaces with the network devices in the data plane and has special APIs to communicate with different applications in the application layer. On the other hand, the data plane is abstracted by the forwarded devices called SDN switches. These switches maintain forwarding tables that are populated with flow table entries received by the controller.

The communication between different SDN planes is governed by the Openflow which is a standard communication interface. According to this standard, whenever a new packet is received by an Openflow enabled switches (OF-Switches) with no matching entry in its forwarding table, a PACKET-IN packet is sent to the controller. In turn, the controller updates the flow table of that switch sending a PACKET-OUT packet including a flow table entry added to its flow table. The architecture of SDN network is shown in Fig. 1. The control plane consists of the controller and different network applications.

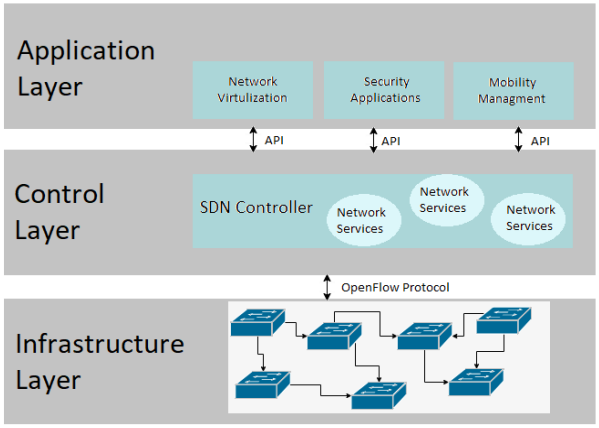

Figure 1 – Software defined network architecture

Similar to traditional networks, software defined networks are vulnerable to distributed denial of service attacks as shown in many research studies (e.g., [5]-[7]). In particular, the centralization of the SDN controller makes it an attractive target for these attacks because overloading the controller with huge packet volume would result in bringing the whole network down or degrade its performance. Moreover, DDoS attacks may have the objective of flooding a network segment with huge traffic volume targeting single or multiple end systems. Recent DDoS attack incidents confirm the devastating effects of these attacks and calls for efficient methods to detect and mitigate them. DDoS attacks are hard to be detected, because the traffic generated during attacks is similar to legitimate traffic (e.g., SYN flooding and ICMP flooding). In these cases, the victim cannot decide whether the received packets are malicious or not. Also in DDoS attacks, it is hard to detect the attacker because attack packets usually carry spoofed source IP addresses [8].

Several research efforts adopted entropy as a main method for DDoS attack detection in both traditional networks and SDN networks. This is because entropy provides a measure of statistical randomness of a certain variable and any sudden change of its value could be a strong indication of an attack given that the entropy value is associated with suitable network traffic parameters. The work presented in this paper is based on the concept of entropy for DDoS attack detection. However, in contrast with the previous work, we monitor entropy value associated with a number of distinct IP destination addresses and the entropy value associated with the number of distinct IP source addresses as observed by the SDN controller. Moreover, we dynamically adjust the entropy threshold value based on network dynamics. The rest of this paper is organized as follows: Section II discusses related work. Section III-B discusses proposed mechanism. Evaluation is presented in Section IV. Finally, conclusion is presented in Section V.

2. RELATED WORK

In this section, we discuss the main research efforts in countering DDoS attacks in software defined networks. For example, in [9] NetSight was proposed as a platform that captures packet histories and enables applications to retrieve interesting packet histories. For NetSight flexibility, there are four main applications developed on top of it: network debugger, live invariant monitor, path packet logger, and a categorized network profiler. The main goal for network debugger is to provide interactive debugging features for networks. The live invariant monitor is used to specify network behavior and launch an alarm when a violation happens. Path packet logger filters the packets with their paths and header values at each hop. The network profiler goal is link utilization by understanding the network characteristics and routing decisions.

A Distributed and Collaborative per-flow Monitoring (DCM) was proposed in [10]. DCM uses Bloom filters that represent monitoring rules in a small size memory. It installs a monitoring tool into the switch data plane. DCM uses this tool as two-stage Bloom filters which are the admission of the Bloom filter and a group of actions to perform different measurement. Moreover, for dynamicity, SDN allows DCM to perform updates of the two-stage switch data plane. Sahay et al. [11] proposed a solution based on SDN for DDoS detection and mitigation. Their approach is done using ISP level monitoring of traffic. The traffic is tagged by OpenFlow switches. This traffic is monitored to calculate statistics from it. The statistics are forwarded to the detection engine. The engine then generates policy rules depending on the statistics. These rules are enforced by SDN controllers of the customers to local routers. The previous tagged

information is sent from customer controller to the ISP controller. The malicious traffic will be directed to an appropriate middle box for mitigation.

FloodGuard was proposed in [12] is an approach for DDoS detection in SDN networks. FloodGuard contains two modules: proactive flow rule analyzer module, and packet migration module. The analyzer module ensures the functionally of the network when an attack happens. The migration module is responsible for sending un-malicious packets to the controller without overwhelming its resources. Chin et al. [13] introduced a collaborative approach for DDoS attack detection. This approach provides one or many monitors to observe network traffic. Also, it contains correlators that respond to alerts from monitors. Thus, when the number of packets exceeds the control threshold, the monitor generates an alert and sends it to the correlator to take action based on the alert type.

Zhou et al. [14] combined SDN with NFV (Network Function Virtualization) for DDoS detection. This combination ensures the control and date plane separation from SDN and enables flexible resource allocation and development from NFV. The SDN module is responsible for data collection and analysis. It runs appropriate mitigation method when an attack happens. The NFV module is responsible for virtualizing and managing virtual machines on DDoS mitigation requirement. Nguyen et al. [15] proposed an IDS with sampling method for DDoS detection in SDN. It detects the attacks even with small volume by choosing the appropriate sample rate. Also, it detects the attacks at the edge router, which will prevent them from going through core network. This method consists of different sample collectors, which will forward the samples to IDS for analysis. Then, the controller generates a rule based on the IDS results to block the traffic or forward it.

