Mid Defense for the degree of Masters in Computer Engineering

Optimizing of Controller Placement based on Hybrid GENCLUST Algorithm for DDOS Mitigation on Software Defined Network

Sahaj Shakya
(2019-1-39-0020)

Nepal College of Information Technology
Faculty of Science and Technology
Pokhara University, Nepal

April, 2022

Mid Defense for the degree of Masters in Computer Engineering

Optimizing of Controller Placement based on Hybrid GENCLUST Algorithm for DDOS Mitigation on Software Defined Network

Supervised by Prof. Roshan Chitrakar, Ph. D

A thesis submitted in partial fulfillment of the requirement for the

Degree of Master in Computer Engineering

Sahaj Shakya
(2019-1-39-0020)

Nepal College of Information Technology
Faculty of Science and Technology
Pokhara University, Nepal

April, 2022

ACKNOWLEDGEMENT

I would like to express my deep gratitude to the Masters department of engineering of NCIT, for providing me with the opportunity to do this project.

I would like to extend my sincere gratitude to my supervisor "**Prof. Dr Roshan Chitrakar, Ph. D**" for his valuable suggestions and guidance throughout the course of this project. I am also highly thankful to our program coordinator "**Mr. Saroj Shakya**" of Master's in Computer Engineering for his constant support and guidance as well as to the department. Also, I would like to thank all my classmates and teachers for their constant support and suggestions on this project.

Keywords: SDN, DDOS, K-means Clustering, Genetic Algorithm, OpenFlow Controller

………………………

Sahaj Shakya

2019-1-39-0020

4/20/2022

ABSTRACT

Today's widely used networks are complex and difficult to manage. To implement high-level network guidelines in traditional IP-based networks, network administrators must configure individual network devices with manufacturer-specific commands. As a result, it becomes very difficult to implement the desired policies and reconfigurations of network devices in today's IP-based networks. Software Defined Networking (SDN) has been adopted for the Flexible Internet and has the potential to be used for the next generation of the Internet by using a controller to separate the control plane from the data plane. A distributed denial of service (DDoS) attack can make on the SDN controller unable to process legitimate flow requests from the switch. The main approach is to protect a controller from DDoS attacks is based on attack detection which results in a high rate of false-negative and false-positive results. Existing mitigation techniques are basically based on external and additional resource or network traffic analysis and tend to be computationally intensive or have a high rate of false positives and / or false positives. The management of 150 switches in a system can only handle up to 20,000 new flow requests. Therefore, the controller must do a lot of work to handle network traffic efficiently. However, additional computational overhead can occur for multiple parameters and optimizing the controllers' positions. The purpose methods reduce the overhead in three steps, firstly selecting the controller for overhead traffic, secondly using genetic algorithm to elect the leader and third select the optimize position for controller from clustered controllers.

Keywords: SDN, DDOS, K-means Clustering, Genetic Algorithm, OpenFlow Controller

Table of content

List of Tables

List of Figures

List of abbreviation/acronyms

DDOS	Distributed Denial of Service
SDN	Software Defined Network
ANN	Artificial Neural Network
ANP	Analytic network process
SOM	Self-Organizing Map
DR	Detection Rate
TP	True Positive
FN	False Negative
FP	False Positive
SVM	Support Vector Machine
KNN	K Nearest Neighbor
SSIP	Speed if IP Sources
SFE	Speed of flow entries
RPE	Ratio of pair flow entries

CHAPTER 1

INTRODUCTION

A distributed denial of service (DDoS) attack occurs when one or more attackers attempt to block the delivery of a service. This can be achieved by forcing access to virtually everything, including servers, devices, services, networks, applications, and even specific transactions within an application. A DoS attack is a system that sends malicious data and requests. DDoS attacks come from multiple systems. Impacts range from service interruptions to entire websites, applications, and even business failures, from minor annoyances. However, a system can be developed to mitigate such problems. The system can be trained based on dataset. For this purposed methodology, NSL-KDD can be used.

Many researchers have proposed and implemented various models for IDS but they often generate too many false alerts due to their simplistic analysis. An attack generally falls into one of four categories [16].

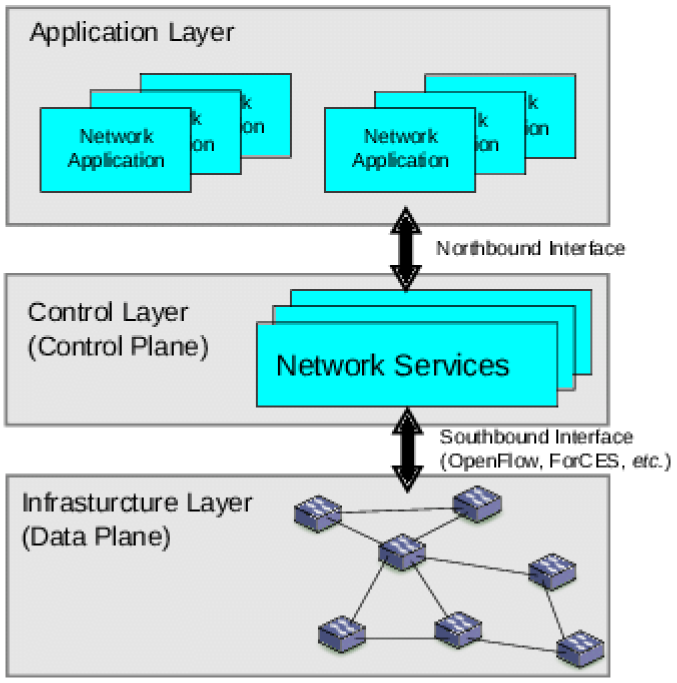

Application Layer
Network Application
Network Application
Northbound Interface
Control Layer (Control Plane)
Network Services
Southbound Interface (OpenFlow, ForCES, etc.)
Infrasturcture Layer (Data Plane)

Figure 1-1 SDN Layers

1.1 Types of Attacks

1.1.1. Denial-of-Service(DoS):

Attackers tries to prevent legitimate users from using a service. For example, there are smurf, neptune, back, teardrop, pod and land.

1.1.2. Probe:

The attacker attempts to obtain information about the target host. Port scans or sweeps of a specific range of IP addresses usually fall into this category. (e.g. saint, ipsweep, portsweep and nmap).

1.1.3. User-to-Root(U2R):

Attacker attempts to gain super user privileges by accessing the victim's computer locally. For example, these are buffer_overflow, rootkit, landmodule and perl.

1.1.4. Remote-to-Local(R2L):

The attacker does not have an account on the victim's computer and is trying to access it. For example, these are guess_passwd, ftp_write, multihop, phf, spy, imap, warezclient and warezmaster.

SDN controller causes potential threats due to its single point failure. The major vulnerabilities in SDN can be categorized into three major attacks based on the different plane of SDN architecture [17].

1.2 Attacks in SDN Layers

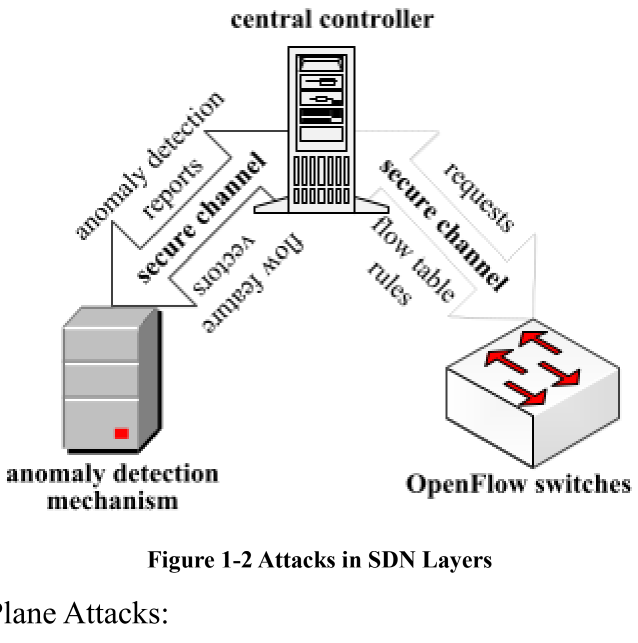


Figure 1-2 Attacks in SDN Layers

1.2.1 Data Plane Attacks:

Space constraints in the data plane lead to buffer overflows and stream table overflows, which are the main causes of security issues challenges in data plane attacks. It is difficult to identify malicious and true flows in the data plane due to the presence of a discharge switch and the role of the controller in decision-making. As data traffic increases, congestion on the data control plane link causes the control plane and data plane to shut down. Also, corruption of the SDN controller will corrupt the data plane resources. Therefore, a data plane attack depends on the security of the control plane [17].

1.2.2 Control Plane Attacks:

Due to the centralized structure of the controller, the control plane is the focus of the majority of attacks. It comprises dangers from the application, scalability threats, and availability threats. The majority of unaddressed data plane issues result in control plane saturation attacks. The controller is in charge of performing a tailored security check of various applications, including application authentication and resource authorization for resources that have not yet been established. Furthermore, today's SDN controllers are incapable of handling network traffic on a 10 Gbps link in a high-speed network. DoS attacks in SDN are made easier by the SDN controller's lack of scalability and the unavailability of network resources. Multi-controller is not a good solution for DDoS attacks as it can result in cascading failure of all controllers if there is not optimizing mechanism for switching between them [17].

The DDoS attacks are categorized under the availability threat of the controller. The reasons for DDoS vulnerabilities are as follows:

- Due to the limited memory space available to buffer the information, buffer saturation occurs.
- Due to overhead in the controller caused by the controller's centralized architecture is known as controller saturation.
- due to limited TCAM memory which causes Flow Table overflow
- Due to control-data plane link's communication overhead generates a bottleneck for genuine users.

1.2.3 Application Plane Attacks:

It includes authentication and authorization issues, as well as access control and accountability concerns. Every request from the application to access network resources must be authenticated. Authentication of a large number of SDN apps, on the other hand, is difficult. Furthermore, due to a lack of access control and accountability, the malicious program can evade the SDN network architecture [17].

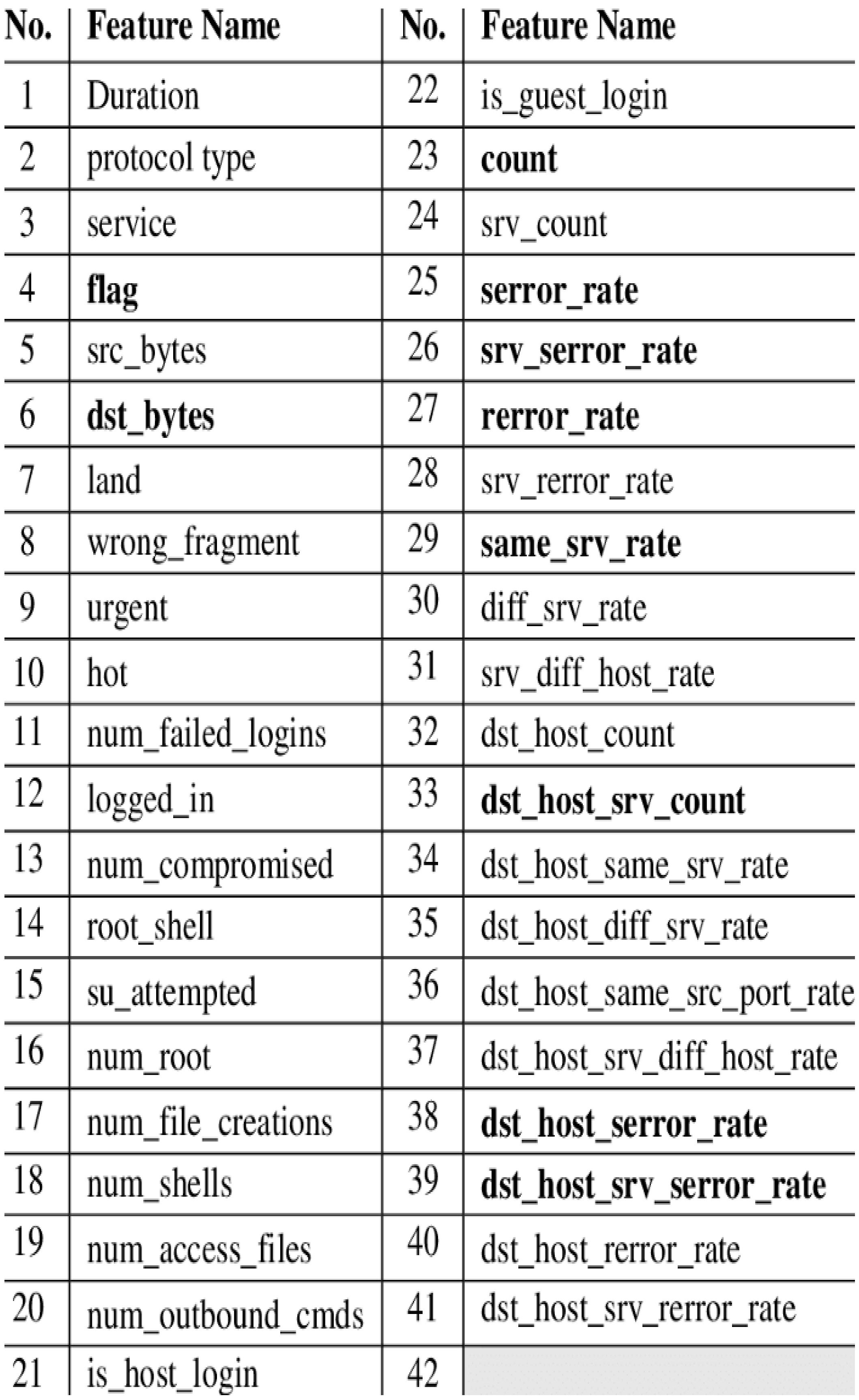
1.3 NSL-KDD

NSL-KDD is an up to date model of KDD cup99 statistics set which counseled to remedy a few troubles of preceding model. This statistics set is a powerful benchmark for researchers to evaluate special kinds of Intrusion detection system (IDS) methods, construct an Intrusion detection system (Host primarily based totally or Network primarily based totally), doing for a few experiments in Cyber safety region like smart there may be such a lot of advantages. And additionally, there are lot of statistics sets (ADFA-ID, ISCX-UNB etc) which use on this field [17].

There are 8 types of different data sets in the NSL-KDD.

- KDDTrain+.ARFF The full NSL-KDD train set with binary labels in ARFF format.
- KDDTrain+.TXT The full NSL-KDD train set including attack-type labels and difficulty level in CSV format.
- KDDTrain+_20Percent.ARFF A 20% subset of the KDDTrain+.arff file.
- KDDTrain+_20Percent.TXT A 20% subset of the KDDTrain+.txt file.
- KDDTest+.ARFF The full NSL-KDD test set with binary labels in ARFF format.
- KDDTest+.TXT The full NSL-KDD test set including attack-type labels and difficulty level in CSV format.
- KDDTest-21. ARFF A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21.
- KDDTest-21.TXT A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21.

In each record there are 41 types of features and those are assigned to attack or normal type. Each feature is categorized in to 3 types of attribute value types. (Nominal, binary and numeric)

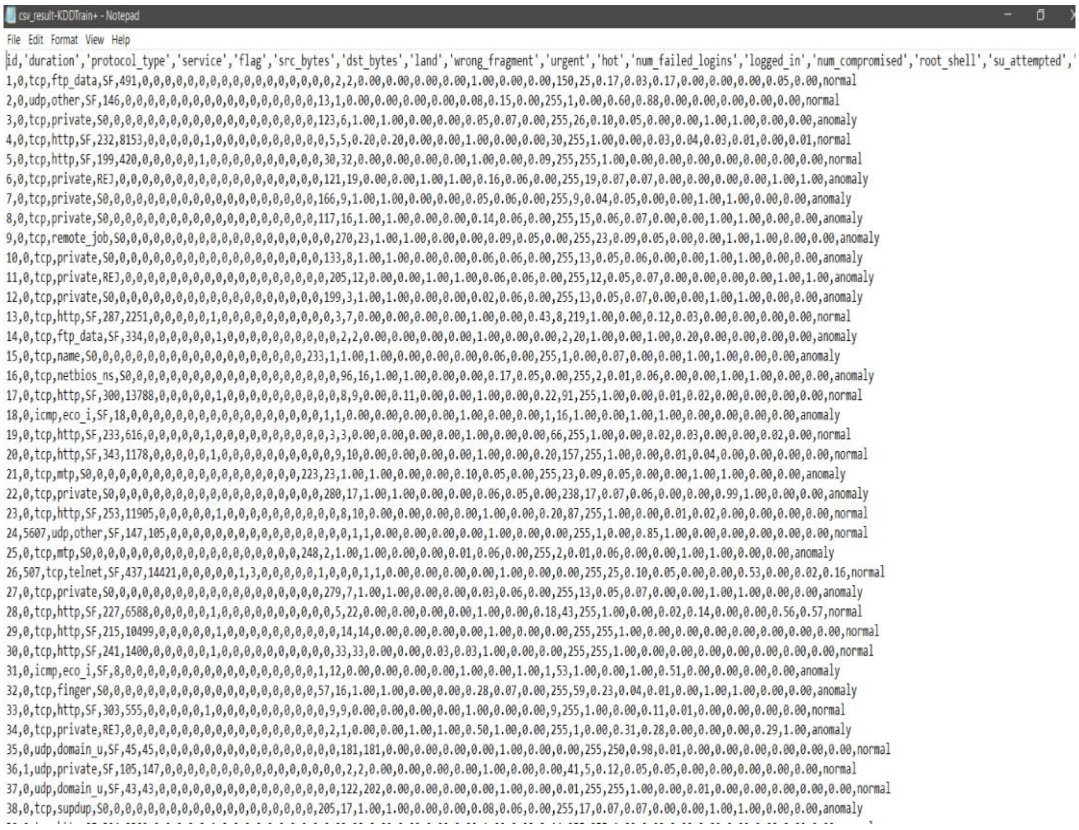


No.	Feature Name	No.	Feature Name
1	Duration	22	is_guest_login
2	protocol type	23	**count**
3	service	24	srv_count
4	**flag**	25	**serror_rate**
5	src_bytes	26	**srv_serror_rate**
6	**dst_bytes**	27	**rerror_rate**
7	land	28	srv_rerror_rate
8	wrong_fragment	29	**same_srv_rate**
9	urgent	30	diff_srv_rate
10	hot	31	srv_diff_host_rate
11	num_failed_logins	32	dst_host_count
12	logged_in	33	**dst_host_srv_count**
13	num_compromised	34	dst_host_same_srv_rate
14	root_shell	35	dst_host_diff_srv_rate
15	su_attempted	36	dst_host_same_src_port_rate
16	num_root	37	dst_host_srv_diff_host_rate
17	num_file_creations	38	**dst_host_serror_rate**
18	num_shells	39	**dst_host_srv_serror_rate**
19	num_access_files	40	dst_host_rerror_rate
20	num_outbound_cmds	41	dst_host_srv_rerror_rate
21	is_host_login	42	

Figure 1- 3.1 Features of NSL-KDD

Attacks in this Dataset is categorized into 4 parts.

