APPENDIX 1

DETECTION OF DISTRIBUTED DENIAL OF SERVICE ATTACKS IN SDN USING MACHINE LEARNING TECHNIQUES

A PROJECT REPORT

Submitted by

HARINI V (2010105013)

JAYAASHREE P (2019105541)

SHWETHA A (2019105576)

YAMINI K (2019105605)

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY::CHENNAI 600 025

MAY 2023

APPENDIX 1

DETECTION OF DISTRIBUTED DENIAL OF SERVICE ATTACKS IN SDN USING MACHINE LEARNING TECHNIQUES

A PROJECT REPORT

Submitted by

HARINI V (2010105013)

JAYAASHREE P (2019105541)

SHWETHA A (2019105576)

YAMINI K (2019105605)

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY::CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"Detection of Distributed Denial of Service Attacks in SDN using Machine Learning Techniques"** is the bonafide work of **"Harini V, Jayaashree P, Shwetha A and Yamini K"** who carried out the project work under my supervision.

SIGNATURE

Dr. M. Meenakshi

HEAD OF THE DEPARTMENT

Professor

Department of ECE

College of Engineering, Guindy

Anna University,

Chennai-600025

SIGNATURE

Dr. Manimekalai T

SUPERVISOR

Associate Professor

Department of ECE

College of Engineering, Guindy

Anna University,

Chennai-600025

ACKNOWLEDGEMENT

We express our wholehearted and sincere gratitude to our Head of the Department Dr. M. MEENAKSHI, Professor, Department of Electronics & Communication Engineering for her enthusiastic encouragement and support throughout the project.

We manifest our sincere thanks and gratitude to our project supervisor, Dr. Manimekalai T, Associate Professor, Department of Electronics and Communication Engineering for her ardent support, patience, valuable guidance, technical expertise and encouragement in our project.

We also thank all the teaching and non-teaching staff of Department of Electronics and Communication Engineering, for their kind help and co-operation during the course of our project.

Last but not the least, we would not have made this project without the support from our beloved family and friends.

ABSTRACT

Distributed Denial of Service (DDoS) attacks pose a significant threat to network availability and security. This project focuses on developing a DDoS attack detection system by leveraging Machine Learning (ML) algorithms within the context of Software-Defined Networking (SDN). The proposed system collects network traffic data from various devices, trains an ML model using historical data, and applies anomaly detection or supervised learning techniques to identify potential DDoS attacks. Upon detection, proactive mitigation measures are triggered, facilitated by the integration of the ML-based system with the SDN controller. Through centralized control, programmability, and ML's pattern recognition capabilities, this project aims to enhance network security and resilience, ensuring the availability and integrity of SDN-enabled networks.

Upon detecting a potential DDoS attack, the system triggers proactive mitigation measures, utilizing the programmability of SDN to dynamically adjust network configurations. This may involve actions such as traffic redirection, rate limiting, or prioritization of critical flows. The integration of the ML-based detection system with the SDN controller enables seamless communication and real-time decision-making, facilitating rapid and automated response actions.

Through the combination of SDN's centralized control, programmability, and ML's pattern recognition capabilities, this project aims to enhance network security and resilience against DDoS attacks. The proposed system offers the potential for proactive threat detection, efficient resource utilization, and minimal disruption to network services. By ensuring the availability and integrity of SDN-enabled networks, this project addresses the critical need for robust DDoS attack detection mechanisms in modern network environments.

சுருக்கம்

விநியோகிக்கப்பட்ட சேவை மறுப்பு (DDOS) தாக்குதல்கள் நெட்வொர்க் கிடைப்பதற்கும் பாதுகாப்பிற்கும் குறிப்பிடத்தக்க அச்சுறுத்தலாக உள்ளன. மென்பொருள்-குறிப்பிட்ட நெட்வொர்க்கிங் (எஸ். டி. என்) சூழ்நிலையில் இயந்திரக் கற்றல் வழிமுறைகளைப் பயன்படுத்தி டிடிஓ-வின் தாக்குதல் கண்டறிதல் அமைப்பை உருவாக்குவதில் இந்தத் திட்டம் கவனம் செலுத்துகிறது. உத்தேசிக்கப்பட்டுள்ள அமைப்பு பல்வேறு சாதனங்களிலிருந்து நெட்வொர்க் தரவைச் சேகரிக்கிறது, வரலாற்றுத் தரவுகளைப் பயன்படுத்தி ஒரு ML மாதிரியைப் பயன்படுத்துகிறது, மேலும் சாத்தியமான DDOS தாக்குதல்களை அடையாளம் காண ஒரு குறைபாடு கண்டறிதல் அல்லது மேற்பார்வை கற்றல் நுட்பங்களைப் பயன்படுத்துகிறது. கண்டறியப்பட்டவுடன், ML-அடிப்படையிலான அமைப்பை SDN கட்டுப்பாட்டாளருடன் ஒருங்கிணைப்பதன் மூலம், தீவிர தணிப்பு நடவடிக்கைகள் மேற்கொள்ளப்படுகின்றன. மையப்படுத்தப்பட்ட கட்டுப்பாடு, நிரலாக்க திறன் மற்றும் ML-ன் மாதிரி அங்கீகார திறன்கள் மூலம், இந்தத் திட்டம் நெட்வொர்க் பாதுகாப்பு மற்றும் மனத் திடத்தை மேம்படுத்துவதை நோக்கமாகக் கொண்டுள்ளது, SDN-செயல்திறன் கொண்ட நெட்வொர்க்குகள் கிடைப்பதை உறுதி செய்கிறது.

