SDN Threat Vectors and Security Solutions

Anand Kumar Nandan Goudar Sachin Uplaonkar Amogh Eshappa Setra

San Jose State University San Jose State University San Jose State University San Jose State University

[Anand.kumar@sjsu.edu](mailto:Anand.kumar@sjsu.edu) [Nandan.goudar@sjsu.edu](mailto:Nandan.goudar@sjsu.edu) [sachin.uplaonkar@sjsu.edu](mailto:sachin.uplaonkar@sjsu.edu) [amogh.eshappasetra@sjsu.edu](mailto:amogh.eshappasetra@sjsu.edu)

Abstract—Software-defined networking architectural framework eases the life of the network administrators by isolating the data plane from the control plane. This facilitates easy configuration of the network, provides a programmable interface for developing applications related to management, security, logging etc. and the centralized logical controller gives more control over the entire network [3], which has the total visibility of the network. These advantages of SDN also expose the network to the vulnerabilities and the impact of the attacks are much severe when compared to conventional networks, where the network devices in itself provided protection from the attacks and limits the scope of the attacks. In this paper, we explore various attacks that can be launched on SDN at different layers. We also evaluate some of the existing security methods in mitigating the attacks. We also explore a possible solution to prevent DDoS attacks using entropy. A Distributed Denial of Service (DDoS) attack is a DoS attack utilizing multiple distributed attack sources. Every network in the system has an entropy. Increase in randomness causes decrease in entropy. To mitigate this threat, this project 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 project 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. Based on this value if it drops below threshold , we are blocking the specific port in the switch and bring the port down. This method is able to detect DDoS within the first five hundred packets of the attack traffic.

Keywords—SDN, security, threat vectors, networking, DDoS, Entropy,threshold, randomness, programmable, launch attack ,generate traffic .

# **Introduction**

Why is SDN security important? We know SDN network has separated data plane and control plane unlike the conventional network architecture. This network architecture can be implemented as cloud service. Cloud services are exploding and big organizations and enterprise network administrators are migrating to the SDN-based network implementations. These virtual technologies provide predictability, manageability and good quality of service. In parallel with added advantage, importance of security provision in this centralized managed network has become one of the main concerns. With the single centralized virtual server running as controller, which basically install and manages the flows in the data plane network agents through overflow communication protocol which makes the controller a primary victim for the attacker.

* Security standards implementation in overflow communication has not been defined and product developers are implementing their proprietary methods.
* The programmable aspect of Software Defined Networks also makes them more vulnerable to a number of malicious code exploits and attacks.
* The southbound interface of an SDN can also easily be targeted with diverse denial of service and side channel attacks.
* Configuration errors of SDN can have more serious consequences than in traditional networks.
* Establishing trust is crucial.

# **Problem Statement**

With a listing of different security concerns in Software Defined Networks, one of the main security threats we are concentrating upon in this research work is on Distributed Denial-Of-Service. When a large number of packets are forwarded to a network device with an intent to either stop the service or decrease the performance then such attacks are termed as Distributed Denial-of-service attacks. 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 they usually are, the switch will not find a match and has to forward the packet to the controller. The collection of legitimate and the DDoS spoofed packets can bind the resources of the controller into continuous processing that exhausts them. 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

This kind of attacks can be detected at an early stage by monitoring few hundreds of packets based on the entropy changes. The early detection of DDOS attack prevents the controller going down. The term “early” is subjected to tolerance level and traffic being handled by the controller [6] [7]. If detection happens early say first few hundreds of packet then, the impact of flooding of malicious packets can be controlled significantly. The early detection mechanism must be of light weight and should have a high response time. The high response time saves the controller in the period of attack to regain the control by terminating the DDOS attack.