- DoS- Denial of service.
- Probing- Surveillance and other probing attacks.
- U2R- Unauthorized access to local super user.
- R2L- Unauthorized access from a remote machine.



```
id,'duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','
1,0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,150,25,0.17,0.03,0.17,0.00,0.00,0.00,0.05,0.00,normal
2,0,udp,other,SF,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,1,0.00,0.00,0.00,0.00,0.08,0.15,0.00,255,1,0.00,0.60,0.88,0.00,0.00,0.00,0.00,0.00,normal
3,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,123,6,1.00,1.00,0.00,0.00,0.05,0.07,0.00,255,26,0.10,0.05,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
4,0,tcp,http,SF,232,8153,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.20,0.20,0.00,0.00,1.00,0.00,0.00,30,255,1.00,0.00,0.03,0.04,0.03,0.01,0.00,0.01,normal
5,0,tcp,http,SF,199,420,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,30,32,0.00,0.00,0.00,0.00,1.00,0.00,0.09,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal
6,0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,19,0.00,0.00,1.00,1.00,0.16,0.06,0.00,255,19,0.07,0.07,0.00,0.00,0.00,0.00,1.00,1.00,anomaly
7,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,166,9,1.00,1.00,0.00,0.00,0.05,0.06,0.00,255,9,0.04,0.05,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
8,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,117,16,1.00,1.00,0.00,0.00,0.14,0.06,0.00,255,15,0.06,0.07,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
9,0,tcp,remote_job,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,270,23,1.00,1.00,0.00,0.00,0.09,0.05,0.00,255,23,0.09,0.05,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
10,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,133,8,1.00,1.00,0.00,0.00,0.06,0.06,0.00,255,13,0.05,0.06,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
11,0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,205,12,0.00,0.00,1.00,1.00,0.06,0.06,0.00,255,12,0.05,0.07,0.00,0.00,0.00,0.00,1.00,1.00,anomaly
12,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,3,1.00,1.00,0.00,0.00,0.02,0.06,0.00,255,13,0.05,0.07,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
13,0,tcp,http,SF,287,2251,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,7,0.00,0.00,0.00,0.00,1.00,0.00,0.43,8,219,1.00,0.00,0.12,0.03,0.00,0.00,0.00,0.00,normal
14,0,tcp,ftp_data,SF,334,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,2,20,1.00,0.00,1.00,0.20,0.00,0.00,0.00,0.00,anomaly
15,0,tcp,name,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,233,1,1.00,1.00,0.00,0.00,0.00,0.06,0.00,255,1,0.00,0.07,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
16,0,tcp,netbios_ns,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,96,16,1.00,1.00,0.00,0.00,0.17,0.05,0.00,255,2,0.01,0.06,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
17,0,tcp,http,SF,300,13788,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,9,0.00,0.11,0.00,0.00,1.00,0.00,0.22,91,255,1.00,0.00,0.01,0.02,0.00,0.00,0.00,0.00,normal
18,0,icmp,eco_i,SF,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,16,1.00,0.00,1.00,1.00,0.00,0.00,0.00,0.00,anomaly
19,0,tcp,http,SF,233,616,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,66,255,1.00,0.00,0.02,0.03,0.00,0.00,0.02,0.00,normal
20,0,tcp,http,SF,343,1178,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,9,10,0.00,0.00,0.00,0.00,1.00,0.00,0.20,157,255,1.00,0.00,0.01,0.04,0.00,0.00,0.00,0.00,normal
21,0,tcp,mtp,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,23,1.00,1.00,0.00,0.00,0.10,0.05,0.00,255,23,0.09,0.05,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
22,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,280,17,1.00,1.00,0.00,0.00,0.06,0.05,0.00,238,17,0.07,0.06,0.00,0.00,0.99,1.00,0.00,0.00,anomaly
23,0,tcp,http,SF,253,11905,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,10,0.00,0.00,0.00,0.00,1.00,0.00,0.20,87,255,1.00,0.00,0.01,0.02,0.00,0.00,0.00,0.00,normal
24,5607,udp,other,SF,147,105,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,1,0.00,0.85,1.00,0.00,0.00,0.00,0.00,0.00,normal
25,0,tcp,mtp,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,248,2,1.00,1.00,0.00,0.00,0.01,0.06,0.00,255,2,0.01,0.06,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
26,507,tcp,telnet,SF,437,14421,0,0,0,0,0,1,3,0,0,0,0,0,1,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,25,0.10,0.05,0.00,0.00,0.53,0.00,0.02,0.16,normal
27,0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,279,7,1.00,1.00,0.00,0.00,0.03,0.06,0.00,255,13,0.05,0.07,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
28,0,tcp,http,SF,227,6588,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,22,0.00,0.00,0.00,0.00,1.00,0.00,0.18,43,255,1.00,0.00,0.02,0.14,0.00,0.00,0.56,0.57,normal
29,0,tcp,http,SF,215,10499,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,14,14,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal
30,0,tcp,http,SF,241,1400,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,33,33,0.00,0.00,0.03,0.03,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal
31,0,icmp,eco_i,SF,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,12,0.00,0.00,0.00,0.00,1.00,0.00,1.00,1,53,1.00,0.00,1.00,0.51,0.00,0.00,0.00,0.00,anomaly
32,0,tcp,finger,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,57,16,1.00,1.00,0.00,0.00,0.28,0.07,0.00,255,59,0.23,0.04,0.01,0.00,1.00,1.00,0.00,0.00,anomaly
33,0,tcp,http,SF,303,555,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,9,9,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,255,1.00,0.00,0.11,0.01,0.00,0.00,0.00,0.00,normal
34,0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,0.00,0.00,1.00,1.00,0.50,1.00,0.00,255,1,0.00,0.31,0.28,0.00,0.00,0.00,0.29,1.00,anomaly
35,0,udp,domain_u,SF,45,45,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,181,181,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,250,0.98,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal
36,1,udp,private,SF,105,147,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,41,5,0.12,0.05,0.05,0.00,0.00,0.00,0.00,0.00,normal
37,0,udp,domain_u,SF,43,43,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,122,202,0.00,0.00,0.00,0.00,1.00,0.00,0.01,255,255,1.00,0.00,0.01,0.00,0.00,0.00,0.00,0.00,normal
38,0,tcp,supdup,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,205,17,1.00,1.00,0.00,0.00,0.08,0.06,0.00,255,17,0.07,0.07,0.00,0.00,1.00,1.00,0.00,0.00,anomaly
```

Figure 1-3.2 Parts of NSL-KDD Dataset

Dataset	Number of Records:					
	Total	Normal	DoS	Probe	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.11%)	52 (0.04%)	995 (0.85%)
KDDTest+	22544	9711 (43%)	7458 (33%)	2421 (11%)	200 (0.9%)	2654 (12.1%)

Figure 1-3.3 Records of NSL-KDD Dataset

1.4 Statement of the problem

Distributed denial of service (DDoS) attacks are on the rise, and many organizations are either completely unprepared or poorly defended. The impact of a DDoS attack usually consists of loss of sales and customers, damage to the brand, and use as a smoke screen for further inbound attacks. Companies with business-critical online resources are not only exposed to greater losses, but are also more likely to be attacked by attackers. For this reason, enterprises need to be proactive in addressing DDoS threats [1].

In DDos detection and prevention based on Ann, there are mainly two issues

1.4.1 Maximum Controller detection overhead

The management of 150 switches in a system can handle up to 20,000 new flow requests Therefore, the controller must do a lot of work to handle network traffic efficiently. However, additional computational overhead can occur for multiple parameters which might impact controller performance. The number of parameters used for DDoS detection can be reduced in an ANN-based way. However, it can also be improved by using multiple machine learning algorithms to reduce the complexity of the controller. Similarly, multiple OpenFlow controllers can be also used to mitigate the problem [2].

1.4.2 False Negative and False Positive Value

False negative values are common when the attack rate parameter is set to a low value. However, an attack with a low packet flow rate will do less damage to the victim's system [3]. However, this problem can be mitigated by using multiple machine learning algorithms [4].

1.2 Research objectives

The major objectives of this paper are

- To mitigate the Single Point Failure and overhead of Controller
- To reduce the false positive detection in case of Slow Attack
- To introduce Multiagent system in SDN

1.6 Significance/Rationale of the study

With the rapid development in the field of IT infrastructure, the size of the network, its complexity has increased many times. This makes it increasingly difficult to guarantee key network characteristics such as integrity, confidentiality, authentication, information availability, and non-repudiation. In recent years, many researchers and industries have shifted their focus to developing more robust, scalable, and more secure networks. Today's widely used networks are complex and difficult to manage. To implement high-level network guidelines in traditional IP-based networks, network administrators must configure individual network devices with manufacturer-specific commands. As a result, it becomes very difficult to implement the desired policies and reconfigurations of network devices in today's IP-based networks [5].

The latest advances, such as SDN (Software Defined Networking), are a step towards the dynamic and centralized nature of networks compared to traditional network static distributed environments. Recently, Software Defined Networking (SDN) has been adopted for the Flexible Internet and has the potential to be used for the next generation of the Internet. The concept is to control the network through software. This approach facilitates network management and enables new applications in virtual environments. SDN was developed with extended network support from the central controller. SDN is a new network architecture that gives hope to an efficient network infrastructure. First, vertical integration is eliminated by separating the control plane (network control logic) from the data plane (routers and switches that route network traffic according to the network control logic). This isolation control then provides a simple packet forwarding device that simplifies flexibility, implementation speed, programmability, and network management with a logically centralized controller and network logic installed on the network switch. The SDN architecture can improve the security of the network with the help of a centralized controller, but it creates global transparency for the network and, if necessary, traffic routing rules. However, security issues still remain in the SDN. [4].

However, SDN still has its own concerns and challenges regarding network security, scalability, and support capabilities. Security is paramount because of all these issues. Since the central controller is responsible for managing the network, a failure of this controller will affect the entire network. Central control and communication between the controller and the switch can be the target of advanced DDoS attacks.

Detecting flood-based distributed denial of service (DDoS) attacks is one of the biggest challenges for Internet security today. The DDoS attack mechanism is based on exploiting the huge resource asymmetry between the Internet and the victim's server limitations in handling a large number of fake requests. As a result, system resources are exhausted and the victim is taken from the Internet, so legitimate user requests are not processed [6].

With the advent and development of software-defined networks over the last decade, the ability to combat DDoS attacks in cloud environments has expanded. SDN has been determined to be an integral part of cloud and service providers that make networks programmable [7]. One of the problems is changing the packet header fields, similar to regular fields. As a result, it is very difficult to distinguish between legitimate regular traffic packets and useless packets sent from the compromised host to the victim. The second problem is the large number of packets that need to be analyzed. These are challenges that make detection difficult and slow response times.

The SDN controller can be a single point of failure. Security challenges are expected to increase as SDN technology becomes increasingly available. These SDN vulnerabilities focus on different layers of the SDN architecture. Analysis different vulnerabilities, security challenges can be summarized at different levels [4].

CHAPTER 2

LITERATURE REVIEW

Machine learning algorithms can use training data to automatically create classification models and use flow functions to classify network flows. In a real-time network environment, machine learning algorithms can detect known and unknown DDoS attacks. In the area of network security, the benefits of this SDN allow developers to easily and flexibly update their classification mechanisms to detect anomalous attacks on the network control plane. The SDN controller quickly collects and analyzes information from the switch and sends operational decisions back to the switch. Due to this flexible and effective process, the detection of SDN-based DDoS attacks has recently attracted the attention of the research community.

Initial was of mitigation mechanism uses HTTP-based and XML-based DDoS attacks preventions against cloud computing environments using a filter tree. This tree-based filtering scheme to prevent DDoS attacks uses five levels of filtering to mitigate the impact of HTTP-based and XML-based DDoS attacks. This tree-based DDoS attack prevention approach with filtering analyzes suspicious packets in the puzzle resolver to address issues caused by malicious data packets generated based on SOAP headers. This filter tree scheme identifies the IP address that was initially initiated by the malicious message to send the puzzle, and the puzzle sent is resolved to determine the actual client [8].

In case of "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow", the researcher has used multiple parameters for calculation of the current state of the network. Researchers have presented a lightweight method for detecting DDoS attacks based on traffic flow characteristics that extract such information with very little effort compared to traditional approaches [6]. This is made possible by the use of the NOX platform, which provides a programmatic interface that facilitates the processing of switching information. Other important contributions by researchers include high detection rates and very low false alarm rates achieved by flow analysis using self-organizing maps.

OpenFlow-based SDN protection against DDoS attacks was provided to prevent problems arising from smooth packet replay attacks. This OpenFlow-based SDN framework leverages the mutual benefits of the data plane and control plane used in the investigation to control the strength of DDoS attacks. This OpenFlow-based SDN framework is also important for managing and monitoring traffic flows that utilize SDN in the cloud. A fast and accurate

SDN-based DDoS attack prevention scheme has been proposed to solve the problems caused by flood-based DDoS attacks using the process of entropy variation. It has been found that this entropy coefficient of variation used in this accurate SDN-based DDoS attack defense scheme can enable an accurate and clear classification of normal and malicious traffic. The researcher found that the false positive rate of this entropy variable factor-based DDoS attack mitigation scheme was reduced by 12% compared to the mitigation framework focused on the OpenFlow-based monitoring process [9].

As per "Evaluating the Controller Capacity in Software ", an important issue with the OpenFlow architecture is the capacity of the controller. This can be defined as the number of switches that the controller can manage. This white paper models the switch-to-controller flow setup requirements as a batch arrival process Mk / M / 1. In addition, queuing theory is used to analyze controller performance to derive an equation for average flow uptime. In the situation of limited flow rate setting time, the number of switches is determined, which provides a way to evaluate the capacity of the regulator. The centralized controller is only responsible for creating policies. The researcher state that on management of 150 switches, only 20,000 new flow requests can be handled. However, this additional computational overhead for multiple parameters can impact controller performance. [1].

As per the paper "QoS improvement with an optimum controller selection for software-defined networks", the controller has multiple functions that guide the network from the center point and respond to updates related to topology changes. However, the ability to support these features is strong on one controller and weak on another. Choosing the best SDN controller can be considered a multiple-criteria decision-making problem (MCDM), as multiple controllers and each controller have many features. Here the researchers have proposed a two-step approach for choosing an SDN controller. First, the researcher classifies the controllers using the Analytical Network Process (ANP) according to the qualitative characteristics that affect the performance of these controllers, and then perform a performance comparison to see the improvement in QoS. Controllers with high weights from feature-based comparisons are analyzed quantitatively by experimental analysis. The researcher's main contribution is to confirm the applicability of ANP to controller selection in SDN, taking into account feature and performance analysis in real-world Internet and Brite topologies [3]. Choosing the best controller with ANP results in faster topology discovery times and delays in normal and traffic load scenarios. Researchers have also seen an increase in throughput with the controller, which makes good use of the central processing unit.

KNN classifies flows by measuring the distance between flow feature vectors. It's simple and effective [10]. Peng et al. We proposed a KNN-based method using a transudative confidence machine (TCM) for SDN anomaly detection [11]. Nam et al. Combined ANN and SOM to address the DDoS flood. Linear ANNs are very time consuming, so the Kth Dimensional tree (KD tree) stores training points in a tree structure for quick queries on ANNs [4]. However, the KD tree must create an index chain for all training units. Changes in training sessions affect detection accuracy [3].

Shin et al. Have proposed a system called Avant-Guard that can identify DDoS of TCP SYN floods including alarm services [12]. Line Switch then extended it to include a statistics collector. Also, Kotani et al. We proposed a packet filter mechanism to prevent attacks by Open flow for TCAM. The implementation here is focused only on software switches and open switch [13].

PATGEN, a Protocol to reduce the effects of DDoS Attacks using an advanced Genetic algorithm with optimized and new operators, thereby significantly reducing the effects of attacks and increasing the efficiency of the multi-controller SDN [14].

CHAPTER 3:

CONCEPTUAL MODEL

The Conceptual model is classified into two parts. One is Training and testing the proposed algorithm on NSL-KDD Data set and another part is to use the algorithm on simulated Environment.

3.1 **Conceptual Algorithm based on NSL-KDD Dataset**

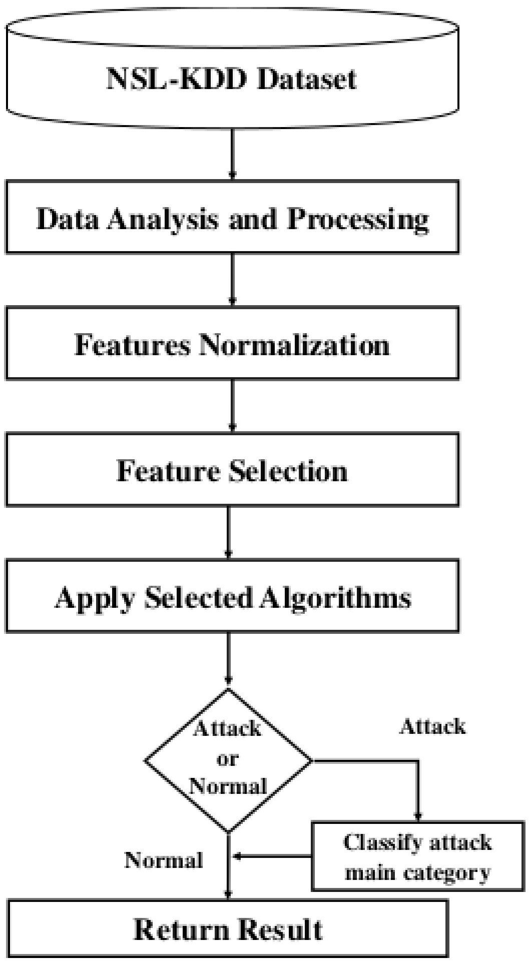


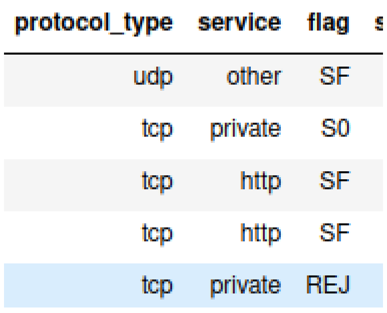
Figure 3-1 Conceptual Model of NSL-KDD Dataset

The procedure begins with the data source, the NSL-KDD dataset, and continues with data analysis and data cleaning processes. In order to implement the later specified tree-based machine learning algorithm to categorize whether or not there is an attack and the types of attack, feature normalization and feature selection are conducted [13].

3.1.1 Data Analysis and Processing

The Training set is divided as:

- Feature 'protocol_type' has 3 categories
- Feature 'service' has 70 categories
- Feature 'flag' has 11 categories
- Feature 'label' has 23 categories



protocol_type	service	flag
udp	other	SF
tcp	private	S0
tcp	http	SF
tcp	http	SF
tcp	private	REJ

Figure 3-1.1 Features in NSL-KDD Dataset

Similarly, distribution of categories in service can be done as

Protocols	Count
http	40338
private	21853
Domain_u	9043
smtp	7313
ftp_data	6860

Table 3-1.2 Distribution Domain in NSL-KDD Dataset

3.1.2 Features Normalization

For simplicity all the normal traffic is assumed as 0 and rest of all the traffic are assumed to be 1 for easy correlation of flags with the traffic. In Short Normalization is done by normalizing the standardized data into [0, 1]. It is assumed that Xij is the normalized value of X'ij [13].

$$X''ij = \frac{X'_{ij} - Xmin}{Xmax - Xmin} \quad (3.1.2)$$

Where,

$X_{min} = \min[X'_{ij}]$

$X_{max} = \max[X'_{ij}]$

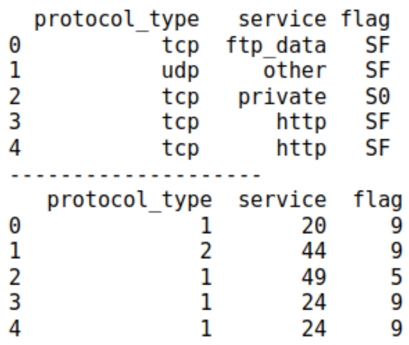
3.1.3 Standardization

Similarly, by analyzing the labels of attack, the attack types are evaluated

Attacks	Types
DoS_attacks	'apache2','back','land','neptune','mailbomb','pod','processtable', 'smurf','teardrop','udpstorm','worm
Probe_attacks	'buffer_overflow','loadmdoule','perl','ps','rootkit','sqlattack', 'xterm'
U2R	'ipsweep','mscan','nmap','portsweep','saint','satan'
R2L	'ftp_write','guess_passwd','http_tunnel','imap',' multihop','named','phf','sendmail','snmpgetattack', 'snmpguess','spy','warezclient','warezmaster','xclock','xsnoop

Table 3-1.3.1 Distribution of Attacks in NSL-KDD Dataset

Similarly label encoder and onehotEncoder are used to assign the string value to binary values.



```
  protocol_type   service flag
0           tcp  ftp_data   SF
1           udp     other   SF
2           tcp   private   S0
3           tcp      http   SF
4           tcp      http   SF
--------------------
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
3              1       24     9
4              1       24     9
```

Figure 3-1.3.2 One Hot Encoding of Protocols in NSL-KDD Dataset based on flags

$$X'_{ij} = \frac{X_{ij} - Mean_j}{AvgDev_j} \quad (3.1.3.1)$$

Where,

$$\text{Mean} = \frac{X_{1j}+X_{2j}+\cdots+X_{nj}}{n}$$

$$\text{AvgDev}_j = \frac{|X_{1j}-Mean_j|+\cdots+|X_{nj}-Mean_j|}{n}$$

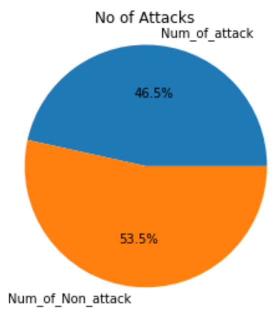


Figure 3-1.3.3 Features in NSL-KDD Dataset

Standardization is given by

$$\text{Standardization} = \frac{\text{Sum of type of Traffic(Normal or Attack)}}{\text{length}}) \qquad (3.1.3.2)$$

where Attack types can be 'DoS/DdoS, Probe, U2R, R2L

```
Normal =  0.5346505572666942
DoS/DDoS =  0.3645810180040009
Probe =  0.09252849839646905
U2R =  0.00034134569586892325
R2L =  0.007898580636966945
```

Figure 3-1.3.4 Distribution of Type of Attacks in NSL-KDD Dataset

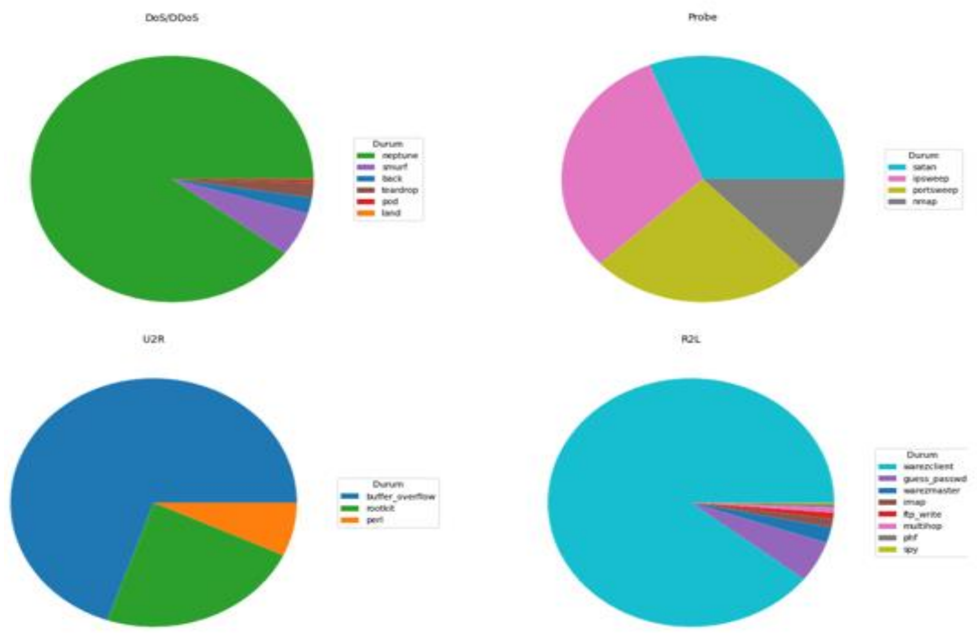


Figure 3-1.3.5 Standard Distribution of Type of Attacks in NSL-KDD Dataset

3.1.4 Features Selection

To improve the effectiveness of data mining algorithms, feature selection is critical. The majority of the data contains features that are irrelevant, redundant, or noisy. Feature selection is a technique in data mining for dimension reduction that involves picking a subset of original features based on particular criteria. It is an important and widely used approach. It brings about palpable results for applications by reducing the amount of features, removing unnecessary, redundant, or noisy features, and speeding up a data mining method, boosting learning accuracy, and leading to higher model comprehensibility. For feature reduction, there are two main ways. A Wrapper analyzes the usefulness of features using the intended learning process, whereas a filter evaluates features using heuristics based on the data's general properties. Although the wrapper approach is thought to create superior feature subsets, it is slower than a filter. In this purpose two feature subset selection methods are used Correlation-based Feature Selection (CFS) and Information Gain (IG) to compare our proposed method [19].

