



Developing a model to detect and prevent DDoS attacks in SDN environments using machine learning

Raad Abdo Mohammed Al Selwi

Department of IT, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

Department of Cyber Security, Engineering and IT Faculty, Al Saeed University, Yemen

raad.alselwi@alsaeeduni.edu.ye

Mogeeb Abdulhakim Saeed

Department of CNDS, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

mogeeb1982@taiz.edu.ye

Mohammed Ahmed M. Saif

Department of CNDS, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

mohammedahmed775979036@gmail.com

Ammar Hasan M. Almagashi

Department of CNDS, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

Ammar7760www@gmail.com

Abdullah Mofareh Saleh Saeed

Department of CNDS, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

Fuad Abdo Mahyoub Mohammed

Department of CNDS, Al Saeed faculty for Engineering and IT, Taiz University, Yemen

Received: 19/10/2024

Accepted: 13/9/2024

Journal Website:

<https://journal.alsaeeduni.edu.ye>

تطوير نموذج لاكتشاف ومنع هجمات رفض الخدمة الموزعة DDoS في بيئات الشبكات المعرفة برمجياً باستخدام التعلم الآلي

الباحث/ رعد عبده محمد علي الصلوي

قسم تقنيات المعلومات، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن
قسم الأمن السيبراني، كلية الهندسة وتقنية المعلومات، جامعة السعيد، اليمن

الباحث/ مجيب عبدالحكيم سعيد

قسم هندسة الشبكات والنظم الموزعة، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن

الباحث/ محمد احمد محمد سيف

قسم هندسة الشبكات والنظم الموزعة، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن

الباحث/ عمار حسن محمد يحيى المجعشي

قسم هندسة الشبكات والنظم الموزعة، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن

الباحث/ عبدالله مفرح صالح سعيد

قسم هندسة الشبكات والنظم الموزعة، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن

الباحث/ فؤاد عبده مهيب محمد

قسم هندسة الشبكات والنظم الموزعة، كلية السعيد للهندسة وتقنية المعلومات، جامعة تعز، اليمن

المخلص

تتميز بنية الشبكات المعرفة بالبرمجيات SDN المتطورة بالديناميكية، وسهولة الإدارة، وقابلية التحكم فيها، والقابلية للتكيف مع الشبكات الأخرى مما يجعلها مثالية للطبيعة الديناميكية ذات النطاق الترددي العالي لتطبيقات اليوم. يتم فصل وظائف التحكم في الشبكة وإعادة توجيهه في هذا التصميم، مما يسمح بالبرمجة المباشرة للتحكم في الشبكة واستخراج البنية التحتية الأساسية للتطبيقات وخدمات الشبكة إحدى أنواع الهجمات السيبرانية التي تؤثر على البنية التحتية لهذه الشبكات هو هجوم رفض الخدمة الموزع DDoS، عندما يكون الضحية هدفاً لهجوم رفض الخدمة الموزع يتم تشويش الخوادم، أو إغراقها بحركة المرور الضارة لمنع المستخدمين الشرعيين من الوصول إلى حساباتهم، أو الخدمات المشروعة عبر الإنترنت، لاكتشاف هذا الهجوم ومنعه، لذلك قمنا في هذه الورقة البحثية بتطوير نموذج لاكتشاف ومنع هجمات رفض الخدمة الموزع DDoS في بيئات الشبكات المعرفة برمجياً باستخدام التعلم الآلي. ويعمل النموذج المقترح على اكتشاف الهجوم من خلال تحديد معرف المبدل، ورقم المنفذ الخاص بالمهاجم، ويقوم بجمعها كرقم فريد يتم إضافته إلى قائمة خاصة قبل معالجة حركة مرور البيانات لأي جهاز، يتم دمج رقم المعرف ورقم المنفذ كرقم فريد، ويتم البحث في القائمة إذا تطابقت، يتم تجاهل هذه البيانات، دون أن يتم إغلاق وحدة التحكم.

الكلمات المفتاحية: الشبكات المعرفة بالبرمجيات، هجوم رفض الخدمة الموزع، التعلم الآلي، اكتشاف ومنع هجوم رفض الخدمة الموزع.

Developing a model to detect and prevent DDoS attacks in SDN environments using machine learning

Raad Abdo Mohammed Al Selwi

Department of IT, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Department of Cyber Security, Engineering and IT Faculty,
Al Saeed University, Yemen

Mogeeb Abdulhakim Saeed

Department of CNDS, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Mohammed Ahmed M. Saif

Department of CNDS, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Ammar Hasan M. Almagashi

Department of CNDS, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Abdullah Mofareh Saleh Saeed

Department of CNDS, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Fuad Abdo Mahyoub Mohammed

Department of CNDS, Al Saeed faculty for Engineering and IT,
Taiz University, Yemen

Abstract

A developing architecture called Software-Defined Networking (SDN) is dynamic, manageable, affordable, and adaptive, making it perfect for the high-bandwidth, dynamic nature of today's applications. The network control and forwarding functions are separated in this design, allowing for direct programming of the network control and the abstraction of the underlying infrastructure for applications and network services. One type of cyber-attack that affects the infrastructure of these networks is a Distributed Denial-of-Service (DDoS) attack. When a victim is the target of a DDoS attack, the servers are jammed or overwhelmed with the malicious traffic to prevent the legitimate users from accessing their accounts or legitimate online services. To detect and prevent this attack, in this paper we developed a model to detect and prevent a DDoS attack in SDN environments using machine learning. The proposed model work to detect the attack by identifying the attacker's switch ID and port number, and collect them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, without closing the controller.

Keywords: Software Defined Networking, Denial-of-Service attack, Machine learning, Detect and prevent DDoS attack.

Introduction:

A Distributed Denial-of-Service (DDoS) attacks are widely used to run out the target's network bandwidth or process resources. DDoS attacks are not only effective in traditional networks, but also available in software-defined networking (SDN) environments [1]. SDN controller is subjected to many different types of attacks. The most critical security issues for the SDN controller are DDoS attack, since if the controller is brought down, the whole network will be stopped. The main aim of a DDoS attack is to make the computing resources unavailable for the users. SDN separates a network's control logic from switches, simplifies network management. It allows network administrators to manage network services through abstraction of lower-level functionality. SDN is meant to address the fact that the static architecture of traditional network doesn't support dynamic, scalable computing and storage needs of more computing environments such as data centers. This is done by decoupling or disassociating the system that makes decisions about where traffic is sent (The Control Plane) from the underlay systems that forward traffic to the selected destination (The Data Plane). These specific capabilities make SDN deployable in many network environments, from home and enterprise network to data centers in cloud networks.

To detect DDoS attack in SDN they were used many of methods and algorithms as Sample entropy and Principal Component Analysis (PCA), but all these methods when detect the attack, led to stop the control unit in network.

Sample entropy, a general method for DDoS detection in SDN, is conducted by collecting the flow statistics or traffic features from the switches, and calculating the entropy measure randomness in the packets that are coming to a network. The higher the randomness, the higher is the entropy and vice versa. By setting a threshold, if the entropy passes it or below it, depending on the scheme, an attack is detected [2].

Principal Component Analysis (PCA) is a coordinate transformation method that maps the measured data onto a new set of axes. These axes are called the principal axes or components, where each principal component has the property that it points in the direction of maximum variation or energy

remaining in the data, given the energy already accounted for, in the preceding components [1].

In this paper, to detect and prevent this attack, we developed a model to detect and prevent a DDoS attack in SDN environments using machine learning. The proposed model work to detect the attack by identifying the attacker's switch ID and port number, and collect them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, without closing the controller.

Problem Statement:

DDoS attacks are widely used to run out the target's network bandwidth or process resources. DDoS attacks are effective and available in SDN environments. The most critical security issues for the SDN controller are DDoS attack, since if the controller is brought down, the whole network will be stopped. When using sample entropy and PCA algorithms to detect DDOS attacks in SDN, these algorithms run in case of attack detection to shut down the microcontroller, which causes the network to go down.

Research Aim:

This study aims to develop and evaluate a model to detect and prevent distributed denial of service attacks in software-defined networks using machine learning, while keeping the network controller running non-stop.

Research objectives:

The research aims to achieve the following objectives:

- 1- Configure and setup of the virtual work environment for the network using VMware and Mininet network emulator.
- 2- Creating SDN network topology.
- 3- Launch of DDoS attacks in the created topology.
- 4- Use POX controller to detection and prevention of DDoS attacks using Sample entropy and PCA algorithms.
- 5- Develop a model to detect and prevent distributed denial of service attacks in software-defined networks using machine learning.
- 6- Evaluating the performance of the proposed model to detect and prevent attacks with Sample entropy and PCA algorithms.

Research Importance:

The importance of the research lies in evaluating the performance of the algorithms of a Sample entropy and PCA in terms of detecting and preventing DDoS attacks in a SDN environment, and develop the model to detect and prevent which leads to improving network performance and not stopping it due to attacks.

Research questions:

According to the problem statement mentioned above, the following questions are generated:

1. What method is used to generate data traffic and attack?
2. How was the attack detecting?
3. How was the attack prevention?
4. Is the proposed algorithm more efficient than sample entropy and PCA algorithms in detecting and preventing DDoS attacks in an SDN environment?
5. Will the proposed algorithm be able to detect and prevent DDoS attacks in an SDN environment without stopping the network?

