

A Major Project Report
On
Detection of DDoS attacks on SDN network using Machine Learning

(Submitted in partial fulfillment of the academic requirements of B.Tech)

In
Department of Computer Science and Engineering

By
Bhanudeep Simhadri (18B61A05D6)

Nidhish Baheti (18B61A05B4)

R. Vineeth (18B61A05C6)

S. Gnaneshwar Reddy (18B61A05D1)

Under The Esteemed Guidance of

Dr. B. K. Madhavi

CSE Head of the Department



NALLA MALLA REDDY ENGINEERING COLLEGE

Autonomous Institution

(Approved by AICTE, New Delhi and Affiliated to JNTUH)

Accredited by NAAC with “A” Grade

NBA Accredited B. Tech Programs (CSE, ECE, EEE, Mech)

2021-2022



Nalla Malla Reddy Engineering College

Divya Nagar, Kachivanisingaram Post
Ghatkesar, Medchal. Dist.500 008

Phones: 08415-256001,02,03, Fax: 08415-256000

Email: info@nmrec.edu.in Website: www.nmrec.edu.in

CERTIFICATE

This is to certify that the project report entitled “**DETECTION OF DDOS ATTACKS ON SDN NETWORK USING MACHINE LEARNING**” being submitted by

BHANUDEEP SIMHADRI (18B61A05D6)

NIDHISH BAHETI(18B61A05B4)

RECHINTALA VINEETH (18B61A05C6)

S.GNANESHWAR REDDY(18B61A05D1)

in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide

Dr.B.K.Madhavi

HOD

Department of CSE

Head of the Department

Dr.B.K.Madhavi

HOD

Department of CSE

External Examiner

Principal

Dr.M.N.V.Ramesh

DECLARATION

I declare that this project report titled **DETECTION OF DDOS ATTACKS ON SDN NETWORK USING MACHINE LEARNING** submitted in partial fulfillment of the degree of **B. Tech in Computer Science and Engineering** is a record of original work carried out by me under the supervision of **Dr. B. K. Madhavi**. And has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Bhanudeep Simhadri (18B61A05D6)

Nidhish baheti (18B61A05B4)

Rechintala Vineeth (18B61A05C6)

S.Gnaneshwar Reddy (18B61A05D1)

ACKNOWLEDGMENTS

Any endeavor in the field of development is a person's intensive activity. A successful project is a fruitful culmination of efforts by many people, some directly involved and some others who have quietly encouraged and supported.

Salutation to the beloved and highly esteemed institute **NALLA MALLA REDDY ENGINEERING COLLEGE**, for grooming us into Computer Science and Engineering graduate. We wish to express our heart full thanks to the Director **Dr. Divya Nalla** whose support was indispensable to me during the course. We would also thank Principal **Dr. M.N. V. Ramesh** for providing great learning environment.

We wish to express profound gratitude to **Dr. B. K. Madhavi Head of the Department**, Computer Science and Engineering, for her continuous encouragement to ensure the successful results in all my endeavors.

We would like to thank **Dr. B. K. Madhavi HOD**, Department of Computer Science and Engineering, who has patiently guided and helped us throughout our project.

We take this opportunity to thank department Project Co-Ordinator **Mr. V. Mohan** for all the review meetings, suggestions and support throughout the project development.

Last but not the least, we thank our family members who are the backbone and provided support in every possible way.

By

Bhanudeep Simhadri (18B61A05D6)

Nidhish Baheti (18B61A05B4)

Rechintala Vineeth (18B61A05C6)

S. Gnaneshwar Reddy (18B61A05D1)

ABSTRACT

Title: Detection of DDoS attacks on SDN Network using Machine Learning

Software defined networking is going to be an essential part of networking domain which moves the traditional networking domain to automation network. Data security is going to be an important factor in this new networking architecture. The packets which travel through the network are prone to attacks and this paper aims to classify the traffic into normal and malicious classes based on features given in dataset by using various machine learning techniques using RapidMiner tool. The generated dataset is processed and applied on various Machine learning models. Out of which best performing model will be chosen and used to detect attacks in real-time on our SDN.

This project aims to implement different Machine Learning (ML) algorithms in RAPIDMINER tool to analyze the detection performance for DDoS attacks using realtime dataset generated on our SDN network. This research has used four different types of ML algorithms which are K_Nearest_Neighbors (K-NN), Super Vector Machine (SVM), Deep Learning (DL) and Random Forest (RF). The best accuracy result in the presented evaluation was achieved when utilizing the Random Forest (RF) algorithms.

TABLE OF CONTENTS

SNO	DESCRIPTION	PAGE
1	CERTIFICATE	i
2	DECLARATION	ii
3	ACKNOWLEDGEMENTS	iii
4	ABSTRACT	iv
5	TABLE OF CONTENTS	v
6	LIST OF FIGURES	vi
7	ABBREVIATIONS/ NOTATIONS/ NOMENCLATURE	ix
8	CHAPTER 1: INTRODUCTION	1-6
9	CHAPTER 2: EXISTING METHODOLOGY	7-17
10	CHAPTER 3: LITERATURE SURVEY	18-22
11	CHAPTER 4: PROBLEM DEFINITION	23-25
12	CHAPTER 5: PROPOSED SOLUTIONS	26-51
13	CHAPTER 6: SYSTEM REQUIREMENTS AND SYSTEM DESIGN	52-55
14	CHAPTER 7: UML DESIGN/ DIAGRAMS	56-58
15	CHAPTER 8: CODE	59-95
16	CHAPTER 9: RESULT AND DISCUSSIONS	96-100
17	CHAPTER 10: CONCLUSION	101-102
18	CHAPTER 11: FUTURE WORK	103-104
19	REFERENCES	105-110

LIST OF FIGURES**PAGE NUMBER**

Fig 1.1	SDN Network Architectures	3
Fig 2.1	Flowchart of attack detection method	10
Fig 2.2	Designed Network Topology	12
Fig 2.3	Information entropy variation	15
Fig 2.4	Comparison of log energy entropy	16
Fig 5.1	Graphical image of ML	28
Fig 5.2	SDN architecture	30
Fig 5.3	SDN network topology	33
Fig 5.4	RYU Architecture	34
Fig 5.5	Classification of DOS	35
Fig 5.6	Single UDP stream	37
Fig 5.7-5.9	Generating Normal Traffic	38
Fig 5.10-5.12	Generating DDoS Traffic	40
Fig 5.14-5.25	Operators Used in RapidMiner	42
Fig 5.26	Main process in RapidMiner	44
Fig 5.27-5.29	Building , Training, Testing K-NN Model	45
Fig 5.30-5.32	Building , Training, Testing RF Model	47
Fig 5.33-5.35	Building , Training, Testing SVM Model	49
Fig5.36-5.38	Building , Training, Testing DL Model	50
Fig 5.39	Comparing Performance of Models	41
Fig 7.1	Flow diagram for building and training the machine learning model	56
Fig 7.2	Flow diagram for testing and categorizing the data	57
Fig 9.1-9.6	Network Logs of Detection of Normal and Attack traffic	96

LIST OF FIGURES

PAGE NUMBER

Fig 9.7	Block Diagram for Detection of DDoS attacks on SDN network using ML	57
---------	---	----

LIST OF TABLES	PAGE
NUMBER	
Table 2.1 POX controller table	13
Table 2.2 Experiments results and discussion	14
Table 2.3 Log Energy Entropy values	16

ABBREVIATIONS/NOTATIONS/NOMENCLATURE

DOS: Denial of Service

DDoS: Distributed Denial of Service

SDN: Software Defined Network

RF: Random Forest

DL: Deep Learning

K-NN: K Nearest Neighbours

ML: Machine Learning

NN: Neural Networks

SVM: Support Vector Machine

IP: Internet Protocol

TCP: Transmission Control Protocol

UDP: User Datagram protocol

CHAPTER-1

INTRODUCTION

Introduction

SDN breaks the shackles of traditional network complexity and coupling and makes it possible for network architecture to satisfy flexibility, reliability and security at the same time. It separates the control plane from the data plane and separates the control function of the network from the data forwarding function. The control plane is only responsible for routing decisions, while the data plane realizes these decisions by forwarding packets and other behaviors. The separation of the two planes can improve the abstraction and programming ability of the network and makes the network structure less tedious and redundant. Although SDN architecture has many advantages compared with traditional network, it is often subjected to network threats and attacks. Network threats to SDN are mainly reflected in security device licensing and global view acquisition. Because packets are passed according to flow rules, physical security devices do not have the right to decide, and attackers can bypass security devices before deployment. The controller is the core of the entire network and can obtain various network status information. The attacker can use the controller to directly grasp the global view of the network to launch serious attacks. SDN has a clear plane structure, and the attack objects at different planes are different. At the control plane, because the controller manages the entire network, an attacker damaging the controller can cause serious damage. Controllers are the main targets of attacks in recent years.

In emerging SDN deployment scenarios such as data centers, the centralized control plane is the main cornerstone, which makes SDN more scalable. The controller has the right of absolute control over the whole SDN. They communicate with switches through commands and perform network operations through packet switching and routing SDN applications. It acts as a core brain, controlling the forwarding operation of the data plane and managing the traffic behavior of the entire network. The controller has the unique property of providing a global view of the network, making it the highest priority target in the net. Among the confirmed security vulnerabilities, DDoS is still one of the most important security problems at the control plane. In view of the importance of controllers

in software-defined networks, DDoS attacks on controllers are very dangerous and protecting controllers from attacks is also a major concern of researchers.

DDoS attacks send a large amount of traffic to the network and consume network resources and cause network congestion. Many DDoS attacks are launched from distributed hosts. A DDoS attack is an aggressive and destructive network attack that causes the system to stop working by depleting system resources. It can destroy the user's available network services, thus seriously threatening the network. When malicious data packets are sent by attackers on the network, normal traffic is processed or even cannot be processed due to the consumption of network resources. As a result, the network and servers become jammed and normal services are interrupted. Attackers who apply DDoS often target SDN mainly because of its unique characteristics.

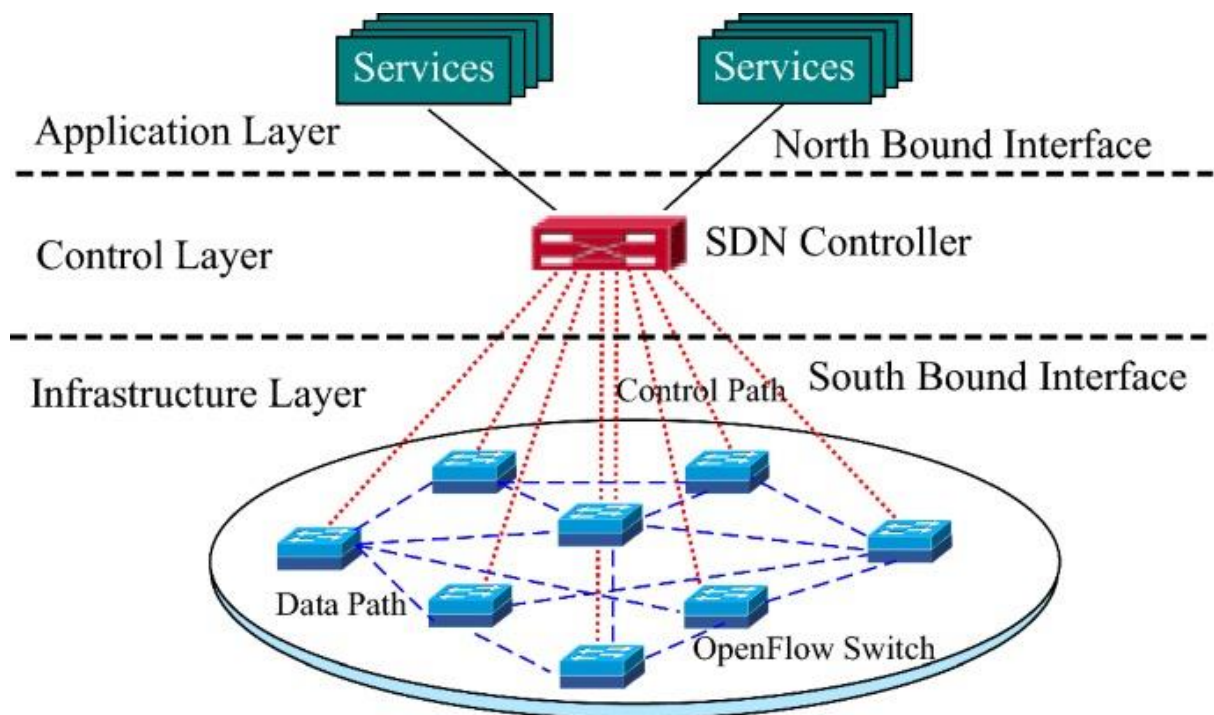


Fig 1.1 SDN Network Architecture

Distributed Denial of Service in SDN:

A DDoS attack is one of the biggest security threats to SDN network in recent ten years. It can not only prevent legitimate users from accessing and using network resources, but also destroy the entire network. Therefore, it is vital to protect the SDN network from DDoS attacks.

Attackers combine multiple hosts to form zombie host groups that meet their attack requirements. These zombie hosts send a large number of useless data packets to the target, making it consume a large amount of resources such as CPU and bandwidth to process these packets. Once the target host receives far more data packets than the load, it will not work properly and cannot process legitimate data packets. DDoS attacks are easy to implement and favored by attackers.

When the controller is attacked, the switch receives the attack packets and matches them with flow entries one by one. The attack packet is not valid and of course cannot match the flow table in the flow table entry. In this case, the switch encapsulates the packet as packet-in message and sends it to the controller. Then the controller makes decisions on the direction of the data packet. Attackers send a large number of attack packets so that packet-in messages continuously enter the controller and occupy a large number of controller resources . As a result, the controller cannot process legitimate traffic data, and the controller cannot work properly or is faulty.

DDoS attacks are of various types, such as TCP flood , UDP flood, and ICMP flood. An attacker sends a large amount of garbage traffic to the target network by operating the attack source, which sharply reduces the available bandwidth and prevents the target host from communicating with the outside world. TCP flood and UDP flood attacks use massive TCP and UDP packets to attack victims. ICMP flood is applied by displaying ICMP request packets to disturb normal traffic destined for the target host. In a TCP flood attack, the attacker sends a large number of forged IP address packets to the destination host. As the IP addresses of the packets are forged, the destination host cannot receive the response from the sender. The difference between UDP flooding attacks and TCP flooding attacks is that UDP is connectionless, which is commonly used in voice and video applications. An attacker initiates UDP flooding attacks by generating excessive

UDP packets to random ports of the destination host and prevents the attack target from responding to legitimate users.

Entropy: This entropy detection method is mainly used to calculate the distribution randomness of some attributes in the network packets' headers. These attributes could be the packet's source IP address, TTL value, or some other values indicating the packet's properties. For example, the detector captures 1000 continuous data packets at a peak point, and calculates the frequency of each distinct IP address among these 1000 packets. By further calculation of this distribution, we could measure the randomness of these packets randomness. Common entropy includes information entropy, mean energy, mean Teager Kaiser energy, shannon wavelet entropy and log energy entropy. Considering the probability of using destination IP address in this paper, for the case of only one variable, this paper mainly uses information entropy and log energy entropy and achieves the purpose of improving the detection effect by using complementarity through fusion.

To calculate the information entropy, the first is to calculate the probability of the destination IP address. The variable x is used to define the destination IP address of the packet, and the probability of x is calculated by using Equation Set the number of packets in the window to n . The probability of each element in the window is defined as p .

$$p_i = \frac{x_i}{\sum_{i=1}^n x_i}$$

In Equation (1), $i = \{1, 2, 3, \dots, n\}$, $0 \leq p_i \leq 1$.

Then calculate information entropy, the calculation formula is Equation (2). H is the information entropy of the destination IP address of packets appearing in a specific window.

$$H1 = - \sum_{i=1}^n p_i \log p_i$$

$$H2 = - \sum_{i=1}^n \log p_i^2$$

CHAPTER-2

EXISTING METHODOLOGY

Existing System:

Entropy is an important part of information theory. Entropy can measure the randomness of packets entering the network, which is the main reason for using entropy in DDoS detection. Entropy increases with the increase of randomness and decreases with the decrease of randomness. Common entropy includes information entropy, mean energy, mean Teager Kaiser energy, shannon wavelet entropy and log energy entropy. Considering the probability of using destination IP address in this paper, for the case of only one variable, this paper mainly uses information entropy and log energy entropy and achieves the purpose of improving the detection effect by using complementarity through fusion.

To calculate the information entropy, the first is to calculate the probability of the destination IP address. The variable x is used to define the destination IP address of the packet, and the probability of x is calculated by using Equation (1). Set the number of packets in the window to n . The probability of each element in the window is defined as p .

$$p_i = \frac{x_i}{\sum_{i=1}^n x_i}$$

In Equation (1), $i = \{1, 2, 3 \dots n\}$, $0 \leq p_i \leq 1$.

Then calculate information entropy, the calculation formula is Equation (2). H is the information entropy of the destination IP address of packets appearing in a specific window.

$$H1 = - \sum_{i=1}^n p_i \log p_i$$

Log energy entropy as another kind of entropy, its calculation formula is shown in Equation (3). In Equation (3), n and p_i still represent the number of packets and the probability of destination IP addresses

$$H2 = - \sum_{i=1}^n \log p_i^2$$

it is found that the entropy value of information entropy will be significantly reduced when an attack occurs, but the attack cannot be detected quickly, while the log energy entropy can quickly detect the attack, but the entropy value is not as obvious as the information entropy. This paper considers the fusion of the two entropies through weighting, so as to achieve complementary effects. Since p ranges from 0 to 1, according to the mathematical properties of the logarithmic function, the log energy entropy will get a negative value. In order to better integrate with the information entropy, we multiply the log energy entropy by minus one and weighted with information entropy. This change is shown in Equation (3). The selection of weight is based on the change rate of entropy decline of the two kinds of entropy when the attack occurs. The fusion entropy calculation is shown in Equation (4). In Equation (4), 0.75 in weight w_1 is the change rate of entropy corresponding to information entropy, while 0.13 in weight w_2 is the change rate of entropy corresponding to log energy entropy. The subsequent experimental results show that the fusion entropy effectively realizes the complementary advantages of information entropy and log energy entropy, which can not only quickly detect the attack but also have a high decline rate of entropy.

$$H3 = w_1 * H1 + w_2 * H2$$

$$w_1 = \frac{0.75}{0.75 + 0.13}$$

$$w_2 = \frac{0.13}{0.75 + 0.13}$$

When multiple data packets are received on the same host or switch port in a specific window and the number of data packets exceeds the threshold, DDoS attacks are detected. During an attack, if the computation entropy of a specified window continuously drops below the threshold, the target port on the specified switch is blocked. In the experiment, data packets will continuously flow into the network. For each

incoming packet, the entropy is calculated. If the entropy value is higher than or equal to the threshold, the next calculation will be carried out normally. If the entropy value falls below the threshold, the packet is logged. The threshold is determined based on the entropy fluctuation range in normal traffic scenarios. In the later experiments, the threshold will be described in more detail.

Packet Generation

Packet generation is done by scapy. It is a very powerful tool for package generation, scanning, sniffing, attacking, and forgery. Two parameters set in scapy are packet type and packet generation interval. The packet type includes TCP and UDP packets. UDP packets are used to cheat the source IP addresses of packets. The interval is set to fit the test case.

In this experiment, using scapy to generate network traffic so that normal and attack traffic can be simulated. With scapy, forged IP packets can be generated, in which way the identity of the real attacker is hidden. Run the scapy program on the host to send packets, both normal packets and attack packets (UDP packets and TCP packets) to the host in the designed network topology.