In [16], Mehdi et al. used three detection algorithms: Threshold Random Walk with Credit Based (TRW-CB), Rate Limiting, and Network Traffic Anomaly Detector (NETAD). TRW-CB uses sequential hypothesis testing like ratio testing to determine if the internal host has an infection or not. Also, it is based on the rule that the probability of a successful connection of non-malicious host is larger than that of a malicious one. Rate Limiting is used to limit the connection rate due to the fact that the infected machine has different connection characteristics than uninfected one. NETAD is a rulebased traffic filtering mechanism which removes unwanted traffic based on the first few packets of a connection. NETAD computes a score for each packet depending on the time and frequency of each byte of the packet. Then a threshold is applied on each score to distinguish the unwanted packets.

AVANT-GUARD [17] deals with two challenges in SDN networks: secure the interface between control plane and data plane and improve responsiveness. Securing the interface between the two planes is done by using migration techniques on the data plane to protect control plane from attacks. Responsiveness is improved by creating triggers that can be inserted by control plane and adding flow rules that will be activated when a trigger is detected. Lim et al. [18] proposed a method called DBA (DDoS Blocking Application). DBA is a defense mechanism for DDoS botnet-based attacks. DBA can distinguish normal traffic from abnormal traffic. When the transmission rate suddenly increases, the client is considered as a bot. DBA will notify the controller that an attack has happened, and the packets should be dropped.

Dharma et al. [19] proposed a method that can detect and mitigate the effects of DDoS attacks in SDN networks. It focuses on the destination address of the packets and the time needed for generating high traffic rate. If the destination address is not valid or unknown, the controller will forward this packet to the flow collector. When invalid packets increase significantly within a time window, the flow collector sends a notification to the controller. The controller then uses this notification to forward any future invalid packets directly to the flow collector.

Xu et al. [20] proposed a method for DDoS detection under SDN context. This method consists of two procedures: victim detection and post-detection. Victim detection considers the flow volume feature and the flow rate asymmetry feature. Thus, if these features showed a DDoS attack, the victims IP address will be determined. Post-detection has two ways to react for DDoS detection: passive processing by asking the victim to change his service to a new IP, and active processing by finding the attackers IP addresses and install rules into switches to drop the packets coming from these attackers.

Dong et al. [21] introduced an approach that uses Sequential Probability Ratio Test (SPRT), which is a powerful statistic tool. This method can quickly detect the attack after a small number of successive flows. Also, this method can detect the attack regardless the type of the flooding packets (e.g. TCP, ICMP or any other flooding of requests). The detection is done using percentage count and entropy data flows. Mousavi et al. [22] have an early detection within hundreds of packets of the attack. This approach uses entropy to calculate randomness based on destination IP addresses. Its main idea is comparing the entropy to a threshold value. If the entropy is lower than the threshold, an attack is

detected, otherwise there is no attack. The threshold value is calculated based on many times experiment.

The work presented in this paper is different from the work presented in [22] in the sense that entropy threshold value is not static as it changes based on network dynamics that is due to link failure or hosts joining and leaving the network. Another important difference is that we consider entropy values for both the IP destination addresses and IP source addresses in such a way to detect attacks that target multiple victims aiming at flooding the network segment.

3. PROPOSED WORK

In this section, we present the proposed adaptive entropy based DDoS detection and mitigation scheme. The core of our proposed scheme is based on the concept of entropy.

3.1 ENTROPY-BASED DDOS DETECTION

The randomness associated with a random variable is typically measured using Entropy which is a well-known concept in information theory. Higher entropy of a random variable indicates higher randomness of that variable. Several DDoS detection schemes used entropy associated with different traffic parameters such as IP destination addresses and TCP flags, etc. In these schemes, entropy value associated with each parameter is monitored during specific time intervals. At the end of each interval, a decision is made whether there is an attack or no based on comparing current entropy value with a pre-defined threshold value. In the proposed scheme we monitor two entropy values: one is associated with destination IP addresses and other is associated with source IP addresses, such that we can detect attacks not only targeting specific end system, but also attacks that aim at flooding one or more network segments by high packet volume destined to multiple IP addresses belonging to the targeted network segments.

In normal operation of SDN network (i.e., with no attack scenario), packets can be destined to any end system in the network without a concentration on one or more end system. Therefore, communication between end systems is expected to be randomly distributed as pointed out in [22] and [23]. However, in the presence of a DDoS attack event, it is expected to observe either (i) large number of packets with the same destination IP address resulting in low entropy value associated with the IP destination address parameter or (ii) large number of packets with randomly spoofed IP source addressees and destined to multiple IP addresses belonging to one or more targeted network segment. This would result in high entropy associated with the IP source address parameter.

The SDN controller groups incoming Packet-IN packets in batches each of size *N*. For each batch of packets (also called *window*), it calculates the entropy value associated with each variable (i.e., IP detestation addresses and IP source addresses) using equation 1. Here *Pi* represents the probability of IP address (*IPi*) in the given window, which can be calculated using equation 2. Here *xi* represents the number of occurrences of IP address *IPi* during the given window.

$$H = -\sum_{k=1}^{N} Pi\ Log(Pi) \quad (1)$$

$$Pi = \frac{xi}{N} \quad (2)$$

It is to be mentioned that the same method is used to calculate entropy for both variables.