Related Work:

A DDoS attack aims to disable network services by using up all available resources. An SDN is more vulnerable to the effects of this DDoS attack than a conventional one. This is because each time a new Origin-Destination pair is created, the controller will be called. Therefore, if an attacker floods a network with too many packets in a short period of time, the controller will be damaged, which will cause the network to crash.

Many previous studies dealt with the detection and mitigation of DDoS attacks on SDN using different types of AI techniques and machine learning algorithms, including:

Di Wu, Jie Li, Sajal K. Das, Jin Song Wu, and Yusheng Ji. [1] they introduce a novel DDoS scheme using principal component analysis, to detect DDoS attack on SDN environment. Then, they put it into test, made a comparison with Sample entropy, a popular used scheme. They show that this scheme has clearer result than another. Meanwhile, they identify a novel DDoS attack aiming on SDN environment, which could cause more damage on SDN, and used the two-detection method on this novel DDoS attack, and

found this novel attack is hard to be detected by sample entropy, and still be captured by PCA.

Aswanth, Mohammed Ameen and Joe Antony [3] To successfully detect DDoS attacks, they have put two strategies into practice. The findings indicate that PCA is a superior strategy than sample entropy because: In sample entropy, any packet with an entropy value below the threshold is recorded in a dictionary, and if an entry appears more than five times, it is assumed to be a DDoS attack. This knob value of 5 is topologically dependent. Therefore, a smaller topology may potentially exhibit DDoS attacks for regular traffic.

But in PCA analysis, we are taking characteristics of each packet to update the principal component axis. Each packet destination value is compared with current principal component axis. During an attack, the principal component axis is gradually shifted towards the destination value. Hence, there will be successive decrease in delta Y values, which is an indication of DDoS attack. However, both algorithms used in this study detect the attack and, as a handler, close the microcontroller, causing the network to stop.

Qin et al. in their study [4]. They were suggested approach with a window of 0.1 seconds and three threshold levels. This approach focuses on preventing false positive and false negative network results. The strategy, however, takes more time and resources, as the authors themselves note.

Ra et al.'s [5] proposal for a quicker method of computing entropy bases it on both the kind and quantity of packets in the network. This approach likewise employs a window of time. The threshold was determined by the authors using numerous datasets and is a multiple of the entropy values' standard deviation. Compared to other approaches, this method has more false negatives and less false positives. There is no mention of accuracy percentage. Additionally, there is no mention of the tools utilized for quick computation.

The work presented by the researchers in ways to analyses the incoming traffic in SDN has shown that every network flow has certain parameters and features which can be monitored and collected to extract the exact features required to detect malicious DDOS traffic in a network. Our presented method uses both statistical and machine learning methods to detect and mitigate DDOS attacks in an SDN. The proposed model work to detect the

attack by identifying the attacker's switch ID and port number, and collect them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, without closing the controller.

Background:

SDN offers a fresh viewpoint on networking. The main objective is to get more control over network resources. Vendors translate control and forwarding functions, which frequently include proprietary software, into functional networking hardware today. Figure (1) shows the SDN architecture between the control plane and forwarding allows network managers to assume control of the control plane [6]. This separation is made possible by altering the network so that the switch gets commands to forward rather than spending its resources to handle incoming messages. Databases with streams for sending commands will be present in the adaptor.

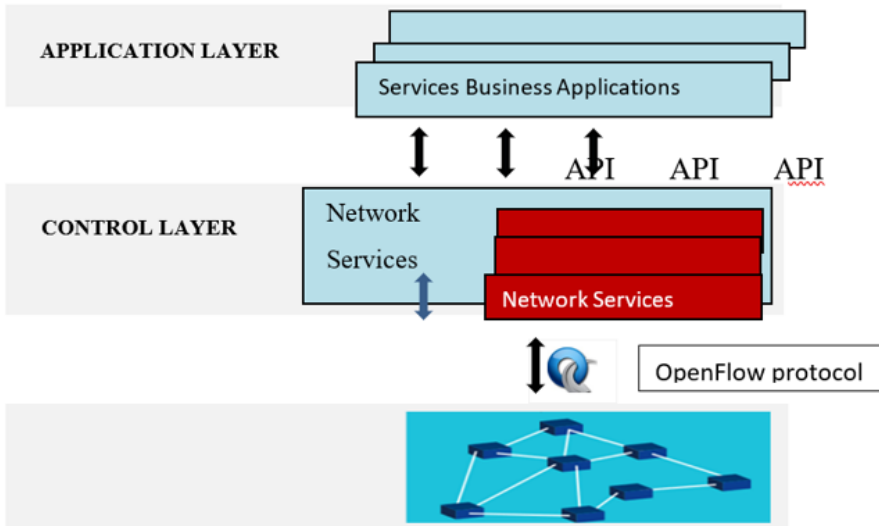


Figure (1). SDN architecture

DDoS attacks in SDNs:

Some of the most common yet dreadful attacks on SDNs are DoS and DDoS attacks. Such attacks affect performance and network behavior. By exhausting resources, they disable or downgrade network services, and

legitimate hosts are not able to communicate with the SDN controller or to send packets on the network [7].

DoS or DDoS attacks are achieved in SDNs by creating several new flows that flood the bandwidth of the control plane, the OpenFlow switches, and the SDN controller, which results in network failure for legitimate hosts. More specifically, attackers generate several new flows that have spoofed IP addresses but are sent from a single source (DoS) or multiple sources (DDoS). These spoofed addresses do not match any existing flow rules in the flow table of the OpenFlow switch, resulting in a table-miss situation. Such a situation results in generating massive packet-in messages sent to the SDN controller from the victim OpenFlow switch, which consumes communication bandwidth, memory, and CPU in both the control and the data plane of SDN [8]. Moreover, since the OpenFlow switch buffers packet-in messages before sending them to the controller, if several new flows are received within a very short time, the buffer fills up. This results in sending the entire packets of the new flow to the controller instead of sending the new flow's headers-only packet-in messages, thus resulting in higher consumption of the control plane's bandwidth and potentially delaying installation of new flow rules received from the SDN controller. Another situation in which massive new flows could result is filling up the forwarding table of the OpenFlow switch. As mentioned earlier, such a table includes different flow rules that direct the switch about forwarding the packets, and it is updated and managed by the controller [9]. Having several new flows results in installing new rules to the forwarding table of the switch. At some point, the forwarding table fills up, and therefore, upon receiving a new flow rule from the controller, it is unable to install it—hence dropping the packet and sending an error message to the controller [10]. Moreover, the switch would not be able to forward packets until there is free memory in its forwarding table, resulting in delays and dropping of incoming packets [11].

On the controller side, a high arrival rate of packet-in messages exceeding the controller processing capability results in overwhelming the controller and making it unreachable to legitimate traffic. This could result in the failure of the entire network, as the controller implements the logic of the SDN and manages the applications and the OpenFlow switches [11]. Figure (2), shows a schematic view of a DoS/DDoS attack in SDNs.

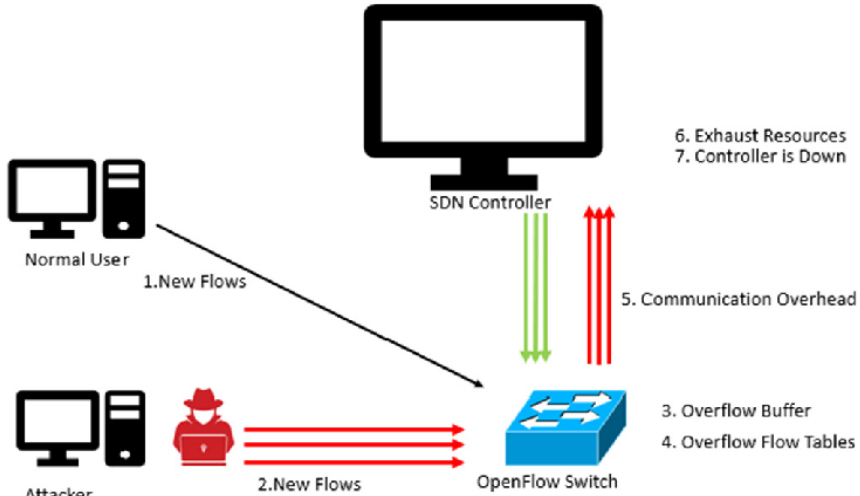


Figure (2). DDoS attacks in SDNs.

Anomaly Detection for DDoS Mitigation:

The irregular traffic provided to the victim is a recurring theme in several DDoS attack types. There is a pattern to the network activity under typical conditions, and a standard rate of bandwidth use. This is frequently seen as abnormal if there is an abrupt rise in traffic, delay, CPU consumption, or abrupt decrease in performance of any of the network assets. Such network anomalies will be sought for by any DDoS detector. Anomalies typically have to do with the type of data being transmitted over the network [12]. The assault may target the application layer, causing CPU resource exhaustion, or the network layer, creating a bottleneck. The first step in finding anomalies in a network is to understand the type of data and its characteristics. These traits can include protocol, packet size, delay, header information, and packet header information. For instance, attacks are most likely TCP SYN request flooding in a server that answers to TCP SYN queries. This is also where anomaly detection is most likely to take place. In actuality, the nature of the network determines the sort of intrusion. The detection and mitigation of the danger must be matched to the sort of threat that a network is vulnerable to.