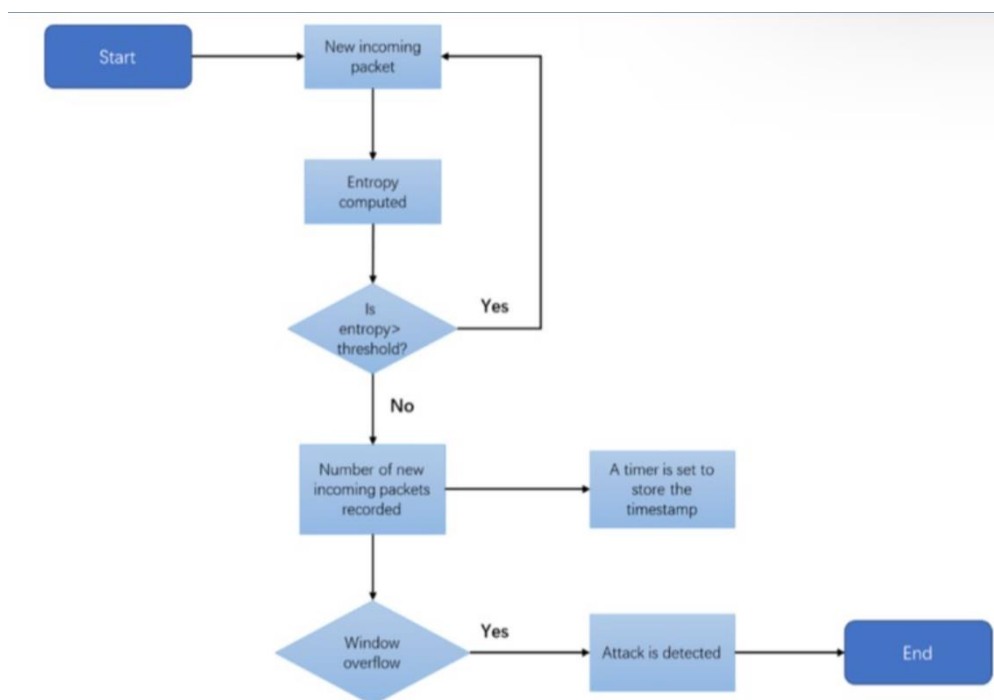


Fig 2.1 Flowchart of attack detection method

Window Size:

The window size should be set to less than or equal to the number of hosts to provide accurate calculations. In this paper, the window size is set to 50. The primary reason for choosing 50 is the limited number of incoming new connections to each host in the network. In SDN, once a connection is established, packets will not pass through the controller unless there is a new request.

The second reason is that each controller can only connect to a limited number of switches and hosts. Finally, the third reason for choosing this size is the amount of computation done for each window. A list of 50 values is much faster to compute than a list of 500 values, and attacks within the 50 packets window are detected earlier. Given the controller's limited resources, this window size is ideal for networks with only one controller and a few hundred hosts.

DDoS Attacks Detection on Simulated SDN Network:

The experimental operating system adopts Linux Ubuntu 20.04 with 20 GB memory and uses Ubuntu image files to run in Oracle VM VirtualBox software. The version of mininet is 2.3.0 and runs locally on Linux. The network topology shown in figure is created using mininet. The network is a tree structure with a depth of 2. The switch adopts OpenFlow switch, which can support OpenFlow protocol.

In order to realize the function of the controller, it is necessary to turn on the controller to observe the changes of traffic in the network during simulation. In SDN, when a data flow arrives, the Openflow switch will parse the data packet and match it with the local flow entry. If the match is successful, the data flow will be processed according to the forwarding policy. The flow table counter corresponding to this flow table entry is increased by 1.

If the match fails, the packet encapsulated as a packet_in message and sent to the controller through the Openflow switch. Then, the controller sends the updated flow table information to the Openflow switch. The flow table includes three modules: matching domain, counter and forwarding policy.

In the experiment, the destination IP address of the packet is extracted from the matching and the corresponding number of packets is obtained from the counter. Before the experiment, the window size and threshold size need to be set in advance. window setting is too small, not enough samples will be obtained, which will affect the accuracy of attack detection. If the window setting is too large, the computing load of the controller will increase.

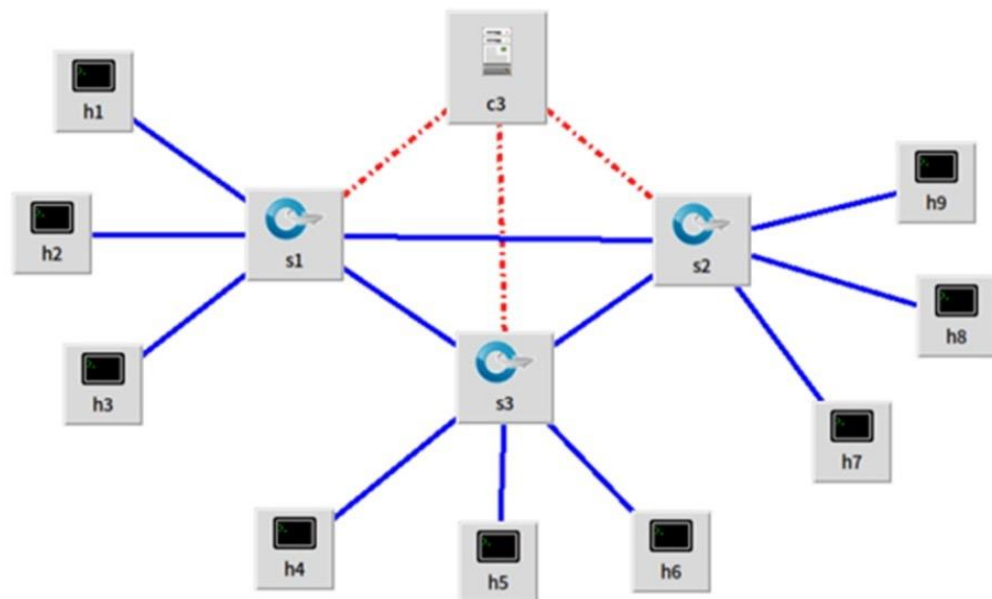


Fig 2.2 Designed network topology

Pox Controller:

The pox controller is developed by Stanford and is based on an Openflow. It is very necessary to choose a suitable controller. At present, there are many controllers available

for researchers and developers. some representative controllers and compares their basic information, as shown in Table.

Controller	Language	Openflow	Thread	Release
Ryu	Python	1.0~1.4	Single	03/2015
Pox	Python	1.0	Single	10/2013
Nox	C++	1.0&1.3	Multiple	02/2014
Beacon	Java	1.0	Multiple	09/2013
Floodlight	Java	1.0	Multiple	11/2012
Opendaylight	Java	1.0&1.3	/	03/2015
ONOS	Java	1.0&1.3	/	03/2015

Table 2.1

The Pox controller is used in this experiment. It is widely used in experiments with high speed and light weight. It is designed as a platform, so a user-defined controller can be built on it. Pox is an open source controller written in Python. Its advantage is to facilitate the application into face to the controller so that they can run in parallel with the controller.

The kernel is the assembly point for all components, and components can interact with each other through the kernel. Pox provides Openflow interface for topology discovery and path selection and can customize components to realize specific functions. Pox controller supports rapid development of controller prototype functions and it can produce superior performance applications, which decreases the burden of developers and improves development efficiency.

In addition to the advantages of programming language, Pox also has the advantage of clear architecture, good performance and strong scalability, so it has attracted the attention of many developers and researchers. Based on the above reasons, the Pox controller was selected for the simulation experiment.

Experimental Results and Discussion:

In this experiment, each test contains 250 data packets, divided by window value of 50, 5 results will be obtained, and a total of 16 times of running, and we will get a total of 80 data results. The results are plotted as a discounted graph, where the horizontal axis represents representative packets and the vertical axis represents the value of calculated entropy. This table shows the information entropy values of partial data packets in the normal and attack scenario. The entropy values of the data packets with 10 nodes in the above representative data packets are selected. The entropy value calculated in the normal scenario is that host 1 randomly sends normal traffic to the simulated network topology. To simulate the attack scenario, two hosts 4 and 6 send attack traffic to the target host 56.

Information Entropy		Incoming Packets
Normal	Attack	
1.11439	1.13712	1
1.1174	1.13712	10
0.84288	0.20198	20
1.0235	0.02878	30
1.0235	0.20452	40
1.05419	0.20017	50
1.0235	0.01419	60
1.0235	1.0235	70
0.84288	1.10643	80

Table 2.2

This figure shows the comparison of information entropy between the two scenarios. Among the 80 packets, the green line represents the information entropy value in the normal scenario, and the blue line represents the information entropy value in the attack scenario. The green line shows the packet's information entropy fluctuates between 0.8 and 1.4. The minimum entropy value of 0.8 was adopted as the threshold of our experiment, which could ensure the minimum error.

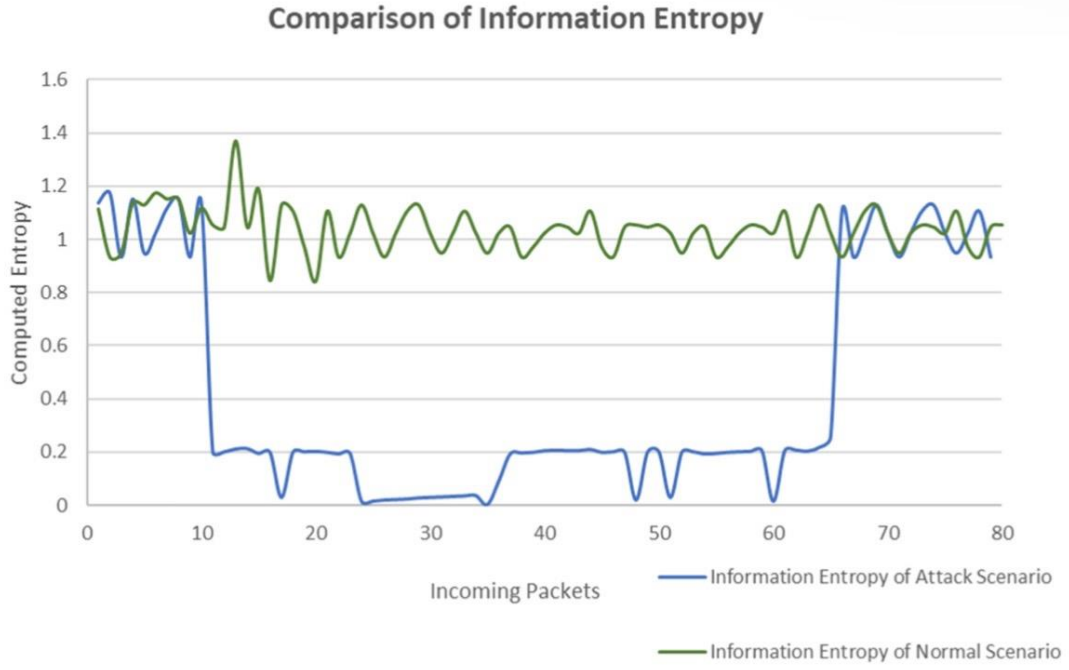


Fig 2.3 Information entropy variation

From the 10th packet to the 65th packet, the entropy value drops sharply and remains in the low value region between 0 and 0.21. The lowest point of normal flow entropy is 0.84288, and the highest point of attack flow entropy is 0.20882. The difference between the two is 0.63406, indicating a 75% decrease in information entropy under attack in the normal scenario. According to the experimental results of information entropy, when an attack occurs, the entropy value decreases obviously, but the attack is not detected immediately. Without changing other experimental conditions, log energy entropy is used to measure the randomness of the network. Table 3 shows the calculated log energy entropy values of partial data packets in the normal and attack scenario. The entropy values of the data packets with 10 nodes in the above representative data packets are selected.

Log Energy Entropy		Incoming Packets
Normal	Attack	
2.66956	2.67964	1
3.05772	1.23388	10
2.97717	1.0398	20
3.05772	1.0398	30
2.97717	1.04654	40
3.05772	1.04654	50
2.97717	1.09536	60
3.05772	1.0398	70
2.97717	2.96453	80

Table 2.3 Log Energy Entropy values.

The above table shows the comparison of log energy entropy under the two scenarios. In normal traffic scenarios, the entropy value of packets ranges from 1.7 to 3.5. We use the minimum entropy value of 1.7 as the threshold to ensure a small error. It can be seen from the figure that in the attack scenario, the value of log energy entropy starts to decline from the third packet, showing an earlier change.

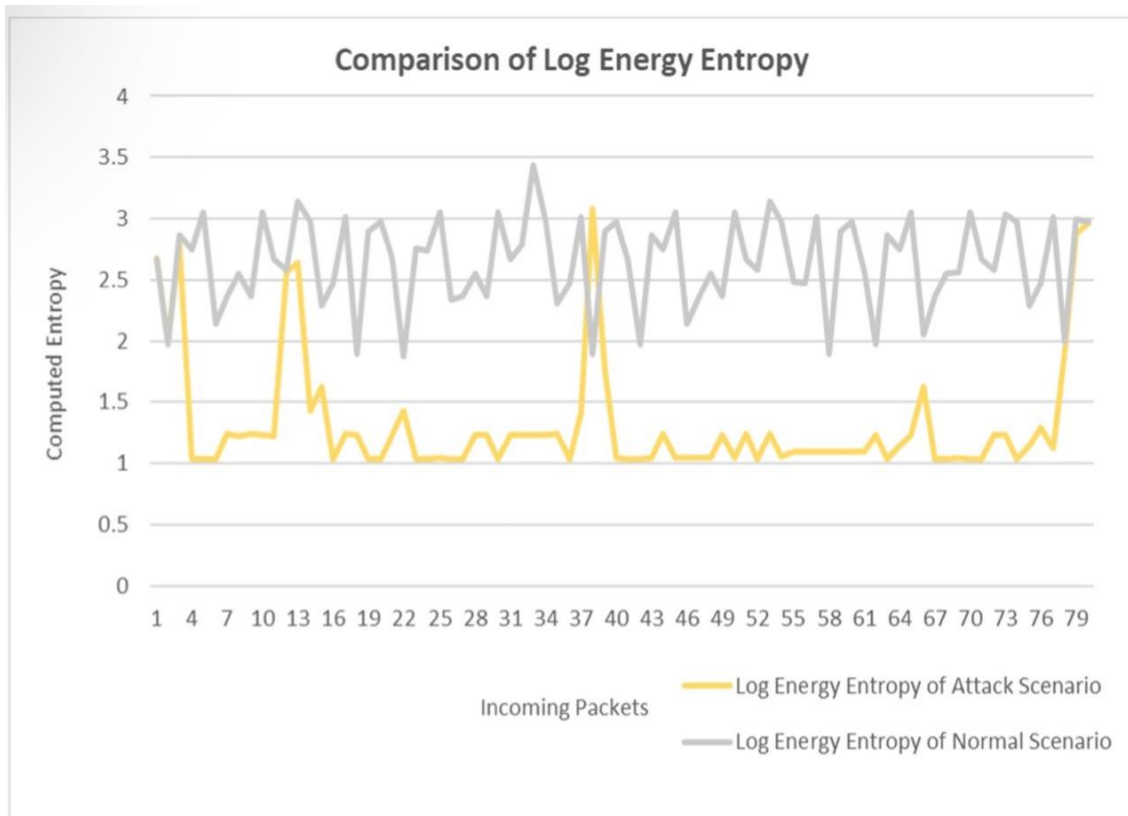


Fig 2.4

The lowest point of entropy of normal flow is 1.87379. The maximum entropy of attack traffic is 1.62796 when very few abnormal data are discarded. The difference between the two is 0.24583, indicating that the log energy entropy decreases by 13% when the attack occurs. According to the experimental results of log energy entropy, when an attack occurs, the attack can be detected immediately, but the change of entropy is small, and even a few extremely high entropy values will appear.

CHAPTER 3

LITERATURE SURVEY

“Early investigation of IDS was carried out by **Georgios Loukas and Glay** , who began their work with the time-line significant DOS.” In September 1996, a SYN Flood attack was discovered, a smurf attack began in January 1998 and a HTTP flood was the modern DOS that began in 2004. Specialists in the area suggest complete protection architecture through detection which can either be anomaly-based or signature based, or a hybrid of these two methods. Classifications such as neural networks, radial basis functions and genetic algorithms are increasingly used in DoS detection because of the automatic classification they can offer. The protection system either drops the attacking packets in a timely fashion or renders them inoperable. Traffic rate, SYN and URG flags, as well as some specific ranges of ports, are the most significant for the identification of a DoS attack, as some investigators have tested in their surveys.

Some work has been done in predict DDoS attack. “The early work of **Zhang et al. (2009)** focuses to shorten the delay of detection, short-term traffic prediction approach is used and prediction values were used in the detecting process.” The predicting approach is error-adjusted LMS (EaLMS), which has shorter predicting delay and less prediction error for the short term real-time prediction to smoother traffic than to violent fluctuating traffic.

Then, “some work of **Yi et al. (2006)** is to detect abnormal states with monitoring available service rates over the short time interval.” They manage to induce available service rates as a measurement to qualify server's availability based on autoregressive integrated moving average (ARIMA) to predict the attack. Then this technique was improved by Cheng et al. (2009) and Nezhad et al. (2016) by creating the detection technique more insensitive to the site and also correlated to its pattern with (CUSUM) algorithm. Qibo et al. (2009) try to Induce available service rates as the measurement to qualify server's availability based on Autoregressive integrated moving average (ARIMA) by detecting abnormal states with monitoring available service rates over the short time interval.

“**Fachkha et al. (2013)** proposed a technique based on ARMA linear prediction model (DDAP) as its prediction model and IP Flow features value (FFV) to detect the DDoS

attack. This technique based on four essential features namely flow dissymmetry, abrupt traffic change and distributed source IP addresses.”

“Some authors like **Chen et al. (2013)** are using rendered by extracting backscattered data and session flows from darknet traffic”. Then, DDoS activities are inferred and consequently tested for predictability. Finally, prediction techniques are applied on DDoS traffic. They used these approach extracting backscattered packets, extracting session flows, inferring DDoS activities, testing for predictable in such time series by using detrended fluctuation analysis (DFA) technique and forecasting DDoS by using ARIMA and GARCH. This researcher, Shin et al. (2013) proposed novel network anomaly detection algorithm (NADA) to detect the abnormal traffic and preprocessing network traffic predicted method (PNTPM) for prediction attack. This technique involves firstly they preprocessing network traffic by cumulatively averaging it with a time range, and using the simple linear AR model, and then generate the prediction of network traffic. Secondly, assuming the prediction error behaves chaotically, they used chaos theory to analyse it and then proposed a novel network anomaly detection algorithm (NADA) to detect the abnormal traffic. With this abnormal traffic, they train a neural network to detect DDoS attacks.

“**Xylogiannopoulos et al. (2014)** proposed the advanced probabilistic approach for network-based IDS (APAN) is proposed to forecast and detect network intrusions effectively.” This approach divided into three main phases using a Markov chain for probabilistic modelling. First, K-means clustering is performed to define network states, and then the concept of the outlier factor is newly introduced. Next, based on the defined states, a Markov model including a state transition probability matrix and the initial probability distribution is built. Finally, the degree of abnormality of incoming data is stochastically measured using the model in real-time.

Recently, “**Olabeur et al. (2015)** using data mining technique is introduced in order to early warning the network administrator of a potential DDoS attack.” This method uses the advanced all repeated patterns detection (ARPAD) algorithm, which allows the

detection of all repeated patterns in a sequence. The proposed method can give very fast results regarding all IP prefixes in a sequence of hits and, therefore, it warns the network administrator if a potential DDoS attack is under development.

“**Xia** (2014) proposed Shannon-entropy concept and clustering algorithm” of relevant feature variables by initial evaluation of the proposed framework targeting forecasting and early detection of DDoS attack by engaged two categorical alert features namely the destination port number and the alert type message, and the concept of Shannon-entropy is employed to measure the degree of divergence of these features and to detect outliers. “**Zargar et al.** (2013) proposed algorithm that analyses the historical traffic data to predict the expected behaviour and also they used regression models and autoregressive integrated moving average (ARIMA) models for forecasting the time series data.” Besides that, they also propose to defence node placement using optimal approach, maximum-coverage-node-first (MCNF) and weak-path-first (WPF) approach. The defence nodes are selected off-line in the k-hop radius of the victim node. Each defence node is equipped with the capability to perform traffic differentiation and rate limiting. Each defence node is also equipped with the attack predictor mechanism. The defence node on predicting an impending attack sends a warning message to all other defence nodes. The warning message contains the time to attack, intensity of attack, and the potential attacker nodes. They assume a separate control channel of communication between defence nodes.

A method call fields programmable gate arrays (FPGA) is proposed by Hoque et al. (2017). This technique capable to detect DDoS attack faster in real-time environment was implemented on software and hardware platform. This device requires less than one microsecond to identify an incoming traffic as normal or attacks.