I. Correlation-based Feature Selection

CFS has two concepts. One is the feature-classification (r cfi) correlation and another is the feature-feature (r fifj) correlation. These two ideas are based on the theory that "good feature subsets comprise features that are substantially linked with the classification but not with each other." The feature-classification correlation shows how closely a characteristic is linked to a certain class. The correlation between two characteristics is known as the feature-feature correlation [19].

$$\text{Ms(k)} = \frac{K(r_{ef})^c}{\sqrt{k+k(k=1)(r_{ff})^c}} \qquad (3.1.4.1)$$

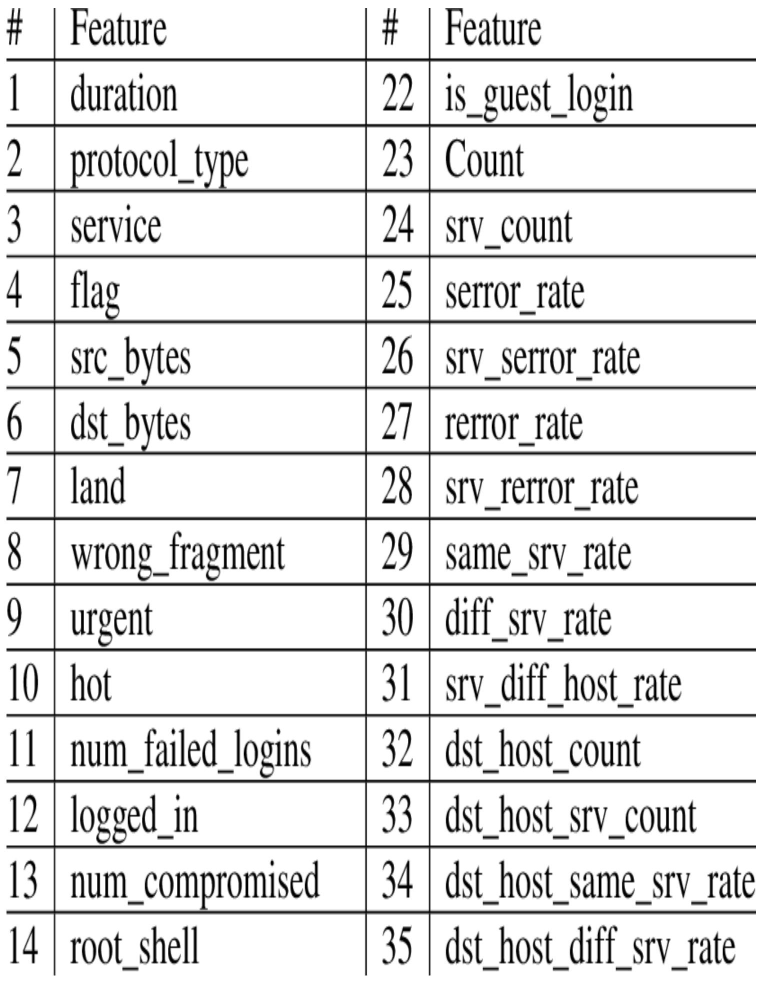
Which is average feature classification correlation to average feature to feature correlation

II. Information Gain (IG)

By analyzing the gain of each variable in the context of the targeted variables, information gain may also be utilized for feature selection. The calculation is referred to as mutual information between the two random variables in this slightly different usage [19].

$$Gain(F) = I(C_1, \ldots, C_m) - E(F) \tag{3.1.4.2}$$

NSL-KDD Consist of 41 different features where only few features play role on our purpose algorithm



#	Feature	#	Feature
1	duration	22	is_guest_login
2	protocol_type	23	Count
3	service	24	srv_count
4	flag	25	serror_rate
5	src_bytes	26	srv_serror_rate
6	dst_bytes	27	rerror_rate
7	land	28	srv_rerror_rate
8	wrong_fragment	29	same_srv_rate
9	urgent	30	diff_srv_rate
10	hot	31	srv_diff_host_rate
11	num_failed_logins	32	dst_host_count
12	logged_in	33	dst_host_srv_count
13	num_compromised	34	dst_host_same_srv_rate
14	root_shell	35	dst_host_diff_srv_rate

Figure 3-1.4.1 different features in NSL-KDD Dataset

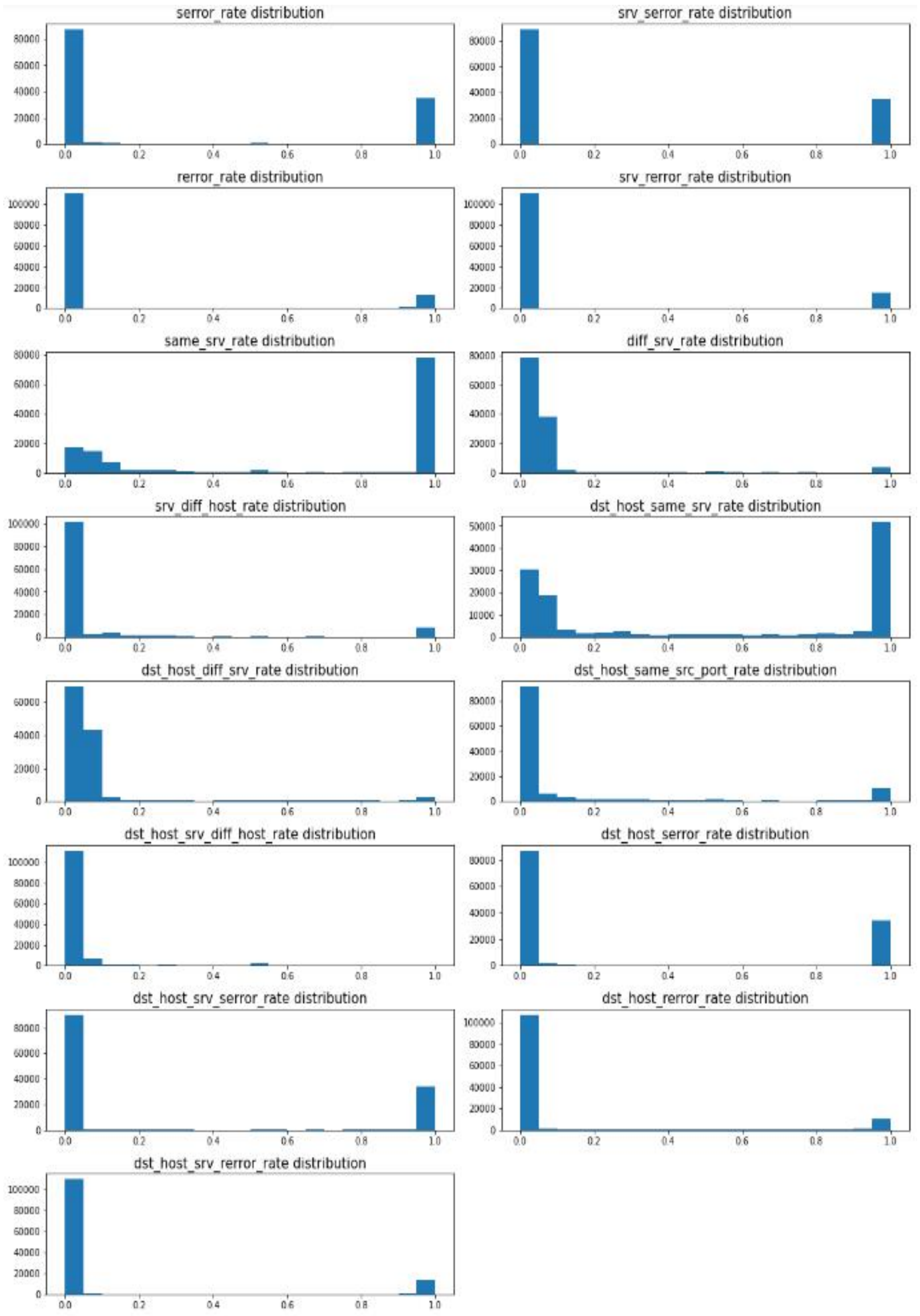


Figure 3-1.4.2 Correlations between 42 different features in NSL-KDD Dataset

Various elements such as flag, protocol, and services are collected from the data collection and examined and connected with the attack and attack status using correlation and information gain (mutual information between two variables). The above features are used for classifications into different types of attack.

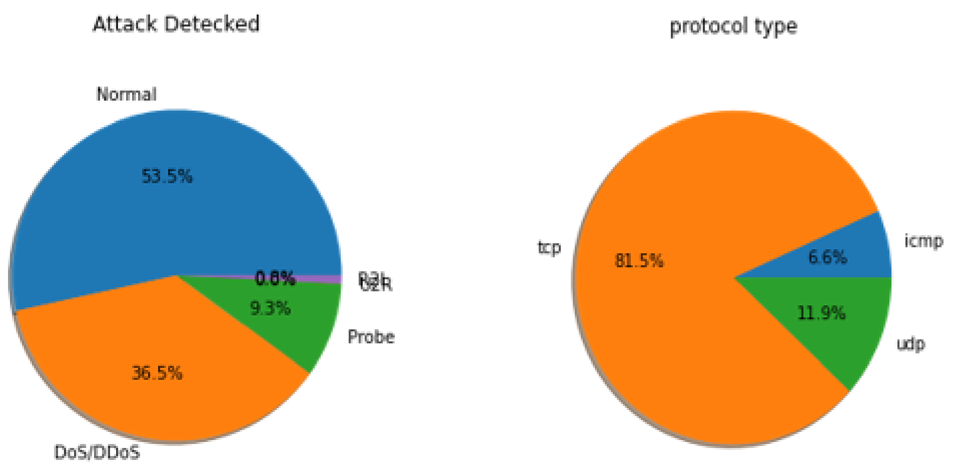


Figure 3-1.4.3 Correlations between Attack type and protocol with correlated of features

From above features and correlation to types of attack, volume based attacks are separated based on their attack types. These are Volume Based DDoS Attack which can be done by manipulation as shown below

3.1.5 Types of Attacks

Types of Attacks	**Carried out by**
UDP and SYN packets Protocol Based DDoS Attack	With vulnerabilities on TCP/IP and OSI packets
Flood DDoS Attack Ping Flood DDoS Attack	Via ICMP protocol
SYN Flood DDoS Attack	With TCP/IP manipulation UDP Flood is in the form of DDoS Attack Distribution of protocol types in DDoS attacks on the dataset

Table 3-1.5 Type of attack with vulnerabilities in NSL-KDD Dataset

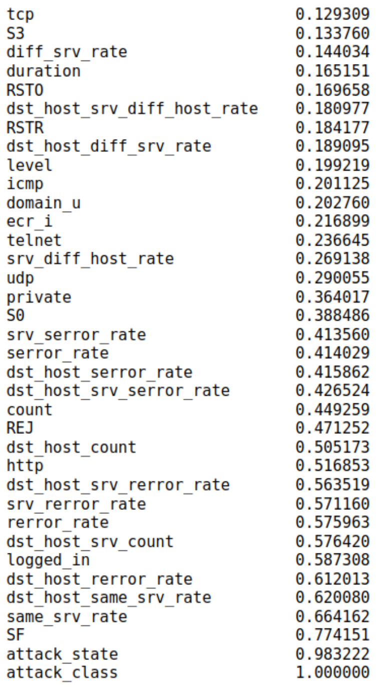
Generally, DoS/DDoS attacks have the highest rate of 36% in the dataset, where U2R and R2L attacks are very rare [Performance Analysis of Network Intrusion Detection System using Machine Learning, Abdullah Alsaeedi]. This shows that our dataset is directly proportional to today's internet traffic attacks where For attack class is chosen as 1 for the target variable.

3.1.6 Applying selected algorithm

By default, NSL-KDD dataset are divided into trained and test data set. From the training data set, the column labeled for ‘attack’ is dropped and correlation between test data set and training data set without attack column is evaluated.

The correlated data is used to evaluated weather the traffic along with its protocol associated with its flag is weather “Normal” or “Attack” Traffic.

For a training set that contains n elements $\{(x_1 , y_1), \cdot\cdot\cdot , (x_n , y_n)\}$, $X_i (x_{i1} , x_{i2} , \cdot\cdot\cdot , x_{it})$ is the value set of the t features that are extracted from the detection point i, while i is the category that the detection point i belongs to. Usually, it ranges from 1 to c [20]. Elements of the anomaly detection systems are divided into two parts, namely, normal elements and Attack Traffic to classify the detection points, the following algorithms can be used.



```
tcp                             0.129309
S3                              0.133760
diff_srv_rate                   0.144034
duration                        0.165151
RSTO                            0.169658
dst_host_srv_diff_host_rate     0.180977
RSTR                            0.184177
dst_host_diff_srv_rate          0.189095
level                           0.199219
icmp                            0.201125
domain_u                        0.202760
ecr_i                           0.216899
telnet                          0.236645
srv_diff_host_rate              0.269138
udp                             0.290055
private                         0.364017
S0                              0.388486
srv_serror_rate                 0.413560
serror_rate                     0.414029
dst_host_serror_rate            0.415862
dst_host_srv_serror_rate        0.426524
count                           0.449259
REJ                             0.471252
dst_host_count                  0.505173
http                            0.516853
dst_host_srv_rerror_rate        0.563519
srv_rerror_rate                 0.571160
rerror_rate                     0.575963
dst_host_srv_count              0.576420
logged_in                       0.587308
dst_host_rerror_rate            0.612013
dst_host_same_srv_rate          0.620080
same_srv_rate                   0.664162
SF                              0.774151
attack_state                    0.983222
attack_class                    1.000000
```

Figure 3-1.6.1 Correlation of Attack in NSL-KDD Dataset

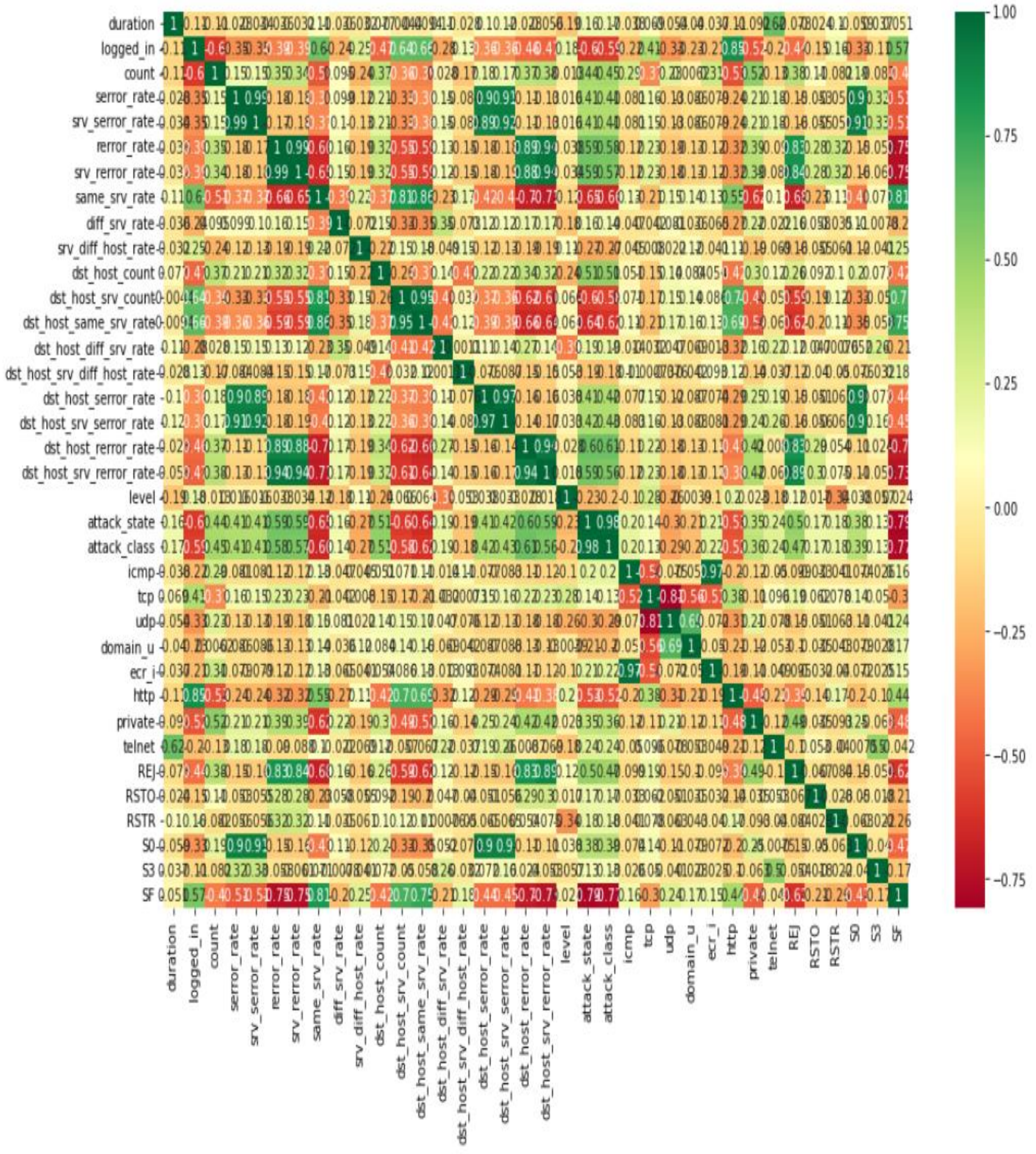
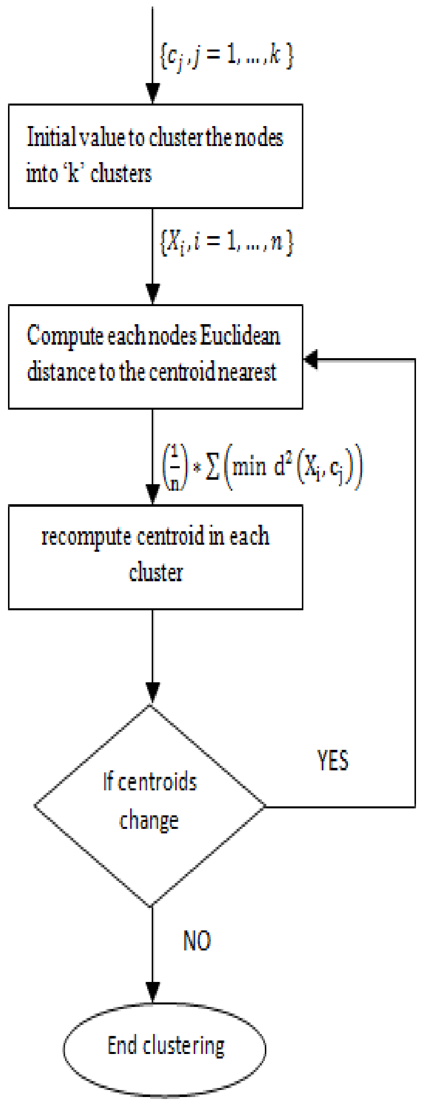


Figure 3-1.6.2 Correlation of Attack to the attributes in NSL-KDD Dataset

For detection of attack correlated with the attributes along with protocols and flag, various algorithms can be used. For this, the machine learning algorithm with best accuracy and precision along with response time is chosen.

I. K-means cluster (purposed)



$\{c_j, j = 1, \ldots, k\}$
Initial value to cluster the nodes into 'k' clusters
$\{X_i, i = 1, \ldots, n\}$
Compute each nodes Euclidean distance to the centroid nearest
$\left(\frac{1}{n}\right) * \sum\left(\min d^2(X_i, c_j)\right)$
recompute centroid in each cluster
If centroids change
YES
NO
End clustering

Figure 3-1.7.1 K means analysis on NSL-KDD Dataset

From the above selection and normalization criteria along with the flags and traffic type, the total number is evaluated which is 2 I.e. one cluster for normal traffic type and another for attack type. Then assign each data point to the closest centroid for predefined value of k and calculate variance for a new place of centroid for each cluster. In this case Euclidean distance is used for the nearest centroid value.

Where, $d = \sqrt((x1-y1)^2 + (x2-y2)^2$ (3.1.7.1)

This process is repeated until the model gets ready.

