Early Detection of DDoS Attacks against SDN Controllers

Seyed Mohammad Mousavi and Marc St-Hilaire
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
seyed_mousavi@carleton.ca, marc_st_hilaire@carleton.ca

***Abstract*—A Software Defined Network (SDN) is a new network architecture that provides central control over the network. Although central control is the major advantage of SDN, it is also a single point of failure if it is made unreachable by a Distributed Denial of Service (DDoS) Attack. To mitigate this threat, this paper proposes to use the central control of SDN for attack detection and introduces a solution that is effective and lightweight in terms of the resources that it uses. More precisely, this paper shows how DDoS attacks can exhaust controller resources and provides a solution to detect such attacks based on the entropy variation of the destination IP address. This method is able to detect DDoS within the first five hundred packets of the attack traffic.**

***Keywords*—DDoS attack; SDN; Controller; Entropy**

I. Introduction

Software Defined Networks (SDN) provide a new and novel way to manage networks. In SDN, switches do not process incoming packets - they simply look for a match in their forwarding tables. If there is no match, packets are sent to the controller for processing. The controller is the operating system of SDN. It processes the packets and decides whether the packets will be forwarded in the switch or dropped. By applying this procedure, SDN separates the forwarding and the processing planes [1]. If the connection between the switches and the controller is lost, the network will lose its processing plane. This means that packet processing is no longer done in the controller and by losing the controller, the SDN architecture is lost.

One of the possibilities that can cause the controller to be unreachable is a Distributed Denial of Service (DDoS) attack. In DDoS attacks, a large number of packets are sent to a host or a group of hosts in a network. If the source addresses of the incoming packets are spoofed, which is usually the case [2], the switch will not find a match and has to forward the packet to the controller. The collection of legitimate and DDoS spoofed packets can bind the resources of the controller into continuous processing up to the point where they are completely exhausted. This will make the controller unreachable for the newly arrived legitimate packets and may bring the controller down causing the loss of the SDN architecture. Even if there is a backup controller, it has to face the same challenge.

The main goal of this paper is to detect a DDoS attack in its early stages. Since the controller software can be run on a laptop or a powerful server, the term "early" depends on the tolerance of the device and traffic properties. However, if the detection happens in the first few hundred packets, the mitigation can be applied before the controller is completely swamped with the large number of malicious packets. To accomplish this goal, a fast and effective method is needed that works within the controller. At the same time, it must be lightweight to avoid excessive processing power usage, specially, at the peak of an attack.

In the proposed solution, randomness of the incoming packets is measured. A good measure of randomness is entropy. Entropy measures the probability of an event happening with respect to the total number of events. For example, in a network of 64 hosts, all hosts should have a reasonably close probability of receiving new incoming packets[1]. This will results in, reasonably, high entropy. If one or a number of hosts start to receive excessive incoming packets, the randomness decreases and the entropy drops. This paper makes use of this property of entropy to detect an attack at its early stages. Based on the simulations that are done in this paper, we choose an experimental threshold for entropy and lower than threshold values will be considered attacks. Being programmable is one of the major advantages of SDN. Any time the network configuration changes, the threshold can be adjusted and it can even be adjusted while the network is running live traffic.

This paper is organized as follows. SDN and DDoS detection in SDN are reviewed in Section II. Section III explains how entropy can be used to detect DDoS. Section IV introduces the proposed detection method applied to the SDN environment. Finally, Section V presents the simulation results followed by the conclusion in Section VI.

II. Background and Related Work

In SDN, the Openflow protocol [1] is responsible for the communication between the controller and Openflow switches [3] through the secure channel. Every time a new packet arrives to the switch, it will look for a match in its tables. If a match is found, the packet will be forwarded according to the instructions that are in the table. Otherwise, the packet will be

[1] A new packet in the sense that there is no flow for it in the switch table and it has to be sent to the controller to be validated as a new flow.

sent to the controller for processing. The controller will either add a new flow to the table so the rest of similar packets will be forwarded or add a rule to drop similar packets. In a DDoS attack where packets have spoofed addresses, they will not have a match in the table and therefore, will be forwarded to the controller. If the arrival rate of packets reaching the controller is high, the controller will have all its resources bound into processing malicious packets. A high rate attack can completely overwhelm the controller and make it unreachable to the legitimate traffic of the network which results in losing the SDN architecture of the network.