Types of Anomaly Detection Techniques:

There is a set amount of processing power and bandwidth available in IP networks for transmitting traffic. When some network properties are statistically analyzed, a pattern will emerge for each property [13]. The pattern is more trustworthy the longer the period of time. This is only

accurate, though, if the network experiences constant traffic. In the long run, statistics will stabilize and cannot be fully trusted if there are deviations that are accepted as typical traffic.

Different approaches are taken to data collecting, filtering, and processing for anomaly detection. Two of the often-used techniques for detecting anomalies are statistical analyses and machine learning.

For spotting changes in network traffic, statistical analysis like the entropy and chi-square approaches have been proposed [14].

Another technique for protecting networks from infiltration is machine learning and cognitive detection. An algorithm is trained to continuously update its filtering criteria based on the network events, as opposed to setting up a preset filter.

Neural networks are one such system [15]. Multiple nodes operate in parallel to process data in neural networks. Like the human brain, they function. When neurons or nodes receive extensive training or information, a pattern for processing related input emerges from their collective expertise. Input, output, and hidden layers in the center, which process the input data, make up the three primary layers of neural networks. The nodes are learning more as time goes on and more data is processed, and a more distinct pattern starts to show.

The brains behind these networks' decision-making are algorithms. The Multilayer Perceptron (MLP), Gaussian Classifier (GAU), K-means Clustering (K-M), and Markov model are some popular techniques in network intrusion and anomaly detection [16] [17] [18]. Sample entropy and PCA are the techniques utilized in this research to construct a detection method in the Open flow controller, then we will propose model to detect and prevent DDoS attack in SDN and we evaluate the performance these techniques with proposed model.

Sample entropy for DDoS Detection:

Before implementing a technique in an SDN network, it is crucial to examine how it is utilized in non-SDN networks. We must rely on research conducted in fields other than SDN because there hasn't been any study of this strategy in SDN.

Window size and a threshold are two crucial elements in DDoS detection utilizing entropy [11]. Either time or the quantity of packets is used to

determine the window size. Within this window, entropy is calculated to assess the degree of uncertainty in the incoming packets. An assault detection threshold is required. Depending on the technique, an attack is recognized if the estimated entropy exceeds or falls below a certain threshold.

Entropy has not, as far as we are aware, been used in SDN; rather, it has been used in a number of ways to detect DDoS attacks in the network.

Any detection approach in SDN when forwarding packets to the controller must include the restriction of available resources and the quick identification of attacks.

PCA:

PCA, is a technique for reducing the number of dimensions in large data sets by condensing a large collection of variables into a smaller set that retains the majority of the large set's information.

Accuracy naturally suffers as a data set's variables are reduced, but the answer to dimensionality reduction is to trade a little accuracy for simplicity. since machine learning algorithms can analyze data points considerably more quickly and easily with smaller data sets since there are less extraneous variables to process.

In conclusion, PCA's basic principle is to minimize the number of variables in a data set while maintaining as much accuracy as possible.

Research Methodology:

The proposed methodology will discuss the configuring and setup of the virtual work environment for the network using VMware and Mininet network emulator, creating SDN network topology and the tools and methods that were used for the implementation process, launch of DDoS attacks in the created topology.

This section also will discuss the use POX controller to detection and prevention of DDoS attacks using Sample entropy and PCA algorithms.

This section also will discuss the developing a model to detect and prevent DDoS attacks in SDN using machine learning.

Finally we will be evaluating the performance of the proposed model to detect and prevent attacks with Sample entropy and PCA algorithms. Figure (3), shows the research methodology.

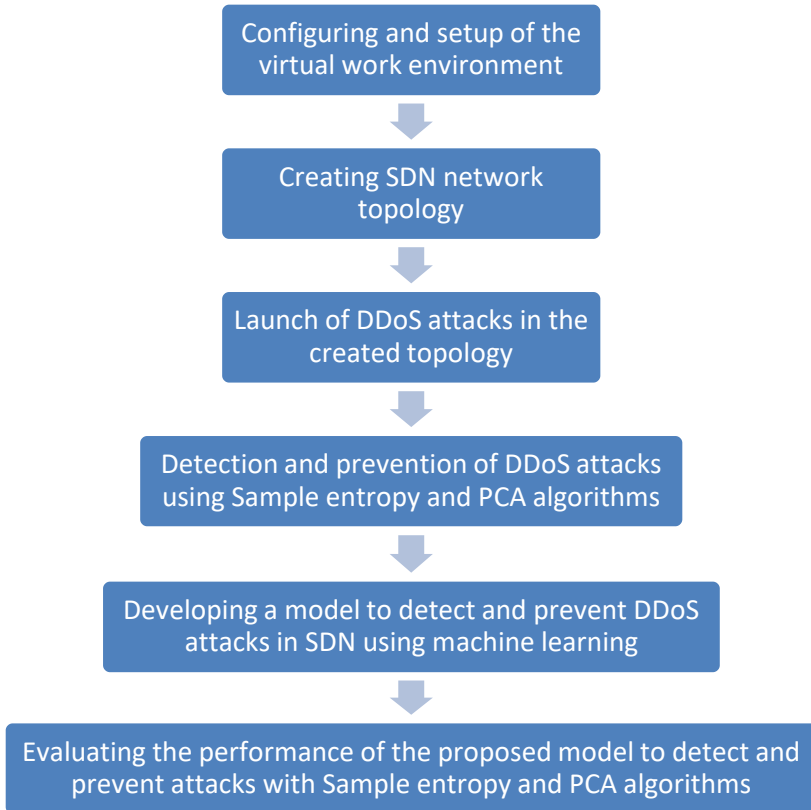


Figure (3). Research methodology

Configuring and setup of the virtual work environment:

Configuring and setup of the virtual work environment for the SDN network using VMware and Mininet network emulator.

POX Controller:

The Open Flow protocol can be used to communicate with SDN switches utilizing the POX architecture. Using Python programming, developers can use POX to create an SDN controller [19]. It is a well-liked teaching and research tool for network programming and SDN. By utilizing the stock components that are provided with it, POX can be deployed right away as a simple SDN controller. When POX is launched from the command line, POX components are extra Python applications that can be used. The SDN network functionality is implemented by these components.

Mininet:

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support Open Flow for highly flexible custom routing and SDN. Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC.

Creating SDN network topology:

In this step, we use Mininet to create a tree topology with a depth of 2 and a fanout of 10 switches with 81 hosts as shown in figure (4).

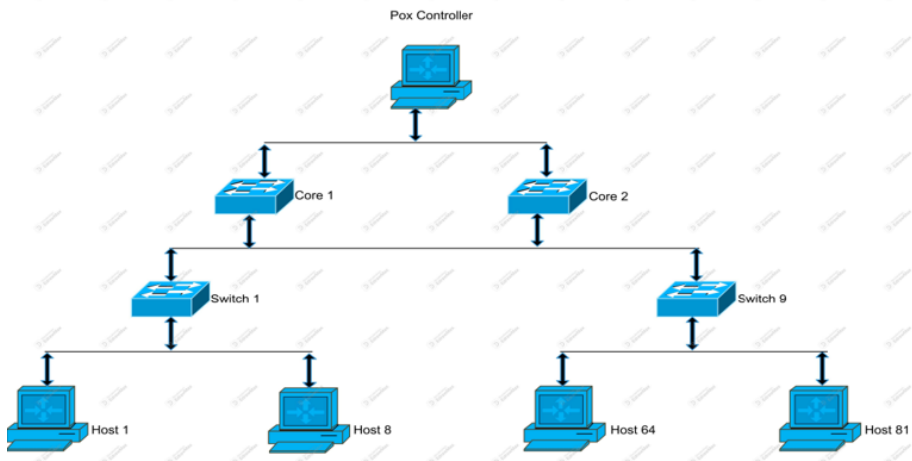


Figure (4). SDN network topology

We also specify the POX controller as the controller for the network and set its IP address to 127.0.0.1 and port to 6633.

Launch of DDoS attacks:

To launch of DDoS attacks in the created topology we used Scapy. Scapy is used for generation of packets, sniffing, scanning, forging of packet and attacking. It also used for generation of UDP packets and spoofing the source IP address of the packets.

Detection and prevention DDoS attacks using the Sample entropy and PCA algorithms:

The presented method uses Sample Entropy both or PCA methods to detect and mitigate DDOS attacks in a software different statistical network. The

implemented method requires to use one of these steps for detecting the attack in a network.

Detection Using Sample Entropy:

- 1. Run the POX controller:** In this step, you navigate to the directory where the POX controller is installed (`cd pox`) and run the controller with the `13_detectionEntropy.py` script. This script is responsible for calculating the sample entropy of the network traffic and detecting anomalies.
- 2. Create a Mininet topology:** In this step, you use Mininet to create a tree topology with a depth of 2 and a fanout of 10 switches. You also specify the POX controller as the controller for the network and set its IP address to 127.0.0.1 and port to 6633.
- 3. Open xterm for hosts:** xterm is a terminal emulator that allows you to run commands on remote hosts. In this step, you open four xterm windows for the hosts h1, h2, h3, and h81. These xterm windows will be used to run commands on the hosts.
- 4. Launch traffic:** In this step, you run the `traffic.py` script on host h1 to generate network traffic between hosts 2 to 82. The `-s` and `-e` arguments specify the start and end hosts for the traffic flow. This traffic will be used to generate a baseline entropy value for normal traffic.
- 5. Generate entropy values:** The POX controller generates a list of entropy values for the traffic generated in step 4. The entropy value is a measure of the irregularity or unpredictability of the network traffic. The least value obtained from this list is considered the threshold entropy for normal traffic. This threshold value will be used to detect anomalies in the network traffic.
- 6. Launch attack:** In this step, you simulate a network attack on host h81 by running the `attack.py` script on hosts h2 and h3. This script generates a large volume of traffic directed at host h81, simulating a DDoS attack. You observe the entropy values calculated by the POX controller during the attack. If the entropy value decreases below the threshold value obtained in step 5, it indicates that an attack is happening, and the algorithm detects it.