II. KNN

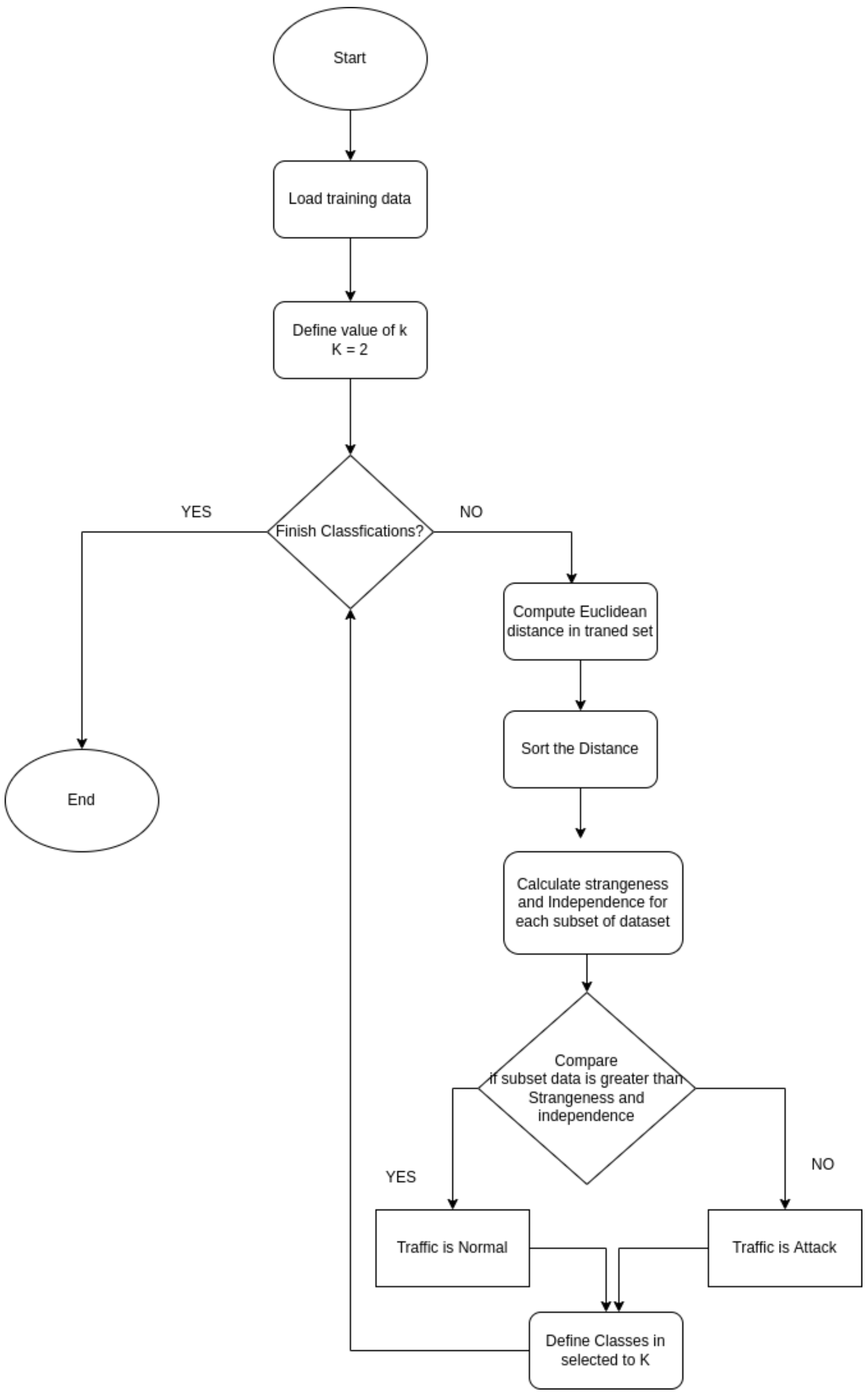

Start
Load training data
Define value of k
K = 2
YES
Finish Classfications?
NO
Compute Euclidean distance in traned set
Sort the Distance
End
Calculate strangeness and Independence for each subset of dataset
Compare
if subset data is greater than Strangeness and independence
YES
NO
Traffic is Normal
Traffic is Attack
Define Classes in selected to K

Figure 3-1.7.2 KNN analysis on NSL-KDD Dataset

The starting value of k is chosen to be 2 based on the selection criteria and features from the training and test data sets. The Euclidean distance between K=2 neighbors is computed. The values are sorted among the K neighbors.

The greater the distance between two points, the bigger the difference between them [21].

$$D_{ij}^{y} = \sqrt{\sum_{a=1}^{t}\left(X_{ia} - X_{ja}\right)^{2}} \qquad (3.1.7.2)$$

The normal and abnormal elements of anomaly detection systems are believed to belong to y and -y, respectively. When the KNN algorithm is used to sort the Euclidean distances between I and the other points in category y. Strangeness is given by

$$\boldsymbol{\alpha}_{\mathbf{iy}} = \frac{\sum_{j=1}^{k} D_{ij}^{y}}{\sum_{j=1}^{k} D_{ij}^{-y}} \qquad (3.1.7.3)$$

The greater the strangeness is, the less likely it is that the point belongs to the category. Independence is the sum of the Euclidean distances between a detection point and its k nearest neighbors, namely, the absolute deviation between the two.

$$\boldsymbol{\theta}_{\mathbf{ij}} = \sum_{j=1}^{k} \boldsymbol{D}_{ij}^{y} \qquad (3.1.7.4)$$

Strangeness describes the distance between a detection point and a category, whereas independence examines the absolute distance between the two and looks into anomalous points that are distant from the normal points. The value of Strangeness and Independence are used to classify whether the traffic is Normal or Attack based on the value of K [21].

III. SVM

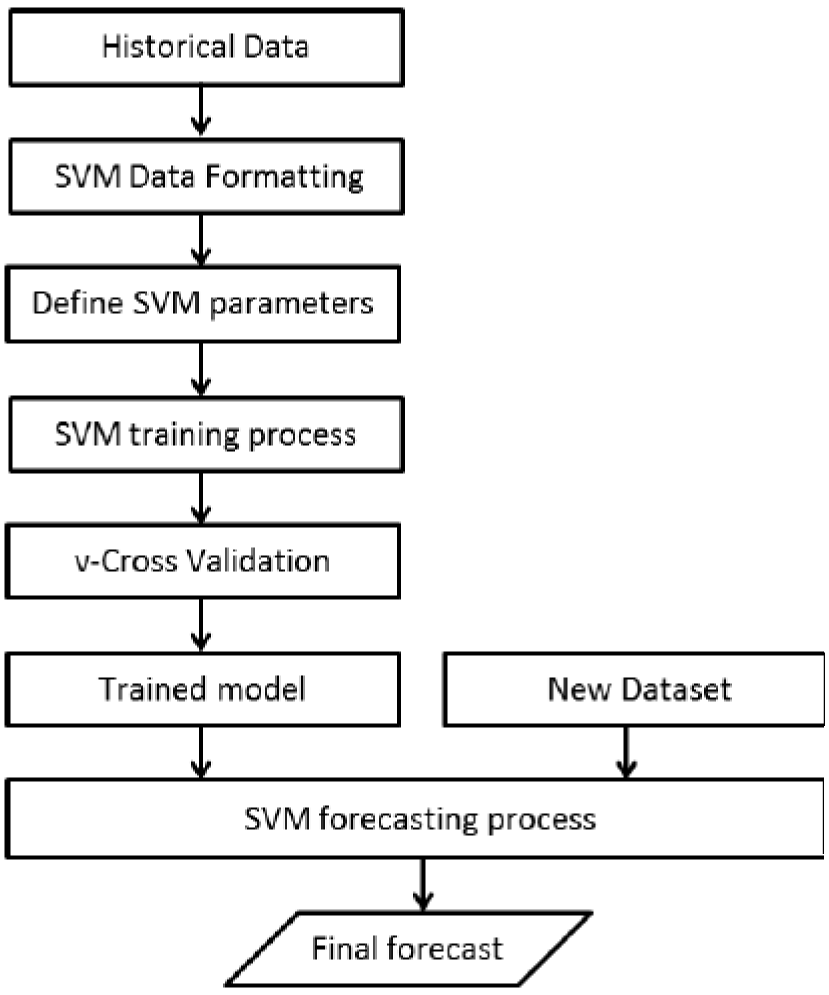


Figure 3-1.7.3 SVM analysis on NSL-KDD Dataset

The suggested technique determined an appropriate hyperplane for classification between two fields based on their attributes. The size of the hyperplane was determined by the attributes. The hyperplane was challenging to construct since the flag value was more than 3d.

$$C(x, y, f(x)) = 0 \ \ if\ y * f(x) \geq 1 \ \ or\ \ 1 - y * f(x) \qquad (3.1.7.5)$$

If the projected and actual values have the same sign, the cost is zero. The regularization parameter's goal is to strike a compromise between margin maximization and loss. After adding the regularization parameter, the cost functions look as

$$minn_w \lambda \mid |w \mid|2 + \sum_{i=1}^{n} \left(1 - y_i(x_i, w)\right)_+ \qquad (3.1.7.6)$$

When there is no misclassification, i.e. our model correctly predicts the class of our data point, only gradient from the regularization is updated

$$w = w - \boldsymbol{\alpha}.\,(\mathbf{2}.\,\boldsymbol{\lambda}.\,\mathbf{w}) \qquad (3.1.7.7)$$

When there is miscalculation, loss and regularization parameters are updates.

$$w = w + \boldsymbol{\alpha}.\,(\mathbf{y_i}.\,\mathbf{x_i} - \mathbf{2}.\,\boldsymbol{\lambda}.\,\mathbf{w}) \qquad (3.1.7.8)$$

3.2 Conceptual Algorithm based on Simulated Environment

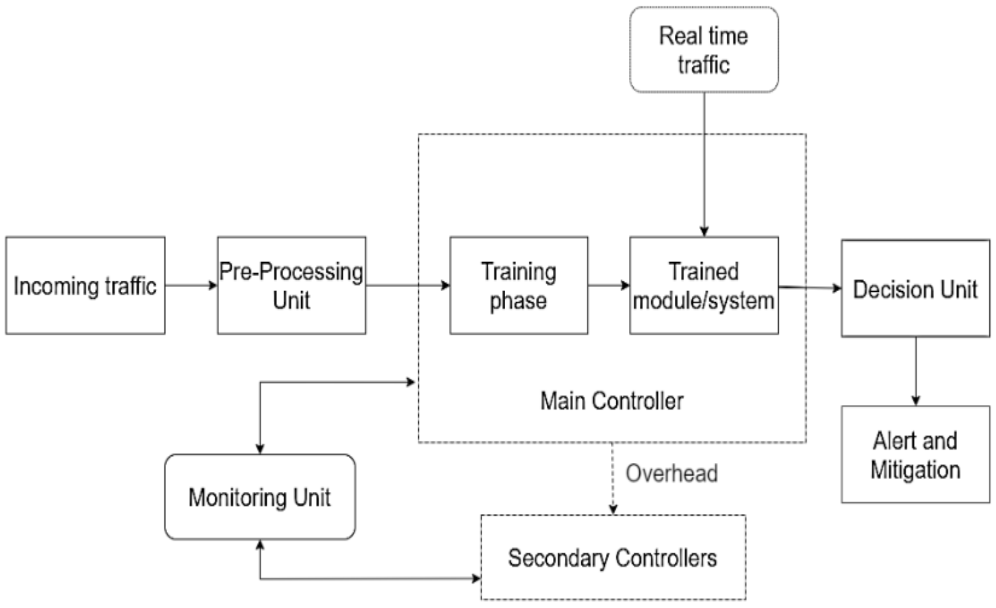


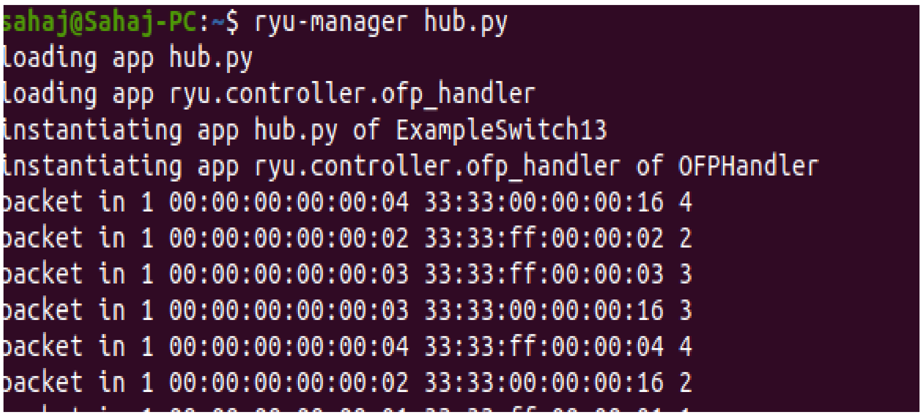
Figure 3-2 Conceptual Algorithm based on Simulated Environment

3.2.1 Incoming traffic:

The normal traffic and attack traffic generated in controlled environment is the input to this block. The traffic is segregated in this block for further processing or precisely to pass to pre-processing unit.

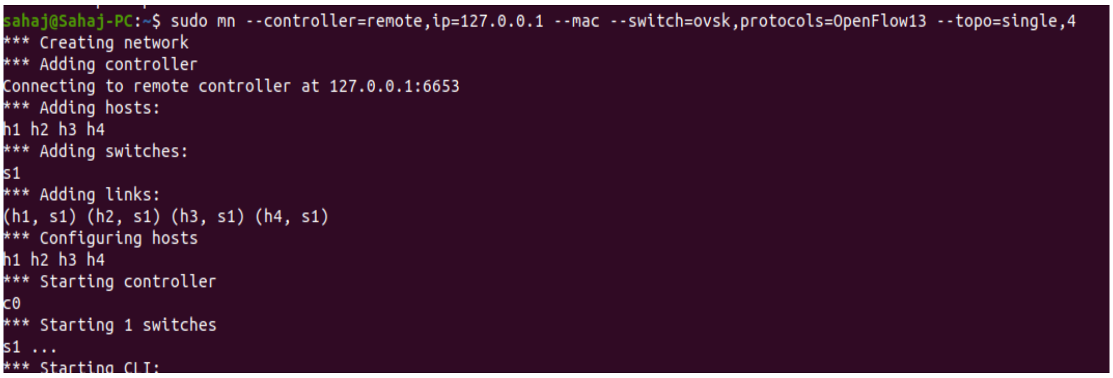
Because TCP contains built-in congestion control and a reliable transmission method, the preceding solution is possible. It's worth noting that TCP is the most common type of Internet traffic, with TCP accounting for up to 90% of DDoS activity. TCP currently accounts for 80% of the total number of flows compared to the total number of packets. As a result, it is critical for DDoS defense techniques to effectively and efficiently accept TCP traffic. The framework in SDN against

DoS attack should address the defense strategies for both data plane attacks and controller plane attacks [21].



```
sahaj@Sahaj-PC:~$ ryu-manager hub.py
loading app hub.py
loading app ryu.controller.ofp_handler
instantiating app hub.py of ExampleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:04 33:33:00:00:00:16 4
packet in 1 00:00:00:00:00:02 33:33:ff:00:00:02 2
packet in 1 00:00:00:00:00:03 33:33:ff:00:00:03 3
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:04 33:33:ff:00:00:04 4
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
```

Figure 3-2.1.1.TCP Packet flow in RYU Switch



```
sahaj@Sahaj-PC:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

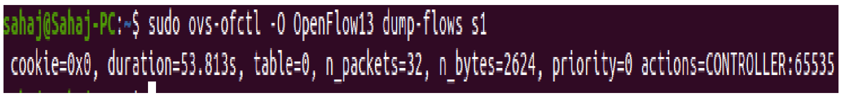
Figure 3-2.1.2 Mininet Topology connections

I. Defense Against DDoS Attack on Data plane

Switches in SDN operate as forwarding devices with little intelligence, and the centralized controller makes all the decisions. When there is a lot of traffic, the controller plane's bandwidth will grow, lowering the controller's performance. As a result of OpenFlow SDN switches' inability to address attacks on their own, these dump switches increase the data plane's vulnerability, as it may be readily targeted by attackers. [23]. When the attack rate increases and there are insufficient resources (switches) for redirection, performance suffers. Furthermore, there is no mechanism for detecting the attack, making it an ineffective tactic for a comprehensive answer to DDoS attacks.

II. Defense Against DDos Attack on Control Plane

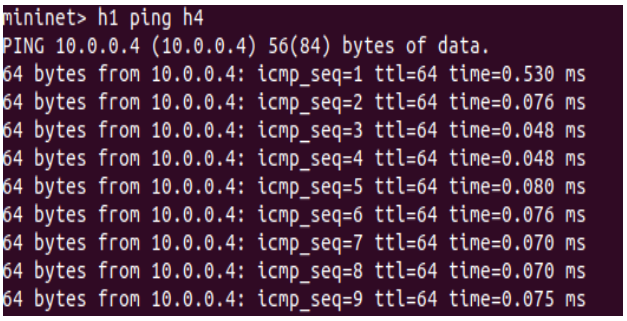
The controller is the brain of the SDN network, providing the network with comprehensive visibility and intelligence. DDoS assaults are most likely to target the centralized SDN controller. As a result, a quick DDoS detection and mitigation approach for the controller is required. The controller's protection mechanisms are primarily focused on avoiding resource saturation as fast as possible [23]. TCP connections impose a large and inevitable delay. Furthermore, this framework is only suitable for TCP-SYN assaults, and the need for switch modification in a practical deployment is undesired. Such attacks can be prevented by using a proactive flow control role, which lowers controller overhead but increases delay.



```
sahaj@Sahaj-PC:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=53.813s, table=0, n_packets=32, n_bytes=2624, priority=0 actions=CONTROLLER:65535
```

Figure 3-2.1.3 Ovs Switch Flow Parameters

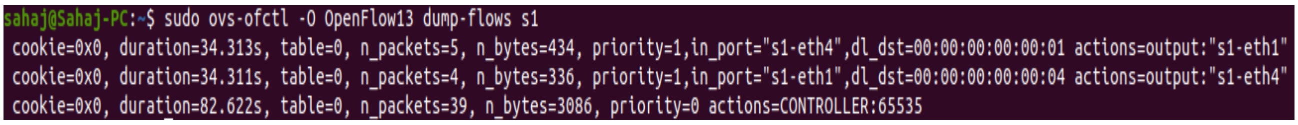
Initially since there is no flow the priority of traffic is 0 with no traffic flow.



```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.530 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.048 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.080 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.076 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.070 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.070 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.075 ms
```

Figure 3-2.1.4 Ping from Host 1 to Host 3

After a flow is introduce to it OpenFlow controller will install flow tables ahead of time for all traffic matches. Since the traffic flow is between h1 to h4, the controller automatically installs the flow ahead with increased priority.

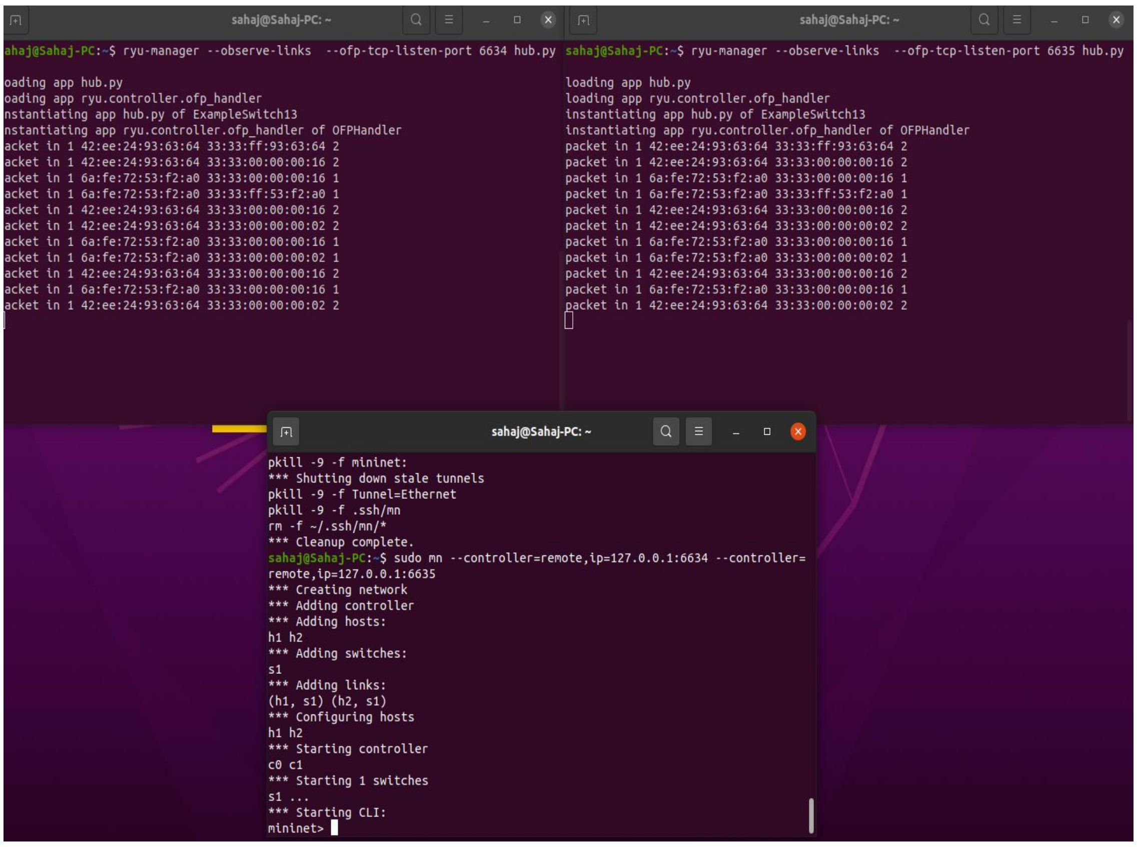


```
sahaj@Sahaj-PC:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=34.313s, table=0, n_packets=5, n_bytes=434, priority=1,in_port="s1-eth4",dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=34.311s, table=0, n_packets=4, n_bytes=336, priority=1,in_port="s1-eth1",dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=82.622s, table=0, n_packets=39, n_bytes=3086, priority=0 actions=CONTROLLER:65535
```

Figure 3-2.1.5 Ovs Switch Flow traffic flow from port 1 to port 3

III. Defense by Integrating Intelligence in Switches

Switches are essentially forwarding devices in the SDN environment because the controller is responsible for all switch decision-making. As a result, the controllers become overloaded, and the channels get congested. The fact that SDN switches are simply forwarding devices increases communication overhead, attack detection delay, and controller congestion. These issues can be overcome by putting intelligence into the switches, which will reduce controller overload and bandwidth [23]. As a result, malicious activity can be immediately recognized at the switch level. The detection of the suspicious traffic is performed by using a multiple machine learning algorithm of K-Means, KNN and SVM using 3 statistical features. the limitation of existing systems includes SDN design problem like the overhead of controller, single-point failure can be mitigated if the traffic can be rerouted to another controllers using intelligent traffic control and controller selection model.