A. DDoS Detection in SDN

Production networks have been studied for a long time and the types of threats to them are mostly known. SDN however, is a new architecture and there are not many papers discussing the topic of DDoS. Existing papers look at DDoS and apply the existing solutions to SDN. This means treating SDN as a normal production network where the role of the controller is ignored. In SDN, the switches have no control over incoming packets and they do not spend any time processing them. This also means that in an attack, the entity affected first and most is the controller.

Braga et al. [4], use Self-Organizing Maps (SOM) [5] machine learning technique for DDoS detection. In this method, SOM is trained by collecting the flow statistics from Openflow switches. The parameters that are checked for training SOM are average packets per flow, average bytes per flow, average duration per flow, percentage of pair flows, growth of single flow, and growth of single ports. The SOM will improve the statistics of its vectors as time passes and more statistics are gathered. However, this paper does not mention the effect of the attack packets on the controller or whether they go through the controller. In a DDoS attack, the packets are always spoofed or coming from several zombie nodes. Hence, packets have to go through the controller to get access to the target through a flow in the switch table. This case was not covered in this paper.

B. SDN for DDoS Detection

Other related work in security used the SDN architecture for intrusion detection in cloud networks. For example, Shin et al. [6] use Openflow as a flow regulation tool to monitor traffic in the cloud. The main idea is to reconfigure the flow of packets by using the Openflow protocol so it takes a path where a Network Intrusion Detection System (NIDS) is installed. By deploying the SDN architecture, there is no need to relocate and reinstall NIDS devices. Openflow will process the packets and add flow rules to routers and switches to go through a monitored path. Few algorithms are offered by the paper to find the shortest path for a secure route and each one is categorized by the time it takes for the controller to find it. In their paper, the SDN controller computes the shortest path to a monitored link in the network and it is not involved in the detection process.

Similarly, Hu et al. [7] propose an intrusion detection system that works on top of Openflow and SDN. This method introduces a central controller that monitors events from sub-controllers and called Event Processing Engine. The event processing software is a hyper controller that receives events from sub-controllers and processes them for possible attacks. The method is in the research phase and there are no experimental results.

III. DDoS Detection Using Entropy

Entropy measures randomness. The main reason entropy is used for DDoS detection is its ability to measure randomness in the packets that are coming to a network. The higher the randomness the higher is the entropy and vice versa. There are two essential components to DDoS detection using entropy: i) window size and ii) a threshold. Window size is either based on a time period or a number of packets. Entropy is calculated within this window to measure uncertainty in the coming packets. To detect an attack, a threshold is needed. If the calculated entropy passes a threshold or is below it, depending on the scheme, an attack is detected.

Let n be the number of packet in a window and p_i is the probability of each element in the window then, entropy (H) will be calculated as:

$$H = -\sum_{i=1}^{n} p_i \log p_i \qquad (1)$$

Entropy represents packet headers as independent information symbols with unique probability of occurrence. By selecting a window of some number, 10,000 for instance, and moving the window forward, a pattern will emerge with probabilities for each type of packet header. Drastic changes in the bins of each header that deviates from the average bin limits will alert the system of anomalies.

Entropy is a, fairly, common method for DDoS detection. Qin et al. [8] propose a method with a window of 0.1 seconds and three levels of threshold. This method is concerned with avoiding false positives and false negatives in the network. However, as the authors themselves mention, the method is time consuming and uses more resources. Ra et al. [9] propose a faster way of computing the entropy by basing the calculation on both the packet type and the volume of packets in the network. This method also uses a time period window. For the threshold, the authors ran several datasets to find a suitable experimental threshold and it is a multiple of the standard deviation of the entropy values. In this method, the false negatives are higher than other methods and false positives are lower. No percentage of accuracy is indicated. There is also no mention of resources used for fast computation.