# **Motivation**

The main goal of this research is detecting a DDoS attack in its early stages. The term early depends on the network itself. Since the controller software can be run on a laptop or a powerful desktop, the term early would depend on the tolerance of the device and traffic properties. However, if the detection happens in the first few hundred packets, the mitigation is applied before the controller is completely swamped with the large number of malicious packets. Figure 1.1 shows a simple DDoS attack on the controller where the normal incoming packet rate is around 100 packets per second. When the attack happens, the rate rises sharply to, approximately, 250 packets per second. The simulated DDoS attack was directed to a SDN controller that is connected to a network with 64 hosts and nine switches. The attack lasted for 40 seconds and sent 500 packets with spoofed source addresses all destined for one host. For the purpose of this research, all packets will have spoofed IP addresses. This way, the switches do not have a match and all the packets are sent to the controller.



**Figure 1.1 Sample attack on the controller**

To accomplish this goal, a fast and effective method is needed that works within the controller. Collecting statistics is one of the functions of the controller. In this study, this attribute is used for adding another set of statistics collection to the controller; destination IP addresses. In our 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 instance, in a network of 64 hosts, all hosts should have a reasonably close probability of receiving new incoming packets. This will results in, reasonably, high entropy. 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 for a new flow. If one or a number of hosts starts to receive excessive incoming packets, the randomness decreases and entropy drops.

The solution is, partially, based on a paper on DDoS detection proposed by Oshima et al. [33] where a small window of 50 packets is used to calculate the entropy of incoming packets.

1. **Project Objective**

# **Technical discussion and R&D**

In this section, we discuss the current threat vectors to SDN and the proposed possible solutions. Below are some of the examples:

* Forged or faked traffic flows caused by faulty devices or malicious users. Here the attacker can use DoS against OpenFlow switches and controller. It can be solved by authentication mechanism and Intrusion Detection System.
* Attacks on vulnerabilities in switches, in this type of attack the packets are dropped or it can slow down the movement of packets in the network, clone or deviate network traffic or overload the controller. One of the solution can be an autonomous trust system and other can be a mechanism to monitor and detect abnormalities in the network.
* Attacks on control plane communication like the DoS/data theft and loopholes in TLS/SSL [5]. These type of attacks on control plane can be solved by considering Oligarchic trust model and through valid certification authorities.
* Another attack on the controller is to exploit the existing vulnerabilities, the solution is replication- to detect, remove or mask abnormal behavior [9]. To have a number of controllers, protocols and programming languages and recovery- periodically refresh to clean and bring back the machine to an original state.
* Lack of mechanism to ensure trust between controller and management application certification between controller and network devices [8]. Solution to such type of attacks is to have a trust management system.
* Host Location Hijacking Attack [1] can be avoided to verify the legitimacy of Host Migration using two conditions, Precondition where a controller must receive a Port Down signal before the host migration finishes. And post condition is to check whether the host entity is unreachable in the previous location after the completion of the host migration.

In this research, we studied SDN to find possible weak points with respect to DDoS attacks. This study led to the controller. We found that the controller is the weak link in a DDoS attack scenario. With protecting the controller in mind, we studied different methods in DDoS detection that could be used in the controller. However, the structure of SDN posed its limitation on the type of the solution and the way it was implemented.

These limitations were:

1. Limited resources of the controller.
2. The need to detect the attack before the controller is out of reach due to the large number of malicious packets.

The main objectives are:

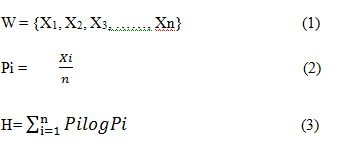
a)To find the weak link in SDN when DDoS happens.

b) Find a solution to detect DDoS in SDN before it overwhelms the controller.

1. **Early Detection of DDoS Using Entropy**
2. **Measuring the Randomness**

The main reason for choosing entropy is its ability to measure randomness in a network. The higher the randomness the higher is the entropy and vice versa.

Let W be a set of data with n elements and X is an event in the set. Then, the probability of X happening in W is shown (2). To measure the entropy, referred to as H, we calculate the probability of all elements in the set and sum that as shown in (3).