```
ahaj@Sahaj-PC:~$ ryu-manager --observe-links  --ofp-tcp-listen-port 6634 hub.py

oading app hub.py
oading app ryu.controller.ofp_handler
nstantiating app hub.py of ExampleSwitch13
nstantiating app ryu.controller.ofp_handler of OFPHandler
acket in 1 42:ee:24:93:63:64 33:33:ff:93:63:64 2
acket in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
acket in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
acket in 1 6a:fe:72:53:f2:a0 33:33:ff:53:f2:a0 1
acket in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
acket in 1 42:ee:24:93:63:64 33:33:00:00:00:02 2
acket in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
acket in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:02 1
acket in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
acket in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
acket in 1 42:ee:24:93:63:64 33:33:00:00:00:02 2
```

```
sahaj@Sahaj-PC:~$ ryu-manager --observe-links  --ofp-tcp-listen-port 6635 hub.py

loading app hub.py
loading app ryu.controller.ofp_handler
instantiating app hub.py of ExampleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 42:ee:24:93:63:64 33:33:ff:93:63:64 2
packet in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
packet in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
packet in 1 6a:fe:72:53:f2:a0 33:33:ff:53:f2:a0 1
packet in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
packet in 1 42:ee:24:93:63:64 33:33:00:00:00:02 2
packet in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
packet in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:02 1
packet in 1 42:ee:24:93:63:64 33:33:00:00:00:16 2
packet in 1 6a:fe:72:53:f2:a0 33:33:00:00:00:16 1
packet in 1 42:ee:24:93:63:64 33:33:00:00:00:02 2
```

```
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
sahaj@Sahaj-PC:~$ sudo mn --controller=remote,ip=127.0.0.1:6634 --controller=
remote,ip=127.0.0.1:6635
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0 c1
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figure 3-2.1.5 Traffic routing between two controllers

3.2.2 Pre-processing unit:

In this block, the input traffic is operated on by statistical analysis to generate various feature vectors also known as representative parameters like Speed of IP sources (SSIP), Speed of flow entries (SFE) and Ratio of pair flow entries (RPE) to generate a set of samples known as training sample dataset [23].

I. Speed of IP sources (SSIP):

The parameter is given by the formula

$$SSIP = \frac{sumIP_{src}}{T} \tag{3.2.2.1}$$

where, the total number of IP sources incoming in each flow is sumIP$_{src}$, and the sampling time intervals are T. The time period T is set to three seconds, causing the detection system to monitor and gather data from flows every three seconds, as well as store the number of source. IPs during that time. For the machine learning system to predict attacks, the controller must have enough data of both regular and attack traffic. The SSIP for benign traffic is usually low, whereas the count for attacks is frequently larger.

II. Speed of flow entries (SFE):

The parameter is given by the formula

$$SFE = \frac{N}{T} \tag{3.2.2.2}$$

where, N gives the total number of flow entries entering into a switch of a network and T denotes a particular time period. This is an extremely relevant feature of attack traffic detection, since in the event of DDOS attacks, the number of flow entries increases significantly in a fixed time interval as opposed to normal traffic flows.

III. Ratio of pair flow entries (RPE):

It is given by the formula

$$RPE = \frac{srcIP_S}{N} \tag{3.2.2.3}$$

where, N represents the total number of Ips, and srcIPs represents the number of collaborative IPs in the network flow. In each sampling, the representative flow table entries in every sampling are calculated to form a sample set P, which is written as Q = (P,S), where P represents flow table entries of triplet-characteristics value matrix, and S denotes the marker vector for category corresponding to P: "0" denotes the normal condition and "1" denotes the attack condition. The generated sample set is then fed to SVM classifier to train the model for further detection of real time traffic. There is a training set D = [(P 1 , s 1), . . . , (P n , s n)], where Pi is the characteristic vector, whereas s i is the corresponding class label. In this experiment, s_i belongs to (-1,+1) takes the value of either 1 or 0.

3.2.3 Main controller:

It is the block where the main operation of classification of DDoS traffic from benign or normal traffic takes place. The block can be further divided into training phase unit and trained module. Training phase unit optimizes the proposed algorithm for further clustering of incoming traffic. In order to determine the type of the traffic received from the network; the characteristics discussed in the theoretical section needs to be modeled. The parameters are calculated on basis of incoming traffic as Trained module is block to which real time incoming traffic is passed for classification into normal traffic and attack traffic.

3.2.4 Secondary controller:

If there is maximum overhead or load on the main controller, the controlling operation is transferred to this secondary controller and all the incoming traffic passes through it consequently.

3.2.5 Decision unit:

This block makes the decision to either classify the incoming traffic as an attack and detect the port from which the attack traffic is coming if there is DDoS attack on the network. There are various roles that a controller can play. The three roles are:

I. Equal

By default, all equal controllers in a network linked to the switch have the ability to add and delete flows from it. It means that all of the switch's controllers have complete control over the flow updates and modifications. The PACKET IN Message must be sent to all Controllers by the Switch. Also, all of the controllers' PACKET OUT, FLOW UPDATES, and so on are processed by the switch.

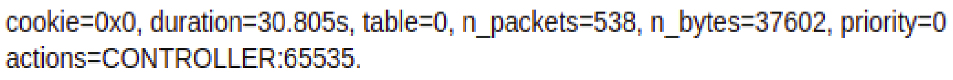
II. Master:

Similar to a switch, except only one master can be connected to it at any given time, and no other flows can intercept the communication between the switch and the master. It means that the MASTER controller will be in charge of the switch openflow dataplane management. Only the MASTER controller will receive control signals from the switch.

III. Slave:

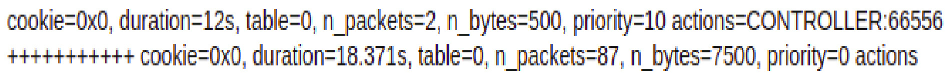
It has a read-only limitation to the switch openflow tables. Slave plays backup role for MASTER. it also receives the HELLO and KEEPALIVE Message. But it cannot send and receive the Control message.

Since the default role of each controller is "Equal". The controller's job will be performed by the SDN switch. This is due to the fact that a single switch can be connected to multiple controllers. With respect to that switch, each controller can play a different role. A single controller can serve as both a slave and a master for different switches. This is why the controller's role is present in the switch, because the controller code is effectively the same.



```
cookie=0x0, duration=30.805s, table=0, n_packets=538, n_bytes=37602, priority=0
actions=CONTROLLER:65535.
```

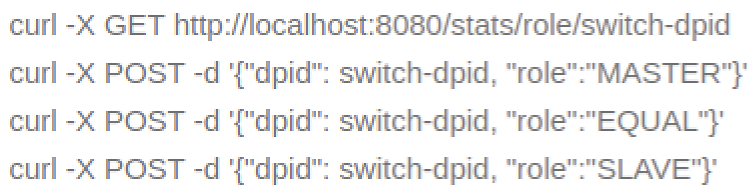
Figure 3-2.5.1 Flow Table for Controller 1



```
cookie=0x0, duration=12s, table=0, n_packets=2, n_bytes=500, priority=10 actions=CONTROLLER:66556
+++++++++++ cookie=0x0, duration=18.371s, table=0, n_packets=87, n_bytes=7500, priority=0 actions
```

Figure 3-2.5.2 Flow Table for Controller 2

The switch can thus be used to retrieve the role. The job of the controller can be modified depending on what function it now performs for the switch.

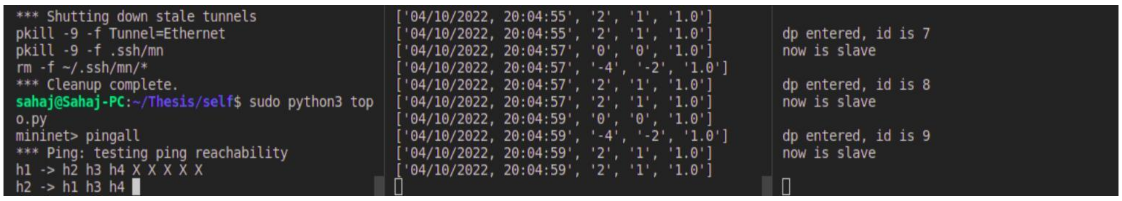


```
curl -X GET http://localhost:8080/stats/role/switch-dpid
curl -X POST -d '{"dpid": switch-dpid, "role":"MASTER"}'
curl -X POST -d '{"dpid": switch-dpid, "role":"EQUAL"}'
curl -X POST -d '{"dpid": switch-dpid, "role":"SLAVE"}'
```

Figure 3-2.5.3 Controller Role Request

IV. ROLE_REQUEST and RESPONSE:

When the controller comes up, it will send the ROLE REQUEST with ROLE MASTER, other controller should send a role as SLAVE. Switch will communicate with MASTER.



```
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
sahaj@Sahaj-PC:~/Thesis/self$ sudo python3 top
o.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X X X X X
h2 -> h1 h3 h4
```

```
['04/10/2022, 20:04:55', '2', '1', '1.0']
['04/10/2022, 20:04:55', '2', '1', '1.0']
['04/10/2022, 20:04:57', '0', '0', '1.0']
['04/10/2022, 20:04:57', '-4', '-2', '1.0']
['04/10/2022, 20:04:57', '2', '1', '1.0']
['04/10/2022, 20:04:57', '2', '1', '1.0']
['04/10/2022, 20:04:59', '0', '0', '1.0']
['04/10/2022, 20:04:59', '-4', '-2', '1.0']
['04/10/2022, 20:04:59', '2', '1', '1.0']
['04/10/2022, 20:04:59', '2', '1', '1.0']
```

```
dp entered, id is 7
now is slave

dp entered, id is 8
now is slave

dp entered, id is 9
now is slave
```

Figure 3-2.5.4 Traffic Flow between master slave with Request and Response

Data intelligence is produced by the controller in a multi controller environment (particularly Master/slave). Many techniques are available, including DB, InMemoryDB, Message Brokers, and so on.

Further experiments for setting up multi controllers are

- Use InmemoryDB or Message Broker for Data Synchronization
- Implement Master Election Algorithm / Autofailover
- Distributed Master/Slaves

The stability of the controller can be used to determine the state of controller overhead. If Stability is 1, the controller is assumed to be insecure, and the entire node is optimized. For controllers having a stability of less than 1 [25]. The flag is set to 0.

The stability parameter is given by

$$Stability = \frac{\text{Number of current requests reaching the controller}}{\text{Number of completed requests of controller}} \qquad (3.2.5)$$

CHAPTER 4

MODEL VERIFICATION/ EXPERIMENTATION

The working principle of the project can be broadly classified into four different subsequent stages: training of model, detection of attack traffic, mitigation of DDoS attack and controller switching on overhead detection.

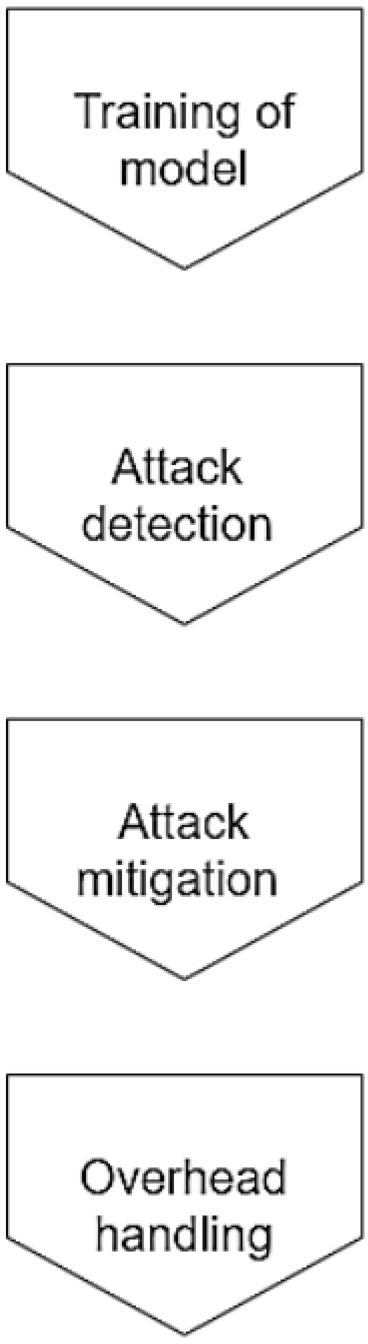


Figure 4 Experimental Model from proposed Algorithm

4.1 Training of model:

The switch communicates with the controller using the OpenFlow switch protocol, and the controller manages the switch. The controller can add, update, and delete flow entries in flow tables both reactively (in reaction to packets) and proactively (in advance) using the OpenFlow switch protocol. Each flow table in the switch has a collection of flow entries, which are made up of match fields, counters, and instructions to apply to matched packets. The matching process begins with the first flow table and may extend to other flow tables in the pipeline. Flow entries are used to match packets in order of priority, with the first matching entry in each table being used. The instructions associated with the specific flow entry are performed if a matching entry is identified. If no match is discovered in a flow table, the outcome is determined by the table-miss flow entry's setting.: for example, the packet may be forwarded to the controllers over the OpenFlow channel, dropped, or may continue to the next flow table. Packet forwarding, packet modification, and group table processing are all described in the instructions. Flow entries can be routed to a specific port. This is normally a physical port, but it could also be a switch-specified logical port (such as link aggregation groups, tunnels, or loopback interfaces) or a reserved port defined by this specification.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 4-1.1 Open Flow Table with fields

A flow table entry is identified by its match fields and priority: the match fields and priority taken together identify a unique flow entry in a specific flow table.

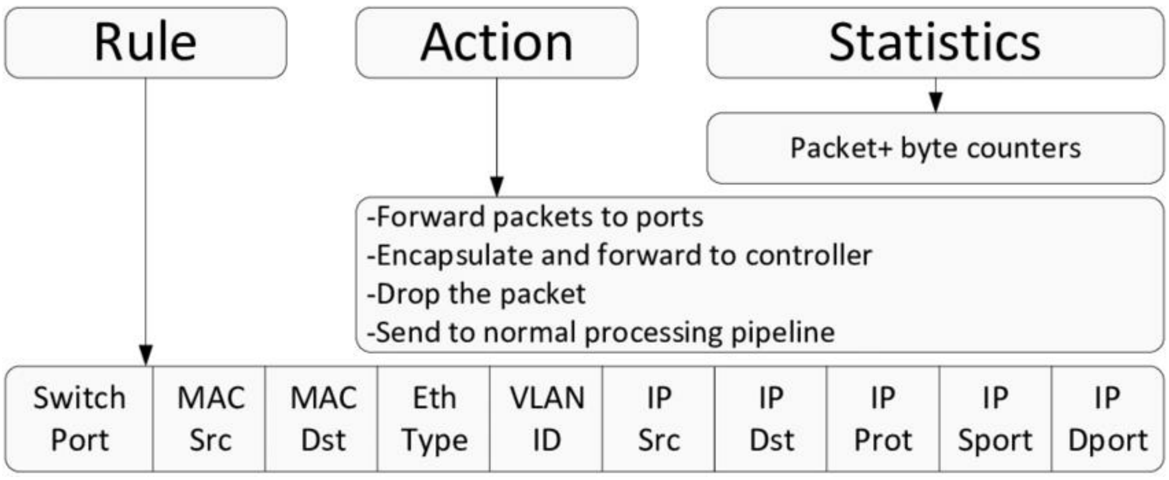
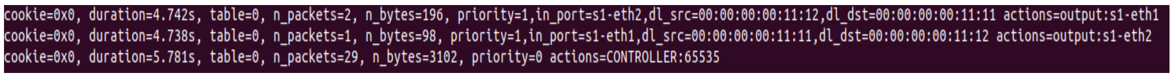


Figure 4-1.2 Detail Open Flow Table with Actions

A typical example of Match can be

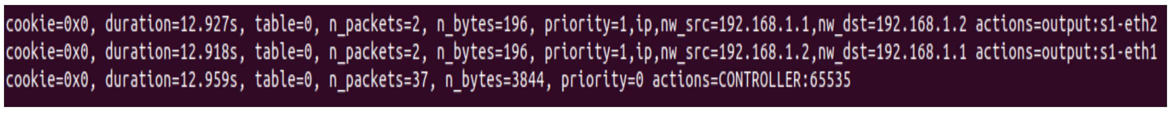
I. Match using Mac



```
cookie=0x0, duration=4.742s, table=0, n_packets=2, n_bytes=196, priority=1,in_port=s1-eth2,dl_src=00:00:00:00:11:12,dl_dst=00:00:00:00:11:11 actions=output:s1-eth1
cookie=0x0, duration=4.738s, table=0, n_packets=1, n_bytes=98, priority=1,in_port=s1-eth1,dl_src=00:00:00:00:11:11,dl_dst=00:00:00:00:11:12 actions=output:s1-eth2
cookie=0x0, duration=5.781s, table=0, n_packets=29, n_bytes=3102, priority=0 actions=CONTROLLER:65535
```

Figure 4-1.3 Matching using MAC Address

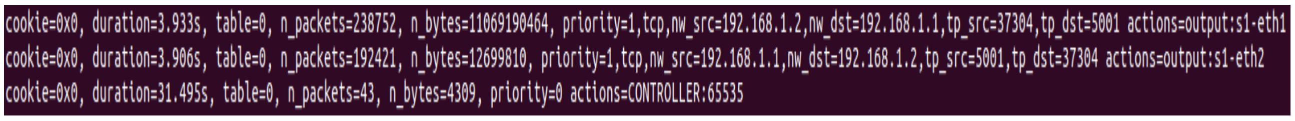
II. Match using Ip address



```
cookie=0x0, duration=12.927s, table=0, n_packets=2, n_bytes=196, priority=1,ip,nw_src=192.168.1.1,nw_dst=192.168.1.2 actions=output:s1-eth2
cookie=0x0, duration=12.918s, table=0, n_packets=2, n_bytes=196, priority=1,ip,nw_src=192.168.1.2,nw_dst=192.168.1.1 actions=output:s1-eth1
cookie=0x0, duration=12.959s, table=0, n_packets=37, n_bytes=3844, priority=0 actions=CONTROLLER:65535
```

Figure 4-1.4 Matching using IP Address

III. Match using Protocol



```
cookie=0x0, duration=3.933s, table=0, n_packets=238752, n_bytes=11069190464, priority=1,tcp,nw_src=192.168.1.2,nw_dst=192.168.1.1,tp_src=37304,tp_dst=5001 actions=output:s1-eth1
cookie=0x0, duration=3.906s, table=0, n_packets=192421, n_bytes=12699810, priority=1,tcp,nw_src=192.168.1.1,nw_dst=192.168.1.2,tp_src=5001,tp_dst=37304 actions=output:s1-eth2
cookie=0x0, duration=31.495s, table=0, n_packets=43, n_bytes=4309, priority=0 actions=CONTROLLER:65535
```

Figure 4-1.5 Matching using Protocol Address

During this step, the model is trained using both normal traffic and attack traffic. This can be done by analyzing the parameters such as timeouts, flags and Priority Fields. Similarly, if the match fields don't match the topology such kinds of traffics can be labeled as Malicious Traffics.

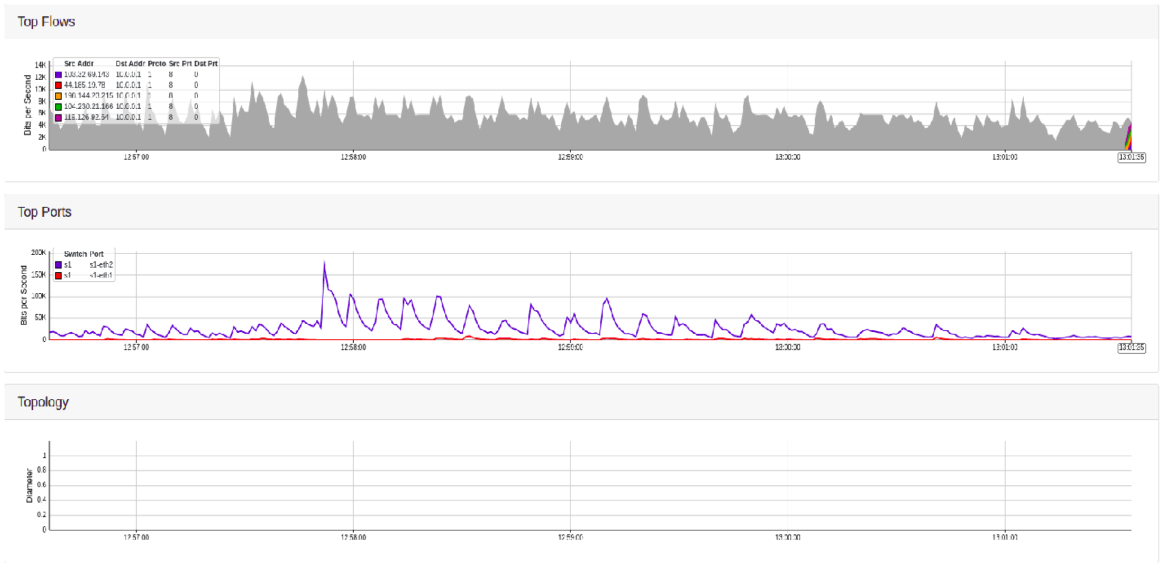


Figure 4-1.6 Traffic on OVS Switch

The incoming traffic to the switch is first stored. As the incoming traffic has lots of parameters, the traffic is applied to statistical analyzer which only generates a data set consisting of useful and meaningful features. The data set consisting of these feature vectors is known as training data. The process of generation of both legitimate and attack traffic is already explained under data set explanation section.