In summary, using sample entropy to detect network attacks involves generating a baseline entropy value for normal traffic and then monitoring the entropy value for any deviations from the baseline. If the entropy value falls

below the threshold, it indicates that an attack is happening, and the algorithm detects it.

Detection Using PCA:

- 1. Run the POX controller:** In this step, you navigate to the directory where the POX controller is installed (`cd pox`) and run the controller with the `l3_detectionPCA.py` script. This script implements the detection algorithm using PCA.
- 2. Create a Mininet topology:** In this step, you use Mininet to create a tree topology with a depth of 2 and a fanout of 10 switches. You also specify the POX controller as the controller for the network and set its IP address to 127.0.0.1 and port to 6633.
- 3. Open xterm for hosts:** In this step, you open four xterm windows for the hosts h1, h2, h3, and h81. These xterm windows will be used to run commands on the hosts.
- 4. Launch traffic:** In this step, you run the `traffic.py` script on host h1 to generate network traffic between hosts 2 to 82. This traffic will be used to generate the PCA axis and the threshold for normal traffic.
- 5. Generate deltaY values:** The POX controller generates a list of deltaY values for the traffic generated in step 4. The deltaY value is the difference between the y-coordinates of a packet and the point obtained by drawing a perpendicular from this packet to the principal component axis. The principal component axis is obtained using PCA, which is a statistical technique used to reduce the dimensionality of data while retaining as much information as possible.
- 6. Launch attack:** In this step, you simulate a network attack on host h64 by running the `attack.py` script on hosts h1. This script generates a large volume of traffic directed at host h81, simulating a DDoS attack. You observe the deltaY values calculated by the POX controller during the attack. If the deltaY values converge to the interval $(-1, 1)$, it indicates that an attack is happening, and the algorithm detects it.

In summary, using PCA to detect network attacks involves generating the PCA axis and the threshold for normal traffic using network traffic and then monitoring the deltaY values for any deviations from the normal range. If the deltaY values converge to the interval $(-1, 1)$, it indicates that an attack is happening, and the algorithm detects it.

Developing a model to detect and prevent DDoS attacks in SDN using machine learning:

The proposed model for detecting and preventing a DDoS attack, which was developed in our research, boils down to identifying the attacker's switch ID and port number, and collecting them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, without closing the console. Figure (5), shows the proposed model for detecting and preventing a DDoS attack in SDN environments.

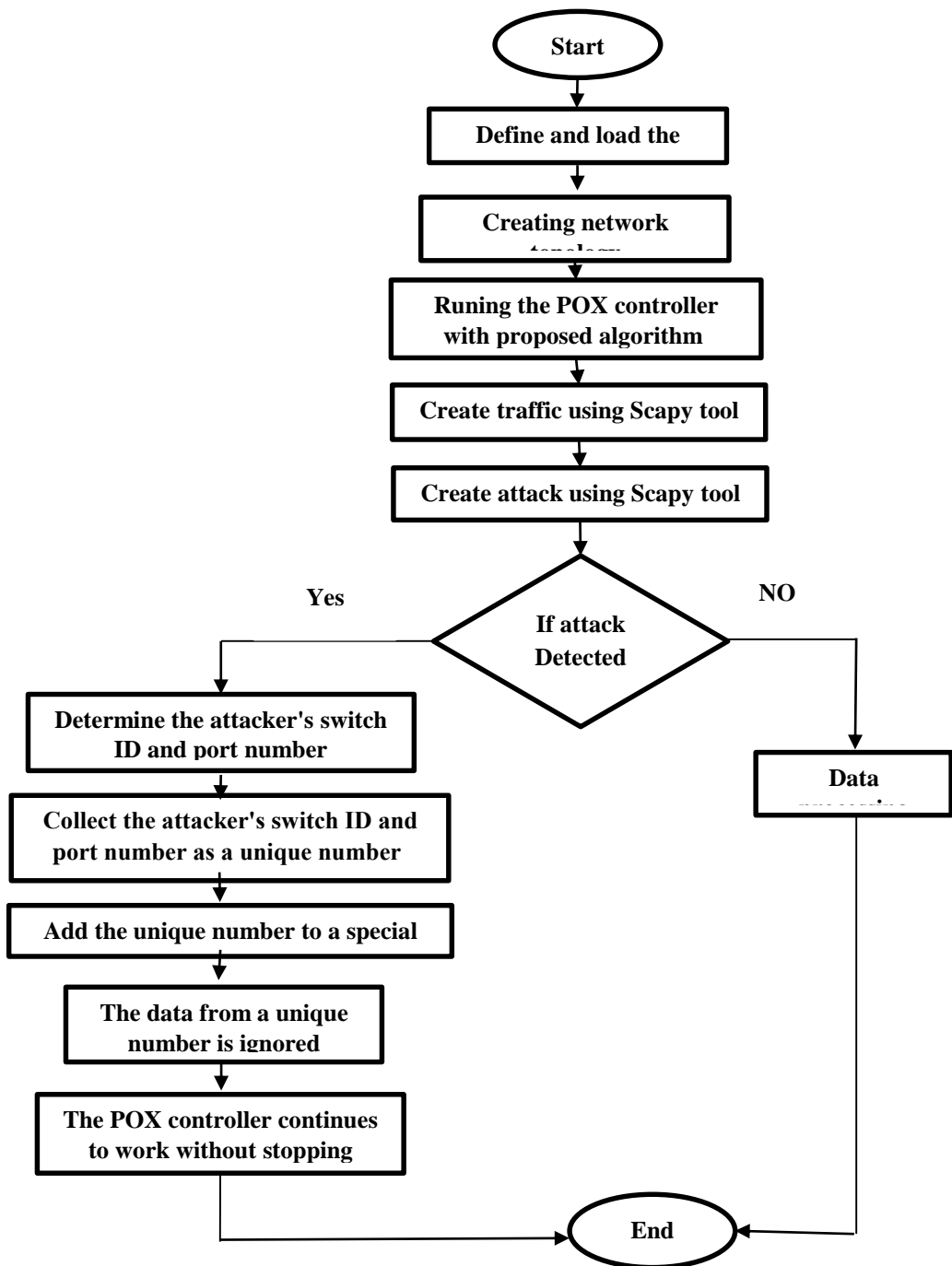
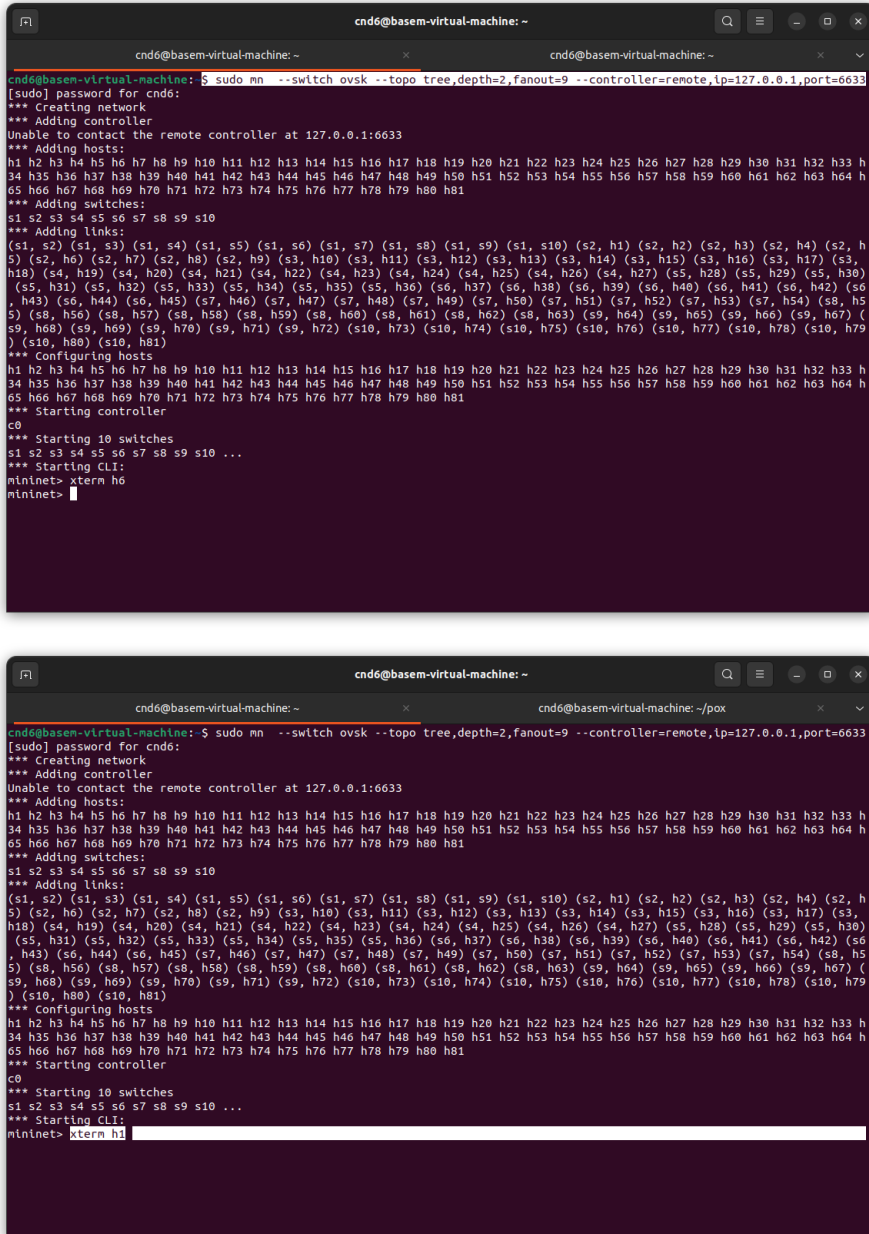


Figure (5). The proposed model for detecting and preventing a DDoS attack in SDN environments.