The entropy will be at its maximum if all elements have equal probabilities. If an element appears more than others, the entropy will be lower. The size of W is called the window size. If there is a continuous stream of incoming data, in our case the data is packet header, it will be divided into equal sets that are called windows. In the window, each element and its occurrence are counted. For instance, if the window has 64 elements and, all elements appear only once, the entropy will be 1.80. If one element appears 10 times, the entropy will be 1.64. This property of entropy will be used for calculating the randomness in the SDN controller.

1. **Entropy for DDoS Detection**

Entropy is the method used in this research to detect DDoS attacks in SDN. A look at the used methods in non-SDN networks is necessary before introducing it in SDN. Since there is no research in using this method in SDN, we have to rely on what is done in non-SDN research. There are two essential components to DDoS detection using entropy; window size and a threshold. Window size is either based on a time period or 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.

Qin et al. [2] propose a method with a window of 0.1 seconds and three levels of threshold. This method is concerned with avoiding false positive and false negatives in the network. However, as the authors themselves mention, the method is time consuming and uses more resources.

Ra et al. [3] propose a faster way of computing entropy by basing the calculation on both 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 threshold and it is a multiple of standard deviation of 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.

Entropy has been used in different ways to detect DDoS attacks in the network but, to the best of our knowledge, it has not been used in SDN. In SDN, when passing packets to the controller, the limitation of available resources and the quick detection of attacks are key features of any detection scheme. In this research, we will apply entropy for DDoS detection with the above limitation of the controller in mind.

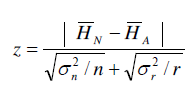
1. **Statistics for Entropy Detection**

Before discussing our solution, we will look at an entropy-based DDoS detection method that is used in a non-SDN network.

Oshima et al. [1] 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 thismethod, different window sizes where tested for optimal entropy measurement. Table 2.1 shows the results of the tests for different window sizes.

**Table 2.1 Entropy of different window sizes [1]**



In Table 2.1, W is the window size, HN is the entropy in normal condition, HA is entropy during an attack, SN and SA are the standard deviation of entropy for normal and attack traffic conditions respectively. Z is the test of significance. More precisely, it is a test of validity for the hypothesis between two averages of different populations. When it is higher than 1.64, the hypothesis is valid. In Table 2.1, it can be seen that for a window size of 50, z is 1.7. The value can be computed using (4).

(4)

(4)

σn and σr are the same as Sn and Sa . n is the population of normal traffic packets (value of n is not given) and r is set to 25. To test the hypothesis, a one-sided test of significance with 5% confidence interval was used. The formula for the one-sided test is shown in (5) where x is the mean of the population,

μo is the sample mean, σ is the standard deviation and n is the sample count.

(5)

We have chosen the window size to be 50 for this research. 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. The other reason is the fact that a limited number of switches and hosts can be connected to each controller. The third reason for choosing this size is the 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. Table 2.2 shows that there is no difference in memory usage but CPU usage increases with window size.

**Table 2.2 Window size comparison**

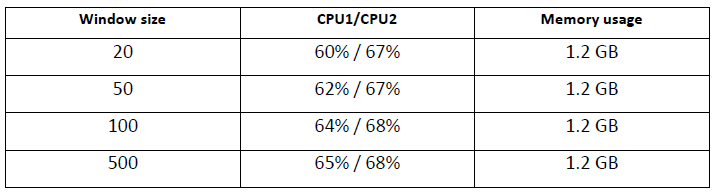
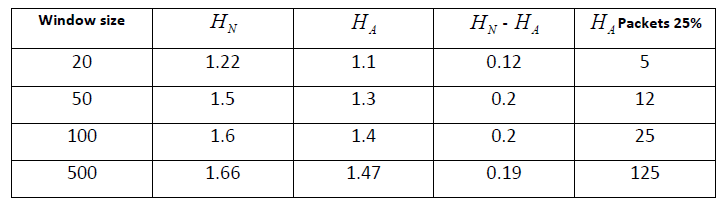


Table 2.3 shows the difference in entropy and the number of attack packets from each window size. HN is the normal traffic entropy, HA is the attack traffic entropy and HN - HA is the difference. Last column shows the number of malicious packets when the attack traffic is 25% of all incoming packets. This is the lowest attack traffic rate that our method can detect with accuracy.