ஒரு சாத்தியமான DDOS தாக்குதலைக் கண்டுபிடித்ததும், இந்த அமைப்பு செயல்திறன் மிக்க தணிப்பு நடவடிக்கைகளை ஊக்குவிக்கிறது, இது SDN இன் நிரல்திறன் திறனைப் பயன்படுத்தி நெட்வொர்க் உள்ளமைவுகளை சீரமைக்கும். இது போக்குவரத்து மறுஇயக்கம், வட்டி விகித வரம்பு அல்லது முக்கிய பாய்வுகளின் முன்னுரிமை போன்ற செயல்களை உள்ளடக்கியது. ML-அடிப்படையிலான கண்டறிதல் முறையை SDN கட்டுப்பாட்டாளருடன் ஒருங்கிணைப்பது, தடையற்ற தகவல்தொடர்பு மற்றும் நிகழ்நேர முடிவுகளை எடுப்பதில் உதவுகிறது, விரைவான மற்றும் தானியக்க செயல்பாடுகளுக்கு உதவுகிறது.

SDN இன் மையப்படுத்தப்பட்ட கட்டுப்பாடு, நிரலாக்க திறன் மற்றும் ML இன் வடிவமைப்பு அங்கீகார திறன்கள் ஆகியவற்றின் மூலம், இந்தத் திட்டம் நெட்வொர்க் பாதுகாப்பு மற்றும் DDOS தாக்குதல்களுக்கு எதிரான தாங்கு திறனை மேம்படுத்துவதை நோக்கமாகக் கொண்டுள்ளது. உத்தேசிக்கப்பட்டுள்ள அமைப்பு, தீவிர அச்சுறுத்தலைக் கண்டறிதல், திறமையான வளப் பயன்பாடு மற்றும் வலையமைப்பு சேவைகளுக்கு குறைந்த இடையூறாக இருக்கும் திறனை வழங்குகிறது. நவீன வலையமைப்பு சூழல்களில் வலுவான DDOS தாக்குதல் கண்டறியும் வழிமுறைகளின் முக்கிய தேவையை இத்திட்டம் நிறைவேற்றும்.

APPENDIX 3

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The detection of Distributed Denial of Service (DDoS) attacks in Software-Defined Networking (SDN) using machine learning techniques has gained significant attention due to the growing threat of such attacks. Machine learning offers the potential to enhance DDoS detection by automatically analyzing network traffic patterns and identifying malicious activities. This approach allows for proactive mitigation measures, ensuring the availability and security of SDN-enabled networks. By leveraging the power of machine learning algorithms, SDN-based DDoS detection systems can adapt to evolving attack techniques and provide real-time threat detection and response.

1.1 MOTIVATION

The motivation for developing a DDoS attack detection system in a Software Defined Network (SDN) using machine learning arises from the increasing threat of DDoS attacks and the need for efficient and proactive network security measures. DDoS attacks can cause severe disruptions to network availability, impacting businesses, organizations, and individuals relying on network services. Traditional network security approaches struggle to effectively detect and mitigate these attacks due to their complex and evolving nature.

SDN provides a promising solution with its centralized control and programmability, enabling dynamic network management. By leveraging the capabilities of machine learning, specifically in analyzing network traffic patterns, we can develop an intelligent system that can identify and respond to DDoS attacks in real-time. Machine learning algorithms can effectively learn and adapt to the ever-changing attack patterns, improving the accuracy of detection and reducing false positives