Results and Discussion:

Results:

Figure (6) shows Creating a Mininet topology.



```

cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=9 --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for cnd6:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h
34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h
65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s1, s10) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h
5) (s2, h6) (s2, h7) (s2, h8) (s2, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s3, h17) (s3,
h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s4, h25) (s4, h26) (s4, h27) (s5, h28) (s5, h29) (s5, h30)
(s5, h31) (s5, h32) (s5, h33) (s5, h34) (s5, h35) (s5, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s6, h41) (s6, h42) (s6
, h43) (s6, h44) (s6, h45) (s7, h46) (s7, h47) (s7, h48) (s7, h49) (s7, h50) (s7, h51) (s7, h52) (s7, h53) (s7, h54) (s8, h5
5) (s8, h56) (s8, h57) (s8, h58) (s8, h59) (s8, h60) (s8, h61) (s8, h62) (s8, h63) (s9, h64) (s9, h65) (s9, h66) (s9, h67) (
s9, h68) (s9, h69) (s9, h70) (s9, h71) (s9, h72) (s10, h73) (s10, h74) (s10, h75) (s10, h76) (s10, h77) (s10, h78) (s10, h79
) (s10, h80) (s10, h81)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h
34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h
65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
*** Starting CLI:
mininet> xterm h0
mininet>
cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~/px
cnd6@basem-virtual-machine: ~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=9 --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for cnd6:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h
34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h
65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s1, s10) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h
5) (s2, h6) (s2, h7) (s2, h8) (s2, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s3, h17) (s3,
h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s4, h25) (s4, h26) (s4, h27) (s5, h28) (s5, h29) (s5, h30)
(s5, h31) (s5, h32) (s5, h33) (s5, h34) (s5, h35) (s5, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s6, h41) (s6, h42) (s6
, h43) (s6, h44) (s6, h45) (s7, h46) (s7, h47) (s7, h48) (s7, h49) (s7, h50) (s7, h51) (s7, h52) (s7, h53) (s7, h54) (s8, h5
5) (s8, h56) (s8, h57) (s8, h58) (s8, h59) (s8, h60) (s8, h61) (s8, h62) (s8, h63) (s9, h64) (s9, h65) (s9, h66) (s9, h67) (
s9, h68) (s9, h69) (s9, h70) (s9, h71) (s9, h72) (s10, h73) (s10, h74) (s10, h75) (s10, h76) (s10, h77) (s10, h78) (s10, h79
) (s10, h80) (s10, h81)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h
34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h
65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
*** Starting CLI:
mininet> kterm h1

```

Figure (6). Creating a Mininet topology

The figure (7) shows running the POX controller forwarding to Sample Entropy is the program file that will be executed.

```

cnd6@basem-virtual-machine: ~/pox
cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~$ python3 ./pox.py forwarding_l3 detectionEntropy
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) ls up.
INFO:openflow.of_01:[00-00-00-00-00-06 1] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-0a 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-08 5] connected
INFO:openflow.of_01:[00-00-00-00-00-02 6] connected
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:openflow.of_01:[00-00-00-00-00-09 8] connected
INFO:openflow.of_01:[00-00-00-00-00-04 9] connected
INFO:openflow.of_01:[00-00-00-00-00-05 10] connected

```

Figure (7). Running the POX controller forwarding to Sample Entropy

The figure (8) shows running the POX controller forwarding to PCA is the program file that will be executed.

```

cnd6@basem-virtual-machine: ~/pox
cnd6@basem-virtual-machine: ~
cnd6@basem-virtual-machine: ~$ python3 ./pox.py forwarding_l3 detectionPCA
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) ls up.
INFO:openflow.of_01:[00-00-00-00-00-06 1] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-0a 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-08 5] connected
INFO:openflow.of_01:[00-00-00-00-00-02 6] connected
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:openflow.of_01:[00-00-00-00-00-09 8] connected
INFO:openflow.of_01:[00-00-00-00-00-04 9] connected
INFO:openflow.of_01:[00-00-00-00-00-05 10] connected

```

Figure (8). Running the POX controller forwarding to PCA

After executing these commands, the POX program will run using Python, and PCA and Sample Entropy will be analyzed to detect DDoS attacks.

Open xterm for hosts:

The figure (9) shows the opening a host h1.

```

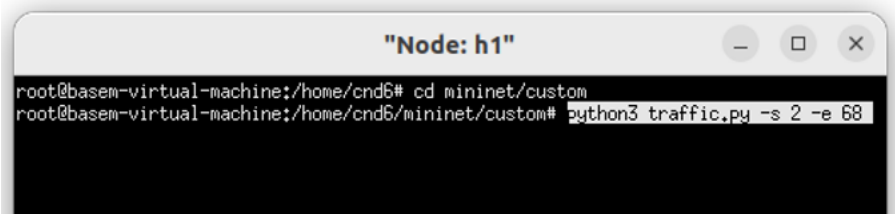
"Node: h1"
root@basem-virtual-machine:~/home/cnd6# cd mininet/custom

```

Figure (9). Open xterm for hosts h1.

Running for generate normal traffic:

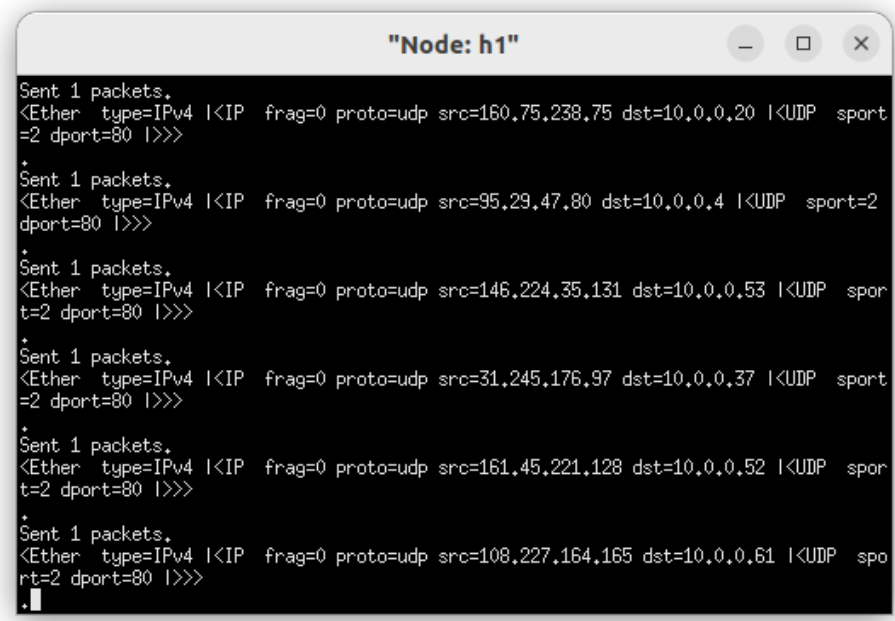
In the xterm window of host h1 running the following command for generate normal traffic as shown in the figure (10).



```
"Node: h1"
root@base-virtual-machine:/home/cnd6# cd mininet/custom
root@base-virtual-machine:/home/cnd6/mininet/custom# python3 traffic.py -s 2 -e 68
```

Figure (10). Running for generate normal traffic

The figure (11) shows normal traffic in host h1.