3.2 PROPOSED MECHANISM

The overall architecture of the proposed solution is shown in Fig. 2. The proposed system consist of two major units that include DDoS detection unit and DDoS mitigation unit. The main objective of the DDoS detection unit is to detect DDoS attack event by calculating entropy associated with source and destination IP addresses included in Packet-IN packets, and comparing its value to a pre-defined threshold. To this end, the entropy is calculated continuously for each window of packets of size *N*. A decision is made whether there is an attack or no based on the entropy values calculated in the previous *k* windows. This is to ensure that there is enough evidence that there is an attack. The output of this stage/unit is the list of end system IP addresses targeted by a DDoS attack, and the source (switch interface) responsible for forwarding incoming attack traffic. The mitigation unit is responsible for or blocks attack traffic identified by the detection unit.

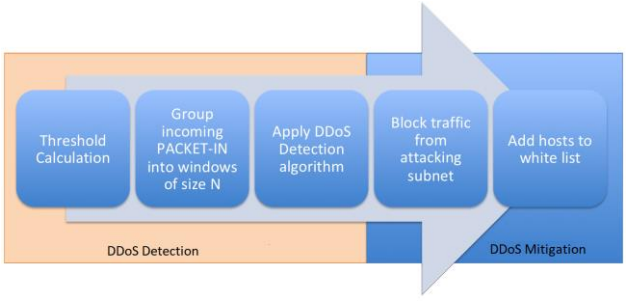

Figure 2 – Proposed detection and mitigation system architecture

Entropy Threshold Calculation: Before we go into the details of the DDoS detection algorithm, we

explain how to set the entropy threshold values. The SDN controller maintains two entropy threshold values: (i) (*ThresholdSrc*) for IP source addresses and (ii) (*ThresholdDst*) for IP destination addresses. The threshold values are provided by equations 3 and 4, respectively.

$$ThresholdSrc = -log\left(\frac{1}{Window\ Size}\right) * 0.8 \quad (3)$$

$$ThresholdDst = -\log\left(\frac{1}{hosts - 1}\right) * 0.8 \quad (4)$$

We set the *ThresholdSrc* to the value given by equation 3 because in the case of random source address spoofing, which is a common feature of DDoS attacks in general, the probability of a given IP source address in a window of packets of size *N* would be 1/*N*. This correspond to an ideal situation where all packets in the monitored window of packets have distinct IP source addresses (due to random spoofing). On the other hand, we set the *ThresholdDst* to the value given by equation 4 because in normal operation of the SDN network, and as an extreme case, the controller is expected to receive PACKET-IN packets that contain IP destination addresses of all hosts in the network except the sender IP address. This leads to the probability of destination IP address for each packet in the window to be equal (1/(hosts-1)). It should be noted that both values are multiplied by 0.8 factor in order to provide a 20% margin of the ideal value.

DDoS Detection: Algorithm 1 depicts the proposed DDoS detection algorithm which is performed by the SDN controller for each window of PACKET-IN packets. In order to minimize false positives, the algorithm takes a decision that there is an attack, only if the entropy values calculated according to equation 1 for both the IP source address and IP destination address exceed the specified thresholds for *k* consecutive windows, where *k* is a design parameter that we study later in Section IV. The algorithm starts by initializing the number of entropy rounds counter *c* to 1. Once the window is full, the entropy of destination IP addresses is calculated. If the entropy value is greater than the *ThresholdDst*, the counter *c* is reset to 1 and the algorithm repeats after receiving new window of PACKET-IN packets. Otherwise (i.e., in case the entropy is less than *ThresholdDst*), the entropy of the source addresses for the packets found in the window is calculated and compared to *ThresholdSrc*. If the entropy value is less than the *ThresholdSrc*, the counter c is reset to 1 and the algorithm repeats after receiving new window of PACKET-IN packets. Otherwise (i.e., in case the entropy is more than *ThresholdSrc*), the counter *c* is incremented and compared to the number of windows parameter *k*. If *c* is larger than *k*, then an attack is detected. Otherwise, the algorithm starts over waiting for new window of packets.

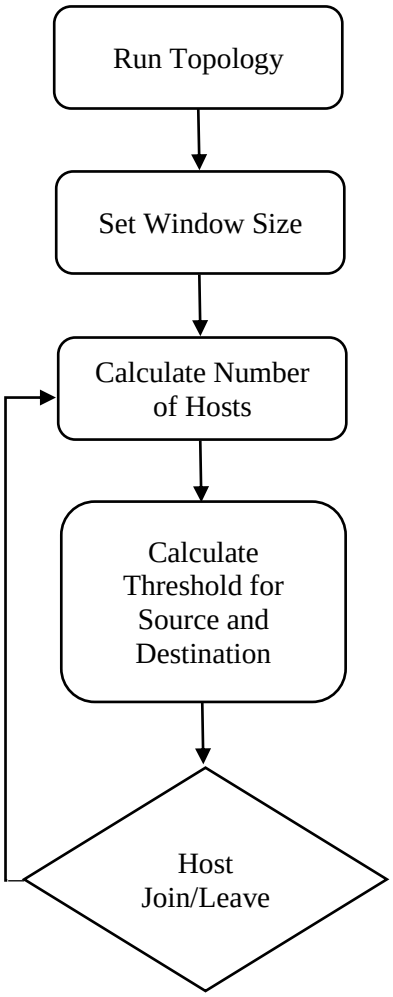

Fig. 3 – Threshold update mechanism

DDoS Mitigation: Mitigation of DDoS attack is done by invoking the mitigation algorithm which is based on the idea of initial blocking of all incoming packets from the switch(s) responsible for forwarding attack traffic. The switch that forwards the largest number of packets during the monitoring interval is considered first. This step is important in order to relief the SDN controller and the network in general from attack traffic overload. This is done by using the OpenFlow tables of the switches where the controller adds specific rules to achieve this goal. As a result, all traffic coming through a specified switch interface and destined to the victim node(s) is blocked. This is followed by adding rules to these switches such that only active hosts connected to these switches are permitted to forward traffic to victim nodes. These hosts can be identified by performing host scanning for the subnetwork behind that switch. In other words, these hosts are put in a

white list. The white list is dynamic, i.e., when a new host is added to the attacks subnet, it will be automatically added to the white list and can forward traffic to victim nodes.