1.2 PROJECT FLOW

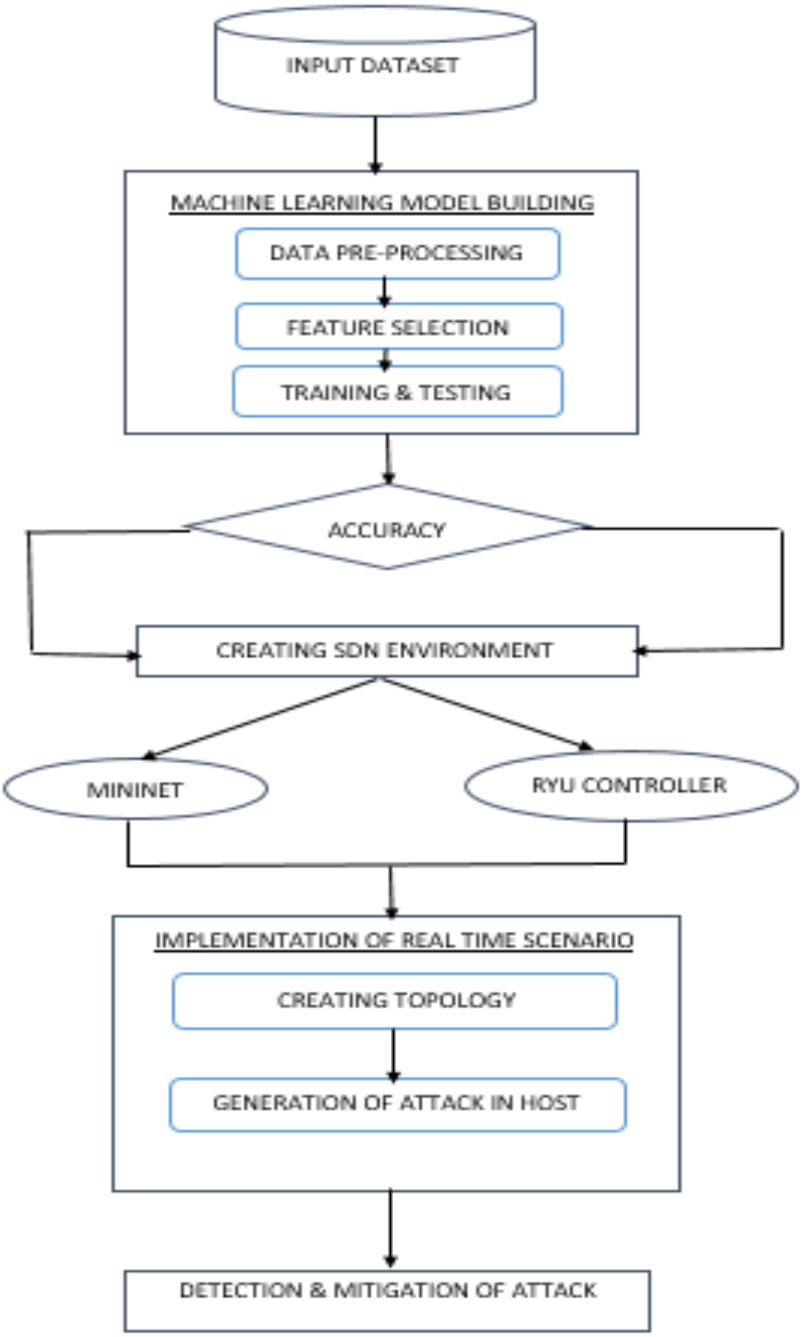


Figure 1.1 Project Work Flow

1.3 SOFTWARE REQUIREMENTS

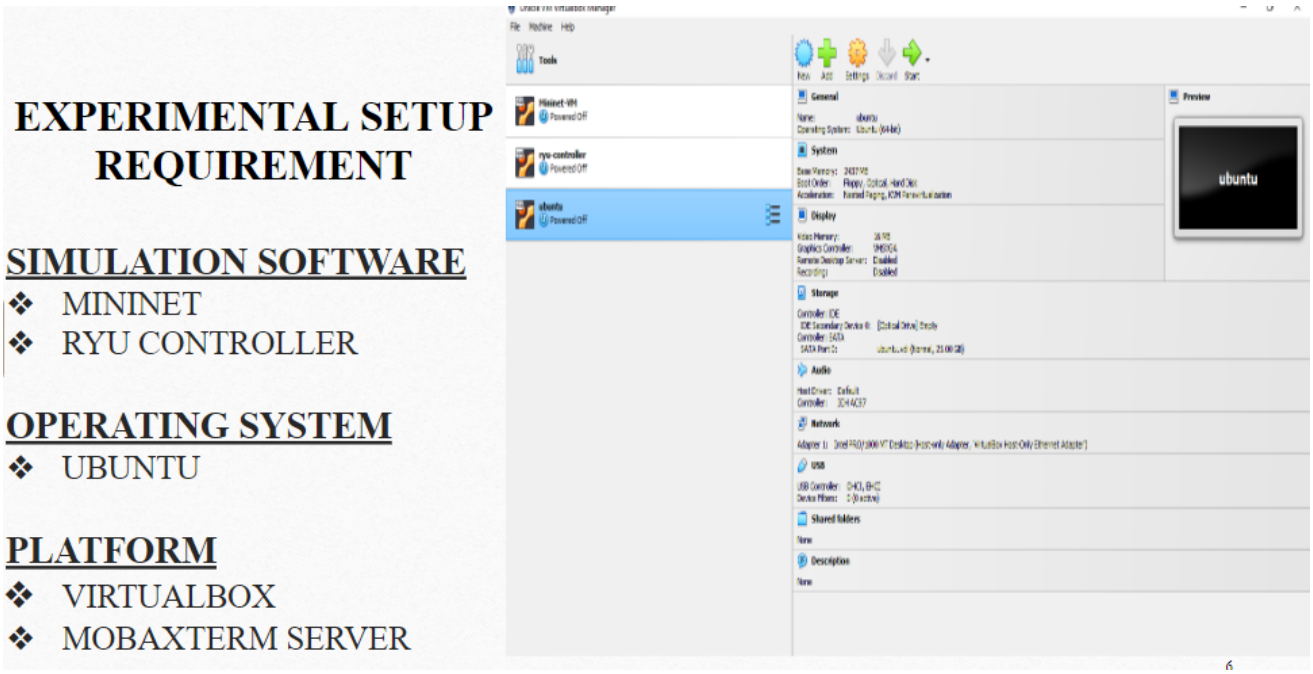


Figure 1.2 Experimental Setup

SIMULATION SOFTWARE:

MININET:

Mininet is an open-source network emulator that allows you to create a virtual network environment on a single machine. It provides a lightweight and scalable platform for testing and developing SDN applications. Mininet enables the creation of a network topology using virtual switches, hosts, and controllers, which can be interconnected to simulate various network scenarios.

Key features of Mininet include:

- **Network Emulation**: Mininet allows the creation of a virtual network topology

with customizable hosts, switches, and links. This enables network emulation for testing and experimentation without the need for physical network equipment.

- **OpenFlow Support**: Mininet integrates with OpenFlow, a protocol used in SDN, allowing you to create software-defined networks with programmable switches. It enables you to simulate OpenFlow-enabled switches and test SDN applications in a controlled environment.

- **Customization**: Mininet provides a Python API that allows users to easily customize the network topology and behavior. You can define custom network configurations, implement custom network protocols, and simulate.

RYU CONTROLLER:

Ryu is an open-source controller framework for SDN. It provides a software platform for developing SDN applications by implementing network control logic. Ryu supports various protocols, including OpenFlow, NETCONF, and others, making it flexible and adaptable to different network environments.

Key features of Ryu Controller include:

- **OpenFlow Controller**: Ryu serves as an OpenFlow controller, allowing it to communicate with OpenFlow-enabled switches and manage their forwarding behavior. It enables the implementation of network control and management applications using OpenFlow-based programmability.

- **Event-Driven Architecture**: Ryu follows an event-driven architecture, where

applications can define event handlers and callbacks to respond to network events and messages. This asynchronous programming model allows for efficient and reactive control of the network.

- **Extensibility**: Ryu provides a modular architecture that allows developers to extend its functionality by implementing custom applications and modules. It offers a rich set of APIs and libraries for rapid development of SDN applications.

- **Python-Based**: Ryu is primarily written in Python, which makes it accessible and easy to use for developers familiar with the language. The Python programming language provides simplicity and flexibility for rapid prototyping and development of SDN applications.

Together, Mininet and Ryu form a powerful combination for SDN development and testing. Mininet allows the creation of virtual network topologies for testing SDN applications, while Ryu provides the controller framework to develop and deploy those applications, enabling network programmability and control.

CHAPTER 2

LITERATURE SURVEY

Here's a brief literature review on the topic of detecting DDoS attacks in SDN using machine learning techniques:

1. Title: "DDoS attack detection using machine learning algorithms in SDN"
   Authors: A. Ramyashri, V. Lalitha, R. S. Moni
   Published: 2017

This study presents a comparative analysis of machine learning algorithms for DDoS attack detection in an SDN environment. The authors evaluate the performance of various algorithms, including Random Forest, Support Vector Machine (SVM), and k-Nearest Neighbors (k-NN), using different feature selection techniques. The results demonstrate the effectiveness of machine learning in accurately detecting DDoS attacks in SDN.

2. Title: "Machine learning-based DDoS detection in SDN using traffic flow analysis"
   Authors: N. Majumder, D. Bhattacharyya
   Published: 2018

The authors propose a DDoS detection approach based on traffic flow analysis in an SDN architecture. They employ machine learning algorithms, specifically Deep Belief Networks (DBNs), to classify normal and attack traffic based on flow features. Experimental results show that the proposed approach effectively detects and mitigates DDoS attacks in an SDN environment.

3. Title: "DDoS attack detection and mitigation using machine learning in SDN"
   Authors: T. Nguyen, K. Nguyen, M. Nguyen, D. Thai, H. Tran
   Published: 2019

This study investigates the application of machine learning algorithms, such as Decision Tree, SVM, and Naïve Bayes, for DDoS attack detection and mitigation in SDN. The authors propose a hybrid approach that combines multiple algorithms to improve detection accuracy. Experimental results demonstrate the effectiveness of the proposed approach in accurately identifying DDoS attacks while minimizing false positives.

4. Title: "Distributed DDoS detection scheme in SDN using machine learning techniques"
   Authors: C. Jiang, Y. Wang, Q. Zhang, W. Chen
   Published: 2020

The authors propose a distributed DDoS detection scheme based on machine learning techniques in an SDN environment. They leverage the power of SDN controllers and apply Random Forest and SVM algorithms to classify normal and malicious traffic. The scheme utilizes multiple SDN controllers in a distributed manner to enhance detection accuracy and minimize detection time.