In a window of size 20, the difference of entropies is less than 10% making it difficult to choose a threshold. With only five packets, probabilities of false positives will increases. On the other side, window of 500 does not offer a better difference of entropies and takes a much longer time than a window of 50 to compute entropy. The difference is 0.19 for 500 packet window size which is 11% drop in normal traffic entropy. The difference in the window size of 50 is 0.12 which 10% drop in normal traffic entropy. Difference of 1% does not justify choosing a 10 times bigger window size.

Window sized of 50 and 100 look close. Because the number of hosts in our test network is less than 100, we chose 50. It is very easy to change window size in the controller and this flexibility is the advantage of SDN.

**Table 2.3 Comparison of five windows**



1. **Early Detection in Openflow Controller**

As it was shown before, one function of the controller is collecting statistics from all Openflow switches to detect inactive flows. These flows will be removed if they do not receive any packets for a period of time. This time period is called time-out in the Openflow specification and it can be set to different values.

For a lightweight solution, we propose adding another set of statistics to the controller. In this work, it is the entropy of the destination IP address in the controller. The function will determine if a higher than normal rate of incoming packets destined to the same destination.

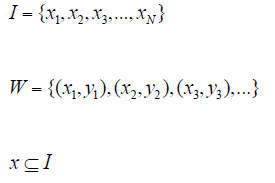
In the previous section, the window size was set to be 50. The assumption is that the network has 50 or more hosts connected to it. One other component of DDoS detection by entropy is selecting an appropriate threshold, which will be discussed in the next chapter.

In the new function, every 50 Packet In messages will be parsed for their destination IP address and the entropy of the list will be computed. The calculated entropy, then, will be compared to a threshold. If the calculated entropy is less than the threshold and it persists for a minimum of 5 consecutive entropy periods, it will be considered an attack. Detection within 5 entropy periods is 250 packets in the attack, which gives the network and early alert of attack. We tested with values one to five consecutive periods and five has the lowest false negative and positive for early detection.

With a window of 50 packets and a network of 50 hosts or more, maximum entropy, HMAX is when each of the 50 packets is equally distributed among all the hosts. When an attack happens, the number of packets going to the same destination host, or the same subnet, is much higher so it will make the target unreachable to legitimate traffic. This would be the main objective of the attack. The other factor in an attack is its flow to the target. The attack packets will be targeting a single host or a subnet. If the rate of attack to a host is higher than the normal traffic level, which is always the case, the number of packets to that particular host in a window will increase. Because of that, the entropy will fall with a certain percentage. If it falls below the threshold, it is an attack.

One advantage of our method is the liberty of testing the controller with different attack rates to calibrate a threshold. In SDN, the controller can be connected to a simulator and tested then be deployed in the field to accept production network traffic. This property allows for the threshold to be tested before applying it.

Let I be the IP addresses of all hosts connected to the network, see (6), and W be the window containing new packets’ destination IP address x and their number of occurrence y , in (7).

 (6)

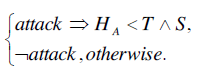
(7)

(8)

Then the entropy will be at its maximum if each destination IP is unique.

 (9)

If the above condition does not hold, then some IP addresses have appeared more than once. Two conditions are chosen to be the trigger for an attack in our method. One is the threshold and the other is the continuity of the attack. There might be glitches in the network that cause irregularity in normal traffic. If a link to a switch goes down or some hosts become temporarily unavailable, the entropy might fall and trigger a false positive. To avoid false positives of this type, we propose a limit for the number of consecutive low entropy windows. Based on that, the condition for declaring an attack is shown in (10) where T is the threshold, S is an array of five windows with lower than T entropy. “-” is the sign of negation. An attack happened if entropy of attack HA, is smaller than the threshold and, having five consecutive lower than threshold entropies is true. Otherwise, there is no attack:



(10)