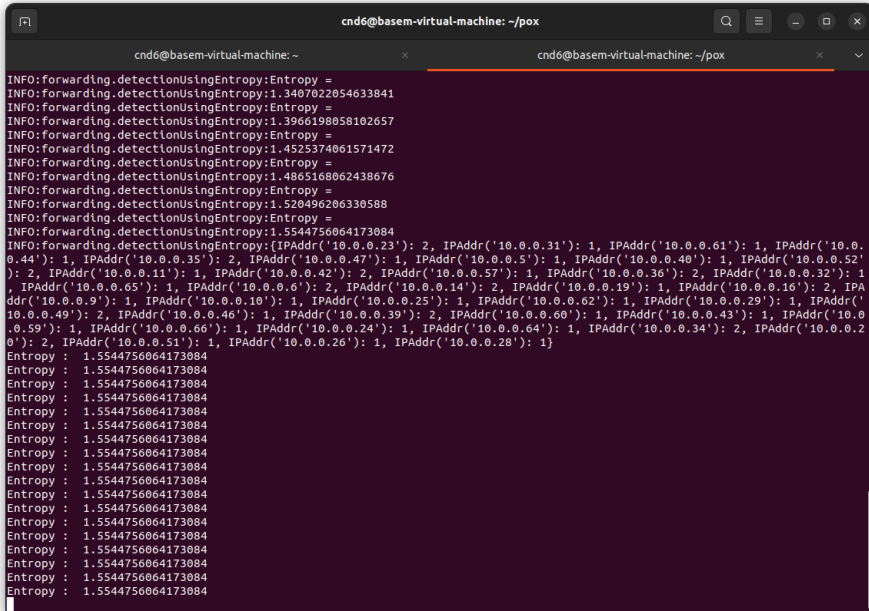


```
"Node: h1"
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=160,75,238,75 dst=10,0,0,20 |<UDP sport=2 dport=80 |>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=95,29,47,80 dst=10,0,0,4 |<UDP sport=2 dport=80 |>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=146,224,35,131 dst=10,0,0,53 |<UDP sport=2 dport=80 |>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=31,245,176,97 dst=10,0,0,37 |<UDP sport=2 dport=80 |>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=161,45,221,128 dst=10,0,0,52 |<UDP sport=2 dport=80 |>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=108,227,164,165 dst=10,0,0,61 |<UDP sport=2 dport=80 |>>
.
```

Figure (11). Normal traffic in host h1.

Normal Traffic in Sample Entropy:

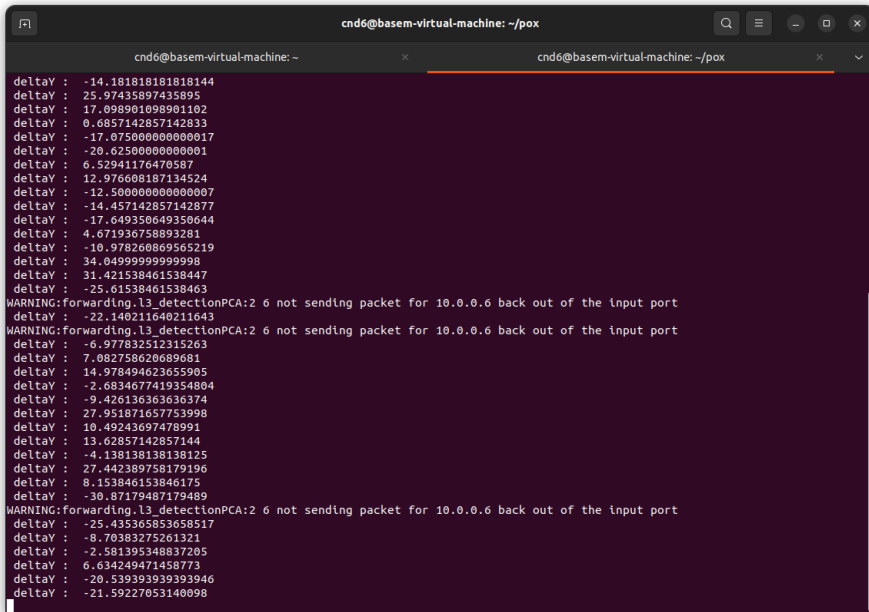
Normal traffic is implemented by host 1 using the Entropy algorithm as shown in figure (12).



```
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.3407022054633841
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.3966198058102657
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.4525374061571472
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.4865169062438676
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.520496206330588
INFO: forwarding.detectionUsingEntropy:Entropy =
INFO: forwarding.detectionUsingEntropy:1.5544756064173084
INFO: forwarding.detectionUsingEntropy:{IPAddr('10.0.0.23'): 2, IPAddr('10.0.0.31'): 1, IPAddr('10.0.0.61'): 1, IPAddr('10.0.0.44'): 1, IPAddr('10.0.0.35'): 2, IPAddr('10.0.0.47'): 1, IPAddr('10.0.0.5'): 1, IPAddr('10.0.0.40'): 1, IPAddr('10.0.0.52'): 2, IPAddr('10.0.0.11'): 1, IPAddr('10.0.0.42'): 2, IPAddr('10.0.0.57'): 1, IPAddr('10.0.0.36'): 2, IPAddr('10.0.0.32'): 1, IPAddr('10.0.0.05'): 1, IPAddr('10.0.0.6'): 2, IPAddr('10.0.0.14'): 2, IPAddr('10.0.0.19'): 1, IPAddr('10.0.0.16'): 2, IPAddr('10.0.0.9'): 1, IPAddr('10.0.0.10'): 1, IPAddr('10.0.0.25'): 1, IPAddr('10.0.0.22'): 1, IPAddr('10.0.0.29'): 1, IPAddr('10.0.0.49'): 2, IPAddr('10.0.0.46'): 1, IPAddr('10.0.0.39'): 2, IPAddr('10.0.0.60'): 1, IPAddr('10.0.0.43'): 1, IPAddr('10.0.0.59'): 1, IPAddr('10.0.0.66'): 1, IPAddr('10.0.0.24'): 1, IPAddr('10.0.0.64'): 1, IPAddr('10.0.0.34'): 2, IPAddr('10.0.0.20'): 2, IPAddr('10.0.0.51'): 1, IPAddr('10.0.0.26'): 1, IPAddr('10.0.0.28'): 1}
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
Entropy : 1.5544756064173084
```

Figure (12). Normal Traffic in Sample Entropy.

Normal traffic is implemented by host 1 using the PCA algorithm as shown in figure (13).

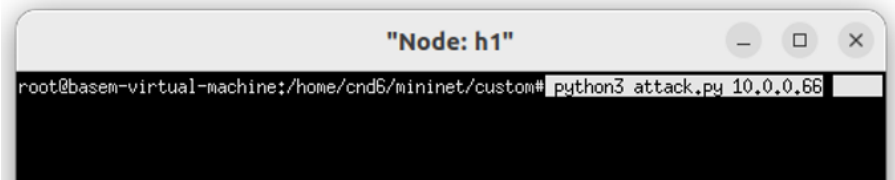


```
deltav : -14.181018101810144
deltav : 25.97435897435895
deltav : 17.098901098901102
deltav : 0.6857142857142833
deltav : -17.075000000000017
deltav : -20.62500000000001
deltav : 6.52941176470587
deltav : 12.976608187134524
deltav : -12.500000000000007
deltav : -14.457142857142877
deltav : -17.649350649350644
deltav : 4.671936758893281
deltav : -10.978260869565219
deltav : 34.049999999999998
deltav : 31.421538461538447
deltav : -25.61538461538463
WARNING: forwarding.l3_detectionPCA:2 6 not sending packet for 10.0.0.6 back out of the input port
deltav : 22.140211640211643
WARNING: forwarding.l3_detectionPCA:2 6 not sending packet for 10.0.0.6 back out of the input port
deltav : -6.977832512315263
deltav : 7.082758020689681
deltav : 14.978494623655905
deltav : -2.6834677419354804
deltav : -9.426136363636374
deltav : 27.351871657733998
deltav : 10.49243697478991
deltav : 13.62857142857144
deltav : -4.138138138138125
deltav : 27.442389758179196
deltav : 8.153846153846175
deltav : -30.87179487179489
WARNING: forwarding.l3_detectionPCA:2 6 not sending packet for 10.0.0.6 back out of the input port
deltav : -25.435365853658517
deltav : -8.7038272561321
deltav : 2.581395348837285
deltav : 6.634249471458773
deltav : -20.539393939393946
deltav : -21.59227053140098
```

Figure (13). Normal Traffic in PCA.

Launch traffic:

Run the attack traffic from h1 xterm windows to attack on 66 as shown in figure (14) and (15).

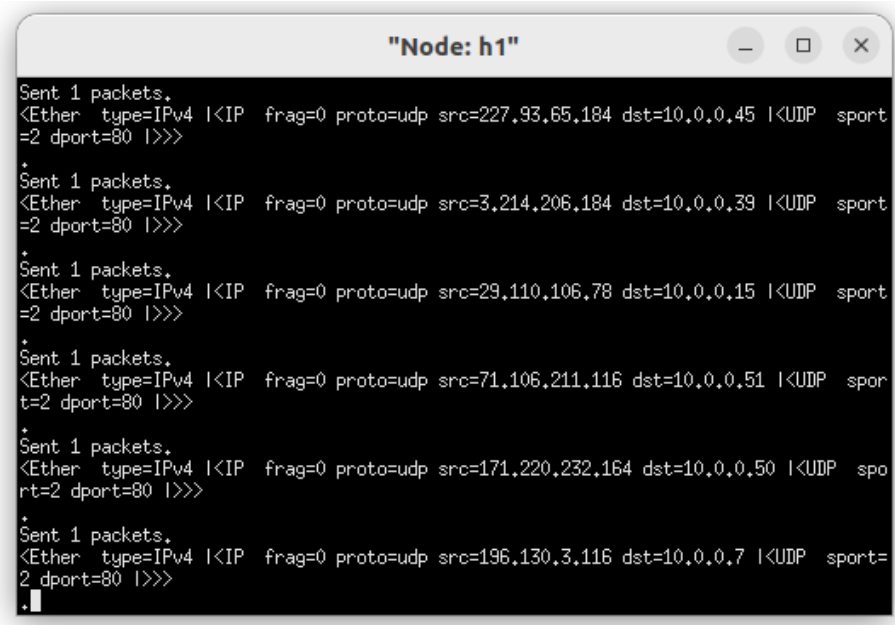


```

"Node: h1"
root@basem-virtual-machine:/home/cnd6/mininet/custom# python3 attack.py 10.0.0.66

```

Figure (14). Run the attack traffic



```

"Node: h1"
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=227.93.65.184 dst=10.0.0.45 |<UDP sport
=2 dport=80 |>>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=3.214.206.184 dst=10.0.0.39 |<UDP sport
=2 dport=80 |>>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=29.110.106.78 dst=10.0.0.15 |<UDP sport
=2 dport=80 |>>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=71.106.211.116 dst=10.0.0.51 |<UDP spo
rt=2 dport=80 |>>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=171.220.232.164 dst=10.0.0.50 |<UDP spo
rt=2 dport=80 |>>>
*
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=udp src=196.130.3.116 dst=10.0.0.7 |<UDP sport=
2 dport=80 |>>>
.