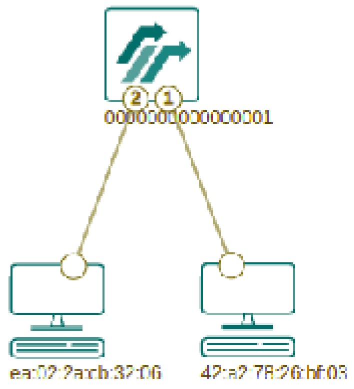
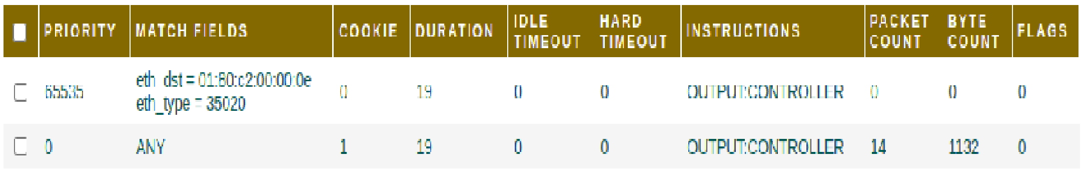


Figure 4-1.7 Multiple controller Topology

Generally, the model is trained using for normal and attack traffic. Out of the multiple host, one host is assuming as an attacker and rest are normal host user. DDos attack is simulated using hping3 and iperf. Controller is responsible to check the flow table periodically to insure that the packets are flooded or not. Similarly, each port of the OVS swtich are also periodically observed for continuous packet monitor and flow control.



	PRIORITY	MATCH FIELDS	COOKIE	DURATION	IDLE TIMEOUT	HARD TIMEOUT	INSTRUCTIONS	PACKET COUNT	BYTE COUNT	FLAGS
☐	65535	eth_dst = 01:80:c2:00:00:0e eth_type = 35020	0	19	0	0	OUTPUT:CONTROLLER	0	0	0
☐	0	ANY	1	19	0	0	OUTPUT:CONTROLLER	14	1132	0

Figure 4-1.8 Flow table for above Topology

After collecting the normal traffic, the result along with its SSIP and SFE are stored for continuous comparisons. The dataset is passed to the purposed algorithm and with the help of these data samples, data point can be classified based on similarity in the specific group of neighboring data points.

4.2 Attack detection:

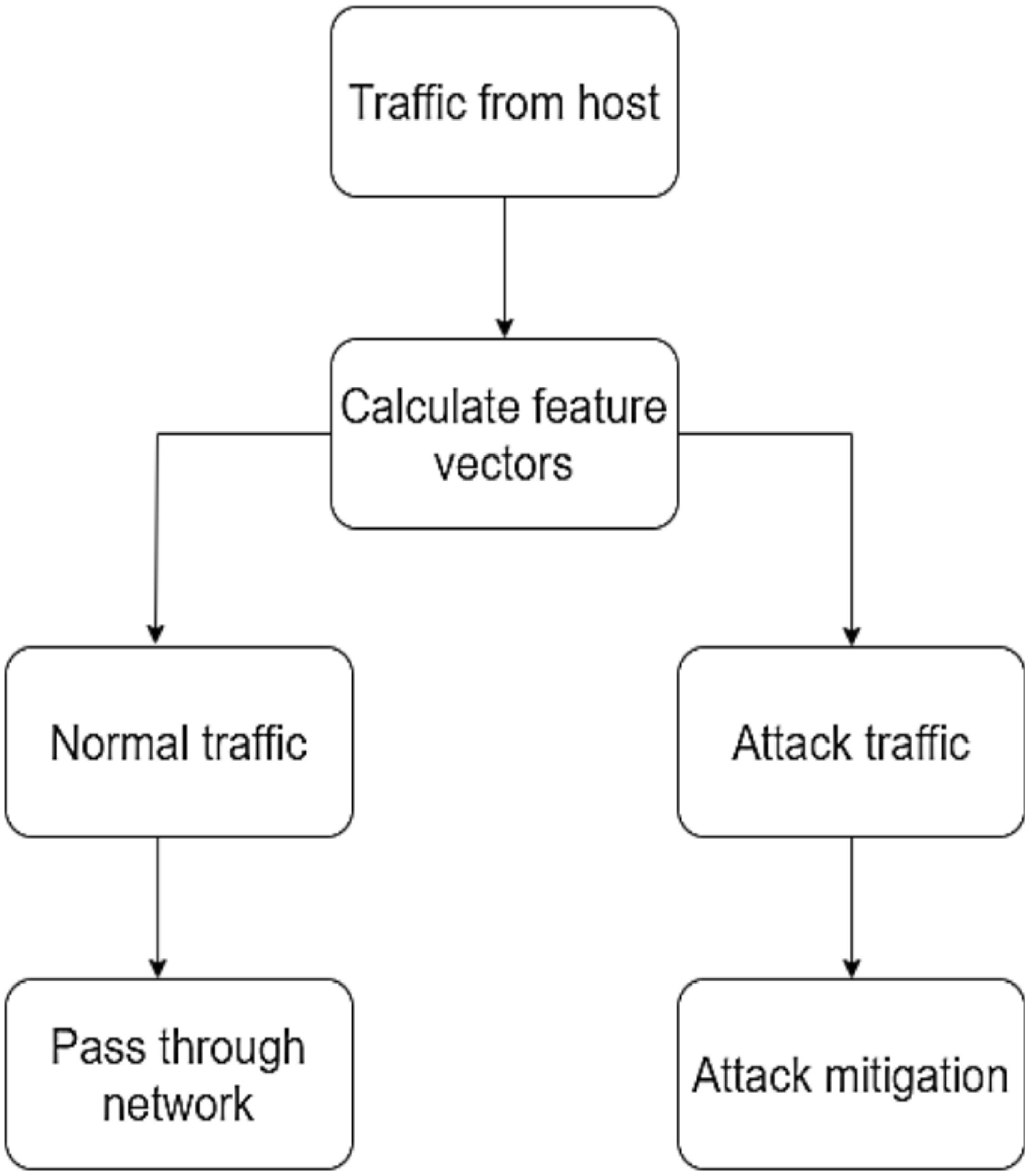


Figure 4-2.1 Attack Detection

Based on the training of the proposed model, the model predicts the nature of incoming traffic based on various feature vectors and classifies it as either benign traffic or attack traffic based on which side of neighboring point the data lies on [24].

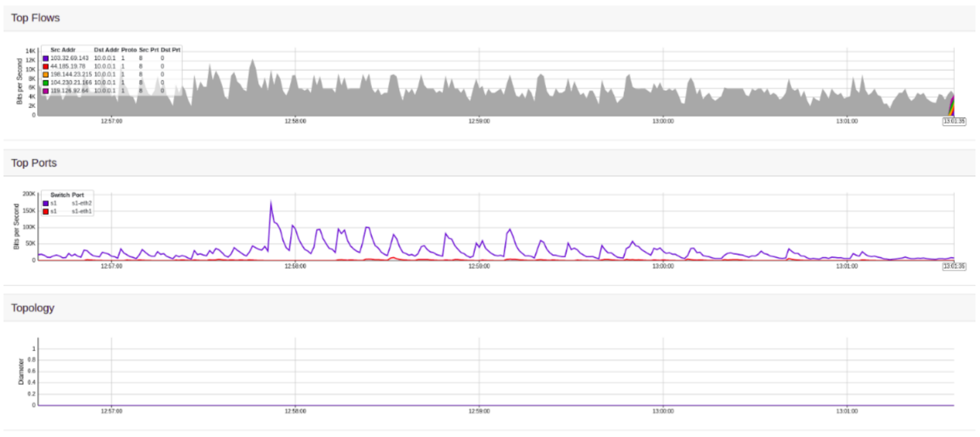
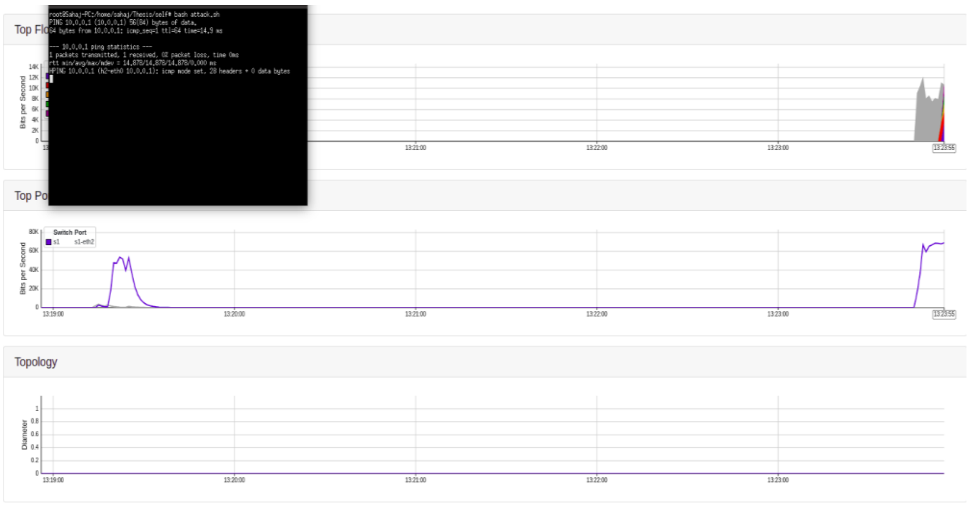


Figure 4-2.2 Normal traffic Flow between 2 host

In case of normal traffic, the traffic towards the switches are easily pass through which can be done by analyzing the delay and timeout parameter obtained from SSIP, SFE and RPE. If a traffic from any host incoming to the host is found to have low SSIP as well as low SFE, the traffic is classified as normal traffic and it is allowed to pass through the network without any tempering. In case of the abnormal traffic, the traffic can be observed on flow table parameters [24].

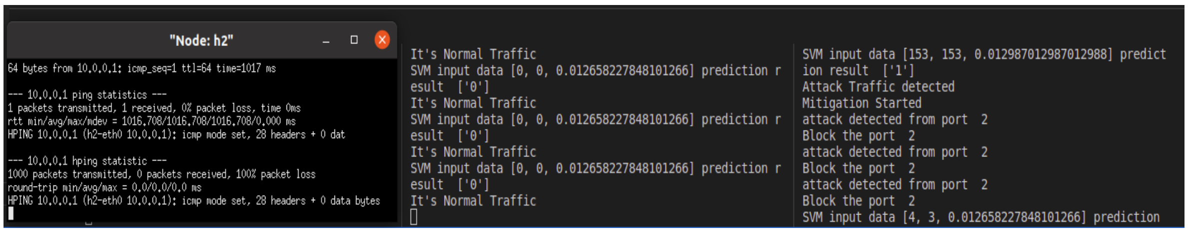


```
root@Sahaj-PC:/home/sahaj/Thesis/self# bash attack.sh
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=14.9 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.878/14.878/14.878/0.000 ms
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
```

Figure 4-2.3 Attack traffic Flow from Host 1

For example, a packet incoming to switch from host h1 connected to port1 of traffic has high SSIP, the point lies above the hyperplane generated from training of model and it is classified as attack traffic in case of SVM. For another layer of surety, another feature like Speed of flow entries is also plotted. If it also classifies the traffic as attack traffic, the host h1 is decided to be the attacker on the network.



```
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1017 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1016.708/1016.708/1016.708/0.000 ms
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 dat

--- 10.0.0.1 hping statistic ---
1000 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
```

```
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
```

```
SVM input data [153, 153, 0.012987012987012988] predict
ion result  ['1']
Attack Traffic detected
Mitigation Started
attack detected from port  2
Block the port  2
attack detected from port  2
Block the port  2
attack detected from port  2
Block the port  2
SVM input data [4, 3, 0.012658227848101266] prediction
```

Figure 4-2.4 Attack traffic Flow from Host 1 in case of two Controllers

The number of hosts were varied from 10 to 20 for multiple iteration of traffic generation. First the normal traffic was generated for 200 seconds while taking sample for every 2 seconds followed by attack traffic for 200 seconds taking sample for every 2 seconds. The process of normal traffic and attack traffic is repeated multiple times.

The normal traffic is generated using ping command whereas the attack traffic is generated using hping3 and iperf depending on requirement of either automatic traffic or manual traffic. The traffic from both switch is collected and stored in csv file [24]. The dataset generated is shown below.

Time	**Sfe**	**Ssip**	**Rfip**	**Flowcount**
03/31/2022, 12:25:01	7	4	1	7
03/31/2022, 12:25:06	6	2	1	13
03/31/2022, 12:25:11	6	3	1	19
03/31/2022, 12:25:16	8	1	1	27
03/31/2022, 12:26:21	177	176	0.011364	177
03/31/2022, 12:26:26	482	482	0.00304	659
03/31/2022, 12:26:31	374	374	0.001938	1033
03/31/2022, 12:26:36	326	326	0.001473	1359
03/31/2022, 12:26:41	326	326	0.001188	1685
03/31/2022, 12:38:14	2	1	1	7

Table 4.2.1 Training dataset generated in tabulated form for two switches

The generated table consists of all the parameters that are used for the classification of traffic as normal and from the tabulated sample.

4.3 Overhead Handling:

Even if the network is under attack, the controller can protect the hosts on the network by blocking the attacker as explained in earlier steps. But, if the controller is attacked, then the problem of overhead arises which is solved by using multi controller environment. In this project, the multi controller network has 2 independent controllers, one operating a main controller and another one being a standby secondary controller. If the controller is in normal operation, there is zero or no delay in receiving of the packet and the flag is set to 0. When an attack is performed on the controller, there is a certain delay in incoming traffic and the flag is set to 1 which is a sign of attack on controller. But as delay can occur due to other factors too, stability check is performed to confirm about the attack. Under attack scenario, the incoming traffic is extremely high compared to traffic processed by the controller due to the arrival of large amount of fake traffic. This result in the controller being unstable which is confirmation of attack on controller.

Once the controller is on attack, the operation is transferred to the standby secondary controller from the main controller and main controller is terminated from the network for certain duration to protect the hosts in the network.

Overhead is calculated based on time period of flow of packers. A flag of set for every controller. Initially flag is set to 0. If flag is equal to 1 then the delay of the flow for particular controller is calculated. Similarly, the stability for all controller is determined. If Stability is equal to 1, the controller is assumed unstable and the whole node is optimized. For controller with stability not equal to 1. The flag is set to 0. The purpose methodology elects the controller with highest performance as Master and loads are transferred to slave controller if overhead is detected. If controller cannot control the overhead, the slave controller becomes master and so-on [25].

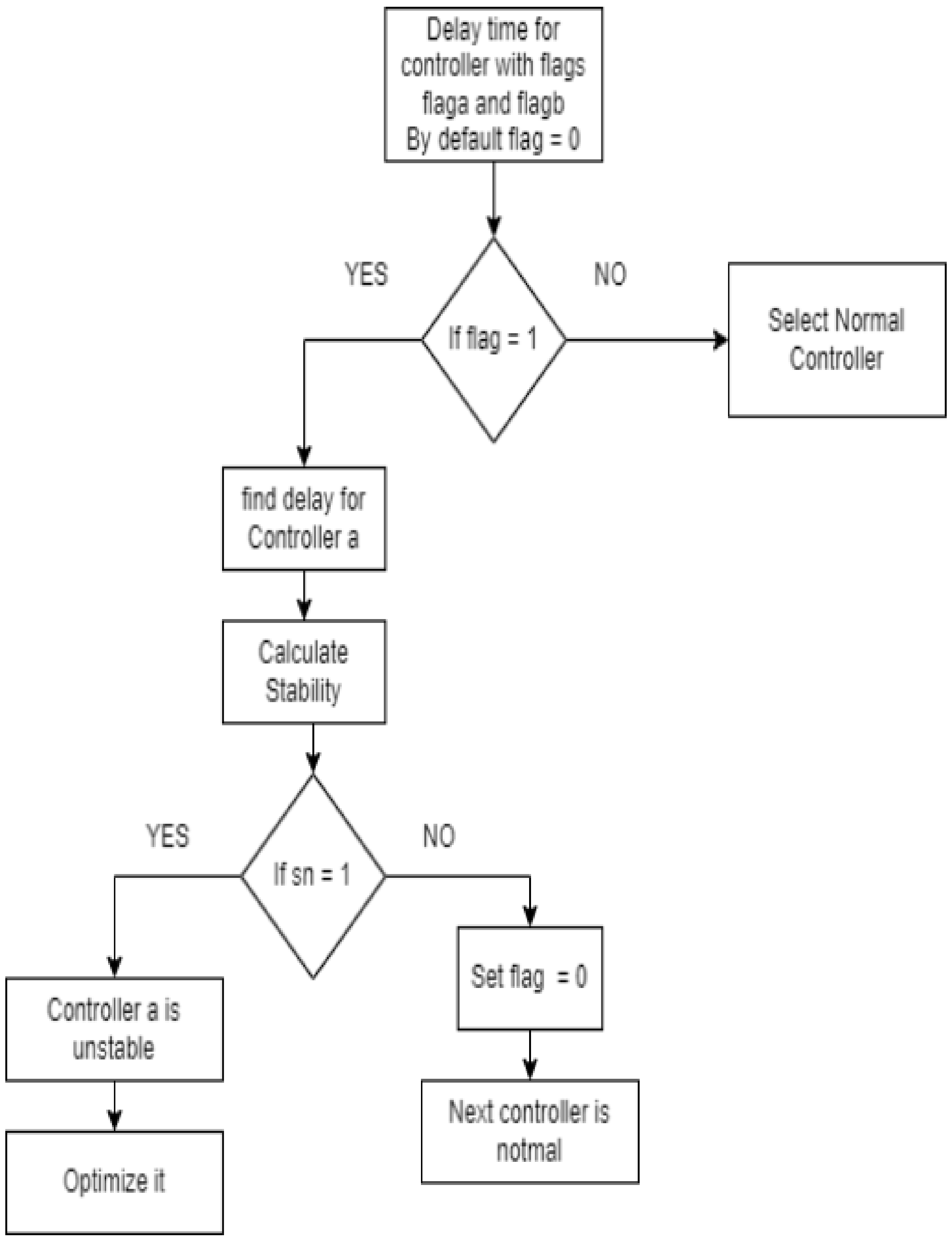

Delay time for controller with flags flaga and flagb By default flag = 0
YES
If flag = 1
NO
Select Normal Controller
find delay for Controller a
Calculate Stability
YES
If sn = 1
NO
Set flag = 0
Controller a is unstable
Next controller is notmal
Optimize it

Fig 4.3 Proposed Selection Methodology for overhead detection

CHAPTER 5

OBSERVATION /RESULTS

Similar to the Conceptual model, the results can also be classified into two parts. One is Training and testing the proposed algorithm on NSL-KDD Data set and another part is to use the algorithm on simulated Environment. The results are calculated based on Accuracy, error, Precisions, recall and F1-measures where

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

$$\alpha - \text{error} = \frac{FN}{FN+TP} \tag{5.2}$$

$$\text{Precision} = \frac{\sum TP}{\sum TP+FP} \tag{5.3}$$

$$\text{Recall} = \frac{\sum TP}{\sum TP+FN} \tag{5.4}$$

$$\text{F} - \text{Measure} = \frac{2*Precision*Recall}{Precision+Recall} \tag{5.5}$$

5.1 Training the Algorithm based on NSL-KDD Dataset

I. Using K means Clustering

As per the proposed algorithm, the detection and training was initially carried using K means clustering. Since the clustering data was of varying size and density, K-means algorithm shows average result from predictions. For training the dataset, the total fields to be analyzes was upto 84 different field, As the number of dimension increase, the mean is distance between the features decreased. However, such problems can be avoided using Spectral clustering by reducing the dimension size of features by using PCA, the accuracy of the proposed algorithm decreases.