Algorithm 1: Entropy-Based DDoS Detection

Input : Sequence of Incoming PACKET-IN packets
Output: DDoS detection alert

1. $c = 1$
2. Form a new incoming window of PACKET-IN packets (window size = N packets)
3. Calculate entropy of IP destination addresses *EntropyDst*.
4. **If** (*EntropyDst* > *ThresholdDst*) then
   - $c = 1$
   - Go to Step2
5. **end**
6. **else** Calculate entropy of IP source addresses *EntropySrc.*
7. **If** (*EntropySrc* < *ThresholdSrc*) then
   - $c = 1$
   - Go to Step2
8. **else**
   - $c++$
9. **end**
10. **If** ($c>k$)
    - DDoS attack is detected
11. **else**
    - Go to Step2
12. **end**

Example: As an illustrative example, we consider the network topology shown in Fig. 4. This network consists of two switches (s1, s2) and four hosts (h1 - h4). The SDN controller initializes the window size to a certain value (for example 50 packets), then it calculates the number of hosts, which is four in this case. Based on equations 3 and 4, the initial value of *ThresholdSrc* and *ThresholdDst* is set to 1.359 and 0.381, respectively. It is clear that the entropy threshold value of the destination IP address is affected by network dynamics. For example, if the link connecting h1 and s1 is broken, the new number of hosts will be three and *ThresholdDst* will be 0.241. On the other hand, when a new host is added, the total number of hosts becomes 5, and *ThresholdDst* becomes 0.482.

Assuming that there is an attack from host h1 to host h3. The SDN controller detects this attack according to algorithm1 as follows: It calculates the entropy of destination IP addresses for each window of packets. The entropy value of the destination IP addresses will be zero according to equation 1 because the probability of destination packets will be one as all the packets are directed to one h3 (i.e., the victim). This value of entropy is considered the minimum value that could be obtained because all packets in the window are directed to one host. This entropy value is compared to *ThresholdDst* (0.381). Next, the controller calculates the entropy of IP source addresses which will be 1.56 based on equation 1. This value is greater than the threshold value *ThresholdSrc* (1.359). As a result of that, the counter will be incremented and the same process will be repeated again for k consecutive windows (We used $k = 5$ in this example). After that, the attack is detected. If the condition of comparing the entropy to the threshold is not met in any round, the counter value will be reset to one to start checking again. This means the attack must be in k consecutive windows to be detected. The mitigation process starts directly after an attack is detected. This starts by adding a rule to the OpenFlow table of switch s1 to block all traffic destined to h3. Therefore, it leads to getting rid of attack traffic directly. Then, the controller add rules to the OpenFlow table of switch s2 to allow h1 and h2 to reach h3. Furthermore, when adding a new host to s1, this host will automatically be added to the OpenFlow table of switch s2.

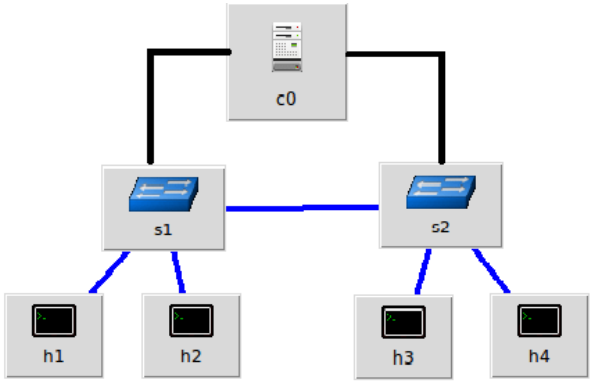

Figure 4 – SDN based network

It should be mentioned that it is possible for an attacker to bypass the proposed detection mechanism in certain attack scenarios. This depends mainly on attacker's ability to exploit entropy skewness inherent limitations. For example, an attacker may distribute his/her traffic to large number of end systems in the network. Therefore, it leads to increasing the entropy value associated with IP destination addresses. Moreover, the attacker can limit IP source address spoofing to certain number of IP addresses during each time interval. Therefore, the entropy value associated IP source addresses decrease, which would result in evading attack detection.

4. EVALUATION

We have conducted extensive simulation experiments to evaluate the proposed algorithm. In this section, we discuss the different aspects of these experiments and the results obtained. Subsection IV-A introduces the simulation environment, the tools and the network topology used to perform the experiments. Subsection IV-B discusses the performance metrics and introduces the simulation parameters. Finally, Subsection IV-C discusses the results obtained.