5. Title: "DDoS attack detection in SDN using machine learning-based traffic flow analysis"
   Authors: S. Gupta, P. Chhabra, A. Sood
   Published: 2021

This research focuses on DDoS attack detection in an SDN architecture using machine learning-based traffic flow analysis. The authors employ machine learning algorithms,

such as k-NN, Decision Tree, and Gradient Boosting, to classify network traffic into normal or malicious. Experimental results demonstrate the effectiveness of the proposed approach in accurately detecting DDoS attacks in an SDN environment.

6. Title: "A Flow-Based Anomaly Detection Approach With Feature Selection Method Against DDoS Attacks in SDNs"
   Authors: Mahmoud Said El Sayed, Nhien-An Le-Khac , Senior Member, IEEE,Marianne. A. Azer , and Anca D. Jurcut
   Published: 2022

This research focuses on DDOS Attack detection in an SDN architecture using machine learning techniques.The auther uses two popular feature selection methods, i.e., Information Gain (IG) and Random Forest (RF) in order to analyse the most comprehensive relevant features of DDoS attacks in SDN networks. Using the most relevant features will improve the accuracy of the anomaly detection system and reduce the false alarm rates. Moreover, we propose a Deep Learning (DL) technique. We perform our analysis and evaluation on three different datasets. The results validate that the DL approach can efficiently identify DDoS attacks in SDN environments without any significant degradation in the controller performance.

CHAPTER 3

SYSTEM ARCHITECTURE

SDN (Software-Defined Networking) system architecture is designed to separate the control plane from the data plane, providing a flexible and programmable approach to network management. The architecture consists of three main components: the application layer, the control layer, and the infrastructure layer.

3.1 OVERVIEW OF SDN ARCHITECTURE

SDN architecture revolutionizes networking by decoupling the control plane from the data plane, enabling centralized control and programmability. It comprises three components: the application layer for network applications to interact with the controller, the control layer housing the controller for centralized network management, and the infrastructure layer consisting of programmable network devices. APIs facilitate communication between the layers, allowing applications to leverage the controller's capabilities, which maintains a global network view and instructs devices on forwarding decisions. SDN offers flexibility, scalability, and simplified management, promoting innovation in network applications and services. Its separation of control and data planes empowers administrators to efficiently optimize networks, enhancing performance and capabilities.

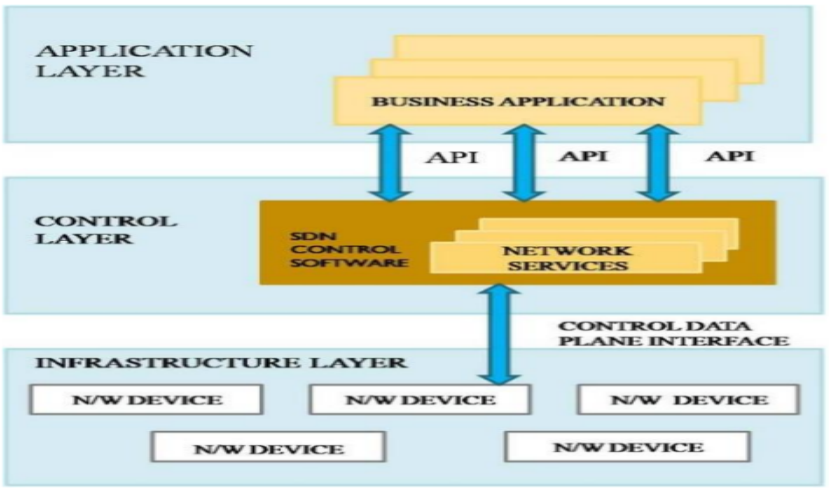


Figure 3.1 SDN Architecture

The infrastructure layer comprises the programmable network devices that forward traffic. These devices, also known as forwarding elements, can be physical or virtual switches, routers, or other network equipment. They rely on the instructions provided by the controller to determine how to process and forward network packets. The communication between the control layer and the infrastructure layer is facilitated through southbound APIs. These APIs allow the controller to configure and manage the behavior of the network devices. They enable the controller to push forwarding rules, modify network flows, and collect network statistics from the devices.

3.2 DDOS ATTACKS TYPE AND CHARACTERISTIC

DDoS (Distributed Denial of Service) attacks are malicious attempts to disrupt the availability of a network, system, or service by overwhelming it with a flood of illegitimate traffic. These attacks can cause significant downtime, loss of revenue, and damage to the reputation of targeted entities.