Oshima et al. [10] propose a short-term statistics detection method based on entropy computation. “Short-term” here refers to calculating entropy in small size windows. The study proposes a window size of 50 packets for gathering statistics. In this method, different window sizes were tested for optimal entropy measurement and a size of 50 was the lowest size that effectively detected attacks. Instead of a threshold, the authors used a one-sided test of significance to confirm whether an attack was in progress or not. In the DDoS detection case, the rate of attack traffic was increased by increments while the normal traffic rate was kept constant. The method proved to be effective the most when the attack traffic was 75% or higher than the normal traffic.

The method described in [10], with its small window size, was partially used in this paper to detect the attacks in the controller. In [10], the entropy is measured for the source IP of all incoming packets regardless of whether a packet is coming to an existing connection or trying to establish one. The assumption is based on the fact that when DDoS attack happens, the source IP addresses are spoofed. Therefore, a high volume attack will increase the number of incoming packets which, in turn, increases the entropy. In our research, the entropy is based on the destination IP address for new incoming packets only. When an attack happens, the entropy will decrease since the IP address of the host(s) under attack will appear more frequently. Another difference is that our proposed detection method is specifically tailored for the SDN environment as the detection happens inside the controller. The proposed method is described in the next section.

IV. Proposed Method

A. *Utilizing SDN Capabilities*

For every new incoming connection, the controller will install a flow in the switch so that the rest of the incoming packets will be directed to the destination without further processing. Hence, any time a packet is seen in the controller, it is new. The other known fact about new packets coming to the controller is that the destination host is within the network of the controller. This assumption is based on the fact that one of the hosts or a subnet of hosts in the network is being attacked. The network consists of the switches and hosts that are connected to it. Knowing that the packet is new and that the destination is in the network, the level of randomness can be quantified by calculating the entropy based on a window size. In this case, maximum entropy occurs when each packet is destined to exactly one host. On the other side, minimum entropy occurs when all the packets in a window are destined for a single host. For instance, if the window has 64 elements and all elements appear only once, then based on Eq. (1), the entropy will be 1.80. If one element appears 10 times, the entropy will be 1.64. This property of the entropy will be used for calculating the randomness in the SDN controller. The central view of the controller over the network gives us the opportunity to evaluate the rate of incoming new packets to the controller and decide whether an attack is in progress or not.

Being able to quantify randomness and have minimum and maximum based on the entropy makes it a suitable method for DDoS detection is SDN. Using entropy, it is possible to see its value drop when a large number of packets are attacking one host or a subnet of hosts.

B. *Statistics Collection for Entropy*

One of the functions of the controller is collecting statistics from the switch tables. The controller monitors the existing flows and if there is an inactive flow for a period of time, it will remove that flow. In this paper, we will make use of that property and add another set of statistics collection to the controller. Since the approximate number of hosts in the network is known, we added the destination IP address of the new incoming packets to be collected into windows of size 50 (discussed later). The entropy of each window is calculated and compared to an experimental threshold. If the entropy is lower than the threshold, an attack is detected. These functions are small (in terms of the amount of code added) and transparent to the other functionalities of the controller. These two properties make the solution applicable to different controllers with minimal changes.

Although we chose the destination IP address, it is possible to add any other field to be collected for the entropy computation. For instance, VLAN tag can be used for virtual machines. It is also possible to add two fields such as destination IP and destination port.