1. **Comparison of Different Detection Methods to SDN Entropy**

In [4], a machine learning method was used to learn the behavior of the network and based on that, decide whether an attack is in progress or not. This method is, mostly, used in non-SDN networks. When used in SDN networks it follows the same procedure and does not take into account the effect of DDoS on the controller. The solution has to run alongside SDN and has to be trained for a few hours before it can be used in the network. One other issue is the fact that SDN may reconfigure the network frequently. This means the Self-Organizing Maps solution has to be trained again for better protection. Finally, as the network expands, the neurons of the Self-Organizing Maps have to increase resulting in expansive neurons in the network and a lightweight solution turns into a heavy drag for the network. Our proposed solution runs within the controller and can be changed to fit the requirements as needed.

Shin et al. [5], propose Openflow for finding the shortest path to Network Intrusion Detection System devices. The solution requires the addition of NIDS devices along the links of the network to monitor traffic for suspicious activity. In our solution, we propose using the controller itself for the detection of any attack. Although our solution is not tested on the cloud, the principle remains the same. The only difference is that the hosts are virtual hosts and the statistics can be changed from IP address to VLAN tag.

Like [5], Xing et al. [6] propose a DDoS detection method alongside SDN. SNORT is a DDoS detection tool itself and its combination with SDN is, again, using a non-SDN tool for SDN. The main purpose of our solution is making the detection process transparent by embedding it in the controller. The closest solution to the one proposed in this project is [7]. This method adds an event processing module on top of the existing controllers and renames them to sub controllers. The event processing module is considered a hyper controller receiving events from the sub-controllers and processes them for possible attacks. It seems that the hyper controller is supposed to have the bigger picture of the network for a better view of the attack. This, however, is the speculation of the writer. No algorithm is mentioned for the hyper controller attack detection and the solution has not been tested. The entropy method uses a different approach by assigning the detection task to each controller thus reducing the complexity. In none of above solutions the controller has been the center of attention. Being the driving force of SDN and the operating system, it is the most essential component in the structure. This work is mainly concerned with detecting DDoS threats that are endangering the controller which is, in part, also protecting the hosts.

1. **Simulation and Results**

In this chapter, an Openflow controller will be connected to a network to form a SDN structure. Then, the entropy of the traffic to the controller is examined under normal and attack conditions.

1. **Controller**

The first part of our experiment is choosing a controller. There are few famous controllers available. The one that is used in this experiment is POX [8]. Pox is widely used for experiments, 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 [9], and both are running on Python. POX works on Linux, Mac OS and windows, and it has topology discovery. For completeness, three other controllers should be mentioned. Floodlight [10] is another widely used controller that is open source and written in Java. One advantage of Floodlight is facilitating application interface to the controller so they can run alongside it. Beacon [11] is another Java based controller that is open-source and has high throughput and low latency. OpenDaylight [12] controller is the most recent addition to Openflow controllers. It meant to be a common platform for all SDN users. Recently, OpenDaylight announced its first release: Hydrogen. All the SDN papers that are mentioned in this project are using NOX. NOX is no longer in development [8], which led to the use of POX in this project.

1. **Network Emulator**

Mininet [13] is the network emulator that is used for this experiment. It is the standard network emulation tool that can be used for SDN. Mininet can prototype a network on a laptop or PC by using kernel namespace feature. Network namespace provides individual processes with their own network interfaces, ARP tables and routing tables. Mininet makes use of this feature of the kernel. It uses process-based virtualization to run switches and hosts on the kernel. Large networks1 with different topologies can be emulated and tested. In fact, the code developed in Mininet emulation can be moved to a real production network.