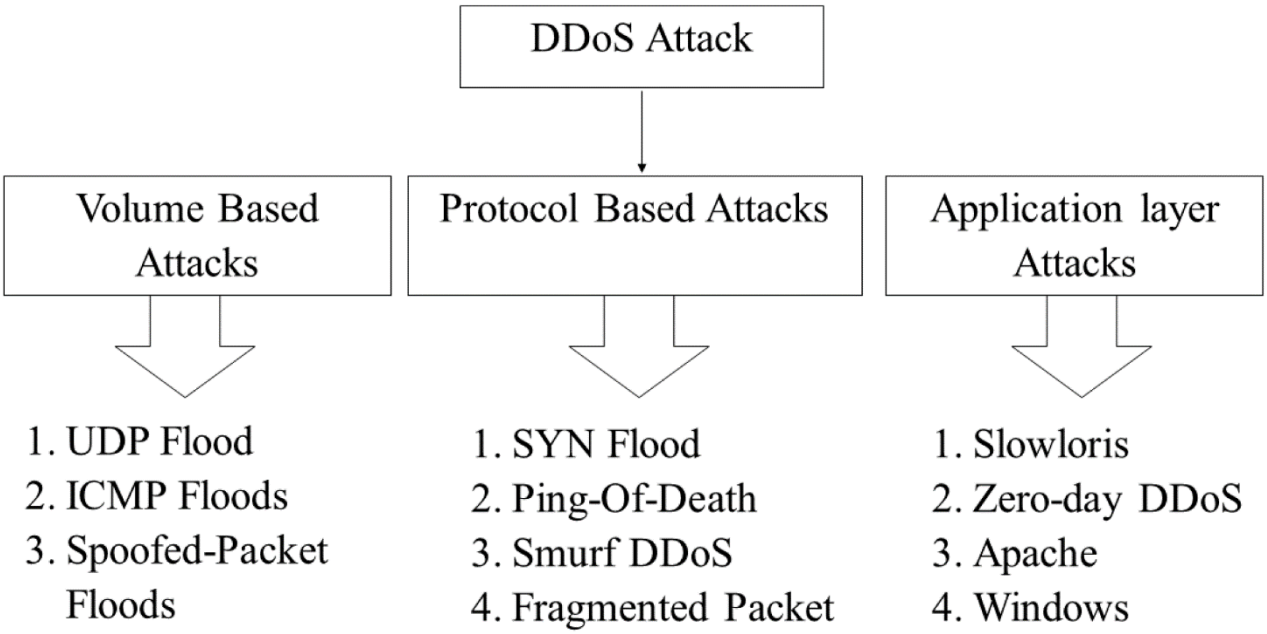


Figure 3.2 DDoS Attack Type

There are several types of DDoS attacks:

- Volumetric Attacks: These attacks aim to overwhelm the target's network bandwidth by flooding it with a massive volume of traffic. Attackers often use botnets, which are networks of compromised computers, to generate and direct the traffic towards the target.

- TCP/IP Protocol Attacks: These attacks exploit vulnerabilities in the TCP/IP protocol stack. Examples include SYN floods, where attackers send a large number of SYN requests to exhaust server resources, and UDP floods, where attackers flood the target with a high volume of UDP packets.

- Application Layer Attacks: These attacks target vulnerabilities in specific applications or services running on the target system. They aim to exhaust server resources or exploit application weaknesses. Examples include HTTP

floods, which overwhelm web servers with a high volume of HTTP requests, and DNS amplification attacks, which use misconfigured DNS servers to flood the target with DNS responses.

- Fragmentation Attacks: These attacks exploit weaknesses in the way network devices handle fragmented packets. Attackers send a large number of fragmented packets, overwhelming the target's resources when it tries to reassemble them.

- Reflective Attacks: These attacks involve using a large number of innocent third-party systems to unwittingly participate in the attack. Attackers spoof the source IP address, making the attack traffic appear to originate from the third-party systems. This makes it harder to trace the attack back to the actual attacker.

DDoS attacks continue to evolve, with attackers employing various techniques to bypass traditional defenses. It is crucial for organizations to implement robust DDoS detection and mitigation strategies to protect their networks and ensure uninterrupted service availability.

3.3 DDOS ATTACK IN SOFTWARE DEFINED NETWORK

The security implications of Software-Defined Networking (SDN) architecture are complex. While SDN offers the potential for improved network security through centralized control and new security tools, it also introduces new vulnerabilities and attack vectors. Attacks on SDN can target the controller or the communication links between the controller and the underlying infrastructure devices. Additionally, traditional attacks that affect current networks can also be directed towards SDN networks, but with potentially more severe consequences.

One of the most significant and damaging attacks in SDN is the Distributed Denial of Service (DDoS) attack. In a DDoS attack, the attacker generates a high volume of traffic from spoofed IP addresses, overwhelming the network and making the controller inaccessible to legitimate users. SDN is susceptible to various types of DDoS attacks that exploit specific weaknesses in the architecture.

Some notable DDoS attacks specific to SDN include Buffer Saturation Attacks, where the attacker fills the switch's buffer memory with a large number of fake packets, causing legitimate packets to be dropped. Flow Table Overflow attacks exploit the limited space of the switch's flow tables, flooding them with useless rules and making it unable to handle legitimate traffic. Link Flooding Attacks overload the bandwidth between flow switches and the controller, creating a bottleneck for legitimate traffic. Controller Saturation attacks target the limited resources of the controller, overwhelming it with a large number of incoming flows and potentially causing the entire network to crash.