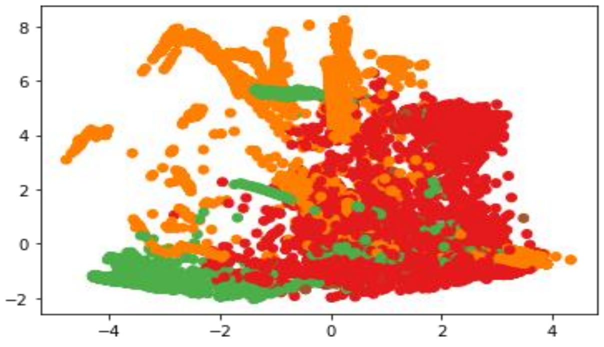
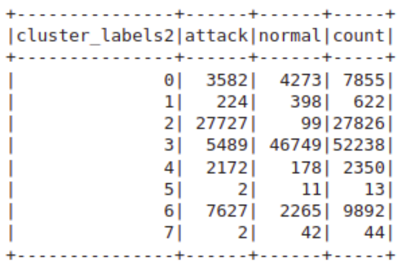


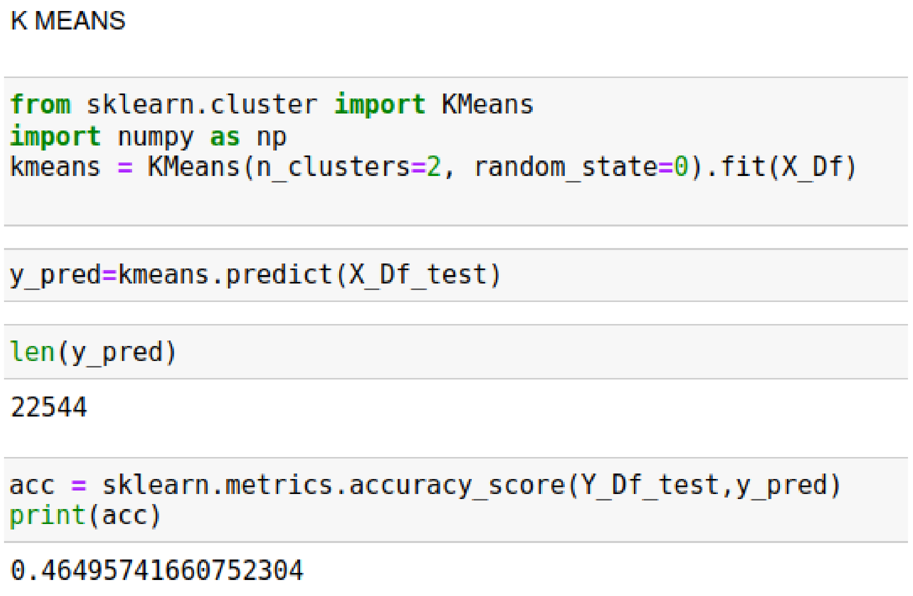
Fig 5.1.1 Cluster of 84 fields using K mean cluster based on PCA on NSL-KDD



```
+--------------+------+------+-----+
|cluster_labels2|attack|normal|count|
+--------------+------+------+-----+
|             0|  3582|  4273| 7855|
|             1|   224|   398|  622|
|             2| 27727|    99|27826|
|             3|  5489| 46749|52238|
|             4|  2172|   178| 2350|
|             5|     2|    11|   13|
|             6|  7627|  2265| 9892|
|             7|     2|    42|   44|
+--------------+------+------+-----+
```

Fig 5.1.2 Traffic Clustering based on various cluster size on NSL-KDD

The accuracy for K means was calculated for the Attack labelled “Ddos ” for K=2



K MEANS

```
from sklearn.cluster import KMeans
import numpy as np
kmeans = KMeans(n_clusters=2, random_state=0).fit(X_Df)
```

```
y_pred=kmeans.predict(X_Df_test)
```

```
len(y_pred)
```

```
22544
```

```
acc = sklearn.metrics.accuracy_score(Y_Df_test,y_pred)
print(acc)
```

```
0.46495741660752304
```

Fig 5.1.3 K mean Clustering Accuracy Calculations on NSL-KDD

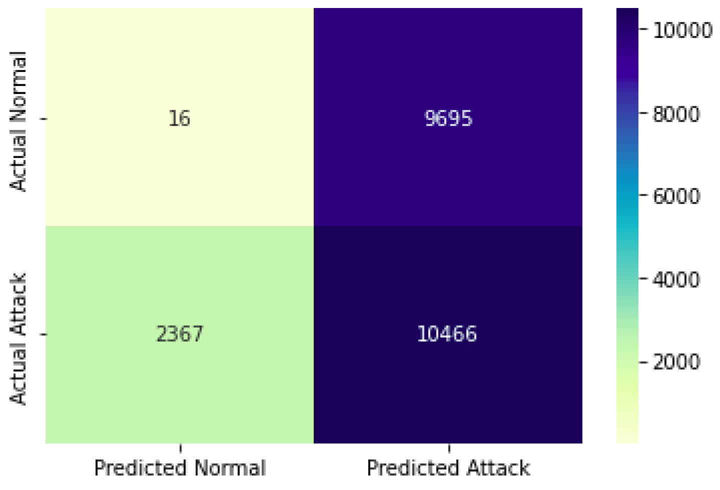
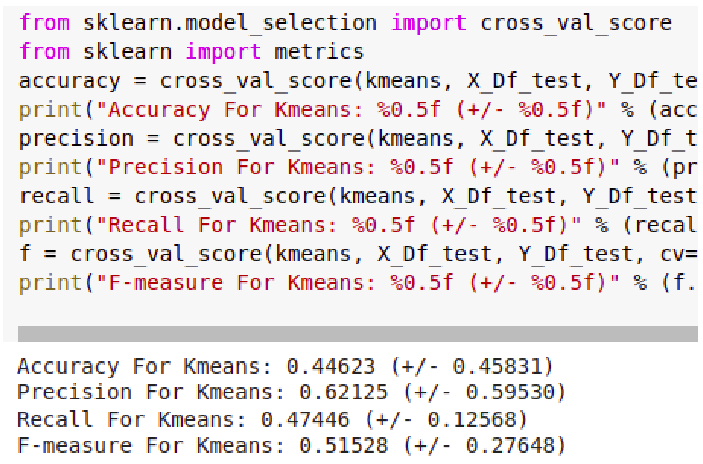


Fig 5.1.4 Confusion matrix obtained from K Mean Clustering on NSL-KDD



```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(kmeans, X_Df_test, Y_Df_te
print("Accuracy For Kmeans: %0.5f (+/- %0.5f)" % (acc
precision = cross_val_score(kmeans, X_Df_test, Y_Df_t
print("Precision For Kmeans: %0.5f (+/- %0.5f)" % (pr
recall = cross_val_score(kmeans, X_Df_test, Y_Df_test
print("Recall For Kmeans: %0.5f (+/- %0.5f)" % (recal
f = cross_val_score(kmeans, X_Df_test, Y_Df_test, cv=
print("F-measure For Kmeans: %0.5f (+/- %0.5f)" % (f.
```

```
Accuracy For Kmeans: 0.44623 (+/- 0.45831)
Precision For Kmeans: 0.62125 (+/- 0.59530)
Recall For Kmeans: 0.47446 (+/- 0.12568)
F-measure For Kmeans: 0.51528 (+/- 0.27648)
```

Fig 5.1.5 Accuracy, Precision, Recall and F-measure of K Mean Clustering on NSL-KDD

II. Using Knn Algorithm

For training the data set, the total fields to be analyzes was up to 84 different field. The data set was initially divided into training and testing set. The accuracy for the logarithm was tested using KneighborsClassifier with metric as 'minikwoski' and neighboring value of 5.

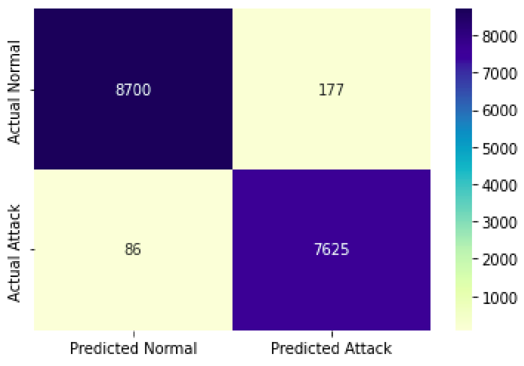
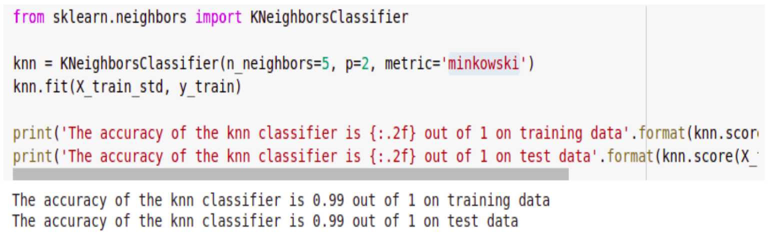


Fig 5.1.6 Confusion Matrix of KNN Algorithm on NSL-KDD

The accuracy of Knn classifer was found to be the best however the precision and F-measure based on the train and test data was found to be weak.



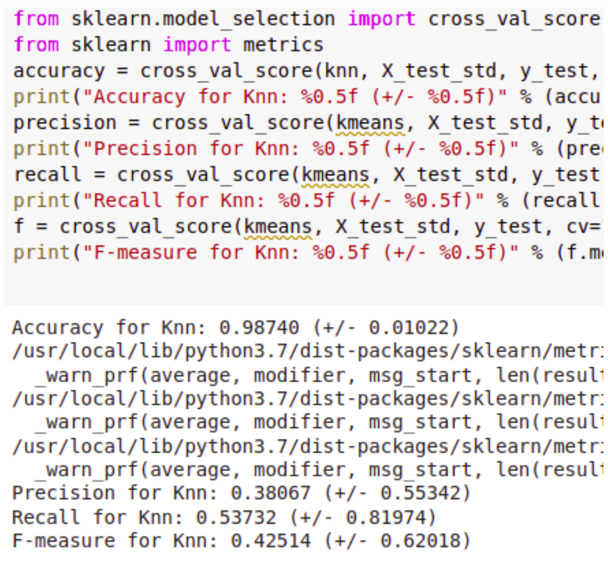
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)

print('The accuracy of the knn classifier is {:.2f} out of 1 on training data'.format(knn.scor
print('The accuracy of the knn classifier is {:.2f} out of 1 on test data'.format(knn.score(X_
```

```
The accuracy of the knn classifier is 0.99 out of 1 on training data
The accuracy of the knn classifier is 0.99 out of 1 on test data
```

Fig 5.1.7 Accuracy of KNN Algorithm on NSL-KDD



```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(knn, X_test_std, y_test,
print("Accuracy for Knn: %0.5f (+/- %0.5f)" % (accu
precision = cross_val_score(kmeans, X_test_std, y_t
print("Precision for Knn: %0.5f (+/- %0.5f)" % (pre
recall = cross_val_score(kmeans, X_test_std, y_test
print("Recall for Knn: %0.5f (+/- %0.5f)" % (recall
f = cross_val_score(kmeans, X_test_std, y_test, cv=
print("F-measure for Knn: %0.5f (+/- %0.5f)" % (f.m
```

```
Accuracy for Knn: 0.98740 (+/- 0.01022)
/usr/local/lib/python3.7/dist-packages/sklearn/metr
  _warn_prf(average, modifier, msg_start, len(resul
/usr/local/lib/python3.7/dist-packages/sklearn/metr
  _warn_prf(average, modifier, msg_start, len(resul
/usr/local/lib/python3.7/dist-packages/sklearn/metr
  _warn_prf(average, modifier, msg_start, len(resul
Precision for Knn: 0.38067 (+/- 0.55342)
Recall for Knn: 0.53732 (+/- 0.81974)
F-measure for Knn: 0.42514 (+/- 0.62018)
```

Fig 5.1.8 Accuracy, Precision, Recall and F-measure of KNN Algorithm on NSL-KDD

III. Using SVM Classifier

For training the data set, the total fields to be analyzes was up to 84 different field. The data set was initially divided into training and testing set. The accuracy for the logarithm was tested using both kernel as 'rbf as nonlinear" and Linear. For lower dimension, the Linear function had better accuracy and was linearly separable but as feature dimension increases, the difference in the space dimension starts to break and becomes more nonlinear. Hence RBG was use to map the complex boundaries.

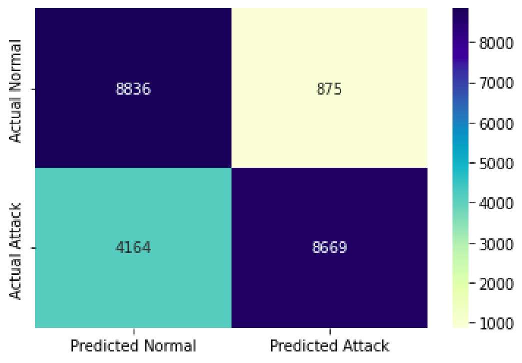
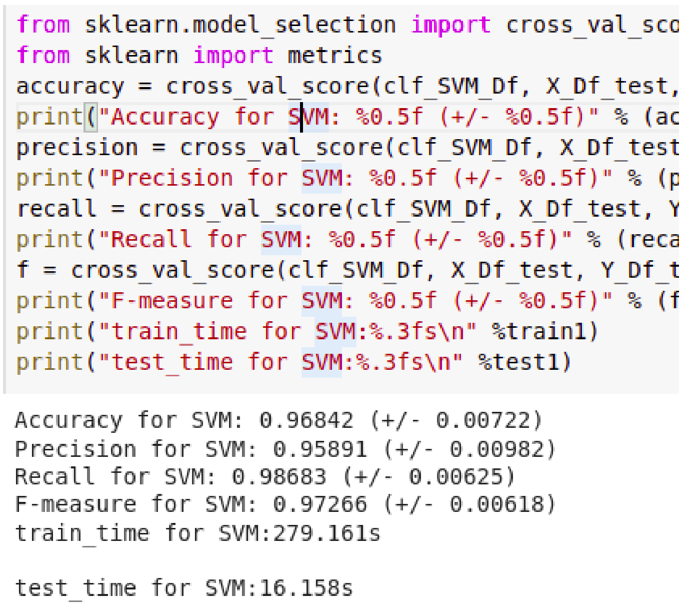


Fig 5.1.9 Confusion Matrix of SVM Classifier on NSL-KDD

The accuracy of svm.SVC classifer was found to be the best as well as the precision and F-measure based on the train and test data was found to be best. So similar algorithm was chosen for classification between Normal and Attack traffic in case of simulation of real world data.



```
from sklearn.model_selection import cross_val_sco
from sklearn import metrics
accuracy = cross_val_score(clf_SVM_Df, X_Df_test,
print("Accuracy for SVM: %0.5f (+/- %0.5f)" % (ac
precision = cross_val_score(clf_SVM_Df, X_Df_test
print("Precision for SVM: %0.5f (+/- %0.5f)" % (p
recall = cross_val_score(clf_SVM_Df, X_Df_test, Y
print("Recall for SVM: %0.5f (+/- %0.5f)" % (reca
f = cross_val_score(clf_SVM_Df, X_Df_test, Y_Df_t
print("F-measure for SVM: %0.5f (+/- %0.5f)" % (f
print("train_time for SVM:%.3fs\n" %train1)
print("test_time for SVM:%.3fs\n" %test1)
```

```
Accuracy for SVM: 0.96842 (+/- 0.00722)
Precision for SVM: 0.95891 (+/- 0.00982)
Recall for SVM: 0.98683 (+/- 0.00625)
F-measure for SVM: 0.97266 (+/- 0.00618)
train_time for SVM:279.161s

test_time for SVM:16.158s
```

Fig 5.1.10 Accuracy, Precision, Recall and F-measure of SVM Classifier on NSL-KDD

5.2 Training the Algorithm based on Simulated Environment

Two ryu controllers and five openflow switches were used to create a network with hosts connected to both switches. The number of hosts per switches can be varied. The topology was created on Mininet based on python programming language.

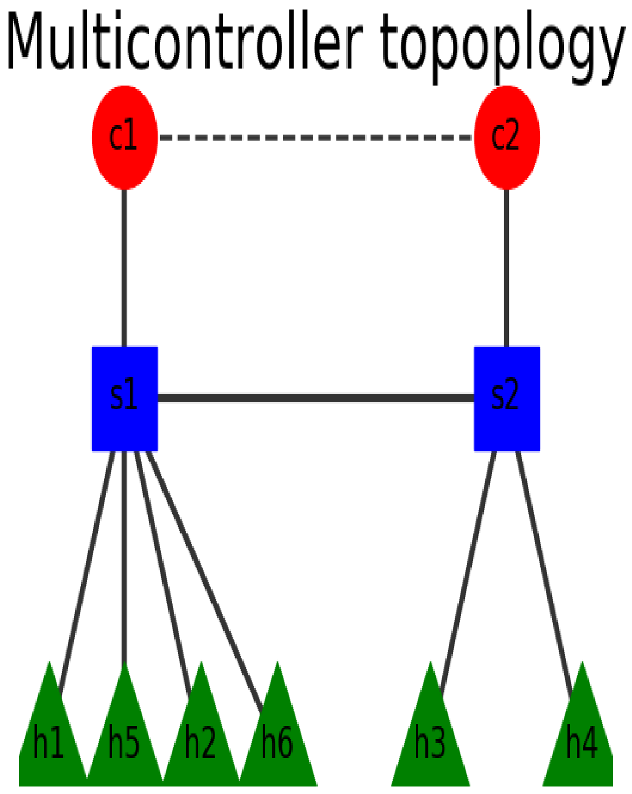


Fig 5.2.1 Mininet Topology with two controllers

The normal traffic is generated using ping command whereas the attack traffic is generated using hping3 and iperf depending on requirement of either automatic traffic or manual traffic. The traffic from both switch is collected and stored in csv file. The dataset generated is shown below.

Time	Sfe	Ssip	Rfip	Flowcount
03/31/2022, 12:25:01	7	4	1	7
03/31/2022, 12:25:06	6	2	1	13
03/31/2022, 12:25:11	6	3	1	19
03/31/2022, 12:25:16	8	1	1	27
03/31/2022, 12:26:21	177	176	0.011364	177
03/31/2022, 12:26:26	482	482	0.00304	659
03/31/2022, 12:26:31	374	374	0.001938	1033
03/31/2022, 12:26:36	326	326	0.001473	1359
03/31/2022, 12:26:41	326	326	0.001188	1685
03/31/2022, 12:38:14	2	1	1	7
03/31/2022, 12:38:19	2	1	1	9
03/31/2022, 12:38:24	14	3	1	23

Table 5.2 Training dataset generated in tabulated form for two switches

The generated table consists of all the parameters that are used for the classification of traffic as normal and from the tabulated sample, the SVM model is trained for classification of real-time traffic. The distribution of feature vectors is seen to be skewed which estimates that there are outliers in the data and affects the performance of the statistical modeling. The skewness in data is handled by SVM. The distribution of data is seen as:

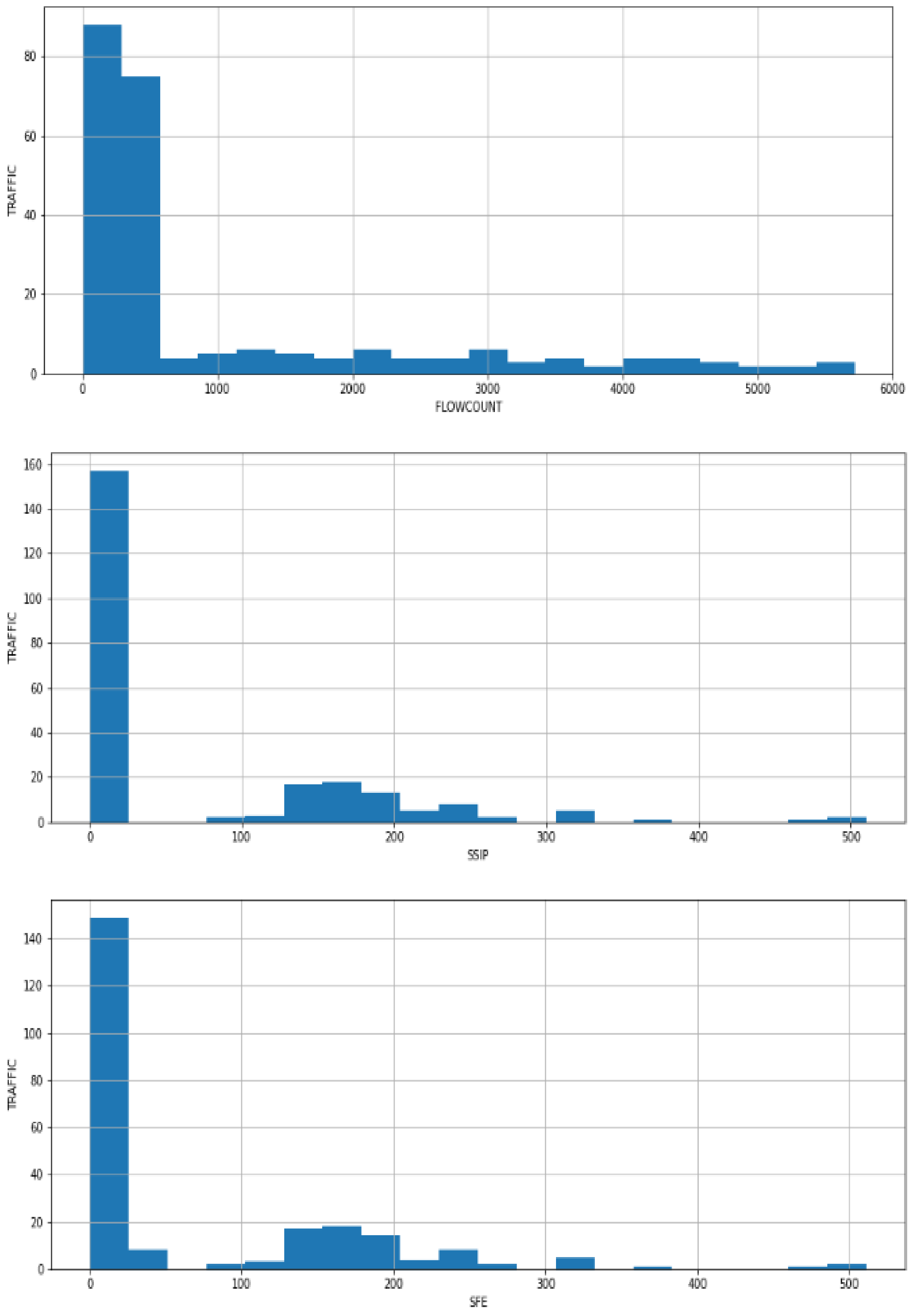


Figure 5-2.2 Distribution of traffic based on various feature vectors or parameters

On basis of parameters the decision boundary with hyperplane is calculated by the process similar to that performed for single controller and single switch configuration.

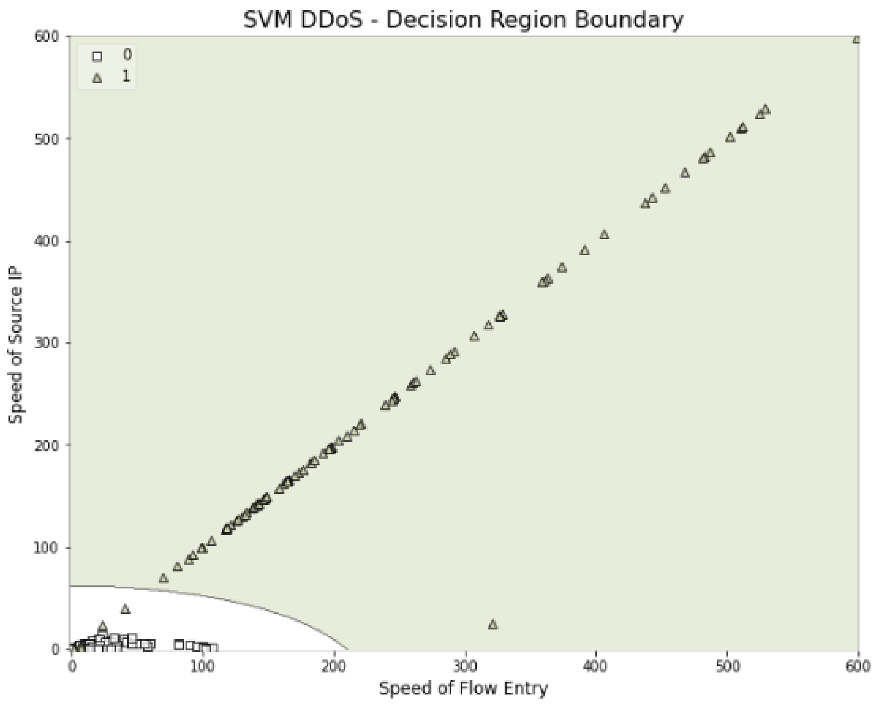


Figure 5-2.3 Hyperplane on basis of SSIP and SFE for two switches

Here square boxes (0) represent the normal traffic and triangle boxes (1) represents attack traffic. Any real time traffic falling below the hyperplane is classified as the benign traffic and the traffic packets falling above the hyperplane is classified as attack traffic during analysis [24].

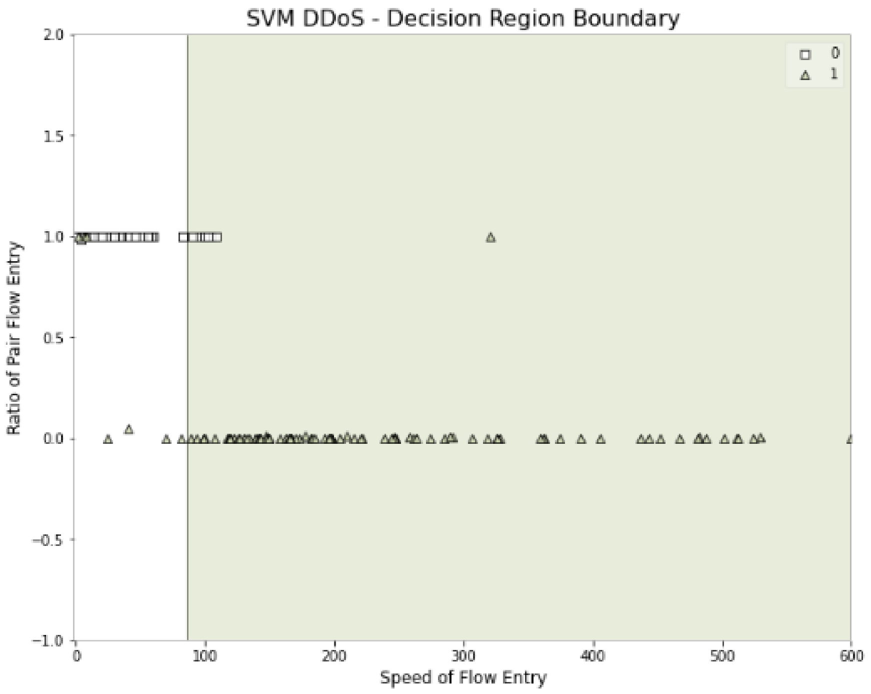
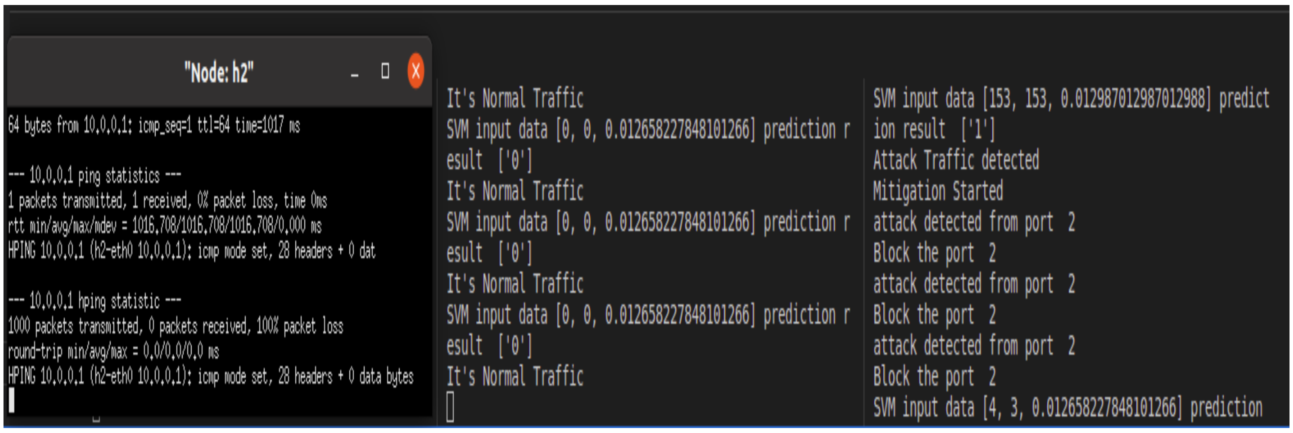


Figure 5-2.4 Hyperplane on basis of RFE and SFE for two switches

Any real time traffic falling on left hand side of the hyperplane is classified as the benign traffic and the traffic packets falling on ride hand side of the hyperplane is classified as attack traffic during analysis. Once the model is trained and ready to be implemented, the real time data is passed through it. When an attack traffic using hping3 is passed into network from host 3 on port 3, the system detects the parameter of the packet and classify it as DDoS attack. The system moves ahead to drop the packet and block the port consequently.