“**Jazi et al.** (2017) proposed DDoS attacks detection and prediction for application layer based on non-parametric CUSUM algorithm which is not only can precisely detect flooding attack but also stealthy”

“Kaliyamurthy, N.M.; Chen, S.; Jiang, H.; Zhou, Y.; Campbell, C. Detection of DDoS Attacks in Software Defined proposed that This study proposes a detection method of Distributed Denial of Service attacks in Software Defined Networking, which uses the property of entropy to measure the occurrence of attack behavior in the network.” The significance of this study is to quickly and effectively detect Distributed Denial of Service attacks in the Software Defined Networking and protect the SDN controller against security threats

“Meenakshi Mittal1 Krishan Kumar · Sunny Behal (2022).; In today’s world, technology has become an inevitable part of human life.” In fact, during the Covid-19 pandemic, everything from the corporate world to educational institutes has shifted from offline to online. It leads to exponential increase in intrusions and attacks over the Internet-based technologies. One of the lethal threat surfacing is the Distributed Denial of Service (DDoS) attack that can cripple down Internet-based services and applications in no time. The attackers are updating their skill strategies continuously and hence elude the existing detection mechanisms. Since the volume of data generated and stored has increased manifolds, the traditional detection mechanisms are not appropriate for detecting novel DDoS attacks. This paper systematically reviews the prominent literature specifically in deep learning to detect DDoS. The authors have explored four extensively used digital libraries (IEEE, ACM, ScienceDirect, Springer) and one scholarly search engine (Google scholar) for searching the recent literature. We have analyzed the relevant studies and the results of the SLR are categorized into five main research areas: (i) the different types of DDoS attack detection deep learning approaches, (ii) the methodologies, strengths, and weaknesses of existing deep learning approaches for DDoS attacks detection (iii) benchmarked datasets and classes of attacks in datasets used in the existing literature, and (iv) the preprocessing strategies, hyperparameter values, experimental setups, and performance metrics used in the existing literature (v) the research gaps, and future directions

CHAPTER 4

PROBLEM DEFINITION

Over the past years, distributed denial-of-service (DDoS) attacks on Internet services and websites have dramatically increased. Several research teams designed defensive methodologies to handle the DDoS attacks. Using machine learning-based solutions have enabled researchers to detect DDoS attacks with complex and dynamic patterns. In this work, a subset of the CICIDS2017 dataset, including 200K samples and 84 features, was used to analyze the features and build models. A correlation analysis, as well as a tree-based feature importance exploration, were performed in the feature engineering step. Next, decision tree and support vector machine models were trained and tested to classify DDoS and Benign attacks. The results revealed that "Flow ID," "SYN Flag Cnt," and "Dst IP" had the most impact on attack detection. Also, the machine learning models classified the DDoS attacks, where the accuracy rates of close to 100% were achieved. The decision tree models showed slightly better performance than linear support vector machines. The results in this work highly matched the outcome of the original paper, which was to replicate.

Despite all its tremendous capabilities, the SDN faces many security issues due to the complexity of the SDN architecture. Distributed denial of services (DDoS) is a Disruptive attack on SDN due to its centralized architecture, especially at the control layer of the SDN that has a network-wide impact.

Network security has become of utmost importance in all areas of business and industry, including bank transactions, Email, social media and university eServices, etc. Recently, web and network services have suffered from intruder attacks. Hackers are continually generating new types of Distributed Denial of Service (DDoS) which work on the application layer as well as the network layer. The vulnerabilities in the above mentioned areas allow hackers to deny access to web services and slow down access to network resources. The IDS system is one of the most common solutions to dealing with the problem of DDoS and preserving the confidentiality, integrity and availability of web services and computer network resources. IDS system uses machine learning techniques to detect and classify types of DDoS in an intelligent way and will eliminate intrusion without referral to the System Security Officer (SSO), however achieving one hundred percent accuracy in detecting and classifying all new types of attacks is hard to achieve.

Unfortunately, most of the common open access data sets have duplicated and redundant instances, which make the detection and classification of the DDoS unrealistic and ineffectual. There are also no available data sets (e.g. KDD 99) which include new DDoS types, such as HTTP flood and SIDDOS. In this research, we collected a completely new dataset in a controlled environment, which includes four harmful types of attack namely: UDP flood, Smurf, HTTP Flood and SIDDOS. Machine learning is used to detect and classify network traffic based on some features (average packet size, inter arrival time, packet size, packet rate, bit rate, etc.) that are used to measure and determine whether the network traffic is normal or is a type of DDoS. DDoS attacks mostly have the same average packet size. The number of packets will increase in the attacked packet rather than the normal packet; also, the inter arrival time will be too small to allow attackers to consume resources rapidly. DDoS packets always have a high bit rate for network layer attack. Attackers focus on any attributes that help them to consume resources and make the service unavailable to end users.

Other work on preventing/ avoiding attacks by means of, for example, fuzzy clustering, genetic algorithm, and artificial neural network (ANN) has been conducted. Ms. Jawale and Prof. Bhusari presented research on ANN that achieved the highest accuracy rate. They proposed a system that uses multilayer perceptrons, back propagation and a support vector machine, consisting of multi modules such as packet collection and preprocessing data. This system achieved 90.78% detection rate.

The proposed system that is used to collect a new dataset is organized as illustrated in Figure 7. The selected classifiers were tested in 734,627 records which they were fully randomized to obtain realistic results.

MLP permits the data flow to travel one way, from input to output. There is no feedback; it tends to be straight-forward networks that companion inputs with outputs. According to [20] [21], any MLP network can be distinguished by a number of performance characteristics.

CHAPTER- 5

PROPOSED SYSTEM

In our proposed system we used Machine Learning based approach to detect DDOS attacks in a SDN Network. So basically, Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for building mathematical models and making predictions using historical data or information. Machine Learning is said as a subset of artificial intelligence that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own. With the help of sample historical data, which is known as training data, machine learning algorithms build a mathematical model that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models. Machine learning constructs or uses the algorithms that learn from historical data. The more we will provide the information, the higher will be the performance.

There are some features of Machine Learning like

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

So Why should we know/Learn about Machine Learning?

Machine Learning can automate many tasks, especially the ones that only humans can perform with their innate intelligence. Replicating this intelligence to machines can be achieved only with the help of machine learning. It also helps in automating and quickly create models for data analysis. Various industries depend on vast quantities of data to optimize their operations and make intelligent decisions. Machine Learning helps in creating models that can process and analyze large amounts of complex data to deliver accurate results. These models are precise and scalable and function with less turnaround time. By building such precise Machine Learning models, businesses can leverage profitable opportunities and avoid unknown risks. The reason for learning machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some

Why Software-Defined Networking is important?

Increased control with greater speed and flexibility: Instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can choose a single protocol to communicate with any number of hardware devices through a central controller.

Customizable network infrastructure: With a software-defined network, administrators can configure network services and allocate virtual resources to change the network infrastructure in real time through one centralized location. This allows network administrators to optimize the flow of data through the network and prioritize applications that require more availability. Robust security: A software-defined network delivers visibility into the entire network, providing a more holistic view of security threats. With the proliferation of smart devices that connect to the internet, SDN offers clear advantages over traditional networking. Operators can create separate zones for devices that require different levels of security, or immediately quarantine compromised devices so that they cannot infect the rest of the network. The key difference between SDN and traditional networking is infrastructure: SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control the network, change configuration settings, provision resources, and increase network capacity — all from a centralized user interface, without the need for more hardware. There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because software-defined networks use a centralized controller, securing the controller is crucial to maintaining a secure network.

SDN Architecture:

A typical representation of SDN architecture comprises three layers: the application layer, the control layer and the infrastructure layer. These layers communicate using northbound and southbound application programming interfaces.

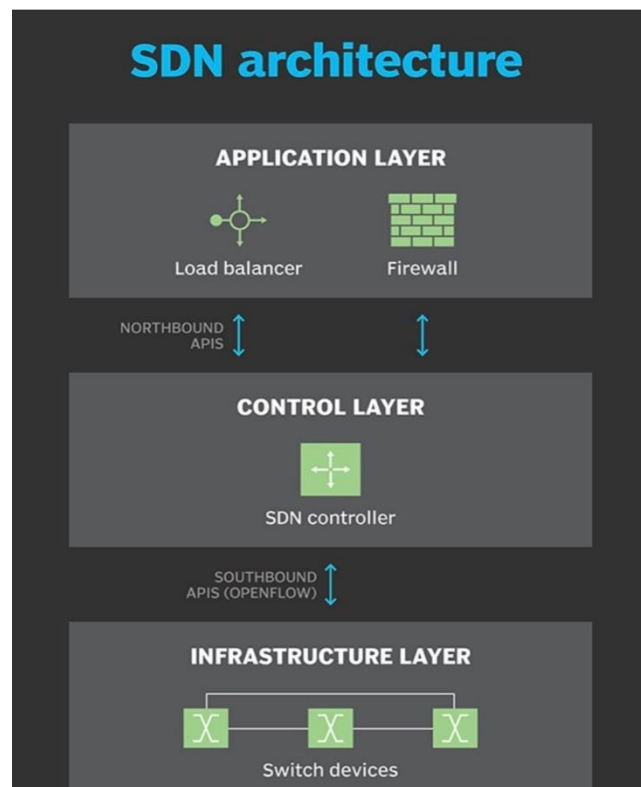


Fig 5.2 SDN Architecture

Application layer:

The application layer contains the typical network applications or functions organizations use. This can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses a controller to manage data plane behavior.

Control layer:

The control layer represents the centralized SDN controller software that acts as the brain of the software-defined network. This controller resides on a server and manages policies and traffic flows throughout the network.

Infrastructure layer: The infrastructure layer is made up of the physical switches in the network. These switches forward the network traffic to their destinations

How Does SDN works?

SDN encompasses several types of technologies, including functional separation, network virtualization and automation through programmability. Originally, SDN technology focused solely on the separation of the network control plane from the data plane. While the control plane makes decisions about how packets should flow through the network, the data plane moves packets from place to place. In a classic SDN scenario, a packet arrives at a network switch. Rules built into the switch's proprietary firmware tell the switch where to forward the packet. These packet-handling rules are sent to the switch from the centralized controller. The switch also known as a data plane device -- queries the controller for guidance as needed and provides the controller with information about the traffic it handles. The switch sends every packet going to the same destination along the same path and treats all the packets the same way. Software-defined networking uses an operation mode that is sometimes called adaptive or dynamic, in which a switch issues a route request to a controller for a packet that does not have a specific route. This process is separate from adaptive routing, which issues route requests through routers and algorithms based on the network topology, not through a controller. The virtualization aspect of SDN comes into play through a virtual overlay, which is a logically separate network on top of the physical network. Users can implement end-to-end overlays to abstract the underlying network and segment network traffic. This micro segmentation is especially useful for service providers and operators with multi-tenant cloud environments and cloud services, as they can provision a separate virtual network with specific policies for each tenant.

Benefits of SDN Network:

- **Automation of Network Provisioning:** Network orchestrators can do this without SDN too. But very few mature and niche products are around.
- **Integration with Public Cloud:** Network provisioning can be done today from a cloud portal without getting into manual device centric configuration mode as earlier.

- **Abstraction of Infrastructure:** Complexity of network infrastructure i.e., vendor specific interfaces, commands, software features are made available to upper layers (like cloud platform or stand-alone orchestrators) as abstracted GUI drop-downs.

Service Chaining and Automation-Seamless insertion and configuration of network services on the fly in an elastic fashion on-demand.

- **Low Capital Expenditure Costs:** Use of less expensive hardware “white box” switches with intelligence centered at SDN controller. Guaranteed Content Delivery-SDN’s ability to shape and control data traffic results in enforcing quality of service (QOS) network fabric delivering guarantees on performance for applications..

- **Physical vs virtual Networking Management:** Physical environments necessitate collaboration among different teams to get a task done. For example, if you require some modification at a physical networking device, it would often take a considerable amount of time and teamwork in most organizations before the task can be accomplished. Software-defined networking provides you the ability to control virtual and physical networking by using a central management tool, sometimes known as a single pane of glass. A virtual administrator can process the necessary changes without needing to collaborate with different teams.

- **Isolation and Traffic Control:** Data center managers can benefit from centralizing the networking control using a central management tool. At the same time, SDN provides several isolation mechanisms such as configuring ACLS and firewalls at the virtual machine NIC level. You can also define the traffic rules using the SDN management console, which helps in providing full control over the network traffic.

Our SDN Topology:

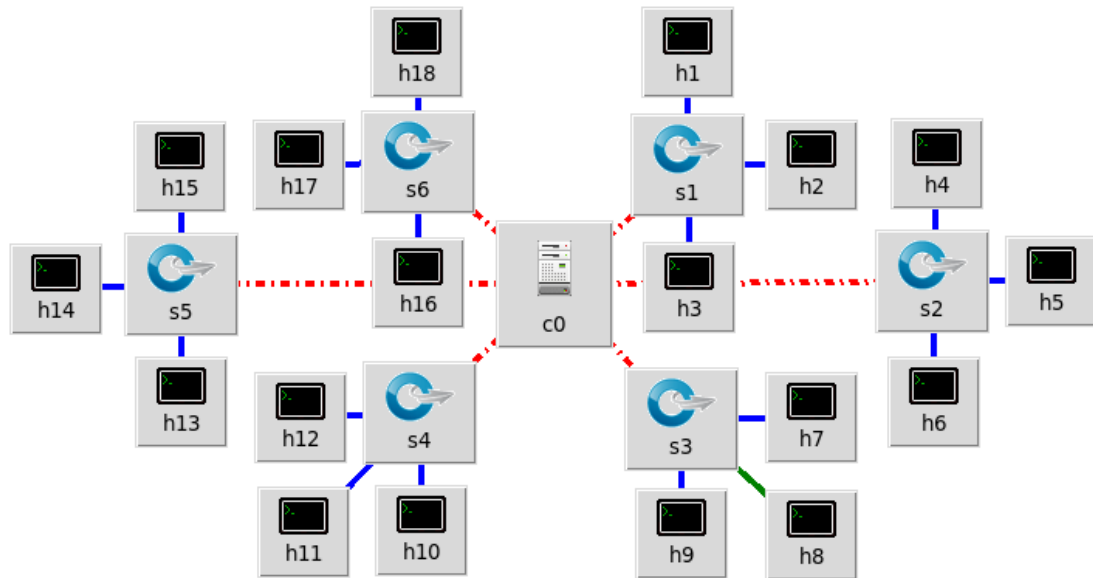


FIG 5.3 SDN Network topology

RYU MANAGER:

RYU is a component-based software defined networking framework. RYU provides software components with well-defined API that make it easy for developers to create new network management and control applications. RYU supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, RYU supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions. Ryu controller is a component of RYU manager. With the help of RYU manager we are using our controller of SDN. Or controller is being controlled by RYU manager.

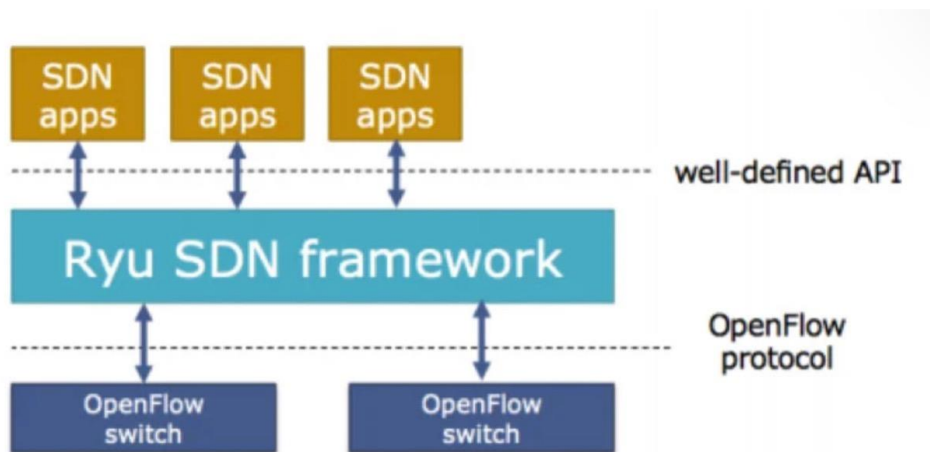


FIG:5.4 RYU ARCHITECTURE

RYU controller is a component of RYU manager. With the help of RYU manager we are using our controller of SDN. Or controller is being controlled by RYU manager

DDOS ATTACKS:

Historically, Denial of Service (DoS) attacks are intended primarily to disrupt computing systems in a network. Fundamentally, these attacks are initiated from a single machine with the illegitimate intention of targeting a server system through an attack. A simple DoS attack could be a PING Flood attack in which the machine sends ICMP requests to the target server and a more complex DoS attack example could be Ping of death attack. DDoS (Distributed Denial of Service) attacks are precursor to DoS attacks, i.e., DoS attacks are forerunner to DDoS attacks. DDoS attacks are the attacks which are carried in distributed environments. Fundamentally, a DDoS attack is an intentional attack type which is usually made in a distributed computing environment by targeting a website or a server so as to minimize their normal performance. To achieve this, an attacker uses multiple systems in a network. Now, using these systems, the attacker makes an attack on the target website or server by making multiple requests to the target system or server. As these types of attacks are carried out in distributed environments, hence, these are also called distributed DDoS attacks.

The conventional way of DDoS attacks is the brute force attack that is triggered using Botnet wherein the devices of the network environment are infected with malware. Based on the target and the behavior, we may classify DDoS attacks into three categories. Thus, though DDoS attacks could be categorized into several types, usually these attacks are mainly classified into three classes. They are (i) Traffic/fragmentation attack, (ii) Bandwidth/Volume attack and (iii) Application attack as shown in Figure. In traffic-based attacks, voluminous UDP or TCP packets are sent to the target system by the attacker and these huge UDP or TCP packets reduce the system performance. In the second type of attack called bandwidth or volumetric attacks, the attacker creates congestion in bandwidth through consuming excessive bandwidth than required legitimately and they also try to flood the target system through sending large amounts of anonymous data. The last type of attack are also specialized attacks as they are aimed at attacking only a specific system or a network. These types of attacks are also difficult to mitigate and throw greater challenges in recognizing them.

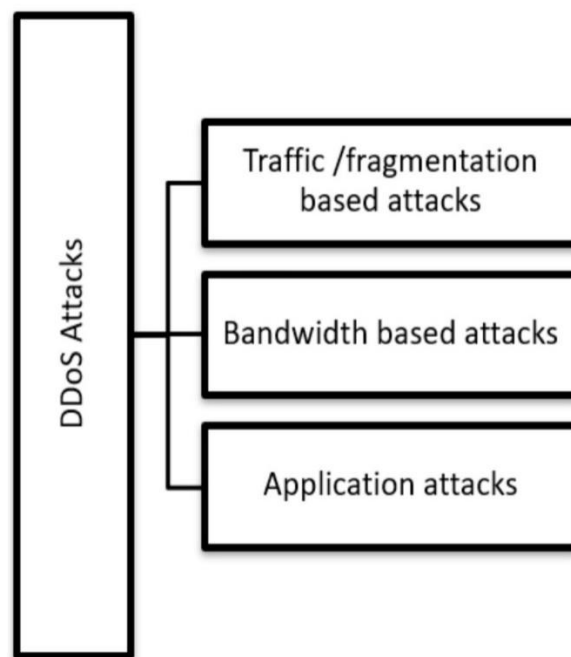


FIG : 5.5 Classification of DoS (Distributed Denial of Service) attacks

In general, a Denial of Service attack, which is usually called a DoS attack, is a purposeful attempt which is initiated so as to make an application or website unavailable to its legitimate users. This is achieved usually by flooding the website or application through network traffic. In order to achieve this, usually, one of the several choices of attackers is to apply diversified techniques that intentionally consume huge network bandwidth, thus causing inconvenience to legitimate users. Alternately, attackers also achieve this by handling system resources in an illegitimate manner. DoS attack is also called Non-distributed Directed attack wherein an attacker initiates DoS attack on the target system. The concept of DDoS attack is similar to DoS attack but the fundamental difference is that in DDoS attacks there are multiple attack sources which are implicitly involved, i.e., in DDoS attacks, the attacker makes an attack by using multiple sources which may include routers, IoT devices, and computers in a distributed environment infected by malware. To make this possible, an attacker looks for availability of any compromised network. By utilizing such compromised networks, an attacker usually attacks the target system through continuously generating packet floods or requests to conquer the target system. The DDoS attacks are common in the Network layer, Transport layer, Presentation and Application layer of the 7-layer OSI reference model. Network layer and Transport layer attacks are usually called Infrastructural attacks whereas the Presentation layer and Application layer attacks are commonly known as Application layer attacks.