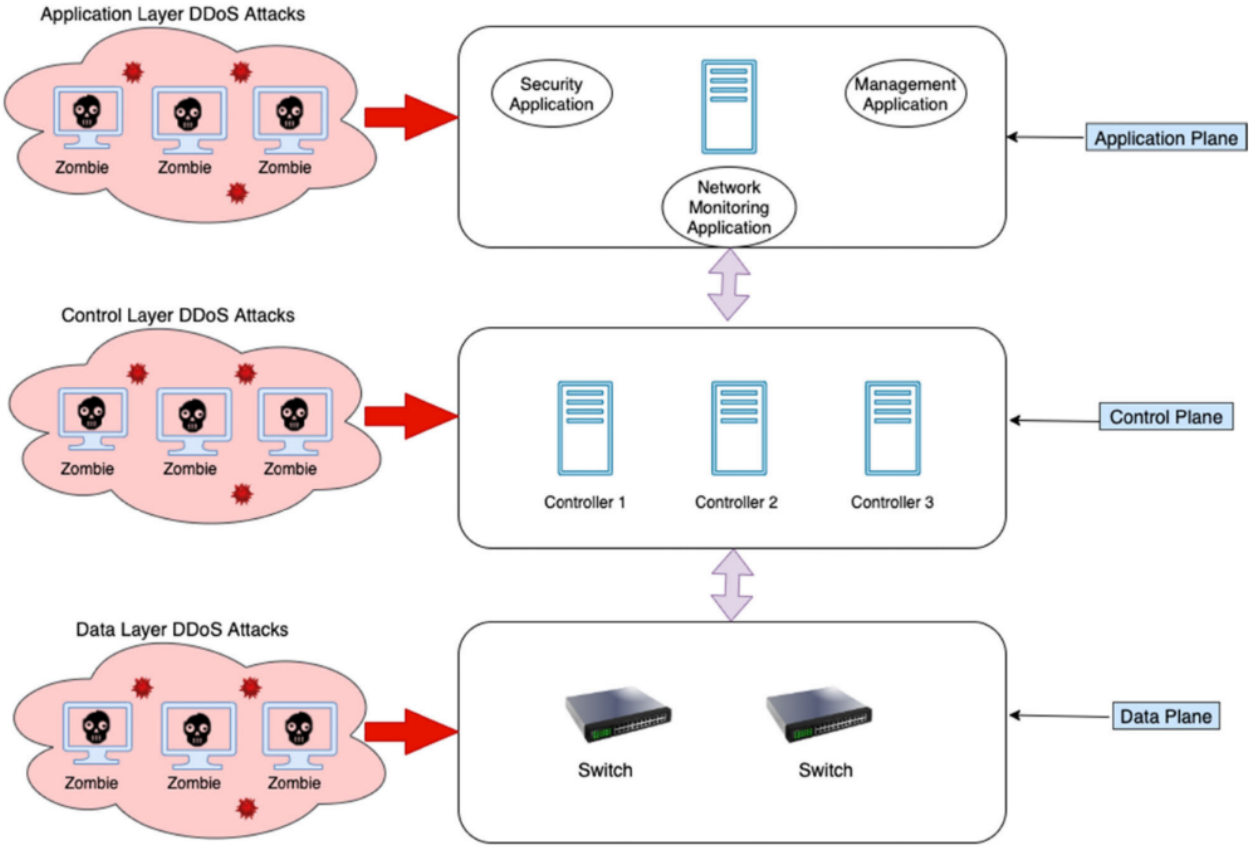


Figure 3.3 DDOS Attack in SDN

Protecting SDN networks from DDoS attacks has become a critical area of research. It is essential to develop effective defense mechanisms and mitigation strategies to safeguard the network resources and ensure the uninterrupted operation of SDN networks.

CHAPTER 4

DATASET

This chapter deals with the SDN specific dataset generated by using mininet emulator and used for traffic classification by machine learning and deep learning algorithms.

4.1 DATASET DESCRIPTION

The project start by creating ten topologies in mininet in which switches are connected to single Ryu controller. Network simulation runs for benign TCP, UDP and ICMP traffic and malicious traffic which is the collection of TCP Syn attack, UDP Flood attack, ICMP attack. Total 23 features are available in the data set in which some are extracted from the switches and others are calculated. Extracted features include Switch-id, Packet_count, byte_count, duration_sec, duration_nsec which is duration in nano-seconds, total duration is sum of duration_sec and duration_nsec, Source IP, Destination IP, Port number, tx_bytes is the number of bytes transferred from the switch port, rx_bytes is the number of bytes received on the switch port.dt field show the date and time which has been converted into number and a flow is monitored at a monitoring interval of 30 second. Calculated features include Packet per flow which is packet count during a single flow, Byte per flow is byte count during a single flow, Packet Rate is number of packets send per second and calculated by dividing the packet per flow by monitoring interval, number of Packet_ins messages, total flow entries in the switch, tx_kbps, rx_kbps are data transfer and receiving rate and Port Bandwidth is the sum of tx_kbps and rx_kbps. Last column indicates the class label which indicates whether the traffic type is benign or malicious .Benign traffic has label 0 and malicious traffic has label 1.