C. *Window Size*

As will be shown in Section V, the window size should be set to be smaller or equal to the number of hosts in order to provide accurate calculations. In this paper, we set the window size at 50. The main reason for choosing 50 is the limited number of incoming new connection to each host in the network. In SDN, once a connection is established, the packets will not pass through the controller unless there is a new request. A second reason is the fact that a limited number of switches and hosts can be connected to each controller. Finally, a third reason for choosing this size is the amount of computation that is done for each window. A list of 50 values can be computed much faster than 500 and an attack in a 50-packet window is detected earlier. We also tested the entropy with three other window sizes and measured the CPU and memory usage. There is no difference in memory usage but the CPU usage slightly increases with respect to the window size. Considering the limited resources of the controller, this window size is ideal for networks with one controller and few hundred hosts.

D. *Attack detection*

To detect an attack in the controller, we monitor the destination IP address of the incoming packets. A function was added to the controller to create a hash table of the incoming packets. If an IP address is new in the table, it will be added with count one. If there is an instance of it in the table, its count will be incremented. After 50 packets, the entropy of the window will be calculated. Eq. (2) shows the hash table where x is the destination IP address and y is the number of times it appeared. To compute the entropy as shown in Eq. (1), we use equations (2) and (3) where W is the window and p_i is the probability of each IP address.

$$W = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\} \tag{2}$$

$$p_i = \frac{x_i}{n} \tag{3}$$

When each IP address appears only once, the entropy will be at its maximum. If an attack is directed towards a host, a large number of packets will be directed to it. These packets will fill most of the window and reduce the number of unique IPs in the window, which in turn, reduces entropy. We made use of this fact and set an experimental threshold. If the entropy drops below this threshold and that five consecutive windows have lower than threshold entropy, then an attack is in progress. Detection within 5 entropy periods is 250 packets in the attack, which gives the network an early alert of the attack. We tested with different values between one and five consecutive periods and, five has the lowest false positive for early detection. The other advantage of having five windows is the possibility of losing a switch or a broken link, which will cut off some hosts and reduce the number of new packets into the controller. This will cause a drop in the entropy and a false positive. Five windows can give the network admin enough time to act.

V. Simulation Results

The first part of our experiment consists in choosing a controller. The one that is used in this experiment is POX [11]. POX is widely used for experiment and it is fast, lightweight and designed as a platform so a custom controller can be built on top of it. It is an improved version of its predecessor NOX [12], and both are running on Python. All the SDN papers that are mentioned in this paper are using NOX. However, NOX is no longer in development [11], which led to the use of POX in this paper. The POX controller was modified to collect the destination IPs of the new incoming packets and a function to compute the entropy was also added.

A. *Network Emulator*

Mininet [13] is the network emulator that is used for this experiment as it is the standard network emulation tool for SDN. In fact, the code developed in Mininet can be moved to a real production network. Using Mininet, a tree-type network of depth two with nine switches and 64 hosts was created. Open Virtual Switch (OVS) [14] was used for network switches. Mininet was run native on a Linux machine running Ubuntu OS.

B. *Traffic Generation*

Packet generation is done by Scapy [15]. Scapy is a very powerful tool for packet generating, scanning, sniffing, attacking and packet forging. Scapy is used here to generate UDP packets and spoof the source IP address of the packets. Two types of traffic were generated: normal and attack traffic. The attack traffic is destined to one host and has a higher rate than normal traffic.

C. *Choosing a Threshold*

The detection mechanism in our solution dictates that if the entropy is lower than the threshold, and it persists for five consecutive windows, an attack is in progress. To find the range for an optimal threshold, we ran a series of experiments to see the effect of an attack on the entropy. In this paper, the experiments cover an attack to a single host. To compare different rates of incoming packets, we controlled the rate of normal and attack traffic to increase and decrease the intensity of DDoS on the controller. Eq. (4) is used for showing the rate R of incoming attack packets to normal traffic attacks. P_a and P_n are the number of attack packets and normal traffic packets respectively.

$$R = \frac{P_a}{P_n} \times 100\% \tag{4}$$

We ran a 25% rate attack on one host for 25 times to find a suitable threshold. This threshold is the highest entropy of all cases so it will enable the controller to detect any attack with packets occupying 25% of the incoming traffic or more. We call it 25% rate attack. Table I shows the threshold and compares it to normal traffic values. The threshold is set to 1.31. To get this value, the following process was done:

a) Calculate the lowest value that normal traffic entropy can reach. This is equal to normal traffic mean entropy minus confidence interval, 1.4665.
b) Calculate the highest value that attack traffic entropy can reach. This is equal to attack traffic mean entropy plus confidence interval, 1.3047.
c) Find the difference between the two, 0.1618. We have a drop of 11%.