4.1 SIMULATION ENVIRONMENT AND NETWORK TOPOLOGY

All the experiments have been carried out using Mininet version 2.2.0 [24] running on Linux Ubuntu 14.4 machine. Mininet is a standard network emulator tool for SDN networks that provides a great way for developing OpenFlow and SDN networks. We have used the popular POX [25] controller as the SDN controller in our experiments. POX is an extension of NOX controller [26]. It is a light-weight and fast SDN controller written in Python that can run on both Linux and Windows platforms. We wrote a Python code that utilizes the Scapy packet generation tool [27] to generate background normal traffic and to perform DDoS attacks with randomly spoofed IP source addresses. All the experiments were accomplished following these steps:

- On a terminal, the POX controller is started.
- On another terminal, Mininet is launched to design a custom topology configured in a separate file.
- Now the network topology is configured through running the POX controller.
- In order to check the connectivity between hosts, a ping is done in Mininet using the *ping* command to check the connectivity between two specific hosts, or the command *pingall* to check the connectivity between all hosts.
- To launch terminals for running hosts the command *xterm* is performed on Mininet (e.g., *xterm h1* will open a terminal for host h1).
- To start the attack by opening a terminal for the attacker host and running the Scapy script that is configured to perform the attack.
- To view the OpenFlow tables in each switch the command *sh ovs-ofctl dump-flows* is performed in Mininet as follows: *sh ovs-ofctl dump-flows s1* shows the OpenFlow table of the switch s1.

Fig. 5 depicts the network topology used to conduct the simulation experiments. The simulation network consists of four switches: three of them are connected to the hosts forming three subnets: 10.0, 10.1 and 10.2. Subnet 10.0 has four hosts representing the servers (target machines). On the other hand, the hosts in subnets 10.1 and 10.2 are used mainly to generate background traffic and DDoS attacks with twenty hosts in each subnet.

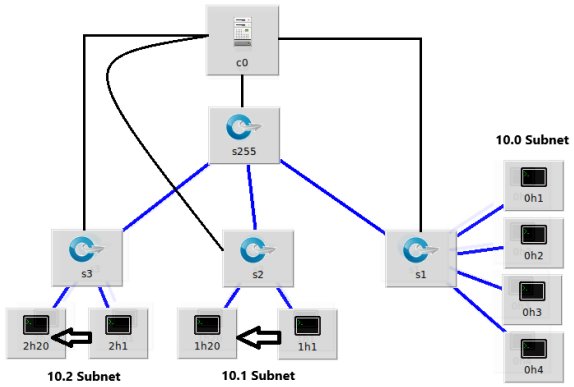

Figure 5 – Network Topology

4.2 PERFORMANCE METRICS AND SIMULATION PARAMETERS

The proposed DDoS detection mechanism has been evaluated in terms of the following performance metrics:

- *Average Detection Time*: Detection time is defined as the time from when an attack starts until it is detected. This metric is very important and should be as minimum as possible. For a short detection time, the damages resulting from the attack can be reduced and eliminated quickly.
- *Entropy*: Entropy as defined before is the randomness of the packets. The randomness of source IP addresses in normal traffic is usually small and from known IP addresses. Furthermore, the randomness of destination IP addresses in normal traffic is usually large due to the wide range of destination IP addresses. However, during an attack, the randomness of source IP addresses is usually large because they are spoofed. On the other hand, the randomness of destination IP addresses during an attack is usually small because the traffic is destined to one host which is the victim. Therefore, it is very important to consider the entropy value for attack detection.
- *OpenFlow tables overhead*: OpenFlow tables overhead is defined as the number of rules that are added to the switches' OpenFlow tables as a result of attack flows. The main objective of the DDoS mitigation component is to reduce network overload and to minimize OpenFlow tables overhead.

- *False positive rate*: False positives are defined as misclassifying non-attack as an attack. This metric is very important in detection algorithms and should be minimized as possible. The effectiveness of the algorithm is to be able to differentiate between attack and non-attack scenarios.

The main simulation parameters that have been used in our experiments were as follows:

- *Attack rate (R):* The Attack rate is defined as the number of attack packets per second. Usually in DDoS attacks, the attack rate is high. High rate of packets makes the controller unable to handle these packets and becomes unavailable. In our experiments, we study the performance of the proposed mechanism under different attack rates.
- *Window Size (W):* Recall that the SDN controller groups incoming Packet-IN packets in batches each of size *N* representing *window* of packets that are used as input for entropy calculation. Choosing an appropriate window size is important to measure the false positives rate and detection time.
- *Number of entropy rounds k*: This parameter specifies the number of rounds used for calculating entropy before deciding that there is an attack. Choosing an appropriate value for this parameter is important as it effects false positive rate and detection time.

Table I summarizes main simulation scenarios and parameters and the range of values used for each parameter. In the simulation experiments, we consider single victim case and multiple victim case in order to evaluate the performance of the proposed detection and mitigation algorithm in both cases.

Table 1. Summary of simulation scenarios and parameters

Scenario	Number of victims	Attack rate (Packets/Sec.)	Window size	Number of rounds
S1	1	Varies from 50 to 500	Varies from 20 to 70	Varies from 2 to 7
S2	4	Varies from 50 to 500	Varies from 20 to 70	Varies from 2 to 7

4.3 EXPERIMENTAL RESULTS

1. *Entropy*: Attack detection depends mainly on entropy variation during attack event. Therefore, we start by presenting entropy value for IP destination addresses and IP source addresses during a DDoS event. Fig. 6 shows that IP destination addresses entropy drops significantly during attack period (between time 1.9 and 3.9 seconds). At the same time, IP source address entropy shown in Fig. 7 increases during the same time period. This variation of entropy in both cases is expected because of the decrease of randomness in destination IPs observed during attack interval, and the significant increase of randomness in source IPs due to random address spoofing.