1. **Packet generation**

Packet generation is done by Scapy [14]. It 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. Python programming language is used in POX. The code for generating random source IP addresses and host IP addresses is in Python. The function “randrange” is used which is inheriting the function “random”. This function produces a uniform random float in the range [0.0, 1.0). The generated float has 53-bit precision and has a period of 219937-1 [15]. This number shows a long period of random number generation which will result in generating random numbers with uniform distribution. These numbers are joined together to form spoofed source IP addresses. Two other parameters that we set in Scapy are: type of packets and interval of packet generation. UDP packets are used for both attack and normal traffic. The interval was set to suit the test case. For instance, for an attack with 25% rate, normal traffic interval is 0.1 seconds and attack traffic is 0.025. This gave us windows with 25% of packets destined to one host. The code for generating normal and attack is shown in Appendix B and C respectively.

1. **Network Setup**

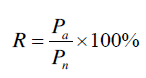
The experiment was done on a MacBook Pro laptop with a dual core processor, 2.8 GHz of power and 8GB of ram. The operating system is OS X and Mininet version 2.2.1 was run on Virtual Box. Using Mininet, a tree-type network of depth two with nine switches and 64 hosts was created. Figure 3.1 shows the network. Open Virtual Switch (OVS) [16] was used for network switches. OVS is a software switch that runs both on hardware and software. For this work, there is no difference between Openflow and OVS switch. Both do the same job and both are supported in Mininet. In Figure 3.1, all switches refers to Openflow enabled switches. The L3\_learning module of POX was used for the controller.



**Figure 3.1 Experiment Network with 9 switches and 64 hosts**

1. **Choosing a threshold**

After setting all the parameters for a full network, a threshold is needed to detect DDoS attacks. The detection mechanism in our solution dictates that if the entropy is lower than the threshold, and it persists for five windows in a row, 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. 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. (11) Is used for showing the rate R of incoming attack packets to normal traffic attacks. Where Pa and Pn are the number of attack packets and normal traffic packets respectively.



(11)

Table 3.1 [17] shows the threshold and compares it to normal traffic values. Theoretically the threshold is set to 1.31. To get this value the following was done:

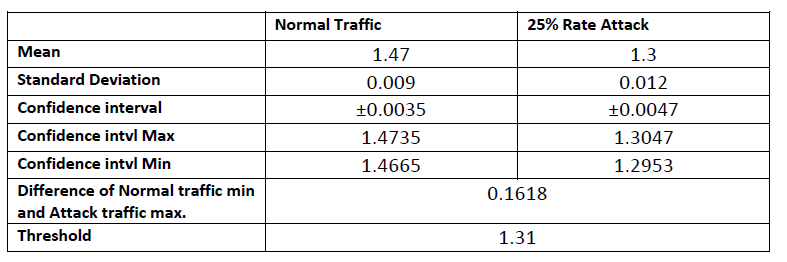
a) Calculated the lowest value that normal traffic entropy can reach. This is equal to normal traffic mean entropy minus confidence interval, 1.4665.

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 of the two, 0.1618. We have a drop of 11%.

Hence, a threshold of 1.31(theoretical value) will give us a clear cut for detecting any DDoS attack that will occupy 25% or more of the incoming traffic.

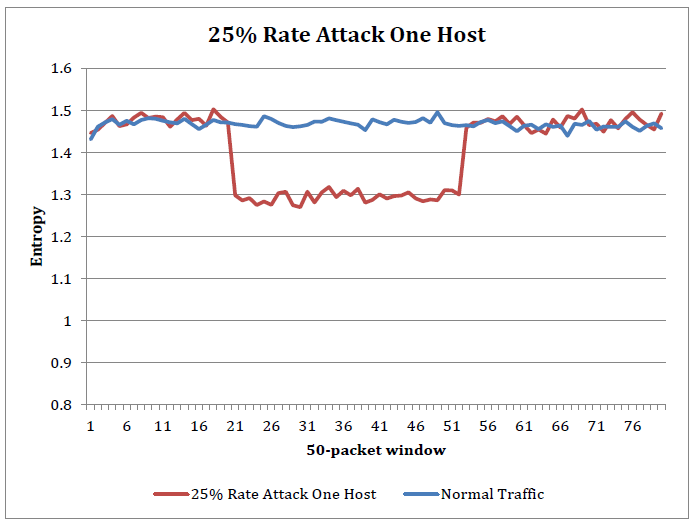
**Table 3.1 Threshold value calculation [17]**