PACKET GENERATION

GENERATING NORMAL TRAFFIC:

IPERF is a traffic generation tool that allows the user to experiment with different TCP and UDP parameters to see how they affect network performance.

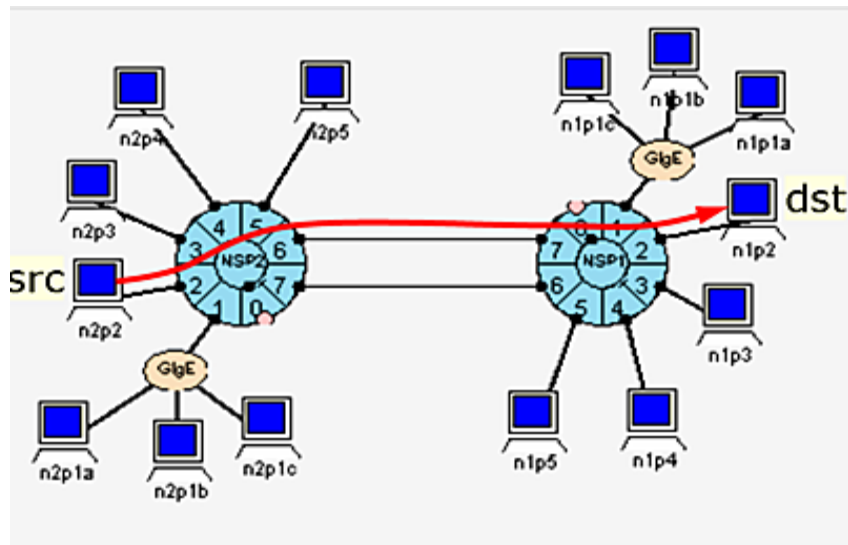


Fig 5.6 Single UDP Stream

The typical way that IPERF is used is to first start one IPERF process running in server mode as the traffic receiver, and then start another IPERF process running in client mode on another host as the traffic sender. In order to send a single UDP stream from n2p2 to n1p2 as shown in Fig we would run IPERF in server mode on n1p2 and IPERF in client mode on n2p2.

Command to generate normal traffic: ping 10.0.0.15,

`iperf -c 192.168.1.102 -i1 -t60`

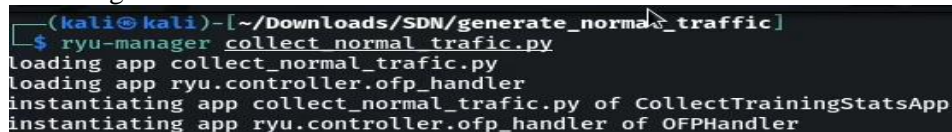
Generating DDOS Traffic:

A Distributed Denial of Service attack (DDOS) is similar to a DOS attack but carried out from different nodes (or different attackers) simultaneously. Commonly DDOS attacks are carried out by botnets. Botnets are automated scripts or programs which infect computers to carry out an automated task (in this case a DDOS attack). A hacker can create a botnet and infect many computers from which botnets will launch DOS attacks, the fact many botnets are shooting simultaneously turn the DOS attack into a DDOS attack (that's why it is called "distributed").

The tool hping3 allows you to send manipulated packets. This tool allows you to control the size, quantity and fragmentation of packets in order to overload the target and bypass or attack firewalls. Hping3 can be useful for security or capability testing purposes, using it you can test firewalls effectivity and if a server can handle a big amount of packets. Below you will find instructions on how to use hping3 for security testing purposes.

Command to generate attack traffic: hping3 -S -V -d 120 -w 64 -p 80 --rand-source --flood 10.0.0.12

Generating Normal Traffic Dataset



```
(kali@kali)-[~/Downloads/SDN/generate_normal_traffic]
$ ryu-manager collect_normal_traffic.py
loading app collect_normal_traffic.py
loading app ryu.controller.ofp_handler
instantiating app collect_normal_traffic.py of CollectTrainingStatsApp
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Fig 5.7

We execute command collect_normal_traffic.py file with ryu-manager. This is used to enable a channel for the Normal traffic flow to our SDN.


```

(kali@kali)-[~/Downloads/SDN/generate_normal_traffic]
$ sudo python generate_normal_traffic.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3)
(h10, s4) (h11, s4) (h12, s4) (h13, s5) (h14, s5) (h15, s5) (h16, s6) (h17, s6)
(h18, s6) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
-----
Generating traffic ...

```

Fig 5.8

We then execute command `generate_normal_traffic.py` using python to start our SDN network and Generate Normal traffic using Iperf tool which is embedded in python code to perform various iterations of Normal packets flow to generate Normal Traffic.

```

h18 Downloading test.zip from h1
-----
Iteration n 73 ...
-----
generating ICMP traffic between h11 and h17 and TCP/UDP traffic between h11 and h1
h11 Downloading index.html from h1
h11 Downloading test.zip from h1
generating ICMP traffic between h14 and h3 and TCP/UDP traffic between h14 and h1
h14 Downloading index.html from h1
h14 Downloading test.zip from h1
generating ICMP traffic between h1 and h6 and TCP/UDP traffic between h1 and h1
h1 Downloading index.html from h1
h1 Downloading test.zip from h1
generating ICMP traffic between h7 and h12 and TCP/UDP traffic between h7 and h1
h7 Downloading index.html from h1
h7 Downloading test.zip from h1
generating ICMP traffic between h18 and h9 and TCP/UDP traffic between h18 and h1
h18 Downloading index.html from h1
h18 Downloading test.zip from h1
generating ICMP traffic between h8 and h16 and TCP/UDP traffic between h8 and h1
h8 Downloading index.html from h1
h8 Downloading test.zip from h1
generating ICMP traffic between h9 and h2 and TCP/UDP traffic between h9 and h1
h9 Downloading index.html from h1
h9 Downloading test.zip from h1
generating ICMP traffic between h7 and h14 and TCP/UDP traffic between h7 and h1
h7 Downloading index.html from h1
h7 Downloading test.zip from h1
generating ICMP traffic between h6 and h3 and TCP/UDP traffic between h6 and h1
h6 Downloading index.html from h1
h6 Downloading test.zip from h1
generating ICMP traffic between h7 and h14 and TCP/UDP traffic between h7 and h1

```

Fig 5.9

We have run 78 such iterations to generate Normal Traffic Dataset.

Generating DDoS traffic Dataset:

```
(kali@kali)-[~/Downloads/SDN/generate_ddos_traffic]
$ ryu-manager collect_ddos_traffic.py
loading app collect_ddos_traffic.py
loading app ryu.controller.ofp_handler
instantiating app collect_ddos_traffic.py of CollectTrainingStatsApp
instantiating app ryu.controller.ofp_handler of OFPHandler
█
```

Fig 5.10

We execute command `collect_ddos_traffic.py` file with RYU-manager. This is used to enable a channel for the DDoS traffic flow to our SDN.