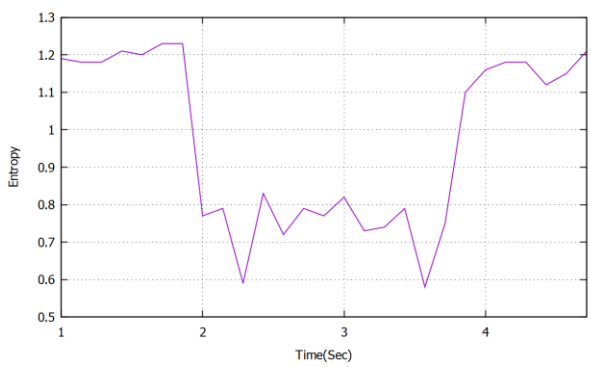

Figure 6 – IP destination addresses entropy

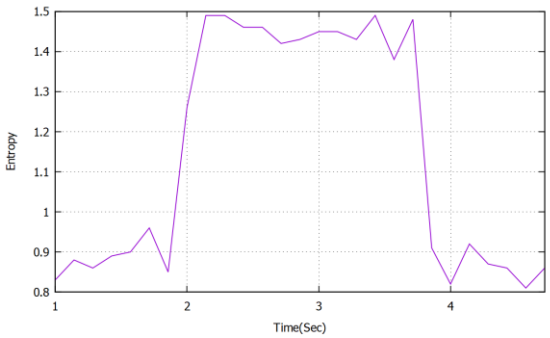

Figure 7 – IP source addresses entropy

2. *Detection Time:* The average attack detection time for single victim and multiple victims scenarios listed in Table 1 is shown in Figs. 8-13. The effect of attack packet rate on the detection time for different values of window size ranging from 20 to 70 packets is shown for scenarios S1 and S2 in Figs. 8 and 9, respectively. It can be seen that the detection time decreases by increasing attack rate because more attack packets will be observed by the SDN controller and this will affect the entropy for each window of packets leading to faster detection. Also, it can be seen that a smaller window size corresponds to smaller detection time because in this case (i.e., when window size is small) it takes the controller small amount of time to collect the packets required for

attack detection. However, it should be noted that there is a tradeoff between window size, number of entropy rounds, and false positive rate, which means that there is a restriction on decreasing the window size. As can be noticed, when targeting multiple victims, the detection time increases because it takes more rounds to discover that there are attacks targeting multiple victims. In general, for lower false positives rate, the controller has to collect enough number of packets and this can be achieved in our algorithm by increasing the window size and the number of entropy rounds.

The effect of window size and number of entropy rounds on the average detection time for different attack rates (50 packets/second and 500 packets/second) for single victim case and multiple victims case is shown in Figs. 10, and 11, 12 and 13. Generally, it is clear that the average detection time increases by increasing the window size and it has higher value for larger number of rounds.

3. *The OpenFlow tables* overhead is calculated by the number of rules that are added in the mitigation process. This overhead depends on the number of subnets from where an attack has been launched. If the attack has been launched from the same subnet, the overhead will be the same regardless the number of attackers. The overhead increases if the attack came from different subnets. Fig. 14 shows this overhead. In the figure, every 20 hosts constitute a subnet. As can be noticed, when the attackers are from the same subnet, the overhead is the same value regardless the number of attackers. This is because of the feature of phase two of the mitigation engine described before. If the attack comes from different subnets the overhead increases.

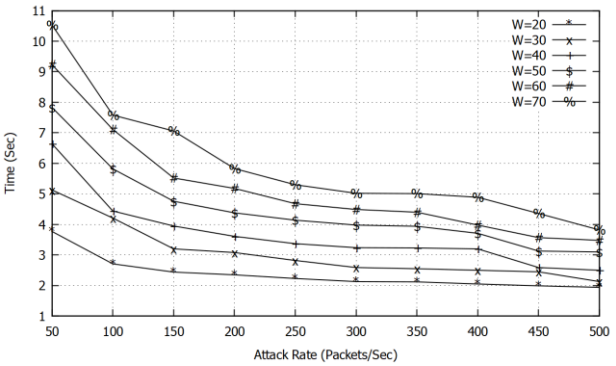

Figure 8 – Effect of attack rate on average attack detection time -Single victim case, entropy rounds (k)=5

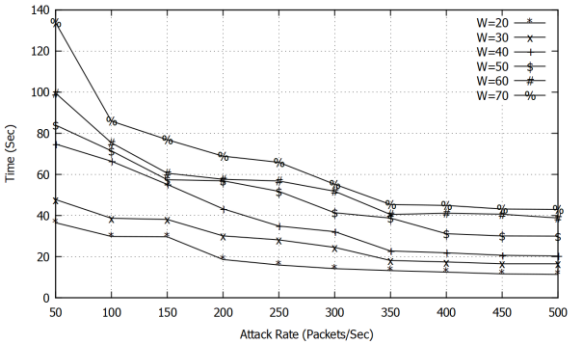

Figure 9 – Effect of attack rate on average attack detection time - Multiple victim case, entropy rounds (k) =5

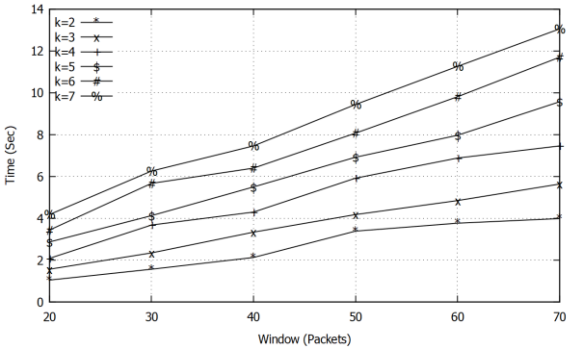

Figure 10 – The effect of window size parameter on the average detection time- Single-victim, Attack rate = 50 Packets/second.