Even though the above calculation shows that 1.3047 could be the threshold, after 25 times running the simulation, we found that 1.31 has much less false negatives. Hence, a threshold of 1.31 will give us a clear cut for detecting any DDoS attack that will occupy 25% or more of the incoming traffic.

TABLE I. Threshold Value Calculations

	Normal Traffic	**25% Rate Attack**
Mean entropy	1.47	1.3
Standard Deviation	0.009	0.012
Confidence interval	±0.0035	±0.0047
Confidence interval Max	1.4735	1.3047
Confidence interval Min	1.4665	1.2953
Difference of Normal traffic min and Attack traffic max.	0.1618	
Threshold	1.31	

D. *Test Cases*

To test this solution, we ran attacks with three different intensities on a single host. Starting with a 25% rate attack, traffic rate was then increased to 50% and 75% rates. In each case, 4,000 packets were sent to the controller. The number is chosen to create a better graph. For the 25% case, 500 attack packets were sent. These packets are sent four times faster than the normal traffic rate, which will occupy 25% of the traffic that goes to the controller (i.e. 25% of the packets in each window). In the 50% and 75% cases, because the number of attack packets in the window increases, the drop in the entropy will be a small gap. Thus, the attack traffic packets were increased to 1,000 packets to make the drop in the entropy more visible in the graph. Fig. 1 shows how the entropy drops, compared to the normal entropy. The 25% rate attack was run 25 times and the other two cases were run 15 times each. In the 25% rate traffic, in 25 runs, we had a success rate of 96% in detecting DDoS attack. For the other two cases, the success rate is 100%. Table III shows the entropy drop in each case.

TABLE II. ENTROPY OF ATTACKS COMPARED TO THRESHOLD

Traffic Type	Mean Entropy	Threshold – attack entropy	Confidence interval
25% rate on host	1.3	0.01	±0.0035
50% rate on host	1.06	0.24	±0.0099
75% rate on host	0.56	0.74	±0.03

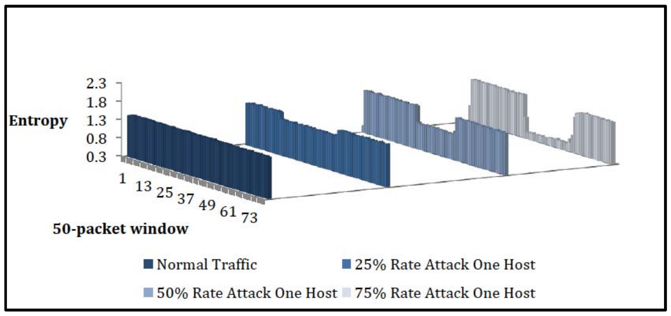

Fig. 1 *Comparison of Entropy for Different Attack Rates*

In Fig. 1, the drop for 25% rate attack is very small compared to the other cases, which shows the effectiveness of this solution in detecting DDoS. We also monitored the CPU usage graph on the Linux machine to examine the effect of adding this DDoS detection method to POX controller. We ran the 25% rate attack in a POX controller without detection function and then, ran the attack with the modified POX controller. The difference is insignificant.

The type of traffic, whether it is UDP, TCP or ICMP, does not affect the solution as long as we look for a valid header field in the incoming packet. We tested the solution with UDP and TCP and the results were the same.