```
(kali@kali)-[~/Downloads/SDN/generate_ddos_traffic]
$ sudo python generate_ddos_traffic.py
[sudo] password for kali:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3)
(h10, s4) (h11, s4) (h12, s4) (h13, s5) (h14, s5) (h15, s5) (h16, s6) (h17, s6)
(h18, s6) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
-----
Performing Smurf
-----
-----
Performing UDP-Flood
-----
```

Fig 5.11

We then execute command `generate_ddos_traffic.py` using python to start our SDN network and Generate DDoS traffic using hping3 tool which is embedded in python code to perform various iterations of DDoS attacks flow to generate Attack Traffic.

			C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
125623	125621	1.59E+09	5	10.0.0.18C	10.0.0.18	0	10.0.0.3	0	1	0	0	91	6.86E+08	20	100	0	91	8918	1	1.33E-07	98	1.30E-05	Normal	
125624	125622	1.59E+09	5	10.0.0.301	10.0.0.3	0	10.0.0.18	0	1	0	8	91	6.93E+08	20	100	0	91	8918	1	1.31E-07	98	1.29E-05	Normal	
125625	125623	1.59E+09	2	10.0.0.18C	10.0.0.18	0	10.0.0.3	0	1	0	0	91	6.88E+08	20	100	0	91	8918	1	1.34E-07	98	1.31E-05	Normal	
125626	125624	1.59E+09	2	10.0.0.301	10.0.0.3	0	10.0.0.18	0	1	0	8	91	7.01E+08	20	100	0	91	8918	1	1.30E-07	98	1.27E-05	Normal	
125627	125625	1.59E+09	4	10.0.0.18C	10.0.0.18	0	10.0.0.3	0	1	0	0	91	6.83E+08	20	100	0	91	8918	1	1.33E-07	98	1.31E-05	Normal	
125628	125626	1.59E+09	4	10.0.0.301	10.0.0.3	0	10.0.0.18	0	1	0	8	91	6.95E+08	20	100	0	91	8918	1	1.31E-07	98	1.28E-05	Normal	
125629	125627	1.59E+09	1	10.0.0.18C	10.0.0.18	0	10.0.0.3	0	1	0	0	91	6.78E+08	20	100	0	91	8918	1	1.34E-07	98	1.32E-05	Normal	
125630	125628	1.59E+09	1	10.0.0.301	10.0.0.3	0	10.0.0.18	0	1	0	8	91	7.02E+08	20	100	0	91	8918	1	1.30E-07	98	1.27E-05	Normal	
125631	125629	1.59E+09	6	10.0.0.18C	10.0.0.18	0	10.0.0.3	0	1	0	0	91	6.91E+08	20	100	0	91	8918	1	1.32E-07	98	1.29E-05	Normal	
125632	125630	1.59E+09	6	10.0.0.301	10.0.0.3	0	10.0.0.18	0	1	0	8	91	6.95E+08	20	100	0	91	8918	1	1.31E-07	98	1.28E-05	Normal	
125633	125631	1.59E+09	1	10.0.0.15C	10.0.0.1	5050	10.0.0.3	60240	6	-1	-1	2	6.84E+08	20	100	0	42809	2814330	21304.5	6.42E-05	1407185	0.001238	Normal	
125634	125632	1.59E+09	1	10.0.0.38C	10.0.0.3	60240	10.0.0.1	5050	6	-1	-1	2	6.71E+08	20	100	0	170201	9.53E+09	85100.5	0.000254	4.76E+09	14.19951	Normal	
125635	125633	1.59E+09	1	10.0.0.18C	10.0.0.18	5050	10.0.0.3	60240	6	-1	-1	17	6.88E+08	20	100	0	157889	10479410	14165.25	0.000287	869117.5	0.0015618	Normal	
125636	125634	1.59E+09	1	10.0.0.336	10.0.0.3	36583	10.0.0.1	5050	6	-1	-1	12	6.75E+08	20	100	0	678652	8.83E+10	56137.67	0.000098	8.19E+09	56.69625	Normal	
125637	125635	1.59E+09	1	10.0.0.336	10.0.0.3	36583	10.0.0.1	5051	17	-1	-1	2	6.36E+08	20	100	0	230	347760	115	3.62E-07	173880	0.000547	Normal	
125638	125636	1.59E+09	1	0.190.18.80.180.18.8	0	10.0.0.3	0	1	0	8	5	4.22E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125639	125637	1.59E+09	1	0.221.100.0.221.100.0	0	10.0.0.3	0	1	0	8	4	7.67E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125640	125638	1.59E+09	1	0.225.172.0.229.172.0	0	10.0.0.3	0	1	0	8	6	9.86E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125641	125639	1.59E+09	1	0.252.180.0.257.180.0	0	10.0.0.3	0	1	0	8	1	5.91E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125642	125640	1.59E+09	1	0.55.18.20.0.55.18.20	0	10.0.0.3	0	1	0	8	8	5.05E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125643	125641	1.59E+09	1	10.11.0.10.10.11.0.10	0	10.0.0.3	0	1	0	8	7	3.57E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125644	125642	1.59E+09	1	100.154.7.100.154.7	0	10.0.0.3	0	1	0	8	3	6.49E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125645	125643	1.59E+09	1	100.227.2.100.227.2	0	10.0.0.3	0	1	0	8	7	4.97E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125646	125644	1.59E+09	1	100.247.1.100.247.1	0	10.0.0.3	0	1	0	8	6	4.55E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125647	125645	1.59E+09	1	100.251.8.100.251.8	0	10.0.0.3	0	1	0	8	0	4.92E+08	20	100	0	0	0	0	0	0	0	0	0	Attack
125648	125646	1.59E+09	1	100.47.10.100.47.10	0	10.0.0.3	0	1	0	8	6	6.74E+08	20	100	0	0	0	0	0	0	0	0	0	Attack

final

ReadyAccessibility: Unavailable

用 田 四

Fig 5.12

We then combined both Normal and Attack Traffic Datasets into one file which finally constitutes 2M rows worth of Data.

Role of Operators used in RapidMiner for Training and Testing Phases :

For training this model we are using RapidMiner. RapidMiner is a comprehensive data science platform with visual workflow design and full automation. It means that we don't have to do the coding for data mining tasks. RapidMiner is one of the most popular data science tools. This is the graphical user interface of the blank process in RapidMiner. It has the repository that holds our dataset. We can import our own datasets. It also offers many public datasets that we can try. We can also work with a database connection.

Retrieve original_dataset

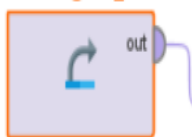


Fig 5.13

Retrieve operator:

Retrieving main dataset into the process.

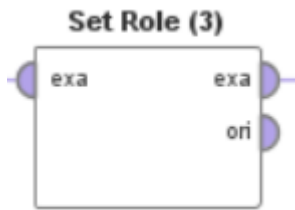


Fig 5.14

Set role operator:

Used to set the role for PKT_CLASS as label. The attribute which is to be predicted.

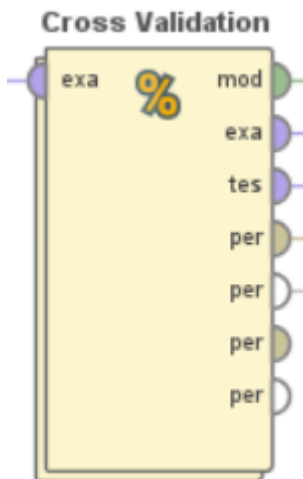


Fig 5.15

Cross validation :

Splits the process into Training and Testing for training and evaluating model's performance in training and testing phase.

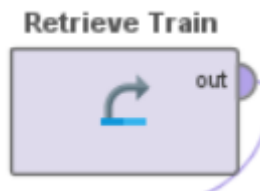


Fig 5.16

Retrieve operator:

Retrieving Train dataset into the process.

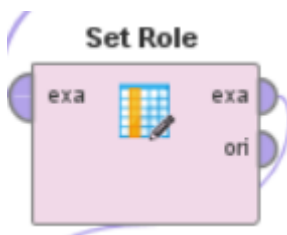


Fig 5.17

Set role operator:

Used to set the role for label as label. The attribute which is to be predicted.

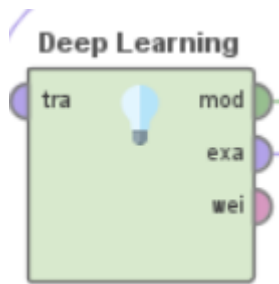


Fig 5.18

Deep learning Model operator:

Receives the input of Training Dataset where role has been applied to the attribute to be predicted and trains the Deep learning model on this dataset.



Fig 5.19

Apply Model :

The Trained model is applied using this operator so that when an unseen data is given as an input it should be able to predict the value.

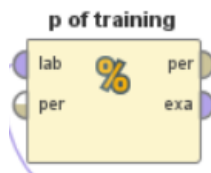


Fig 5.20

Performance (Classification) operator:

Used to determine the performance metrics of Trained Model.

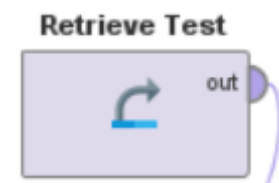


Fig 5.21

Retrieve operator:

Retrieving Test dataset into the process.

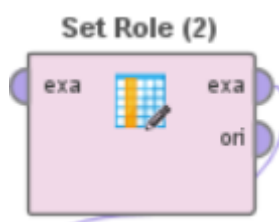


Fig 5.22

Set role operator:

Used to set the role for PKT_CLASS as label. The attribute which is to be predicted

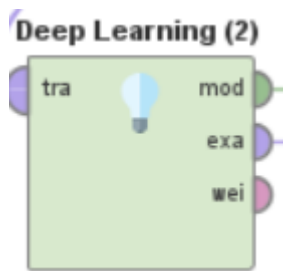


Fig 5.23

Deep learning Model operator:

Receives the input of Testing Dataset where role has been applied to the attribute to be predicted and tests the Deep learning model on this dataset.

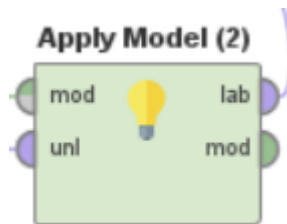


Fig5.24

Apply Model :

The Tested model is applied using this operator so that when an unseen data is given as an input it should be able to predict the value.

After this stage the model will be ready to be Deployed. Therefore, we do not need to run this entire procedure again and again. We can just give our dataset to the deployed model and it will automatically predict .

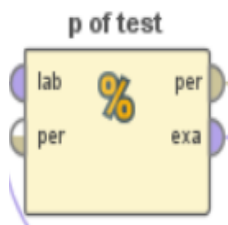


Fig 5.25

Performance (Classification) operator:

Used to determine the performance metrics of Tested Model.

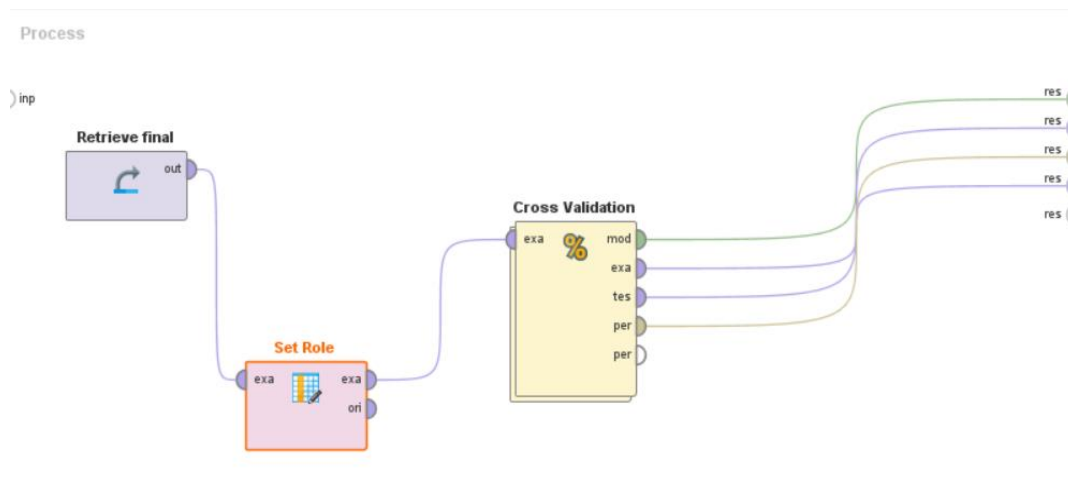


Fig 5.26

Main Process:

This is a common Process for building all the models we have evaluated in this project.

Inside cross validation we have put different models and evaluate their performance.

Evaluating Performance of SVM , Deep Learning , Random Forest , K-NN Machine Learning Models for Training Phase.

Building And evaluating performance of K-NN Model:

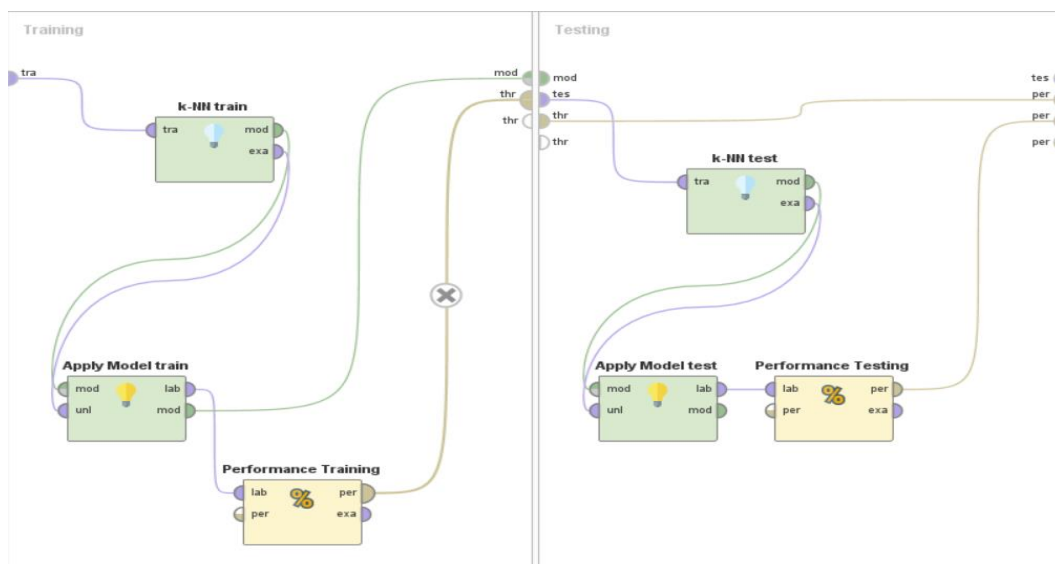


Fig 5.27

Training and Testing phase processes of K-NN model.

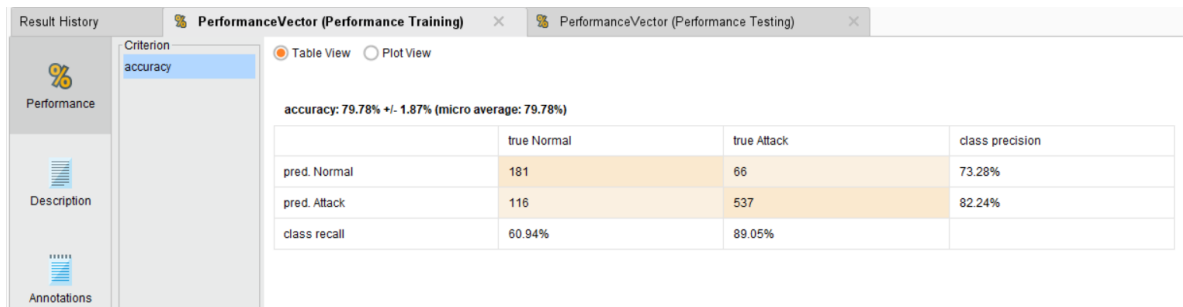


Fig 5.28

Performance vector of K-NN model while training.

Training accuracy=79.78%

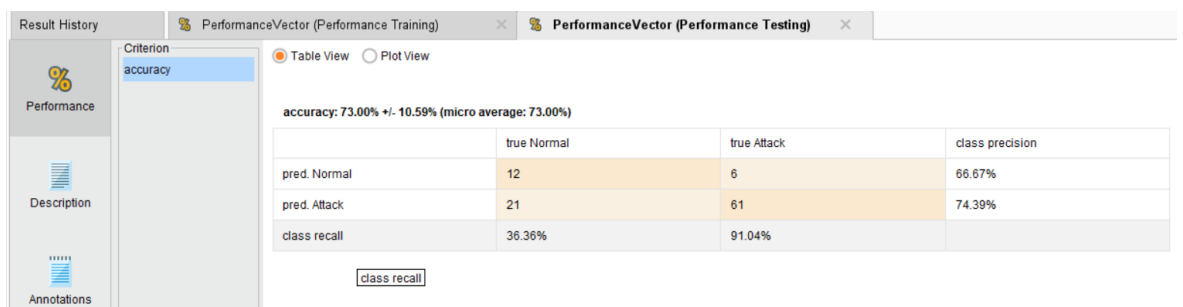


Fig 5.29

Performance vector of K-NN model while testing.

Testing accuracy=73%

Building And evaluating performance of Random Forest Model:

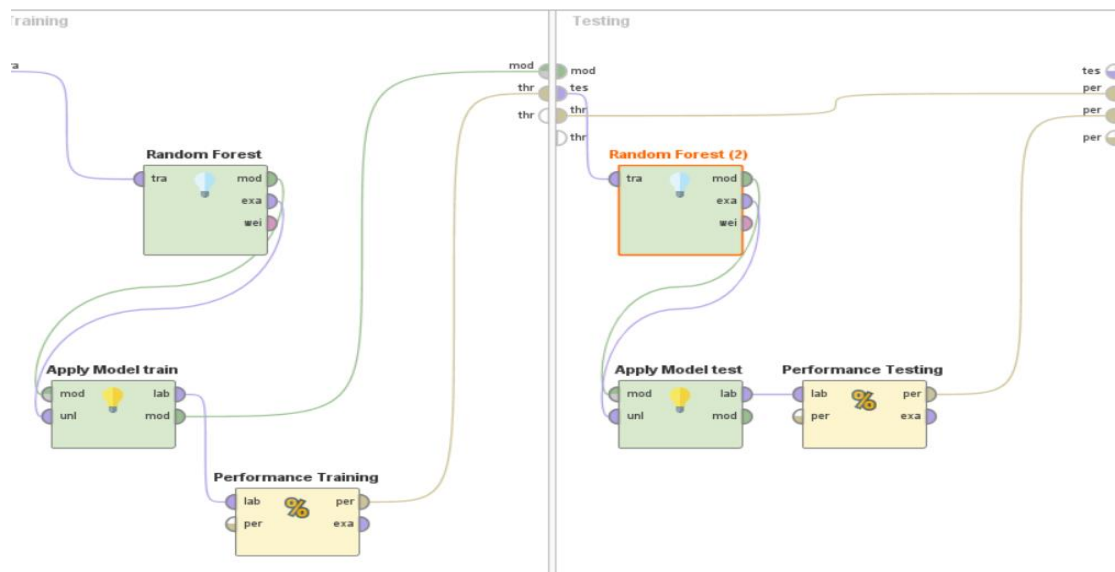




Fig 5.30


Training and Testing phase processes of Random Forest model.



Performance



Description



accuracy

accuracy: 98.42% +/- 0.00% (micro average: 98.76 %)

	true Normal	true Attack	class precision
pred. Normal	297	0	95.00%
pred. Attack	0	603	93.00%
class recall	99.00%	97.00%	

Fig 5.31

Performance vector of Random Forest model while training.

Training accuracy=98.42%

Fig 5.32

Result History

Performance

Description

Annotations

Criterion

accuracy

PerformanceVector (Performance Training)

PerformanceVector (Performance Testing)

Table View

Plot View

accuracy: 96.32 % +/- 0.00% (micro average: 96.45 %)

	true Normal	true Attack	class precision
pred. Normal	33	0	94.00%
pred. Attack	0	67	95.00%
class recall	91.00%	97.00%	

Performance vector of Random Forest model while training.

Testing accuracy=96.32%

Building And evaluating performance of Support Vector Machine Model:

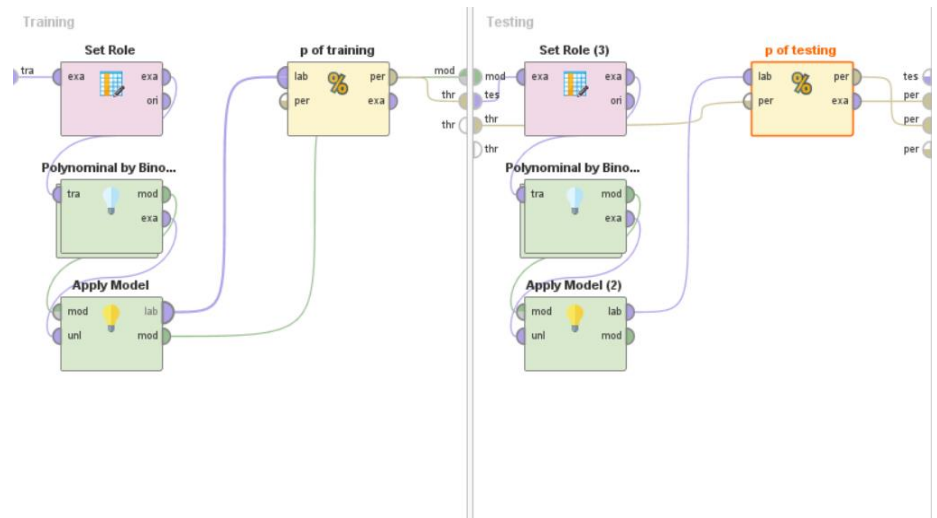


Fig 5.33

Training and Testing phase processes of SVM model.



Fig 5.34

Conversion of Nominal to Numerical values for training SVM model.

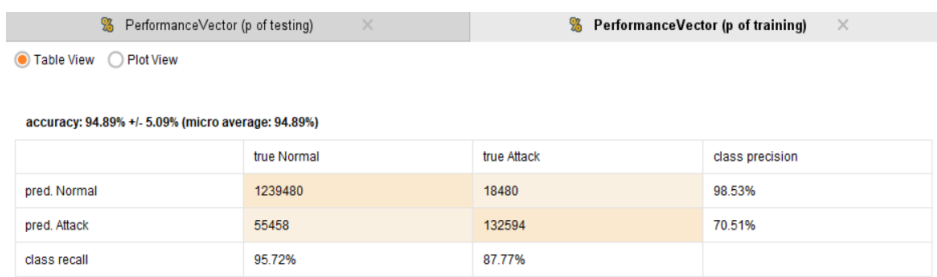


Fig 5.35

Performance vector of Support vector Machine model while training.

Training accuracy=94.89%

PerformanceVector (p of testing)

×

PerformanceVector (p of training)

×

Table View

Plot View

accuracy: 98.63% +/- 0.06% (micro average: 98.63%)

	true Normal	true Attack	class precision
pred. Normal	143836	2152	98.53%
pred. Attack	46	14634	99.69%
class recall	99.97%	87.18%	

Fig 5.35

Performance vector of Support vector Machine model while testing.

Testing accuracy=98.63%

Building And evaluating performance of Deep Learning Machine Model:

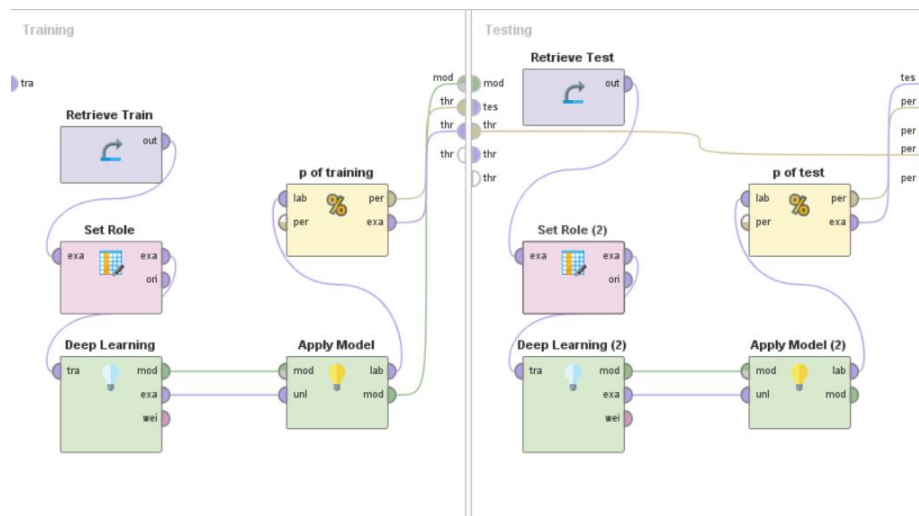


Fig 5.36

Training , Testing phase processes of Deep Learning model.

PerformanceVector (p of testing)			
<input checked="" type="radio"/> Table View <input type="radio"/> Plot View			
accuracy: 94.89% +/- 5.09% (micro average: 94.89%)			
	true Normal	true Attack	class precision
pred. Normal	1239480	18480	98.53%
pred. Attack	55458	132594	70.51%
class recall	95.72%	87.77%	

Fig 5.37

Performance vector of Deep learning model while training.

Testing accuracy=94.8%

PerformanceVector (Performance (2))			
<input checked="" type="radio"/> Table View <input type="radio"/> Plot View			
accuracy: 98.62% +/- 0.03% (micro average: 98.62%)			
	true Normal	true Attack	class precision
pred. Normal	24305	47123	34.03%
pred. Attack	0	0	0.00%
class recall	100.00%	0.00%	

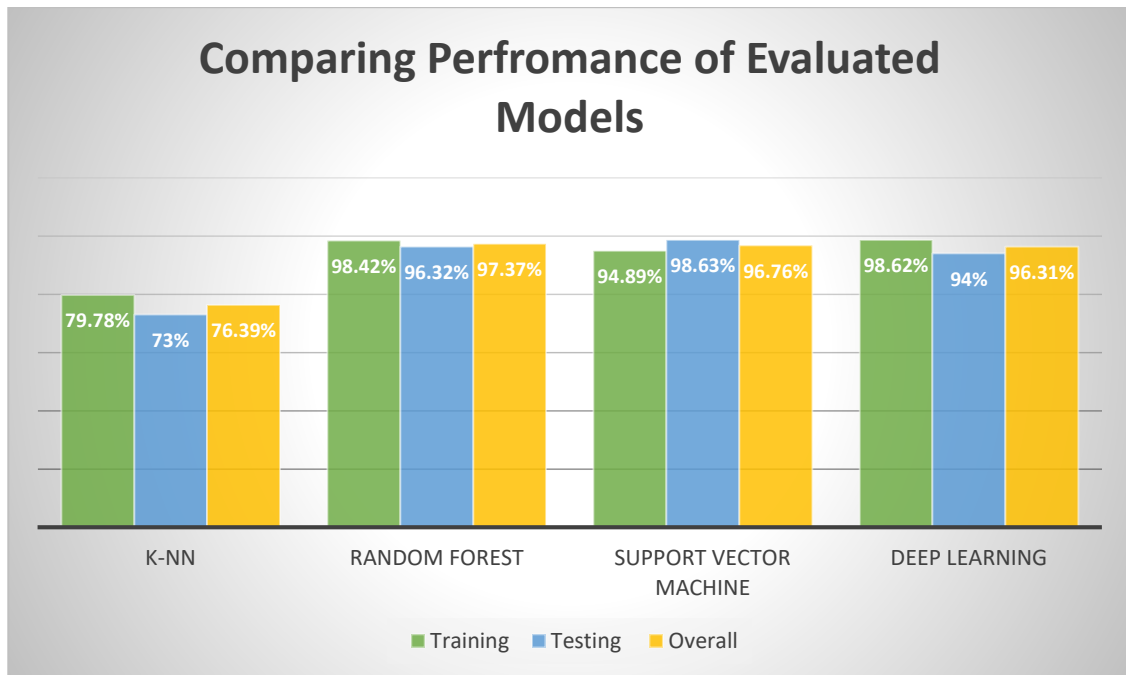
Fig 5.38

Performance vector of Deep learning model on testing.

Testing accuracy=98.62%

Same dataset was used for Training and Testing of all the models.

Analysing the performance of SVM , Deep Learning , Random Forest , K-NN Machine Learning Models.



Overall Accuracy of,

K-NN=**76.39%**

Random Forest=**97.37%**

SVM=**96.76%**

Deep learning=**96.31%**

Random forest model is the best performing among the models we have tested with overall accuracy of **97.37%**.

CHAPTER 6

SYSTEM REQUIREMENTS AND SYSTEM DESIGN

System requirement is a requirement at the system level that describes a function or the functions which the system as a whole should fulfill to satisfy the stakeholder needs and requirements. System requirements are expressed in an appropriate combination of textual statements, views, and non-functional requirements. System requirements express the levels of safety, security, reliability which will be necessary. System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements. System requirements are a broad and also narrow subject that could be implemented to many items. Whether discussing the system requirements for certain computers, software, or the business processes from a broad view point. Also, taking it down to the exact hardware or coding that runs the software. System requirements are the most effective way of meeting the user needs and reducing the cost of implementation. System requirements could cause a company to save a lot of money and time, and also can cause a company to waste money and time. They are the first and foremost important part of any project, because if the system requirements are not fulfilled, then the project is not complete. System requirements describe what the system shall do whereas the user requirements (user needs) describe what the user does with the system. System requirements are classified as either functional or non-functional requirements in terms of functionality feature of the requirement.

Functional Requirements:

Simulating the Network: Simulating the SDN Network using MiniNet and RYU controller.

Generating the dataset: Generating the dataset by simulating SDN network and simulating traffic flow of normal packets (i.e normal traffic) and DDoS attacks (i.e attack traffic).

Preprocessing: Perform data cleaning and removing noisy data from the data.

Training: Training the machine learning model by using the generated dataset.

Deploying the model: Trained model is deployed on to the SDN for monitoring the traffic flow which detects the DDoS attacks.

Non-Functional Requirements:

Usability: The system is designed as an automated process hence there is not much user intervention

Performance: The system is developed with high level languages and uses algorithms of the best accuracy

Reliability: The coding language used to develop the system is very reliable

Readability: The code is written in python which makes it easier to read and understand.

Scalability: The system is designed to work for data of any size and thus has the ability to handle increasing loads

Robustness: The system identifies erroneous or invalid data input with the help of standardization algorithm.

Hardware Requirements

Selection of hardware also plays an important role in existence and performance of any software.

The size and capacity are main requirements.

1.**processor** : intel i5 & above

2.**RAM**: 8 GB/16 GB

Software Requirements

The software requirements specification is produced at the end of the analysis task. Software

Requirements is a difficult task, for developing the application. 1. Operating System:

1. Kali Linux version 20.04

2. Coding language: Python 3.8.1

3. Mininet version 2.3 for creating SDN network.

4. RYU manager version 4.3 to manage traffic flow.

SYSTEM DESIGN

System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system.

The purpose of the System Design process is to provide sufficient detailed data and information about the system and its system elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture.

Elements of a System

- **Architecture** - This is the conceptual model that defines the structure, behavior and more views of a system. We can use flowcharts to represent and illustrate the architecture.
- **Modules** - These are components that handle one specific task in a system. A combination of the modules make up the system.
- **Components** - This provides a particular function or group of related functions. They are made up of modules.
- **Interfaces** - This is the shared boundary across which the components of a system exchange information and relate.
- **Data** - This is the management of the information and data flow

CHAPTER 7

UML DIAGRAMS

Flow diagram for Training process.

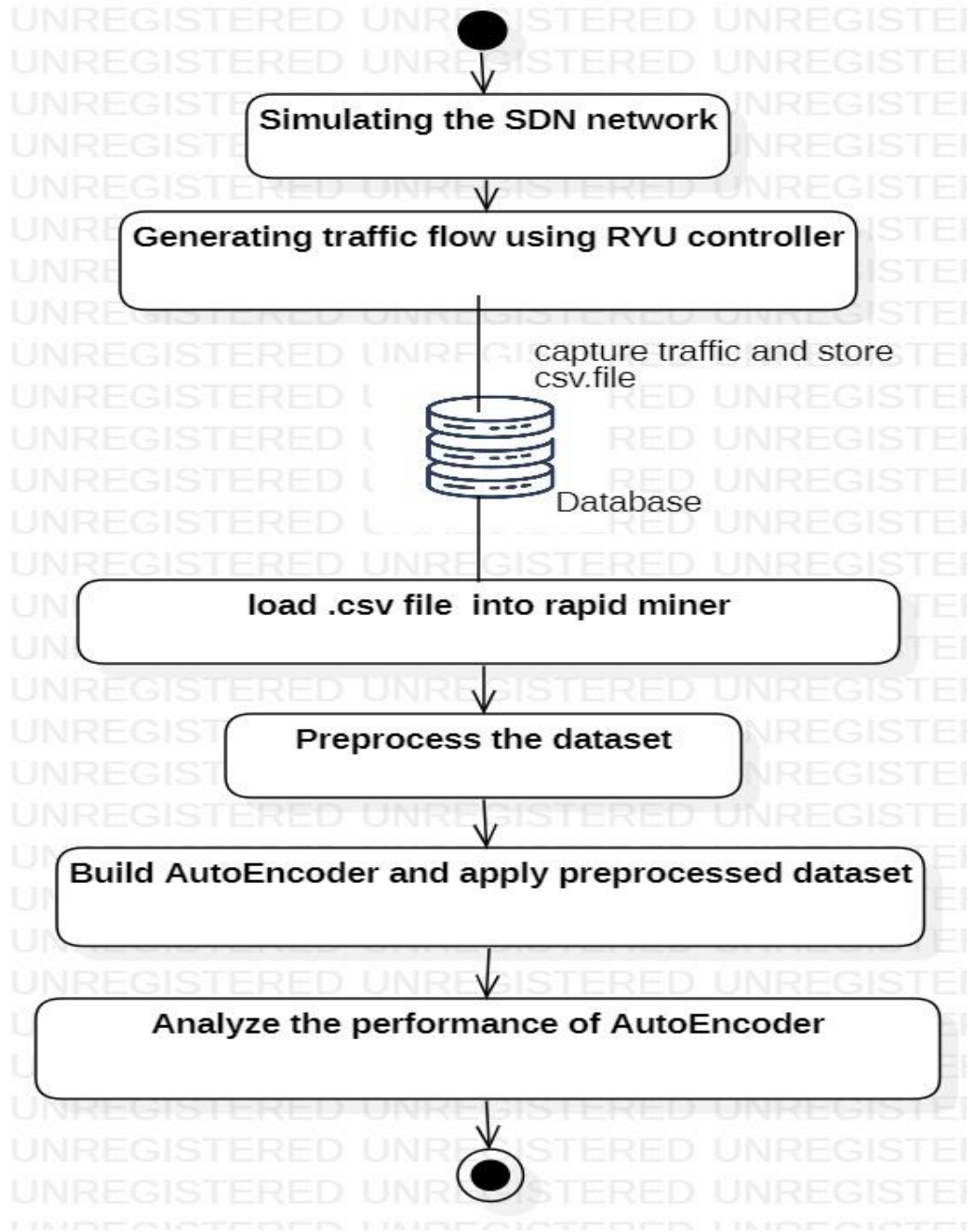


Fig 7.1

Flow diagram for testing and categorizing the data

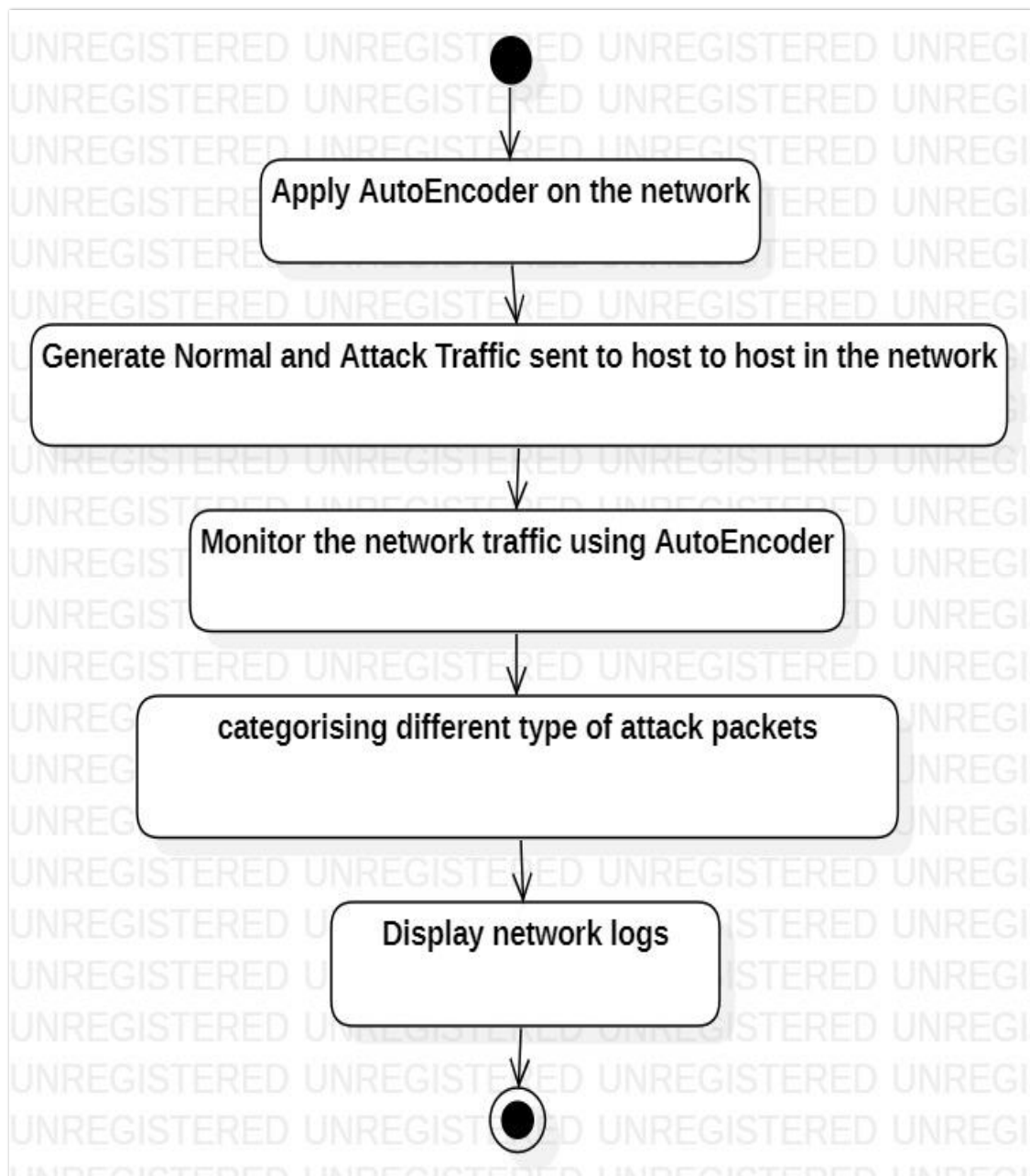


Fig 7.2

CHAPTER 8

CODE

Code for enabling switches:

Common file for all other files(i.e codes).

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
```

```
from ryu.lib.packet import in_proto
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
```

```
class SimpleSwitch13(app_manager.RyuApp):
```

```
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
```

```
    def __init__(self, *args, **kwargs):
```

```
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
```

```
        self.mac_to_port = { }
```

```
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
```

```
    def switch_features_handler(self, ev):
```

```
        datapath = ev.msg.datapath
```

```
        ofproto = datapath.ofproto
```

```
        parser = datapath.ofproto_parser
```

```
        match = parser.OFPMatch()
```

```

actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                   ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions, buffer_id=None, idle=0, hard=0):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                idle_timeout=idle, hard_timeout=hard,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                idle_timeout=idle, hard_timeout=hard,
                                match=match, instructions=inst)

    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