```

Figure (15). Attack traffic to specified address.

Detection and Prevention of DDOS Attack using Entropy:

The figure (16) shows the POX Controller prediction for DDOS UDP traffic, the Sample Entropy detection the attack and close the controller.

```

cnd6@basem-virtual-machine: ~/pox
Empty diction 2 1
Entropy : 0.0
dpid port and its packet count: 2 {1: 2} 2
Entropy : 0.0
dpid port and its packet count: 2 {1: 3} 3
Entropy : 0.0
dpid port and its packet count: 2 {1: 4} 4
Entropy : 0.0
dpid port and its packet count: 2 {1: 5} 5
Entropy : 0.0
dpid port and its packet count: 2 {1: 6} 6
Entropy : 0.0
dpid port and its packet count: 2 {1: 7} 7
Entropy : 0.0
dpid port and its packet count: 2 {1: 8} 8
Entropy : 0.0
dpid port and its packet count: 2 {1: 9} 9
Entropy : 0.0
dpid port and its packet count: 2 {1: 10} 10
Entropy : 0.0
dpid port and its packet count: 2 {1: 11} 11
Entropy : 0.0
dpid port and its packet count: 2 {1: 12} 12
Entropy : 0.0
dpid port and its packet count: 2 {1: 13} 13
Entropy : 0.0
dpid port and its packet count: 2 {1: 14} 14

DDOS DETECTED

{2: {1: 14}}

2024-06-02 22:22:34.738952 : BLOCKED PORT NUMBER : 2 OF SWITCH ID: 1

cnd6@basem-virtual-machine: ~/pox$

```

Figure (16). The Sample Entropy Detection the attack and close the controller

Detection and Prevention of DDOS Attack using PCA:

The figure (17) shows the state of blocked DDOS attack host, PCA detection the attack and close the controller.

```

cnd6@basem-virtual-machine: ~/pox
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
INFO:openflow.of_01:[00-00-00-00-00-0a 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-08 5] connected
INFO:openflow.of_01:[00-00-00-00-00-02 6] connected
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:openflow.of_01:[00-00-00-00-00-09 8] connected
INFO:openflow.of_01:[00-00-00-00-00-04 9] connected
INFO:openflow.of_01:[00-00-00-00-00-05 10] connected
deltaW : 2.842170943040401e-14
deltaY : 4.263256414560601e-14
deltaV : 0.0
deltaW : 0.0
deltaY : -4.263256414560601e-14
deltaV : 0.0
deltaW : 4.263256414560601e-14
deltaY : -2.842170943040401e-14
deltaV : 1.4210854715202004e-14
deltaW : 5.68434188608802e-14
deltaY : -5.68434188608802e-14
deltaV : -1.4210854715202004e-14
deltaW : -2.842170943040401e-14
deltaY : 0.0
deltaV : 0.0

DDOS DETECTED

2024-06-02 22:50:08.712129 : BLOCKED PORT NUMBER : 2 OF SWITCH ID: 1

cnd6@basem-virtual-machine: ~/pox$

```

Figure (17). PCA Detection the attack and close the controller.

Detection and Prevention of DDOS Attack using proposed model:

The figure (18) shows the POX Controller prediction for DDOS UDP traffic, the proposed model with Sample Entropy detection the attack and prevent data processing that comes from the attacker.

```

cnd6@basem-virtual-machine: ~/pox
cnd6@basem-virtual-machine: ~
Entropy : 0.0
dpid port and its packet count: 2 {1: 11} 11
Entropy : 0.0
dpid port and its packet count: 2 {1: 12} 12
Entropy : 0.0
dpid port and its packet count: 2 {1: 13} 13
Entropy : 0.0
dpid port and its packet count: 2 {1: 14} 14

-----

DDOS DETECTED

{2: {1: 14}}

2023-08-02 00:23:41.605331 : BLOCKED PORT NUMBER : 2 OF SWITCH ID: 1

-----

ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
Entropy : 0.0
Empty diction 2 2
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
Entropy : 0.0

```

Figure (18). The proposed model with Sample entropy detects the attack and prevent data processing that comes from the attacker.

The figure (19) shows the proposed model with PCA detection the attack and prevent data processing that comes from the attacker.

```

cnd6@basem-virtual-machine: ~/pox
cnd6@basem-virtual-machine: ~
deltaY : 3.004241800000002e-14
deltaY : -2.842170943040401e-14
deltaY : 7.105427357601002e-15
deltaY : 0.0
deltaY : 0.0

-----
DDOS DETECTED

2023-08-02 00:35:15.797756 : BLOCKED PORT NUMBER : 2 OF SWITCH ID: 1

ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
deltaY : -15.441176470588253
deltaY : -36.07843137254903
deltaY : -18.001871345029242
deltaY : 15.021570047360420

```

Figure (19). The proposed model with PCA detects the attack and prevent data processing that comes from the attacker.

Performance Evaluations and Discussion:

In this section, performance evaluations of the proposed models for DDoS attack detection using Sample entropy and PCA are discussed.

In Sample entropy, any packet with an entropy value below the threshold is recorded in a dictionary, and if an entry appears more than five times, it is assumed to be a DDoS attack. This knobs value of 5 is topologically dependent. Therefore, a smaller topology may potentially exhibit DDoS attacks for regular traffic.

As shown in the figure (16) using Sample entropy, the controller detects the attacker's port number and data continues to flow from the port without stopping, which causes the network controller to stop and thus the entire network to stop.

The figure (20) shows the continuous flow of data from the attacker device and the detection time of the attacks using entropy.

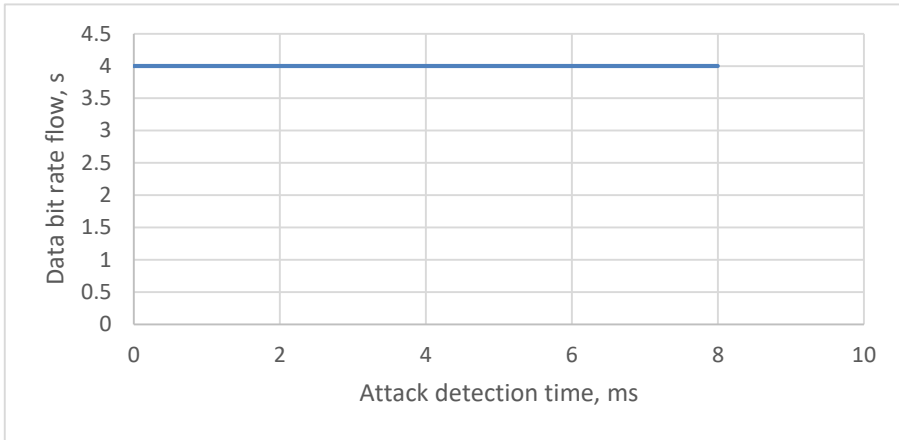


Figure (20). Continued data flow from the attacking device using entropy.

In PCA analysis, we are taking characteristics of each packet to update the principal component axis. Each packet destination value is compared with current principal component axis. During an attack, the principal component axis is gradually shifted towards the destination value. Hence, there will be successive decrease in delta Y values, which is an indication of DDoS attack.

As shown in the figure (17) using PCA, the controller detects the attacker's port number and data continues to flow from the port without stopping, which causes the network controller to stop and thus the entire network to stop.

The figure (21) shows the continuous data flow from the attacker device and the detection time using PCA.

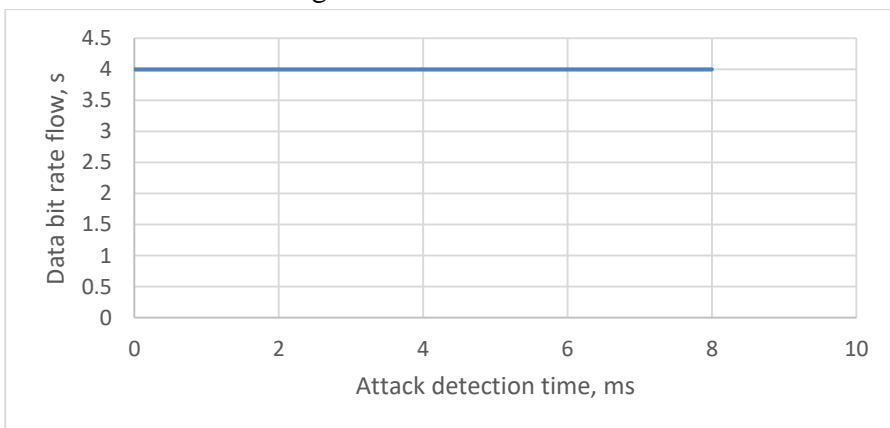


Figure (21). Continued data flow from the attacking device using PCA

As shown in Figure (18) and (19) the proposed model for detecting and preventing a DDoS attack, which was developed in our research, boils down to identifying the attacker's switch ID and port number, and collecting them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, the data flow from the attacking device stops, which allows the network controller to continue working, and thus the network continues to work without stopping.