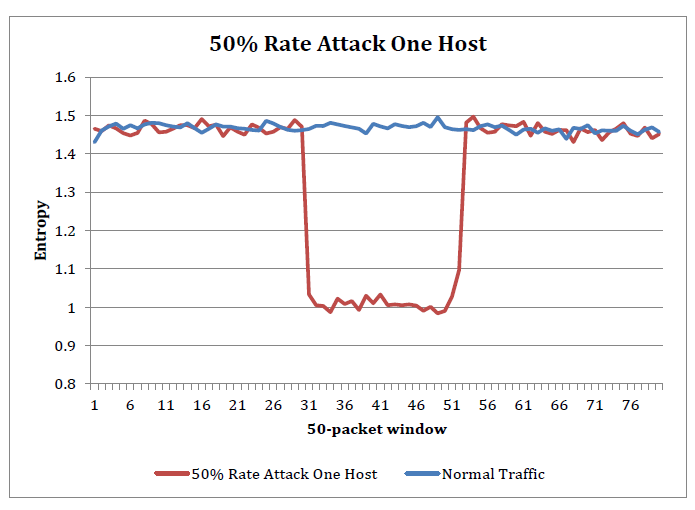
1. **Test Cases**
2. **Attack on One Host**

Figure 4.1 shows change of entropy in 25% rate attack. In all the graphs, the blue line is the normal traffic. The red line shows the transition from normal traffic to attack and back to normal traffic. In Figure 4.3, a distinct difference between the normal and attack traffic entropy can be seen. In the graph, the first six entropies are well below our threshold of 1.47(theoretical value) but the practical value obtained for our test case was 1.05. In table 3.1, the lowest point of confidence interval for normal traffic was 1.4665 and the highest point of attack entropy was 1.3047. The difference of these two values, 0.1618, shows 11% drop in entropy which is 34 bigger than the 25% attack traffic confidence interval, 0.0047. This result shows that this method can, easily, detect any attack occupying 25% or more of the incoming traffic when it is destined for one host. Since 25% rate was run 25 times, we computed the success rate of this method based on this 25 runs of the simulation. Within these runs, only one false negative was detected where an attack was in progress but was not detected by the controller. This shows a success rate of 96% in detecting DDoS attacks within the first 250 incoming packets. For all other cases, the success rate is 100% and no attack passed unnoticed.

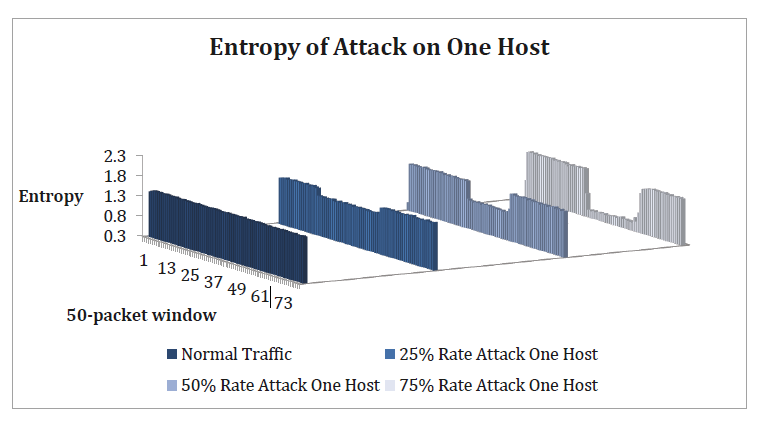
In 50% rate attacks, the window of attack is deeper and narrower showing a bigger drop in entropy. In one host attack, 500 packets are sent as attack traffic. When the rate of attack increases and the number of generated attack packets is fixed, the percentage of attack packets in the window increases. This will results in a deeper and narrower graph of attack [17].