```

```

in_port = msg.match['in_port']

pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ethernet.ethernet)[0]

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    return
dst = eth.dst
src = eth.src

dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

actions = [parser.OFPActionOutput(out_port)]

if out_port != ofproto.OFPP_FLOOD:

    if eth.ethertype == ether_types.ETH_TYPE_IP:
        ip = pkt.get_protocol(ipv4.ipv4)
        srcip = ip.src
        dstip = ip.dst
        protocol = ip.proto

        if protocol == in_proto.IPPROTO_ICMP:

```



```

t = pkt.get_protocol(icmp.icmp)
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                        ipv4_src=srcip, ipv4_dst=dstip,
                        ip_proto=protocol, icmpv4_code=t.code,
                        icmpv4_type=t.type)

elif protocol == in_proto.IPPROTO_TCP:
    t = pkt.get_protocol(tcp.tcp)
    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                            ipv4_src=srcip, ipv4_dst=dstip,
                            ip_proto=protocol,
                            tcp_src=t.src_port, tcp_dst=t.dst_port,)

elif protocol == in_proto.IPPROTO_UDP:
    u = pkt.get_protocol(udp.udp)
    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                            ipv4_src=srcip, ipv4_dst=dstip,
                            ip_proto=protocol,
                            udp_src=u.src_port, udp_dst=u.dst_port,)

if msg.buffer_id != ofproto.OFP_NO_BUFFER:
    self.add_flow(datapath, 1, match, actions, msg.buffer_id, idle=20, hard=100)
    return
else:
    self.add_flow(datapath, 1, match, actions, idle=20, hard=100)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                           in_port=in_port, actions=actions, data=data)

```

```
datapath.send_msg(out)
```

Code to generate normal traffic:

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.node import OVSKernelSwitch, RemoteController
from time import sleep

from datetime import datetime
from random import randrange, choice

class MyTopo( Topo ):

    def build( self ):

        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h1 = self.addHost( 'h1', cpu=1.0/20, mac="00:00:00:00:00:01", ip="10.0.0.1/24" )
        h2 = self.addHost( 'h2', cpu=1.0/20, mac="00:00:00:00:00:02", ip="10.0.0.2/24" )
        h3 = self.addHost( 'h3', cpu=1.0/20, mac="00:00:00:00:00:03", ip="10.0.0.3/24" )

        s2 = self.addSwitch( 's2', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h4 = self.addHost( 'h4', cpu=1.0/20, mac="00:00:00:00:00:04", ip="10.0.0.4/24" )
        h5 = self.addHost( 'h5', cpu=1.0/20, mac="00:00:00:00:00:05", ip="10.0.0.5/24" )
        h6 = self.addHost( 'h6', cpu=1.0/20, mac="00:00:00:00:00:06", ip="10.0.0.6/24" )

        s3 = self.addSwitch( 's3', cls=OVSKernelSwitch, protocols='OpenFlow13' )
```

```

h7 = self.addHost( 'h7', cpu=1.0/20, mac="00:00:00:00:00:07", ip="10.0.0.7/24" )
h8 = self.addHost( 'h8', cpu=1.0/20, mac="00:00:00:00:00:08", ip="10.0.0.8/24" )
h9 = self.addHost( 'h9', cpu=1.0/20, mac="00:00:00:00:00:09", ip="10.0.0.9/24" )

s4 = self.addSwitch( 's4', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h10 = self.addHost( 'h10', cpu=1.0/20, mac="00:00:00:00:00:10", ip="10.0.0.10/24"
)
h11 = self.addHost( 'h11', cpu=1.0/20, mac="00:00:00:00:00:11", ip="10.0.0.11/24"
)
h12 = self.addHost( 'h12', cpu=1.0/20, mac="00:00:00:00:00:12", ip="10.0.0.12/24"
)

s5 = self.addSwitch( 's5', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h13 = self.addHost( 'h13', cpu=1.0/20, mac="00:00:00:00:00:13", ip="10.0.0.13/24"
)
h14 = self.addHost( 'h14', cpu=1.0/20, mac="00:00:00:00:00:14", ip="10.0.0.14/24"
)
h15 = self.addHost( 'h15', cpu=1.0/20, mac="00:00:00:00:00:15", ip="10.0.0.15/24"
)

s6 = self.addSwitch( 's6', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h16 = self.addHost( 'h16', cpu=1.0/20, mac="00:00:00:00:00:16", ip="10.0.0.16/24"
)
h17 = self.addHost( 'h17', cpu=1.0/20, mac="00:00:00:00:00:17", ip="10.0.0.17/24"
)
h18 = self.addHost( 'h18', cpu=1.0/20, mac="00:00:00:00:00:18", ip="10.0.0.18/24"
)

```

```
# Add links
```

```
self.addLink( h1, s1 )
```

```
self.addLink( h2, s1 )
```

```
self.addLink( h3, s1 )
```

```
self.addLink( h4, s2 )
```

```
self.addLink( h5, s2 )
```

```
self.addLink( h6, s2 )
```

```
self.addLink( h7, s3 )
```

```
self.addLink( h8, s3 )
```

```
self.addLink( h9, s3 )
```

```
self.addLink( h10, s4 )
```

```
self.addLink( h11, s4 )
```

```
self.addLink( h12, s4 )
```

```
self.addLink( h13, s5 )
```

```
self.addLink( h14, s5 )
```

```
self.addLink( h15, s5 )
```

```
self.addLink( h16, s6 )
```

```
self.addLink( h17, s6 )
```

```
self.addLink( h18, s6 )
```

```
self.addLink( s1, s2 )
```

```
self.addLink( s2, s3 )
```

```
self.addLink( s3, s4 )
```

```
self.addLink( s4, s5 )
```

```
self.addLink( s5, s6 )
```

```

def ip_generator():

    ip = ".".join(["10","0","0",str(randrange(1,19))])
    return ip

def startNetwork():
    topo = MyTopo()

    c0 = RemoteController('c0', ip='10.0.2.15', port=6653)
    net = Mininet(topo=topo, link=TCLink, controller=c0)

    net.start()

    h1 = net.get('h1')
    h2 = net.get('h2')
    h3 = net.get('h3')
    h4 = net.get('h4')
    h5 = net.get('h5')
    h6 = net.get('h6')
    h7 = net.get('h7')
    h8 = net.get('h8')
    h9 = net.get('h9')
    h10 = net.get('h10')
    h11 = net.get('h11')
    h12 = net.get('h12')
    h13 = net.get('h13')
    h14 = net.get('h14')
    h15 = net.get('h15')
    h16 = net.get('h16')
    h17 = net.get('h17')

```

```

h18 = net.get('h18')

hosts = [h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15, h16, h17,
h18]

print("-----")
print("Generating traffic ...")
h1.cmd('cd /home/mininet/webserver')
h1.cmd('python -m SimpleHTTPServer 80 &')
h1.cmd('iperf -s -p 5050 &')
h1.cmd('iperf -s -u -p 5051 &')
sleep(2)
for h in hosts:
    h.cmd('cd /home/mininet/Downloads')
for i in range(600):

    print("-----")
    print("Iteration n { } ...".format(i+1))
    print("-----")

    for j in range(10):
        src = choice(hosts)
        dst = ip_generator()

        if j < 9:
            print("generating ICMP traffic between %s and h%s and TCP/UDP traffic
between %s and h1" % (src,((dst.split('.')[3]),src))
            src.cmd("ping { } -c 100 &".format(dst))
            src.cmd("iperf -p 5050 -c 10.0.0.1")
            src.cmd("iperf -p 5051 -u -c 10.0.0.1")
        else:

```

```

        print("generating ICMP traffic between %s and h%s and TCP/UDP traffic
between %s and h1" % (src,((dst.split('.')[3]),src))
        src.cmd("ping { } -c 100".format(dst))
        src.cmd("iperf -p 5050 -c 10.0.0.1")
        src.cmd("iperf -p 5051 -u -c 10.0.0.1")

    print("%s Downloading index.html from h1" % src)
    src.cmd("wget http://10.0.0.1/index.html")
    print("%s Downloading test.zip from h1" % src)
    src.cmd("wget http://10.0.0.1/test.zip")

    h1.cmd("rm -f *.* /home/mininet/Downloads")

    print("-----")

    # CLI(net)
    net.stop()

if __name__ == '__main__':

    start = datetime.now()

    setLogLevel( 'info' )
    startNetwork()

    end = datetime.now()

    print(end-start)

```

Code to collect normal traffic:

```
import switch

from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

from datetime import datetime

class CollectTrainingStatsApp(switch.SimpleSwitch13):
    def __init__(self, *args, **kwargs):
        super(CollectTrainingStatsApp, self).__init__(*args, **kwargs)
        self.datapaths = { }
        self.monitor_thread = hub.spawn(self.monitor)

        file0 = open("FlowStatsfile.csv", "w")
        file0.write('timestamp,datapath_id,flow_id,ip_src,tp_src,ip_dst,tp_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,flow_duration_nsec,idle_timeout,hard_timeout,flags,packet_count,byte_count,packet_count_per_second,packet_count_per_nsecond,byte_count_per_second,byte_count_per_nsecond,label\n')
        file0.close()

        @set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
        def state_change_handler(self, ev):
            datapath = ev.datapath
            if ev.state == MAIN_DISPATCHER:
                if datapath.id not in self.datapaths:
```



```

        self.logger.debug('register datapath: %016x', datapath.id)
        self.datapaths[datapath.id] = datapath

    elif ev.state == DEAD_DISPATCHER:
        if datapath.id in self.datapaths:
            self.logger.debug('unregister datapath: %016x', datapath.id)
            del self.datapaths[datapath.id]

    def monitor(self):
        while True:
            for dp in self.datapaths.values():
                self.request_stats(dp)
            hub.sleep(10)

    def request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)

        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):

        timestamp = datetime.now()
        timestamp = timestamp.timestamp()
        icmp_code = -1
        icmp_type = -1
        tp_src = 0
        tp_dst = 0

```

```

file0 = open("Normal.csv","a+")
body = ev.msg.body
for stat in sorted([flow for flow in body if (flow.priority == 1) ], key=lambda flow:
    (flow.match['eth_type'],flow.match['ipv4_src'],flow.match['ipv4_dst'],flow.matc
h['ip_proto']))):

```

```

    ip_src = stat.match['ipv4_src']
    ip_dst = stat.match['ipv4_dst']
    ip_proto = stat.match['ip_proto']

```

```

    if stat.match['ip_proto'] == 1:
        icmp_code = stat.match['icmpv4_code']
        icmp_type = stat.match['icmpv4_type']

```

```

    elif stat.match['ip_proto'] == 6:
        tp_src = stat.match['tcp_src']
        tp_dst = stat.match['tcp_dst']

```

```

    elif stat.match['ip_proto'] == 17:
        tp_src = stat.match['udp_src']
        tp_dst = stat.match['udp_dst']

```

```

    flow_id = str(ip_src) + str(tp_src) + str(ip_dst) + str(tp_dst) + str(ip_proto)

```

```

    try:
        packet_count_per_second = stat.packet_count/stat.duration_sec
        packet_count_per_nsecond = stat.packet_count/stat.duration_nsec
    except:
        packet_count_per_second = 0

```

```

        packet_count_per_nsecond = 0

    try:
        byte_count_per_second = stat.byte_count/stat.duration_sec
        byte_count_per_nsecond = stat.byte_count/stat.duration_nsec
    except:
        byte_count_per_second = 0
        byte_count_per_nsecond = 0

file0.write("{} {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {} \n"
            .format(timestamp, ev.msg.datapath.id, flow_id, ip_src, tp_src, ip_dst, tp_dst,
                    stat.match['ip_proto'], icmp_code, icmp_type,
                    stat.duration_sec, stat.duration_nsec,
                    stat.idle_timeout, stat.hard_timeout,
                    stat.flags, stat.packet_count, stat.byte_count,
                    packet_count_per_second, packet_count_per_nsecond,
                    byte_count_per_second, byte_count_per_nsecond, 0))
file0.close()

```

Code to generate DDoS traffic:

```

from mininet.topo import Topo
from mininet.net import Mininet
# from mininet.node import CPULimitedHost
from mininet.link import TCLink
# from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
# from mininet.cli import CLI
from mininet.node import OVSKernelSwitch, RemoteController
from time import sleep

```

```

from datetime import datetime
from random import randrange, choice

class MyTopo( Topo ):

    def build( self ):

        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h1 = self.addHost( 'h1', cpu=1.0/20, mac="00:00:00:00:00:01", ip="10.0.0.1/24" )
        h2 = self.addHost( 'h2', cpu=1.0/20, mac="00:00:00:00:00:02", ip="10.0.0.2/24" )
        h3 = self.addHost( 'h3', cpu=1.0/20, mac="00:00:00:00:00:03", ip="10.0.0.3/24" )

        s2 = self.addSwitch( 's2', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h4 = self.addHost( 'h4', cpu=1.0/20, mac="00:00:00:00:00:04", ip="10.0.0.4/24" )
        h5 = self.addHost( 'h5', cpu=1.0/20, mac="00:00:00:00:00:05", ip="10.0.0.5/24" )
        h6 = self.addHost( 'h6', cpu=1.0/20, mac="00:00:00:00:00:06", ip="10.0.0.6/24" )

        s3 = self.addSwitch( 's3', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h7 = self.addHost( 'h7', cpu=1.0/20, mac="00:00:00:00:00:07", ip="10.0.0.7/24" )
        h8 = self.addHost( 'h8', cpu=1.0/20, mac="00:00:00:00:00:08", ip="10.0.0.8/24" )
        h9 = self.addHost( 'h9', cpu=1.0/20, mac="00:00:00:00:00:09", ip="10.0.0.9/24" )

        s4 = self.addSwitch( 's4', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h10 = self.addHost( 'h10', cpu=1.0/20, mac="00:00:00:00:00:10", ip="10.0.0.10/24"
)

```