The following figure (22) illustrates the detection of the attack using entropy from the port of the attacking device and preventing the reception of data from it.

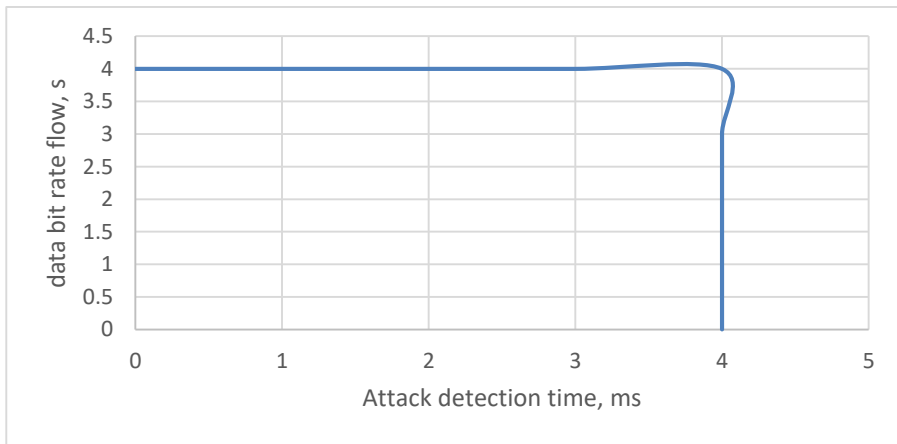


Figure (22). Detecting attacks and stopping data flow from the attacking device using entropy

The following figure (23) shows the detection of the attack using PCA from the port of the attacking device and preventing the reception of data from it.

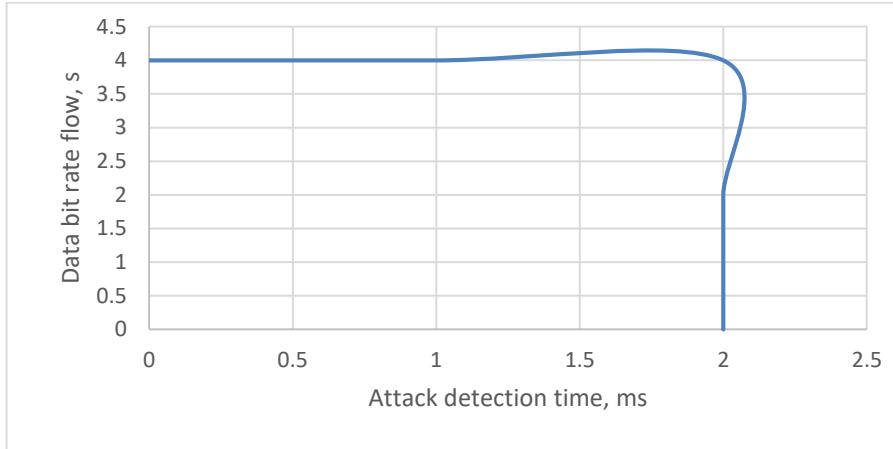


Figure (23). Detecting attacks and stopping data flow from the attacked device using PCA

As shown in Figure (22), the time to detect the attacks was 4 milliseconds and the data flow from the port of the controlling device was stopped. As shown in Figure (23), the time to detect the attacks was 2 milliseconds and the data flow from the port of the attacking device was stopped. From the above-mentioned Figures (22) and (23), it is clear that the PCA algorithm is the best and fastest in detecting distributed denial of service attacks. The two figures also show the validity of using the proposed model in our research paper to detect and prevent distributed denial of service attacks using entropy and PCA algorithms. Thus, the research has achieved the proposed objectives.

Conclusion:

The SDN network is characterized by dynamic programmability in terms of redirecting devices through open interfaces, and separating the control plane and data, as the elements of the control plane are now represented by a single entity, which is the controller. A DDoS attack is an attempt to make the services of network unavailable by exhausting the resources. The effect of this DDoS attack is even worse in an SDN than the traditional one. This is because, the controller will be invoked every time for a new Origin-Destination data pair. Hence within a short span of time if the attacker bursts too much packets into a network it will damage the controller, followed by the collapse of the network.

To detect and prevent DDoS attack, we configure and setup of the virtual work environment for the network using VMware and Mininet network emulator. Then we creating SDN network topology, launch of DDoS attacks in the created topology.

We use POX controller to detection and prevention of DDoS attacks using Sample entropy and PCA algorithms.

We developed a model to detect and prevent a DDoS attack in SDN environments using machine learning. The proposed model work to detect the attack by identifying the attacker's switch ID and port number, and collect them as a unique number that is added to a special list. Before processing data traffic for any device, the ID number and port number are combined as a unique number, and the list is searched. If they match, this data is ignored, without closing the controller. Finally, we evaluating the performance of the proposed model to detect and prevent attacks with Sample entropy and PCA algorithms. Through this evaluation, it was also proven that the PCA algorithm is the best and fastest in detecting and preventing a distributed denial of service attack.

Future Work:

In this paper we are capable to create normal traffic among nodes and capable to measure the entropy of each flow. The next steps:

1. The port of the attacking device is to be shut down by sending a command to the switch.
2. Find a mechanism to measure the randomness of each flow in each node using PCA.
3. Launch DDoS attack and try to capture it with both the methods Sample entropy and PCA.
4. Trying to come up with a new mechanism other than PCA and sample entropy measuring to detect DDoS attack in SDN environments effectively.

References:

- (1)Di Wu; Jie Li; Sajal K. Das; Jinsong Wu; Yusheng Ji; Zhetao Li. A Novel Distributed Denial-of-Service Attack Detection Scheme for Software Defined Networking Environments. IEEE International Conference on Communications (ICC). 20-24 May 2018. Kansas City, MO, USA. Electronic ISSN: 1938-1883.
- (2)Tian, Qiwen, and Sumiko Miyata. "A DDoS Attack Detection Method Using Conditional Entropy Based on SDN Traffic" IoT 4, no. 2: 95-111. 12 April 2023. <https://doi.org/10.3390/iot4020006>.
- (3)Aswanth P. P., Mohammed Ameen, Joe Antony. Analysis of DDoS Attacks in SDN Environments. Report. Department of Computer Science University of Tsukuba, Japan. January 10, 2017.
- (4)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. 67-71.
- (5)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, 1223-1228.
- (6)Open Networking Foundation. (2014, Jan.) ONF. [Online]. <https://www.opennetworking.org/>
- (7)Ombase P.M., Kulkarni N.P., Bagade S.T., Mhaisgawali A.V. Survey on DoS attack challenges in software defined networking Int. J. Comput. Appl., 173 (2) (2017), 19-25.
- (8)G. Shang, P. Zhe, X. Bin, H. Aiqun, R. Kui, Flood Defender: Protecting data and control plane resources under SDN-aimed DoS attacks, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, 1–9.
- (9)Ahmad, I., Namal, S., Ylianttila, M., & Gurtov, A. (2015). Security in Software Defined Networks: A Survey. IEEE Communications Surveys and Tutorials, 17(4), 2317-2346. <https://doi.org/10.1109/COMST.2015.2474118>.
- (10)Kandoi, R., & Antikainen, M. (2015). Denial-of-service attacks in OpenFlow SDN networks. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) (1322-1326).

- (11) H. Wang, L. Xu and G. Gu, "Flood Guard: A DoS Attack Prevention Extension in Software-Defined Networks", 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, 239-250.
- (12) C. Ji M. Thottan, "Anomaly Detection in IP Networks," IEEE Transaction on Signal Processing, vol. 51, no. 8, 2291-2204, Aug 2003.
- (13) Seyed Mohammad Mousavi, Marc St-Hilaire: Early Detection of DDoS Attacks Against Software Defined Network Controllers. J. Netw. Syst. Manag. 26 (3): 573-591 (2018).
- (14) D. Schnackenberg, R. Balupari, D. Kindred L. Feinstein, "Statistical Approaches to DDoS Attack Detection and Response," in DARPA Information Survivability Conference and Expedition, vol. 2003, Apr.
- (15) F. M. Ham, Principles of neurocomputing for Science and Engineering.: McGraw Hill, 1991.
- (16) D. O, Brien S. Seufert, "Machine Learning for Automatic Defence against Distributed Denial of Service Attack," in ICC, 2007, 1217-1222.
- (17) G. Serpen M. Sabhnani. (2014, Jan) BSTU Laboratory of Artificial Neural Networks. [Online]. http://neuro.bstu.by/ai/Todom/My_research/Papers-0/Forresearch/D-mining/Anomaly-D/KDD-cup-99/mlmta03.pdf
- (18) IBM. (2014, Feb) IBM SPSS Modeler. [Online]. http://pic.dhe.ibm.com/infocenter/spssmodl/v15r0m0/index.jsp?topic=%2Fcom.ibm.spss.modeler.help%2Fidh_neuralnet_network.htm.
- (19) Idris Z. Bholebawa, Upena D. Dalal. Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight Wireless Pers Commun DOI 10.1007/s11277-017-4939-z. Springer Science Business Media, LLC 2017.