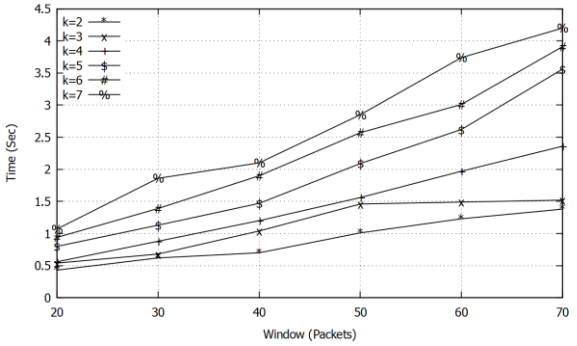

Figure 11 – The effect of window size parameter on the average detection time- Single-victim, Attack rate = 500 Packets/second.

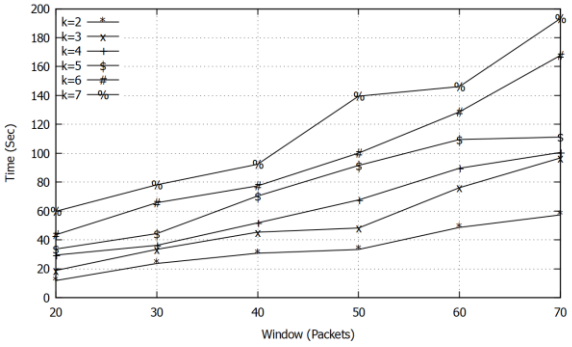

Figure 12 – The effect of window size parameter on the average detection time- Multiple-victims, Attack rate = 50 Packets/second

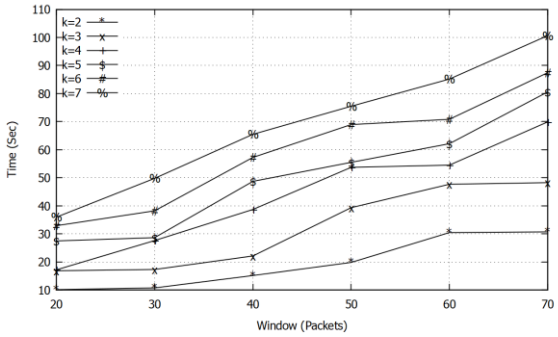

Figure 13: The effect of window size parameter on the average detection time- Multiple-victims, Attack rate = 500 Packets/ second.

As can be noticed, when the attackers are from the same subnet, the overhead is the same value regardless the number of attackers. This is because of the feature of phase two of the mitigation engine described before. If the attack comes from different subnets, the overhead increases, that is, per subnet attack, the overhead is 40 extra rules.

4. *False Positive Rate*: Fig. 15 depicts the false positive rate of the proposed scheme compared to that of the scheme proposed by Mousavi et. al., [22]. In this experiment we set the number of rounds. It can be seen that the false positive rate decreases by increasing the window size in both schemes. This is expected, because both mechanisms rely on collecting enough samples for entropy calculation in order to accurately detect an attack. This emphasizes that the selection of the window size and the number of rounds should take into consideration false positive rate. Also, it can be noted that false positive rate achieved by the proposed scheme is much lower than that obtained in [22] which due to the fact that the proposed scheme calculates the entropy for destination and source IP addresses rather than for destination IP addresses only.

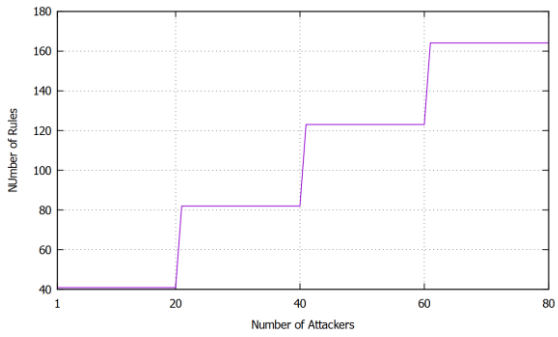

Figure 14 – OpenFlow table's overhead

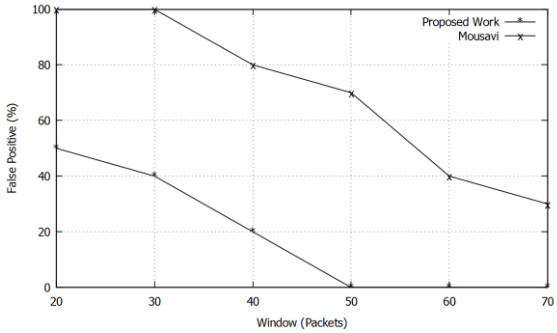

Figure 15 – False Positive Rate

5. CONCLUSIONS

Software Defined Networking (SDN) is a modern approach to network management. SDN provides flexibility for network configuration through a centralized SDN controller. The centralization of the controller makes it more likely for DDoS attacks, such that if the controller goes down, all other network elements become useless. Moreover, DDoS attacks may have the objective of flooding a network segment with huge traffic volume targeting single or multiple end systems. Recent DDoS attack incidents confirm the devastating effects of these attacks and calls for efficient methods to detect and mitigate them. The algorithm presented in this paper detects such attacks based on IP source and IP destination addresses entropies. Furthermore, the proposed algorithm has an adaptive approach to joining or leaving hosts. The algorithm provides also a module for mitigation of the attacks once detected.

The results showed the detection time for a DDoS attack with different attack rate, different Window size, and different entropy rounds. In addition, the results show the entropy variation during normal traffic and attack traffic. The value of entropy for destination decreases during the attack compared with its value during normal traffic. The value of entropy for source, on the other hand, increases during the attack compared with its value during normal traffic. The OpenFlow tables overhead results show that the algorithm has an efficient method for minimizing the overhead, such that when the attack is launched from the same subnet, it will be the same overhead whether the attack is launched from one attacker or many attackers. The overhead is increased when the attack is launched from different subnets.

6. REFERENCES

[1] P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*, Morgan Kaufmann, 2016.