VI. CONCLUSION

This paper presented a lightweight and effective solution for detecting DDoS attack in SDN. By making use of the central role that the controller plays in SDN, we were able to use the destination IP address for detecting attacks within the first 250 packets of the traffic that carries malicious packets. Also, the threshold that was chosen is set to the lowest possible rate of attack traffic. Nonetheless, the detection rate for this threshold was 96%. In addition to early detection of the attack, the solution's addition to the controller is transparent both in terms of resources and functionality. It provides the detection for both the controller and the attacked host.

In [4], the SOM had to be trained for hours and large matrix computations had to be done to provide overall protection. The other problem with that method is the fact that the solution runs alongside the controller. The proposed method in this paper runs inside the controller and complies with the centralized nature of SDN where the operating system controls the network and detects the threat against itself and the network. Its lightweight nature makes its addition to the controller seamless and the amount of resources that are used by it is minimal.

One of the main advantages of this solution is its flexibility. Any parameter in the solution can be modified to fit the requirements of the controller. Detection field (i.e. destination IP address), window size and threshold can all be set to correspond to the desired values even in real-time. This is all possible through the controller's interface. To the best of our knowledge, this is the first work that uses entropy for DDoS detection in SDN controller and the first work to, specifically, address DDoS attacks on the controller.

For future work, we want to investigate how the proposed solution can handle a subnet attack. Instead of having a single host under attack, what if the entire subnet is under attack? We believe the proposed model will be able to detect these attacks. Another interesting avenue consists to perform DDoS detection in a multi-controller network. Finally, once the attack is detected, mitigation of the attack is also an important step that will be looked at.

REFERENCES

[1] "Open Networking Foundation" 2014 [Online]. Available: www.opennetworking.org

[2] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kin- dred, "Statistical approaches to DDoS attack detection and response", Proc. of DARPA Information Survivability Conf. and Exposition, Vol.1, pp.303-314.

[3] "SDN Central". 2014, [Online]. Available at: http://www.sdncentral.com/announced-sdn-products/

[4] E. Mota, A. Passito R. Braga, "Lightwight DDoS flooding attack detection usingNOX/Openflow," IEEE 35th conference on Local Computer Networks, 2010, pp. 408-415.

[5] S. Ostermann, B. Tjaden M. Ramadas, "Detecting anamalous network traffic with self-organizing maps," Recent Advances in Intrusion Detection, 2003, pp. 36-54.

[6] G. Gu S. Shin, "CloudWatcher: Network Security Monitoring Using Openflow in Dynamic Cloud Networks (or: How to provide security monitoring as a service in clouds?)," 20th IEEE International conference on Network Protocols , 2012, pp. 1-6.

[7] W. Su, L. Wu, Y. Huang, S. Kuo Y. Hu, "Design of Event-Based Intrusion Detection System on OpenFlow Network," IEEE International Conference on Dependable Systems and Networks (SDN), 2013, pp. 1-2.

[8] Z. Qin, L. Ou, J. Liu, A. X. Liu J. Zhang, "An Advanced Entropy-Based DDoS Detection Scheme," International Conference on Information, Networking and Automation, 2010, pp. 67-71.

[9] I. Ra G. No, "An efficient and reliable DDoS attack detection using fast entropy computation method," International Symposium on Communication and Information technology, 2009, pp. 1223-1228.

[10] T. Nakashima, T. Sueyoshi S. Oshima, "Early DoS/DDoS Detection Method using Short-term Statistics," International Conference on Complex, Intelligent and Software Intensive Systems, 2010, pp. 168-173.

[11] M. McCauley, "NOXREPO". 2014, [Online]. Available at: http://www.noxrepo.org/

[12] T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker N. Gude, "NOX: Towards an Operating System for Networks," Computer Communication Review, vol. 38, no. 3, pp. 105-110, Jul 2008.

[13] Mininet. 2014, [Online]. Available at: http://mininet.org

[14] Open Vswitch. 2014, [Online]. Available at: http://openvswitch.org

[15] Scapy. 2014, [Online]. Available at: http://www.secdev.org/projects/scapy