```

    h11 = self.addHost( 'h11', cpu=1.0/20, mac="00:00:00:00:00:11", ip="10.0.0.11/24"
)
    h12 = self.addHost( 'h12', cpu=1.0/20, mac="00:00:00:00:00:12", ip="10.0.0.12/24"
)

    s5 = self.addSwitch( 's5', cls=OVSKernelSwitch, protocols='OpenFlow13' )

    h13 = self.addHost( 'h13', cpu=1.0/20, mac="00:00:00:00:00:13", ip="10.0.0.13/24"
)
    h14 = self.addHost( 'h14', cpu=1.0/20, mac="00:00:00:00:00:14", ip="10.0.0.14/24"
)
    h15 = self.addHost( 'h15', cpu=1.0/20, mac="00:00:00:00:00:15", ip="10.0.0.15/24"
)

    s6 = self.addSwitch( 's6', cls=OVSKernelSwitch, protocols='OpenFlow13' )

    h16 = self.addHost( 'h16', cpu=1.0/20, mac="00:00:00:00:00:16", ip="10.0.0.16/24"
)
    h17 = self.addHost( 'h17', cpu=1.0/20, mac="00:00:00:00:00:17", ip="10.0.0.17/24"
)
    h18 = self.addHost( 'h18', cpu=1.0/20, mac="00:00:00:00:00:18", ip="10.0.0.18/24"
)

    self.addLink( h1, s1 )
    self.addLink( h2, s1 )
    self.addLink( h3, s1 )

    self.addLink( h4, s2 )
    self.addLink( h5, s2 )
    self.addLink( h6, s2 )

```

```

self.addLink( h7, s3 )
self.addLink( h8, s3 )
self.addLink( h9, s3 )

self.addLink( h10, s4 )
self.addLink( h11, s4 )
self.addLink( h12, s4 )

self.addLink( h13, s5 )
self.addLink( h14, s5 )
self.addLink( h15, s5 )

self.addLink( h16, s6 )
self.addLink( h17, s6 )
self.addLink( h18, s6 )

self.addLink( s1, s2 )
self.addLink( s2, s3 )
self.addLink( s3, s4 )
self.addLink( s4, s5 )
self.addLink( s5, s6 )
def ip_generator():

    ip = ".".join(["10","0","0",str(randrange(1,19))])
    return ip

def startNetwork():
    topo = MyTopo()

    c0 = RemoteController('c0', ip='10.0.2.15', port=6653)
    net = Mininet(topo=topo, link=TCLink, controller=c0)

```

```

net.start()

h1 = net.get('h1')
h2 = net.get('h2')
h3 = net.get('h3')
h4 = net.get('h4')
h5 = net.get('h5')
h6 = net.get('h6')
h7 = net.get('h7')
h8 = net.get('h8')
h9 = net.get('h9')
h10 = net.get('h10')
h11 = net.get('h11')
h12 = net.get('h12')
h13 = net.get('h13')
h14 = net.get('h14')
h15 = net.get('h15')
h16 = net.get('h16')
h17 = net.get('h17')
h18 = net.get('h18')

hosts = [h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15, h16, h17, h18]

h1.cmd('cd /home/mininet/webserver')
h1.cmd('python -m SimpleHTTPServer 80 &')

src = choice(hosts)
dst = ip_generator()
print("-----")
print("Performing Smurf")

```

```

print("-----")
    src.cmd("timeout 20s hping3 -1 -V -d 120 -w 64 -p 80 --rand-source --flood
{ }".format(dst))
    sleep(100)

src = choice(hosts)
dst = ip_generator()
print("-----")
print("Performing UDP-Flood")
print("-----")
    src.cmd("timeout 20s hping3 -2 -V -d 120 -w 64 --rand-source --flood
{ }".format(dst))
    sleep(100)

src = choice(hosts)
dst = ip_generator()
print("-----")
print("Performing SIDDOS")
print("-----")
    src.cmd('timeout 20s hping3 -S -V -d 120 -w 64 -p 80 --rand-source --flood 10.0.0.1')
    sleep(100)

src = choice(hosts)
dst = ip_generator()
print("-----")
print("Performing HTTP-FLOOD")
print("-----")
    src.cmd("timeout 20s hping3 -1 -V -d 120 -w 64 --flood -a { } { }".format(dst,dst))
    sleep(100)
print("-----")

```



```

# CLI(net)
net.stop()

if __name__ == '__main__':

    start = datetime.now()

    setLogLevel( 'info' )
    startNetwork()

    end = datetime.now()

    print(end-start)

```

Code to collect DDoS traffic:

```

import switch
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

from datetime import datetime

class CollectTrainingStatsApp(switch.SimpleSwitch13):
    def __init__(self, *args, **kwargs):
        super(CollectTrainingStatsApp, self).__init__(*args, **kwargs)
        self.datapaths = { }
        self.monitor_thread = hub.spawn(self.monitor)

```

```

        @set_ev_cls(ofp_event.EventOFPSStateChange,[MAIN_DISPATCHER,
DEAD_DISPATCHER])

def state_change_handler(self, ev):
    datapath = ev.datapath
    if ev.state == MAIN_DISPATCHER:
        if datapath.id not in self.datapaths:
            self.logger.debug('register datapath: %016x', datapath.id)
            self.datapaths[datapath.id] = datapath

    elif ev.state == DEAD_DISPATCHER:
        if datapath.id in self.datapaths:
            self.logger.debug('unregister datapath: %016x', datapath.id)
            del self.datapaths[datapath.id]

def monitor(self):
    while True:
        for dp in self.datapaths.values():
            self.request_stats(dp)
            hub.sleep(10)

def request_stats(self, datapath):
    self.logger.debug('send stats request: %016x', datapath.id)

    parser = datapath.ofproto_parser

    req = parser.OFPFlowStatsRequest(datapath)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):

```

```

timestamp = datetime.now()
timestamp = timestamp.timestamp()
icmp_code = -1
icmp_type = -1
tp_src = 0
tp_dst = 0

file0 = open("Attacks.csv","a+")

body = ev.msg.body
for stat in sorted([flow for flow in body if (flow.priority == 1) ], key=lambda flow:
    (flow.match['eth_type'],flow.match['ipv4_src'],flow.match['ipv4_dst'],flow.matc
h['ip_proto'])):

    ip_src = stat.match['ipv4_src']
    ip_dst = stat.match['ipv4_dst']
    ip_proto = stat.match['ip_proto']

    if stat.match['ip_proto'] == 1:
        icmp_code = stat.match['icmpv4_code']
        icmp_type = stat.match['icmpv4_type']

    elif stat.match['ip_proto'] == 6:
        tp_src = stat.match['tcp_src']
        tp_dst = stat.match['tcp_dst']

    elif stat.match['ip_proto'] == 17:
        tp_src = stat.match['udp_src']
        tp_dst = stat.match['udp_dst']

    flow_id = str(ip_src) + str(tp_src) + str(ip_dst) + str(tp_dst) + str(ip_proto)

```

```

try:
    packet_count_per_second = stat.packet_count/stat.duration_sec
    packet_count_per_nsecond = stat.packet_count/stat.duration_nsec
except:
    packet_count_per_second = 0
    packet_count_per_nsecond = 0

try:
    byte_count_per_second = stat.byte_count/stat.duration_sec
    byte_count_per_nsecond = stat.byte_count/stat.duration_nsec
except:
    byte_count_per_second = 0
    byte_count_per_nsecond = 0

file0.write("{} {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {} \n"
            .format(timestamp, ev.msg.datapath.id, flow_id, ip_src, tp_src, ip_dst, tp_dst,
                    stat.match['ip_proto'], icmp_code, icmp_type,
                    stat.duration_sec, stat.duration_nsec,
                    stat.idle_timeout, stat.hard_timeout,
                    stat.flags, stat.packet_count, stat.byte_count,
                    packet_count_per_second, packet_count_per_nsecond,
                    byte_count_per_second, byte_count_per_nsecond, 1))
file0.close()

```

Code to create SDN network:

```

from mininet.topo import Topo
from mininet.net import Mininet
# from mininet.node import CPULimitedHost

```

```

from mininet.link import TCLink
# from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import OVSKernelSwitch, RemoteController
# from time import sleep
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
class MyTopo( Topo ):

    def build( self ):

        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h1 = self.addHost( 'h1', cpu=1.0/20, mac="00:00:00:00:00:01", ip="10.0.0.1/24" )
        h2 = self.addHost( 'h2', cpu=1.0/20, mac="00:00:00:00:00:02", ip="10.0.0.2/24" )
        h3 = self.addHost( 'h3', cpu=1.0/20, mac="00:00:00:00:00:03", ip="10.0.0.3/24" )

        s2 = self.addSwitch( 's2', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h4 = self.addHost( 'h4', cpu=1.0/20, mac="00:00:00:00:00:04", ip="10.0.0.4/24" )
        h5 = self.addHost( 'h5', cpu=1.0/20, mac="00:00:00:00:00:05", ip="10.0.0.5/24" )
        h6 = self.addHost( 'h6', cpu=1.0/20, mac="00:00:00:00:00:06", ip="10.0.0.6/24" )

        s3 = self.addSwitch( 's3', cls=OVSKernelSwitch, protocols='OpenFlow13' )

        h7 = self.addHost( 'h7', cpu=1.0/20, mac="00:00:00:00:00:07", ip="10.0.0.7/24" )
        h8 = self.addHost( 'h8', cpu=1.0/20, mac="00:00:00:00:00:08", ip="10.0.0.8/24" )
        h9 = self.addHost( 'h9', cpu=1.0/20, mac="00:00:00:00:00:09", ip="10.0.0.9/24" )

```

```

s4 = self.addSwitch( 's4', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h10 = self.addHost( 'h10', cpu=1.0/20, mac="00:00:00:00:00:10", ip="10.0.0.10/24"
)
h11 = self.addHost( 'h11', cpu=1.0/20, mac="00:00:00:00:00:11", ip="10.0.0.11/24"
)
h12 = self.addHost( 'h12', cpu=1.0/20, mac="00:00:00:00:00:12", ip="10.0.0.12/24"
)

s5 = self.addSwitch( 's5', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h13 = self.addHost( 'h13', cpu=1.0/20, mac="00:00:00:00:00:13", ip="10.0.0.13/24"
)
h14 = self.addHost( 'h14', cpu=1.0/20, mac="00:00:00:00:00:14", ip="10.0.0.14/24"
)
h15 = self.addHost( 'h15', cpu=1.0/20, mac="00:00:00:00:00:15", ip="10.0.0.15/24"
)

s6 = self.addSwitch( 's6', cls=OVSKernelSwitch, protocols='OpenFlow13' )

h16 = self.addHost( 'h16', cpu=1.0/20, mac="00:00:00:00:00:16", ip="10.0.0.16/24"
)
h17 = self.addHost( 'h17', cpu=1.0/20, mac="00:00:00:00:00:17", ip="10.0.0.17/24"
)
h18 = self.addHost( 'h18', cpu=1.0/20, mac="00:00:00:00:00:18", ip="10.0.0.18/24"
)

# Add links

self.addLink( h1, s1 )
self.addLink( h2, s1 )

```

```
self.addLink( h3, s1 )
```

```
self.addLink( h4, s2 )
```

```
self.addLink( h5, s2 )
```

```
self.addLink( h6, s2 )
```

```
self.addLink( h7, s3 )
```

```
self.addLink( h8, s3 )
```

```
self.addLink( h9, s3 )
```

```
self.addLink( h10, s4 )
```

```
self.addLink( h11, s4 )
```

```
self.addLink( h12, s4 )
```

```
self.addLink( h13, s5 )
```

```
self.addLink( h14, s5 )
```

```
self.addLink( h15, s5 )
```

```
self.addLink( h16, s6 )
```

```
self.addLink( h17, s6 )
```

```
self.addLink( h18, s6 )
```

```
self.addLink( s1, s2 )
```

```
self.addLink( s2, s3 )
```

```
self.addLink( s3, s4 )
```

```
self.addLink( s4, s5 )
```

```
self.addLink( s5, s6 )
```

```
def startNetwork():
```

```
    topo = MyTopo()
```

```

c0 = RemoteController('c0', ip='10.0.2.15', port=6653)
net = Mininet(topo=topo, link=TCLink, controller=c0)

net.start()
CLI(net)
net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    startNetwork()

```

Code to detect real-time traffic:

```

from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub

import switch
from datetime import datetime

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

class SimpleMonitor13(switch.SimpleSwitch13):

```



```

def __init__(self, *args, **kwargs):

    super(SimpleMonitor13, self).__init__(*args, **kwargs)
    self.datapaths = { }
    self.monitor_thread = hub.spawn(self._monitor)

    start = datetime.now()

    self.flow_training()

    end = datetime.now()
    print("Training time: ", (end-start))

    @set_ev_cls(ofp_event.EventOFPSStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
            elif ev.state == DEAD_DISPATCHER:
                if datapath.id in self.datapaths:
                    self.logger.debug('unregister datapath: %016x', datapath.id)
                    del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

```

```

        self.flow_predict()

def _request_stats(self, datapath):
    self.logger.debug('send stats request: %016x', datapath.id)
    parser = datapath.ofproto_parser

    req = parser.OFPFlowStatsRequest(datapath)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):

    timestamp = datetime.now()
    timestamp = timestamp.timestamp()

    file0 = open("Predict.csv", "w")
    file0.write('timestamp,datapath_id,flow_id,ip_src,tp_src,ip_dst,tp_dst,ip_proto,icmp_code,icmp_type,flow_duration_sec,flow_duration_nsec,idle_timeout,hard_timeout,flags,packet_count,byte_count,packet_count_per_second,packet_count_per_nsecond,byte_count_per_second,byte_count_per_nsecond\n')
    body = ev.msg.body
    icmp_code = -1
    icmp_type = -1
    tp_src = 0
    tp_dst = 0

    for stat in sorted([flow for flow in body if (flow.priority == 1) ], key=lambda flow:
        (flow.match['eth_type'],flow.match['ipv4_src'],flow.match['ipv4_dst'],flow.match['ip_proto'])):

```

```

ip_src = stat.match['ipv4_src']
ip_dst = stat.match['ipv4_dst']
ip_proto = stat.match['ip_proto']

if stat.match['ip_proto'] == 1:
    icmp_code = stat.match['icmpv4_code']
    icmp_type = stat.match['icmpv4_type']

elif stat.match['ip_proto'] == 6:
    tp_src = stat.match['tcp_src']
    tp_dst = stat.match['tcp_dst']

elif stat.match['ip_proto'] == 17:
    tp_src = stat.match['udp_src']
    tp_dst = stat.match['udp_dst']

flow_id = str(ip_src) + str(tp_src) + str(ip_dst) + str(tp_dst) + str(ip_proto)

try:
    packet_count_per_second = stat.packet_count/stat.duration_sec
    packet_count_per_nsecond = stat.packet_count/stat.duration_nsec
except:
    packet_count_per_second = 0
    packet_count_per_nsecond = 0

try:
    byte_count_per_second = stat.byte_count/stat.duration_sec
    byte_count_per_nsecond = stat.byte_count/stat.duration_nsec
except:
    byte_count_per_second = 0
    byte_count_per_nsecond = 0

```

```

file0.write("{} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} \n"
            .format(timestamp, ev.msg.datapath.id, flow_id, ip_src, tp_src, ip_dst, tp_dst,
                    stat.match['ip_proto'], icmp_code, icmp_type,
                    stat.duration_sec, stat.duration_nsec,
                    stat.idle_timeout, stat.hard_timeout,
                    stat.flags, stat.packet_count, stat.byte_count,
                    packet_count_per_second, packet_count_per_nsecond,
                    byte_count_per_second, byte_count_per_nsecond))

file0.close()

def flow_training(self):

    flow_dataset = pd.read_csv('Model.csv')

    flow_dataset.iloc[:, 2] = flow_dataset.iloc[:, 2].str.replace('.', '')
    flow_dataset.iloc[:, 3] = flow_dataset.iloc[:, 3].str.replace('.', '')
    flow_dataset.iloc[:, 5] = flow_dataset.iloc[:, 5].str.replace('.', '')

    X_flow = flow_dataset.iloc[:, :-1].values
    X_flow = X_flow.astype('float64')

    y_flow = flow_dataset.iloc[:, -1].values

    X_flow_train, X_flow_test, y_flow_train, y_flow_test = train_test_split(X_flow,
                                                                              y_flow, test_size=0.25, random_state=0)

    classifier = RandomForestClassifier(n_estimators=10, criterion="entropy",
                                       random_state=0)

    self.flow_model = classifier.fit(X_flow_train, y_flow_train)

```

```
y_flow_pred = self.flow_model.predict(X_flow_test)
```

```
cm = confusion_matrix(y_flow_test, y_flow_pred)  
self.logger.info(cm)
```

```
acc = accuracy_score(y_flow_test, y_flow_pred)
```

```
fail = 1.0 - acc
```

```
def flow_predict(self):
```

```
    try:
```

```
        predict_flow_dataset = pd.read_csv('Predict.csv')
```

```
        predict_flow_dataset.iloc[:, 2] = predict_flow_dataset.iloc[:, 2].str.replace('.', '')
```

```
        predict_flow_dataset.iloc[:, 3] = predict_flow_dataset.iloc[:, 3].str.replace('.', '')
```

```
        predict_flow_dataset.iloc[:, 5] = predict_flow_dataset.iloc[:, 5].str.replace('.', '')
```

```
        X_predict_flow = predict_flow_dataset.iloc[:, :].values
```

```
        X_predict_flow = X_predict_flow.astype('float64')
```

```
        y_flow_pred = self.flow_model.predict(X_predict_flow)
```

```
        legitimate_traffic = 0
```

```
        ddos_traffic = 0
```

```
        for i in y_flow_pred:
```

```
            if i == 0:
```

```

        legitimate_traffic = legitimate_traffic + 1
    else:
        ddos_traffic = ddos_traffic + 1
        victim = int(predict_flow_dataset.iloc[i, 5])%20

    self.logger.info("-----
--")

    if (legitimate_traffic/len(y_flow_pred)*100) > 80:
        self.logger.info("Normal traffic ...")
    else:
        self.logger.info("DDoS traffic ...")
        self.logger.info("victim is host: h{ }".format(victim))

    self.logger.info("-----
--")

    file0 = open("Predict.csv", "w")

    file0.write('timestamp,datapath_id,flow_id,ip_src,tp_src,ip_dst,tp_dst,ip_proto,i
cmp_code,icmp_type,flow_duration_sec,flow_duration_nsec,idle_timeout,hard_timeou
t,flags,packet_count,byte_count,packet_count_per_second,packet_count_per_nsecond,
byte_count_per_second,byte_count_per_nsecond\n')
    file0.close()

except:
    pass

```

Code used to build Random Forest model:

```
from datetime import datetime

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC #loading SVM model
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

class MachineLearning():

    def __init__(self):

        print("Loading dataset ...")

        self.flow_dataset = pd.read_csv('FlowStatsfile.csv')

        self.flow_dataset.iloc[:, 2] = self.flow_dataset.iloc[:, 2].str.replace('.', '')
        self.flow_dataset.iloc[:, 3] = self.flow_dataset.iloc[:, 3].str.replace('.', '')
        self.flow_dataset.iloc[:, 5] = self.flow_dataset.iloc[:, 5].str.replace('.', '')

    def flow_training(self):

        print("Flow Training ...")

        X_flow = self.flow_dataset.iloc[:, :-1].values
        X_flow = X_flow.astype('float64')

        y_flow = self.flow_dataset.iloc[:, -1].values
```

```
X_flow_train, X_flow_test, y_flow_train, y_flow_test = train_test_split(X_flow,
y_flow, test_size=0.25, random_state=0)
```

```
classifier = SVC(kernel='rbf', random_state=0)
flow_model = classifier.fit(X_flow_train, y_flow_train)
```

```
y_flow_pred = flow_model.predict(X_flow_test)
```

```
print("-----")
```

```
print("confusion matrix")
```

```
cm = confusion_matrix(y_flow_test, y_flow_pred)
print(cm)
```

```
acc = accuracy_score(y_flow_test, y_flow_pred)
```

```
print("succes accuracy = {0:.2f} %".format(acc*100))
```

```
fail = 1.0 - acc
```

```
print("fail accuracy = {0:.2f} %".format(fail*100))
```

```
print("-----")
```

```
def main():
```

```
    start = datetime.now()
```

```
    ml = MachineLearning()
```

```
    ml.flow_training()
```

```
    end = datetime.now()
```

```
    print("Training time: ", (end-start))
```

```
if __name__ == "__main__": main()
```


Commands used:

Generating Normal traffic:

run command `ryu-manager collect_normal_traffic.py` in `generate_normal_traffic` folder.

run command `"service openvswitch-switch start"`

run command `sudo python generate_normal_traffic.py` in `generate_normal_traffic` folder.

Restart PC:

Generating DDoS traffic:

run command `ryu-manager collect_ddos_traffic.py` in `generate_ddos_traffic` folder.

run command `"service openvswitch-switch start"`

run command `sudo python generate_normal_traffic.py` in `generate_ddos_traffic` folder.

Restart PC:

Detecting Attacks and Normal Traffic:

run command `service openvswitch-switch start`

run command `ryu-manager RF_controller.py` in `mininet` folder.

run command `sudo python SDN.py` in `mininet` folder

after the above command in the same terminal, run command `xterm h1 h6 h13 h17`.

Normal traffic

ping 10.0.0.15 on node h17 then ping 10.0.0.4 on h6

go to h13 do `cd .. ls, clear, git clone, wget` etc commands

stop

DDOS traffic:

hping3 is a network tool to perform ddos attacks.

now on node h17 ping 10.0.0.18

on node h6 `hping3 -1 -V -d 120 -w 64 -p 80 --rand-source --flood 10.0.0.12 icmp flood`

`hping3 -S -V -d 120 -w 64 -p 80 --rand-source --flood 10.0.0.12 syn flood`

`hping3 -2 -V -d 120 -w 64 -p 80 --rand-source --flood 10.0.0.12 udp flood`

`hping3 -1 -V -d 120 -w 64 -p 80 --rand-source --flood -a 10.0.0.12 10.0.0.12 smurfs`

CHAPTER 9

Result and Discussion

.