[2] L. Chung-Sheng and W. Liao, "Software

defined networks," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 113-113, 2013.

[3] M. Casado, T. Garfinkel, M. Freedman, A. Akella, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15, ser. USENIX-SS'06*, Berkeley, CA, USA, 2006, pp. 137-151.

[4] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, Third Quarter 2014.

[5] S. S. Mohammed, R. Hussain, O. Senko, B. Bimaganbetov, J. Lee, F. Hussain, C. A. Kerrache, E. Barka, and M. Z. A. Bhuiyan, "A new machine learning-based collaborative DDoS mitigation mechanism in software-defined network," *Proceedings of the 14th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2018, pp. 1–8.

[6] K. Bhushan and B. B. Gupta, "Distributed denial of service (DDoS) attack mitigation in software defined network (SDN)-based cloud computing environment," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 5, pp. 1985–1997, May 2019.

[7] K. Kalkan, G. Gur, and F. Alagoz, "Defense mechanisms against DDoS attacks in SDN environment," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 175–179, Sep. 2017.

[8] Q. Yan, F. R. Yu, Q. Gong and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602-622, First quarter 2016.

[9] N. Handigol, B. Heller, V. Jeyakumar, D. Mazires, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014)*, 2014, pp. 71-85.

[10] Y. Ye, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," *Proceedings of the third Workshop on Hot Topics in Software Defined Networking*, ACM, 2014, pp. 85-90.

[11] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, "Towards autonomic DDoS mitigation using software defined networking," *Proceedings of the NDSS Workshop Security Emerging Networking Technologies (SENT)*, San Diego, CA, USA, 2015, pp. 1–7.

[12] H. Wang, L. Xu and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Rio de Janeiro, 2015, pp. 239-250.

[13] T. Chin, X. Mountrouidou, X. Li, K. Xiong, Selective packet inspection to detect DoS flooding using software defined networking, in: (SDN)," *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2015, pp. 95-99.

[14] L. Zhou and H. Guo, "Applying NFV/SDN in mitigating DDoS attacks," *Proceedings of the IEEE Region 10 Conference TENCON 2017*, Penang, 2017, pp. 2061-2066.

[15] S. Nguyen, J. Choi, K. Kim, "Suspicious traffic detection based on edge gateway sampling method," *Proceedings of the 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Seoul, 2017, pp. 243-246.

[16] M. S. Akbar, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, Springer, Berlin, Heidelberg, 2011, pp. 161-180.

[17] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, New York, NY, USA, 2013, pp. 413-424.

[18] S. Lim, J. Ha, H. Kim, Y. Kim and S. Yang, "A SDN-oriented DDoS blocking scheme for botnet-based attacks," *Proceedings of the 2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, Shanghai, 2014, pp. 63-68.

[19] N. I. G. Dharma, M. F. Muthohar, J. D. A. Prayuda, K. Priagung and D. Choi, "Time-based DDoS detection and mitigation for SDN controller," *Proceedings of the 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Busan, 2015, pp. 550-553.

[20] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," *Proceedings of the 35th Annual IEEE International Conference on Computer Communications INFOCOM 2016*, San Francisco, CA, 2016, pp. 1-9.

[21] P. Dong, X. Du, H. Zhang and T. Xu, "A detection method for a novel DDoS attack

against SDN controllers by vast new low-traffic flows," *Proceedings of the 2016 IEEE International Conference on Communications (ICC)*, Kuala Lumpur, 2016, pp. 1-6.

[22] S.M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against software defined network controllers," *Journal of Network and Systems Management*, vol. 26, no. 3, pp. 573-591, 2018.

[23] P. Kumar, M. Tripathi, A. Nehra, M. Conti and C. Lal, "SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN," *IEEE Transactions on Network and Service Management*, vol. 15, issue 4, pp. 1545-1559, 2018.

[24] Mininet. [Online]. Available at: http://mininet.org. last access 10/2/2019.

[25] noxrepo/pox: The POX network software platform – GitHub. [Online]. Available at: https://github.com/noxrepo/pox. last access 10/2/2019.

[26] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105-110, 2008.

[27] Scapy Project. [Online]. Available at: https://scapy.net/. Last access: 10/2/2019.

Jawad Dalou' *is a network engineer at the Information Technology and Communications Center at Jordan University of Science and Technology (JUST), Irbid, Jordan. Prior to that he worked as a teaching assistant in the department of network engineering at JUST from 2012 to 2017. He received his B.S. in computer engineering from JUST in 2011 and his M.S. in computer engineering from Yarmouk University, Irbid, Jordan in 2019. His research interests are in the areas of computer networking and network security.*

Basheer Al-Duwairi *is an Associate Professor at the department of Network Engineering and Security at Jordan University of Science & Technology. He received his B.S. in electrical and computer engineering from Jordan University of Science and University (JUST) in 1999, and his M.S. and PhD in computer engineering from Iowa State University, Ames, IA in 2002 and 2005, respectively. Over the past 15 years, Dr. Al-Duwairi investigated the area of network security focusing mainly on developing efficient schemes for DDoS mitigation, Botnet detection, Email spam filtering, and studying the emerging threat of Fast Flux Networks.*

Mohammad A. Al-Jarrah *is a professor of computer engineering at Yarmouk University, Irbid, Jordan. He Earned his Ph.D. in 2000 from University of Ohio, USA, MS and BS in Computer Engineering from Jordan University of Science and Technology, Jordan in 1992, 1989. Since 2000, he has been working with the Department of Computer Engineering at Yarmouk University. His research interests include image indexing and retrieval, multimedia systems, distributed systems, medical imaging, Network managements and security, data Encryption and many others.*