**Figure 4.1 25% rate attack on one host [17]**



**Figure 4.2 50% rate attack on one host [17]**



**Figure 4.3 Comparison of entropy for different attack rates**

1. **Conclusion and Future Work**
2. **Conclusion**

Protecting the operating system of SDN (i.e. the controller) by detecting DDoS attacks was the center of this research. The challenge in detecting any threats to the controller is early detection. Although the term “early” can be used loosely in detecting an attack, we quantified the early detection to the first 250 packets of traffic as minimum and 500 packets maximum. This solution is not only efficient in detection, it has minimal code addition to the controller program and does not increase CPU load in either normal or attack condition. There are different methods for detecting attacks and each method is used differently. In this project, we focused on a solution that works particularly well for SDN, based on its specifications, points of strength and limitations. We made use of the fact that SDN specification dictates the forwarding of new packets to the controller. We took into account the abilities of the controller and its broad view of the whole network and used that for adding entropy statistics collection. Finally, understanding the importance of keeping the controller connected to the network at all times, we came up with a solution to detect any threat at its very beginning.

By applying entropy as a detection method, we were able to detect attacks on one host or a subnet of hosts in a network. In the case of one host, our detection method proved to be able to, successfully, catch drops in entropy when attack packets are as low as 25% of the incoming traffic to the controller. The success rate this method is 96% when using a threshold of 25% rate attack packets per total traffic. The closest method in non-SDN networks can detect an attack when 75% to 100% of traffic is DDoS. We believe that this is an effective method in addressing the detection of DDoS in SDN with accuracy and efficiency.

1. **Future work**

One limitation that our method has is the detection of attacks when the entire network is being targeted by DDoS. When malicious packets are targeting every host, entropy might not change by a large margin. Detecting such attacks will be an addition to this research.

Having addressed the detection in one controller network, two more tasks to be done are:

i) Detection of attack in a multi-controller SDN structure.

ii) Mitigation of the attack.

In SDN, networks are connected to controllers and, several controllers might be connected to each other. Detecting an attack in one of them could show the source of the attack and make discovery of the source much easier. This method requires an inter-controller communication that sends the threat alert to all the controllers. Adding this communication process to SDN will be an extension to the current work and a topic for future work.

Mitigation of DDoS in SDN is the next future work for this research. The first step of mitigation will be detection of the source or sources of the attack. Adding more statistics collection to the controller will enable it to monitor the flow rate at the switch level where attack flows are directed to the controller. Then, more elaborate techniques can be used to pinpoint the malicious hosts. This is a very interesting future work that can be a baseline for any detection scheme in SDN structure.

**References**

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

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

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

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

[5] 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?)," in 20th IEEE International conference on Network Protocols, 2012, pp. 1-6.

[6] D. Huang, L. Xu, C. Chung T. Xing, "SnortFlow: A openflow-based Intrusion Prevention System in Cloud Environment," Second GENI Research nad Educational Experiment Workshop, pp. 89-92, 2013.

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

[8] M. McCauley. (2013, Nov) NOXREPO. [Online].

<http://www.noxrepo.org/>

[9] 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.

[10] Big Switch Networks. (2014, Mar) Project Floodlight. [Online].

<http://www.projectfloodlight.org/floodlight/>

[11] Linux Foundation. (2014, Jan) Open Daylight. [Online].

<http://www.opendaylight.org/>

[12] Linux Foundation. (2014, Jan) Open Daylight. [Online].

<http://www.opendaylight.org/>

[13] (2014, Feb) Mininet. [Online].

<http://mininet.org/>

[14] (2014, Feb) Scapy. [Online]. <http://www.secdev.org/projects/scapy/>

[15] Python Standar Library. (2014, Jan) Python Douments. [Online].

<https://docs.python.org/2/library/random.html>

[16] (2014, Jan) Open Vswitch. [Online]. <http://openvswitch.org/>

[17] Seyed Mohammad Mousavi and Marc St-Hilaire “Early Detection of DDoS Attacks against

SDN Controllers”, in 2015 International Conference on Computing, Networking and Communications, Communications and Information Security

Symposium.