```
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1017 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1016.708/1016.708/1016.708/0.000 ms
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 dat

--- 10.0.0.1 hping statistic ---
1000 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
HPING 10.0.0.1 (h2-eth0 10.0.0.1): icmp mode set, 28 headers + 0 data bytes
```

```
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
SVM input data [0, 0, 0.012658227848101266] prediction r
esult  ['0']
It's Normal Traffic
```

```
SVM input data [153, 153, 0.012987012987012988] predict
ion result  ['1']
Attack Traffic detected
Mitigation Started
attack detected from port  2
Block the port  2
attack detected from port  2
Block the port  2
attack detected from port  2
Block the port  2
SVM input data [4, 3, 0.012658227848101266] prediction
```

Figure 5-2.5 Real time attack detection and mitigation between two controller network

CHAPTER 6

DATA ANALYSIS AND REPORTING

All the topologies used in the project are analyzed for the accuracy in detection of attack traffic and also the decision rate is evaluated. The accuracy was detected for different kernel types with varying penalty parameter. The accuracy of the SVM model on data generated from this topology was calculated with linear kernel and penalty parameter set to 0.25.

Accuracy = 96.27%

The training set and testing set were split into ratio of 3:2 with respective accuracy of:

Training Set Accuracy = 96.50%

Test Set Accuracy = 96.27%

Since, two values are comparable, there is no chance of over fitting or under fitting.

Null accuracy = 75.37%

Since, the accuracy of model is better than null accuracy, the model is doing good job is classifying the labels.

The confusion matrix for the model is shown below:

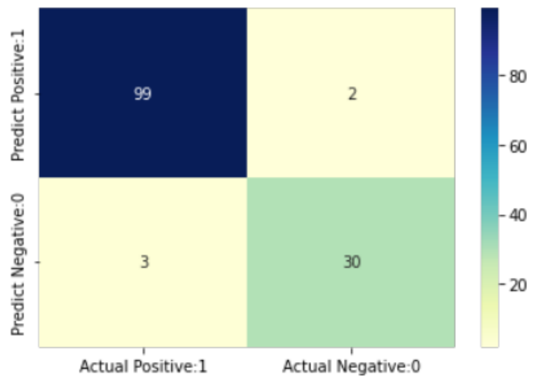
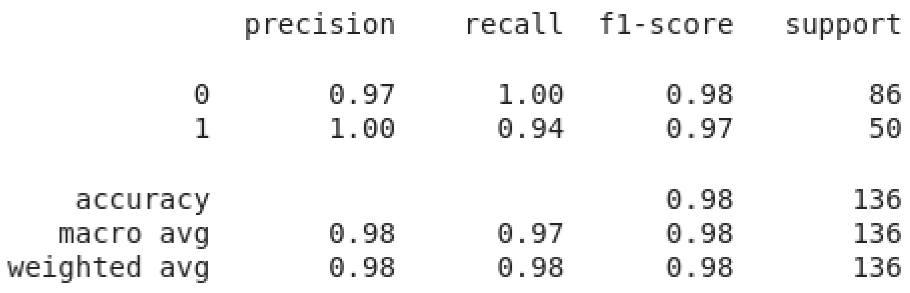


Figure 6-1: Confusion matrix using SVM Classifier for above topology on Simulated Environment

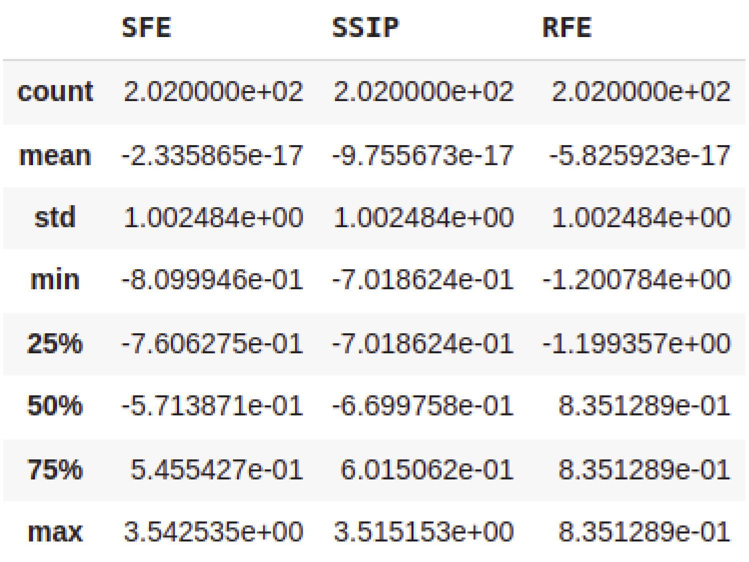


```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        86
           1       1.00      0.94      0.97        50

    accuracy                           0.98       136
   macro avg       0.98      0.97      0.98       136
weighted avg       0.98      0.98      0.98       136
```

Figure 6-2: Precision, Recall and F1 score for above topology on Simulated Environment



	SFE	SSIP	RFE
count	2.020000e+02	2.020000e+02	2.020000e+02
mean	-2.335865e-17	-9.755673e-17	-5.825923e-17
std	1.002484e+00	1.002484e+00	1.002484e+00
min	-8.099946e-01	-7.018624e-01	-1.200784e+00
25%	-7.606275e-01	-7.018624e-01	-1.199357e+00
50%	-5.713871e-01	-6.699758e-01	8.351289e-01
75%	5.455427e-01	6.015062e-01	8.351289e-01
max	3.542535e+00	3.515153e+00	8.351289e-01

Figure 6-3 Variance and correlation between features and traffic

The outcomes are:

True Positives(TP) = 99

True Negatives(TN) = 30

False Positives(FP) = 2

False Negatives(FN) = 3

Thus,

$$\text{Classification Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 96.27\ \%$$

Also,

$$\text{Classification Error} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 3.73\ \%$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 98.02\ \%$$

$$\text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 97.06\ \%$$

True positive rate is also known as detection rate.

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} = 6.25\ \%$$

6.1 Receiver operating characteristic curve and Area Under the ROC Curve

A ROC AUC of 1 indicates a flawless classifier, whereas a ROC AUC of 0.5 indicates a fully random classifier.

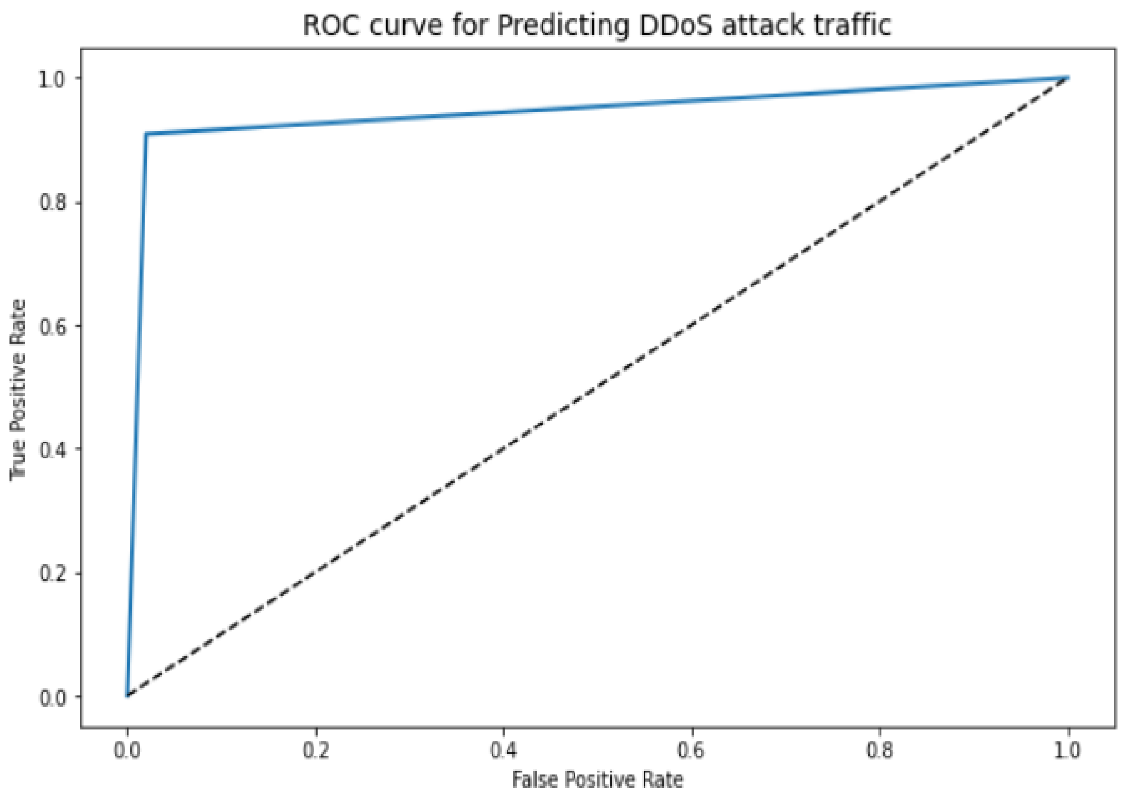


Figure 6.4 ROC curve for two switches topology

From the curve, ROC AUC = 0.9446

k-fold cross-validation is a powerful technique for assessing performance of the model. However, it fails in this case because we have an imbalanced dataset. As a result, stratified k-fold cross-validation is performed in the event of an imbalanced dataset. Average stratified cross-validation score with linear kernel: 0.9640 = 96.40% The stratified cross-validation score with linear kernel after shuffling of dataset is 96.40% which is greater than the accuracy of the model i.e., 96.27%. Thus, stratified cross-validation does not improve the model performance.

6.2 Comparison with work performed by other authors and proposed algorithm

While comparing the accuracy result of a single switch IPv4 topology to the results present in paper "Machine Learning Based Botnet Detection" [9]," A protocol for cluster confirmations of SDN controllers against DDoS attacks [", "Optimizing Controller Placement for Software-Defined Networks]", the outcome of proposed was found to be better than K means, Decision tree and K median while comparable to other methods. The accuracy of the model can be increased by training the model for longer period.

Model	Accuracy
Statistical Analysis + Linear SVM	97.79 %
J48 – decision tree	97.60 %
GA+GD	82.74 %
K median	76.17 %
Capacity Based Greedy Approach	80 %
Random Approach	81.36 %

Table 6.1 Comparison of Algorithm Accuracy with other Author

However, the comparison of algorithm between the data set and simulation environment may not be feasible due to

- Smaller Dataset which led to less training and testing time
- Delay between the packet flow
- Performance of the workstation

Bibliography / References

[1] L. Yao, P. Hong and W. Zhou, "Evaluating the controller capacity in software defined networking," 2014 23rd International Conference on Computer Communication and Networks (ICCCN), 2014, pp. 1-6, doi: 10.1109/ICCCN.2014.6911857.

[2] Ali J, Roh Bh, Lee S, “QoS improvement with an optimum controller selection for software-defined networks,”, 2019, PLOS ONE 14(5): e0217631, doi: 10.1371/journal.pone.0217631

[3] Nam, Tran Manh et al. “Self-organizing map-based approaches in DDoS flooding detection using SDN.” 2018 International Conference on Information Networking (ICOIN) (2018): 249-254.

[4] Jagdeep Singh, Sunny Behal, Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions, Computer Science Review, Volume 37, 2020, 100279, ISSN 1574-0137, doi: 10.1016/j.cosrev.2020.100279.

[5] P. Preamthaisong, A. Auyporntrakool, P. Aimtongkham, T. Sriwuttisap and C. So-In, "Enhanced DDoS Detection using Hybrid Genetic Algorithm and Decision Tree for SDN," 2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2019, pp. 152-157, doi: 10.1109/JCSSE.2019.8864216.

[6] R. Braga, E. Mota and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," IEEE Local Computer Network Conference, 2010, pp. 408-415, doi: 10.1109/LCN.2010.5735752.

[7] Harikrishna, P., Amuthan, A. SDN-based DDoS Attack Mitigation Scheme using Convolution Recursively Enhanced Self Organizing Maps. Sādhanā 45, 104 (2020). Doi: 10.1007/s12046-020-01353-x

[8] Karnwal, Tarun & Sivakumar, T. & Gnanasekaran, Aghila. (2012). A filter tree approach to protect cloud computing against XML DDoS and HTTP DDoS attack. Advances in Intelligent Systems and Computing. 182. 10.1109/SCEECS.2012.6184829.

[9] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," 2015 International Conference on Computing, Networking and Communications (ICNC), 2015, pp. 77-81, doi: 10.1109/ICCNC.2015.7069319.

[10] S. Dong and M. Sarem, "DDoS Attack Detection Method Based on Improved KNN with the Degree of DDoS Attack in Software-Defined Networks," in IEEE Access, vol. 8, pp. 5039-5048, 2020, doi: 10.1109/ACCESS.2019.2963077.

[11] H. Peng, Z. Sun, X. Zhao, S. Tan and Z. Sun, "A Detection Method for Anomaly Flow in Software Defined Network," in IEEE Access, vol. 6, pp. 27809-27817, 2018, doi: 10.1109/ACCESS.2018.2839684.

[12] Seungwon, Shin & Yegneswaran, Vinod & Porras, Phillip & Gu, Guofei. (2013). AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. 10.1145/2508859.2516684.

[13] Ambrosin, Moreno & Conti, Mauro & De Gaspari, Fabio & Poovendran, Radha. (2015). LineSwitch: Efficiently Managing Switch Flow in Software-Defined Networking while Effectively Tackling DoS Attacks. ASIACCS 2015 - Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. 10.1145/2714576.2714612.

[14] Iranmanesh, Amir & Naji, Hamid. (2021). A protocol for cluster confirmations of SDN controllers against DDoS attacks. Computers & Electrical Engineering. Volume 93. 10.1016/j.compeleceng.2021.107265.

[15] Islam, Md & Estivill-Castro, Vladimir & Rahman, Md Anisur & Bossomaier, Terry. (2017). Combining K-Means and a Genetic Algorithm through a Novel Arrangement of Genetic Operators for High Quality Clustering. Expert Systems with Applications. 91. 402-417. 10.1016/j.eswa.2017.09.005.

[16] Upendran, V., and R. Gopinath. "FEATURE SELECTION BASED ON MULTI-CRITERIA DECISION MAKING FOR INTRUSION DETECTION SYSTEM." (2021).

[17] J. E. Varghese and B. Muniyal, "An Efficient IDS Framework for DDoS Attacks in SDN Environment," in IEEE Access, vol. 9, pp. 69680-69699, 2021, doi: 10.1109/ACCESS.2021.3078065.

[18] Alzahrani, Abdulsalam & Alenazi, Mohammed. (2021). “Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks”. Future Internet. 13. 111. 10.3390/fi13050111.

[19] Sang-Hyun Choi and Hee-Su Chae. I. International Conference Data Mining, Civil and Mechanical Engineering (ICDMCME'2014), Feb 4-5, 2014 Bali (Indonesia). “Feature Selection for efficient Intrusion Detection using Attribute Ratio”.

[20] Cherubin, Giovanni and Chatzikokolakis, Konstantinos and Jaggi, Martin. “Exact Optimization of Conformal Predictors via Incremental and Decremental Learning”. Creative Commons Attribution 4.0 International. doi.org/10.48550/arXiv.2102.03236

[21] H. Peng, Z. Sun, X. Zhao, S. Tan and Z. Sun, "A Detection Method for Anomaly Flow in Software Defined Network," in IEEE Access, vol. 6, pp. 27809-27817, 2018, doi: 10.1109/ACCESS.2018.2839684.

[22] Gao, Zhiqiang and Nirwan Ansari. “Differentiating Malicious DDoS Attack Traffic from Normal TCP Flows by Proactive Tests.” *IEEE Communications Letters* 10 (2006): n. pag.

[23] J. E. Varghese and B. Muniyal, "An Efficient IDS Framework for DDoS Attacks in SDN Environment," in IEEE Access, vol. 9, pp. 69680-69699, 2021, doi: 10.1109/ACCESS.2021.3078065.

[24] Kumar Singh, Vishal (2020) DDOS attack detection and mitigation using statistical and machine learning methods in SDN. Master’s thesis, Dublin, National College of Ireland.

[25] Amir Iranmanesh, Hamid Reza Naji, A protocol for cluster confirmations of SDN controllers against DDoS attacks, Computers & Electrical Engineering, Volume 93, 2021, 107265, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2021.107265.