```
kali@kali: ~/Downloads/SDN/Detection
kali@kali: ~/Downloads/SDN/Detec... x kali@kali: ~/Downloads/SDN/Detec... x
(kali@kali)-[~/Downloads/SDN/Detection]
$ sudo python SDN.py
[sudo] password for kali:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3)
(h10, s4) (h11, s4) (h12, s4) (h13, s5) (h14, s5) (h15, s5) (h16, s6) (h17, s6)
(h18, s6) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
mininet>
```

Fig 9.1

Starting the SDN simulation and enabling the ryu controller for traffic flow.

```
kali@kali: ~/Downloads/SDN/Detection
kali@kali: ~/Downloads/SDN/Detec... x kali@kali: ~/Downloads/SDN/Detec... x
(kali@kali)-[~/Downloads/SDN/Detection]
$ sudo python SDN.py
[sudo] password for kali:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3)
(h10, s4) (h11, s4) (h12, s4) (h13, s5) (h14, s5) (h15, s5) (h16, s6) (h17, s6)
(h18, s6) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
mininet> xterm h1 h5
```

Fig 9.2

Xterm h1 h5 opens Host 1 and Host 5 nodes on our network.

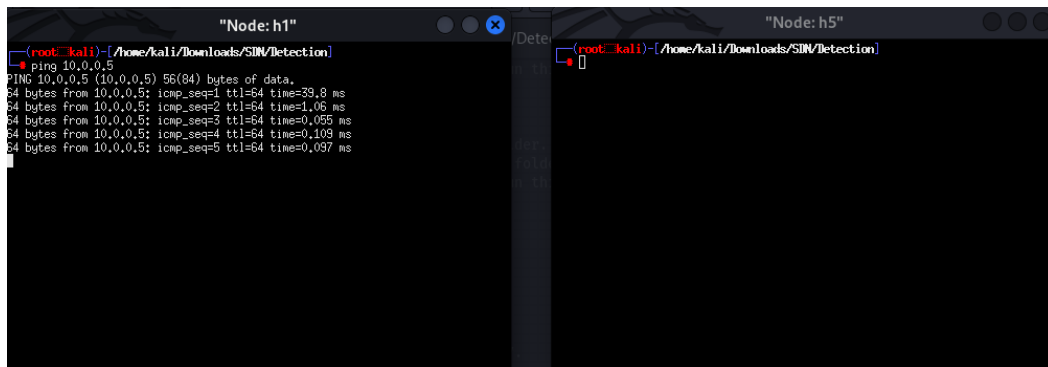


Fig 9.3

Node h1 with ip address 10.0.0.1 and node h5 with ip address 10.0.0.5 are opened.

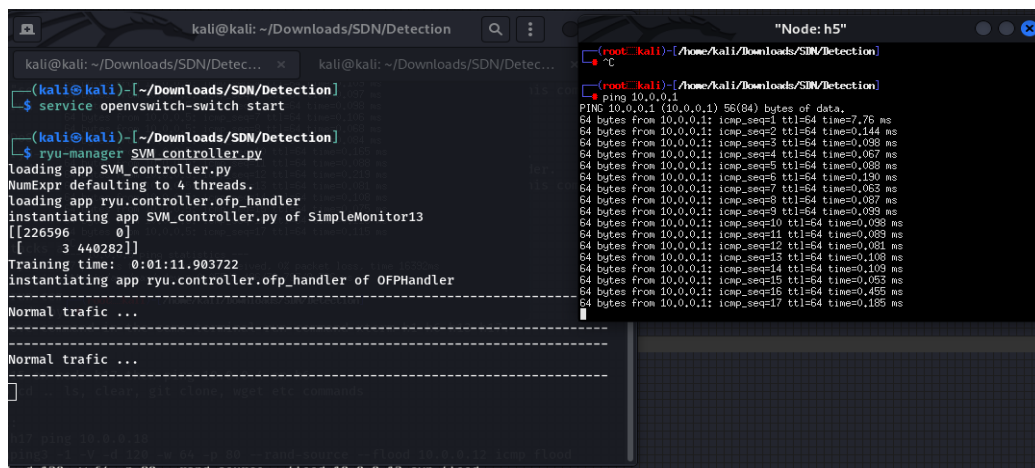


Fig 9.4

We have run command ping 10.0.0.1 from h5 node. Which pings node h1 as ip of node h1 is 10.0.0.1.

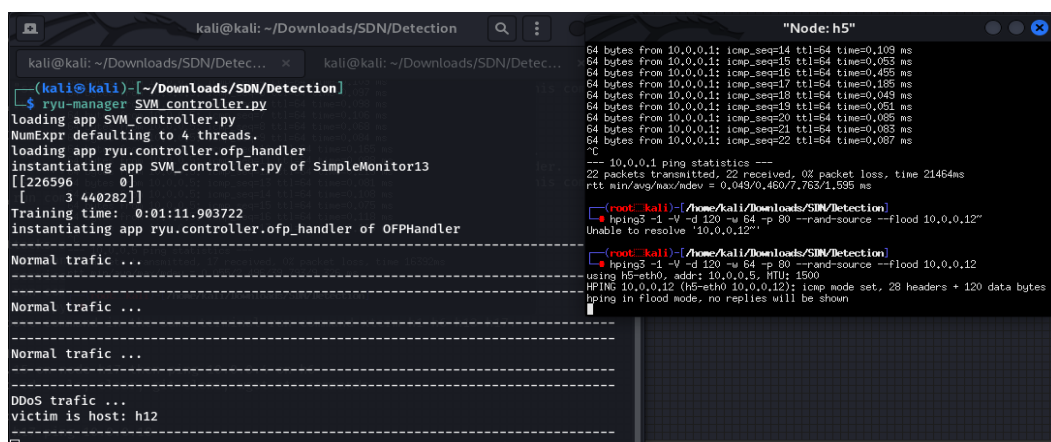


Fig 9.5

When we again attack the network in realtime using hping3 or any other tool our model detects the traffic as DDoS traffic and shows a log of which host is the victim of the attack.

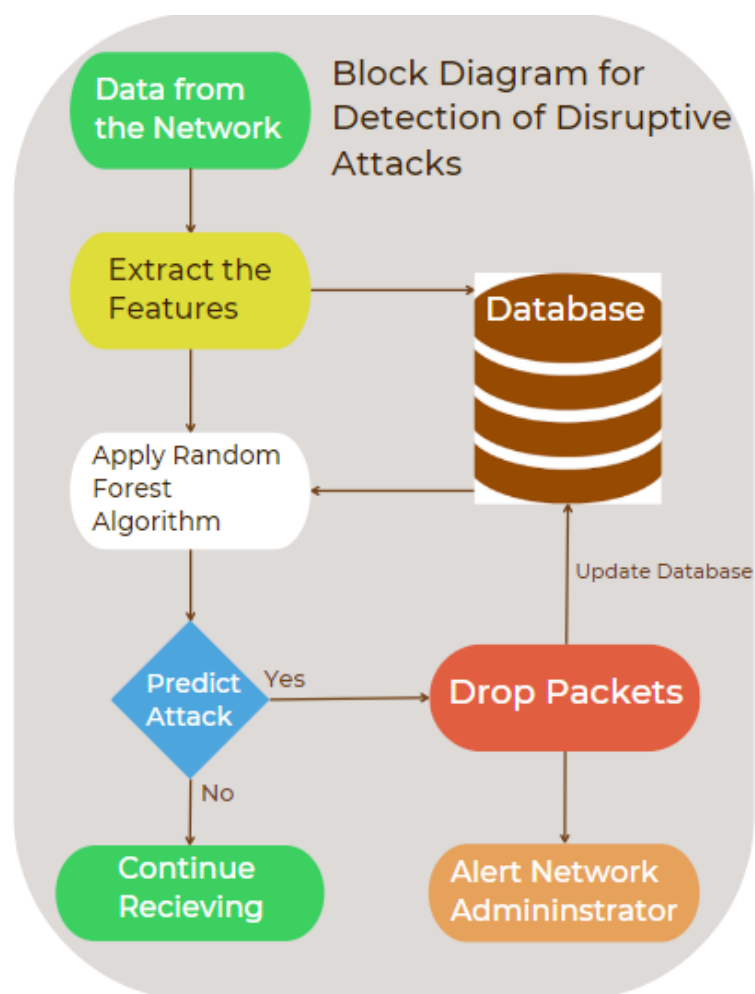


Fig 9.7

CHAPTER 10

CONCLUSION

Conclusion:

We have simulated a SDN Network on our local machine. We have Generated our Dataset on our local machine which we have used for training Machine Learning model. This Model is then Deployed on our SDN network which monitors the Traffic flow on the SDN network.

We chose the best Machine Learning algorithm for the early and accurate detection of DDoS attacks over the SDN. The combination of the machine learning classifiers and the advantages of the SDN protects the SDN controller from DDoS attacks.

Our approach is much more reliable and is quick in detecting DDoS traffic. It takes very less computation time to analyse the packets in the traffic. Which makes our approach much more responsive than the existing system.

We have also compared four other Machine Learning models on our dataset and choose the best Model to be deployed on the network.

The accuracy of the Random Forest machine learning model is **97.37%**

The work done in this project helps in identifying DDoS attacks in a simpler and more efficient manner. It even highlights the performance of machine learning algorithms as a comparative study was conducted between Deep learning , Random Forest, K-NN and SVM . Real-time datasets are used for efficient analysis. Since Random Forest fetched good results, it was preferred.

CHAPTER 11

FUTURE WORK

Future work:

It may fail to detect the attack traffic in a multi-controller environment. So, in the future, these models are also evaluated to detect attacks in a multi-controller context.

The presented model acquired findings from a single dataset, which serves as a limitation of the model. Consequently, a distributed dataset can be examined in order to provide directions for future enhancements.

Memory and other limited resources and computing abilities, as well as a diversity of standards and protocols, characterize the Internet of Things. These variables add significantly to the difficulties in researching IoT security issues, including anomaly mitigation utilizing IDS. In spite of the extensive study on anomaly detection in IoT networks, there are numerous key outstanding challenges that require additional investigation. The following are a few of these issues:.. There are no publicly available IoT network traffic datasets. Because assessing and validating anomaly prevention strategies on a real network will be difficult, efforts to create an IoT dataset are essential. This will make evaluating and validating suggested anomaly mitigation techniques in the IoT much easier. There are not any standard authentication apps for IoT. The validation of implemented structures is critical since it guarantees that they are developed acceptably. The implemented structures are put to the test in a variety of ways, including simulations and tests. However, because of a lack of standard authentication applications, most of implemented IDS structures in the IoT are not evaluated in contrast to other IDS structures in the IoT. As a result, efforts must be made to produce standard authentication, which will assure duplication, reproducibility, and research continuity. RNN and CNN are examples of supervised and unsupervised ML techniques, and both can be discovered using the CICDDoS2019 dataset.. It is possible to gather and examine real-time packets against the classified training dataset. It is possible to use a technique for splitting the data and comparing it with the performance of the classifiers utilized fold cross authentication

References

- 1. **Journal/Article:** Dayanandam, G.; Reddy, E.S.; Babu, D.B. Regression algorithms for efficient detection and prediction of DDoS attacks. In Proceedings of the 2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Tumkur, India, 21–23 December 2017; pp. 215–219.
- 2. **Journal/Article:** Sharma, N.; Mahajan, A.; Mansotra, V. Machine Learning Techniques Used in Detection of DOS Attacks: A Literature Review. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* 2016, 6, 100.
- 3. **Journal/Article:** Somani, G.; Gaur, M.S.; Sanghi, D.; Conti, M.; Buyya, R. DDoS attacks in cloud computing: Issues, taxonomy, and future directions. *Comput. Commun.* 2017, 107, 30–48.
- 4. **Book:** Perera, P.; Tian, Y.-C.; Fidge, C.; Kelly, W. A Comparison of Supervised Machine Learning Algorithms for Classification of Communications Network Traffic. In *International Conference on Neural Information Processing*; Springer: Cham, Switzerland, 2017; pp. 445–454.
- 5. **Book:** Zammit, D. A Machine Learning Based Approach for Intrusion Prevention Using Honeypot Interaction Patterns as Training Data. Bachelor's Thesis, University of Malta, Msida, Malta, 2016.
- 6. **Journal/Article:** Doshi, R.; Apthorpe, N.; Feamster, N. Machine Learning DDoS Detection for Consumer Internet of Things Devices. *arXiv* 2018, arXiv:1804.04159.
- 7. **Journal/Article:** Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* 1997, 9, 1735–1780.
- 8. **Book:** Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated

recurrent neural networks on sequence modeling. In Proceedings of the NIPS 2014 Deep Learning and Representation Learning Workshop, Montreal, QC, Canada, 12 December 2014.

- 9. **Book:** Breitenbacher, D.; Elovici, Y. N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Comput.* 2018, 17, 12–22.
- 10. **Journal/Article:** Zekri, M.; El Kafhali, S.; Hanini, M.; Aboutabit, N. Mitigating Economic Denial of Sustainability Attacks to Secure Cloud Computing Environments. *Trans. Mach. Learn. Artif. Intell.* 2017, 5, 473–481.
- 11. **Journal/Article:** Liao, Q.; Li, H.; Kang, S.; Liu, C. Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching. *Secur. Commun. Netw.* 2015, 8, 3111–3120.
- 12. **Journal/Article:** Xiao, P.; Qu, W.; Qi, H.; Li, Z. Detecting DDoS attacks against data center with correlation analysis. *Comput. Commun.* 2015, 67, 66–74. platerecognition", *Pattern Recognition*, vol. 42.
- 14. **Journal/Article:** Zhong, R.; Yue, G. Ddos detection system based on data mining. In Proceedings of the 2nd International Symposium on Networking and Network Security, Jinggangshan, China, 2–4 April 2010; pp. 2–4.
- 15. **Journal/Article:** Wu, Y.-C.; Tseng, H.-R.; Yang, W.; Jan, R.-H. Ddos detection and traceback with decision tree and grey relational analysis. *Int. J. Ad Hoc Ubiquitous Comput.* 2011, 7, 121–136.
- 16. **Journal/Article:** Li, H.; Liu, D. Research on intelligent intrusion prevention system based on Snort. In Proceedings of the International Conference on Computer,

Mechatronics, Control and Electronic Engineering (CMCE), Changchun, China, 24–26 August 2010; Volume 1, pp. 251–253.

- 17. **Book:** Chen, J.-H.; Zhong, M.; Chen, F.-J.; Zhang, A.-D. DDoS defense system with turing test and neural network. In Proceedings of the IEEE International Conference on Granular Computing (GrC), Hangzhou, China, 11–13 August 2012; pp. 38–43.
- 18. **Journal/Article:** Ibrahim, L.M. Anomaly network intrusion detection system based on distributed time-delay neural network (dtdnn). *J. Eng. Sci. Technol.* 2010, 5, 457–471.
- 19. **Journal/Article:** Fadil, A.; Riadi, I.; Aji, S. Review of Detection DDOS Attack Detection Using Naive Bayes Classifier for Network Forensics. *Bull. Electr. Eng. Inform.* 2017, 6, 140–148.
- 20. **Journal/Article:** Zargar, S.T.; Joshi, J.B.; Tipper, D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Commun. Surv. Tutor.*
- 21. **Journal/Article:** Gupta, B.B.; Misra, M.; Joshi, R.C. FVBA: A combined statistical approach for low rate degrading and high bandwidth disruptive DDoS attacks detection in ISP domain. *IEEE Int. Conf. Netw.* 2008, 1–4.
- 22. **Journal/Article:** Francois, J.; Aib, I.; Boutaba, R. FireCol: A Collaborative Protection Network for the Detection of Flooding DDoS Attacks. *IEEE/ACM Trans. Netw.* 2012, 20, 1828–1841.
- 23. **Book:** Jia, B.; Huang, X.; Liu, R.; Ma, Y. A DDoS Attack Detection Method Based on Hybrid Heterogeneous Multiclassifier Ensemble Learning. *J. Electr. Computer. Eng.* 2017, 2017, 1–9.

- 24.**Journal/Article:** E. W. Dijkstra, "Cooperating sequential processes" in Programming Languages, New York:Academic, pp. 43-112, 1968.
- 25. **Journal/Article:** E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten and E. F. Steffens, "On-the-fly garbage collection: An exercise in cooperation", *Commun. Assoc. Comput. Mach.*.
- 26.**Journal/Article:** Multiprocessors and Parallel Processing, New York:Wiley-Interscience, 1974.
- 27. **Book:** R. S. Fabry, "Dynamic verification of operating system decisions", *Commun. Assoc. Comput. Mach.*, vol. 16, pp. 659-668, Nov. 1973.
- 28. **Book:** M. J. Flynn, "Very high-speed computing systems", *Proc. IEEE*, vol. 54, pp. 1901-1909, 1966-Dec.
- 29. **Book:** M. J. Flynn and A. Podvin, "An unconventional computer architecture: Shared resource multiprocessing", *Computer*, vol. 5, pp. 20-28, Mar. 1972.
- 30. **Journal/Article:** Gupta, B.B.; Misra, M.; Joshi, R.C. FVBA: A combined statistical approach for low rate degrading and high bandwidth disruptive DDoS attacks detection in ISP domain. *IEEE Int. Conf. Netw.* 2008, 1–4.
- 31. **Journal/Article:** Francois, J.; Aib, I.; Boutaba, R. FireCol: A Collaborative Protection Network for the Detection of Flooding DDoS Attacks. *IEEE/ACM Trans. Netw.* 2012, 20, 1828–1841.
- 32. **Journal/Article:** Bhavsar, Hetal, and Amit Ganatra. "A Comparative Study of Training Algorithms for Supervised Machine Learning."

- 33. **Journal/Article:** Liu, Lan, Pengcheng Wang, Jun Lin, and Langzhou Liu. "Intrusion detection of imbalanced network traffic based on machine learning and deep learning." *Ieee Access* 9 (2020): 7550-7563.
- 34. **Journal/Article:** Sultana, N., Chilamkurti, N., Peng, W., & Alhadad, R. (2018). Survey on SDN based network intrusion detection system using machine learning approaches.
- 35. **Journal/Article:** Wagh, Sharmila Kishor, Vinod K. Pachghare, and Satish R. Kolhe. "Survey on intrusion detection system using machine learning techniques." *International Journal of Computer Applications* 78, no. 16 (2013).
- 36. **Journal/Article:** Patgiri, Ripon, Udit Varshney, Tanya Akutota, and Rakesh Kunde. "An Investigation on Intrusion Detection System Using Machine Learning."
- 37. **Journal/Article:** Shon, Taeshik, Yongdae Kim, Cheolwon Lee, and Jongsub Moon. "A machine learning framework for network anomaly detection using SVM and GA." In *Proceedings from the sixth annual IEEE SMC information assurance workshop*, pp. 176-183. IEEE, 2005.
- 38. **Journal/Article:** Patil, Dharmaraj R., and Tareek M. Pattewar. "A comparative performance evaluation of machine learning-based NIDS on benchmark datasets." *International Journal of Research in Advent Technology* 2, no. 2 (2014): 101-106.
- 39. **Journal/Article:** Rupa Devi, T. and Badugu, S., 2020. A review on network intrusion detection system using machine learning. *Advances in Decision Sciences, Image Processing, Security and Computer Vision*, pp.598-607.
- 40. **Journal/Article:** Wagh, Sharmila Kishor, Vinod K. Pachghare, and Satish R. Kolhe.