4.2 DATA PRE-PROCESSING

Data preprocessing refers to the steps taken to transform raw data into a format suitable for machine learning algorithms. It is an essential part of the machine learning pipeline and helps improve the quality and effectiveness of the models built on the data. Some common techniques used are:

- Finding Infinity and Mean Values

FINDING INFINITY AND MEAN VALUES

```
data = pd.read_csv('dataset_sdn.csv')
data = data.replace(np.inf,np.nan)    # replacing inf with nan
data = data.fillna(data.mean(numeric_only=True)) # then converting nan to mean values
```

Figure 4.2.1 Finding Infinity and Mean Values

- Independent and Dependent Variable

```
## independent varibles-X
X=data.iloc[:,:-1]

X.head()
```

	timestamp	datapath_id	flow_id	ip_src	tp_src	ip_dst	tp_dst	ip_proto	icmp_code	icmp_type	...	flow_duration_nsec	idle_time
0	1.679206e+09	4	10.0.0.11010.0.0.601	10.0.0.11	0	10.0.0.6	0	1	0	0	...	28000000	
1	1.679206e+09	4	10.0.0.6010.0.0.1101	10.0.0.6	0	10.0.0.11	0	1	0	8	...	32000000	
2	1.679206e+09	2	10.0.0.1505010.0.0.6585906	10.0.0.1	5050	10.0.0.6	58590	6	-1	-1	...	57000000	
3	1.679206e+09	2	10.0.0.11505010.0.0.6585901	10.0.0.11	5050	10.0.0.6	58590	1	0	0	...	23000000	
4	1.679206e+09	2	10.0.0.65859010.0.0.150506	10.0.0.6	58590	10.0.0.1	5050	6	0	0	...	62000000	

5 rows × 21 columns

```
# y target-dependent variable
y=data.label
y
```

```
0         0
1         0
2         0
3         0
4         0
        ..
898438    1
898439    1
898440    1
898441    1
898442    1
Name: label, Length: 898443, dtype: int64
```

Figure 4.2.2 Independent and Dependent Variable

- Five Number summery

	timestamp	datapath_id	tp_src	tp_dst	ip_proto	icmp_code	icmp_type	flow_duration_sec	flow_duration_nsec	i
count	8.984430e+05	898443.000000	898443.000000	898443.000000	898443.000000	898443.000000	898443.000000	898443.000000	8.984430e+05	
mean	1.679208e+09	3.660857	24423.457933	153.388389	8.491539	-0.769574	1.056207	8.672932	4.918402e+08	
std	1.456409e+02	1.437624	21877.163522	2210.749271	6.287143	0.421106	3.776276	6.315401	2.882243e+08	
min	1.679206e+09	1.000000	0.000000	0.000000	1.000000	-1.000000	-1.000000	0.000000	0.000000e+00	
25%	1.679208e+09	3.000000	1488.500000	0.000000	6.000000	-1.000000	-1.000000	3.000000	2.430000e+08	
50%	1.679208e+09	4.000000	20110.000000	0.000000	6.000000	-1.000000	-1.000000	10.000000	4.860000e+08	
75%	1.679208e+09	5.000000	43438.000000	80.000000	17.000000	-1.000000	-1.000000	14.000000	7.410000e+08	
max	1.679208e+09	6.000000	65535.000000	60828.000000	17.000000	0.000000	8.000000	100.000000	9.990000e+08	

Figure 4.2.3 Five Number summery

- Feature selection

```
In [21]: #FEATURE SELECTION
         # Train a Random Forest classifier
         from sklearn.feature_selection import mutual_info_classif
         from sklearn.ensemble import RandomForestClassifier
         rfc = RandomForestClassifier(n_estimators=100)
         rfc.fit(X, y)

         # Get the feature importance scores
         importances = rfc.feature_importances_

         # Create a DataFrame to display the feature importance scores
         feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
         feature_importances = feature_importances.sort_values('Importance', ascending=False).reset_index(drop=True)

         # Print the top 10 important features
         print(feature_importances.head(10))
         print(feature_importances.tail(10))
```

```
                    Feature  Importance
0                 timestamp    0.267552
1                    ip_src    0.201648
2                byte_count    0.147808
3              packet_count    0.113874
4   packet_count_per_second    0.074799
5      byte_count_per_second    0.059197
6    byte_count_per_nsecond    0.050787
7                    tp_dst    0.045100
8         flow_duration_sec    0.020920
9                   flow_id    0.006195
                    Feature  Importance
11                   ip_dst    0.003820
12                   tp_src    0.001482
13                icmp_type    0.000253
14              datapath_id    0.000240
15                icmp_code    0.000195
16                 ip_proto    0.000164
17  flow_duration_nsec    0.000026
18         idle_timeout    0.000000
```

Figure 4.2.4 Feature selection

- Correlation

```
sns.heatmap(data.corr())
```

```
<AxesSubplot:>
```

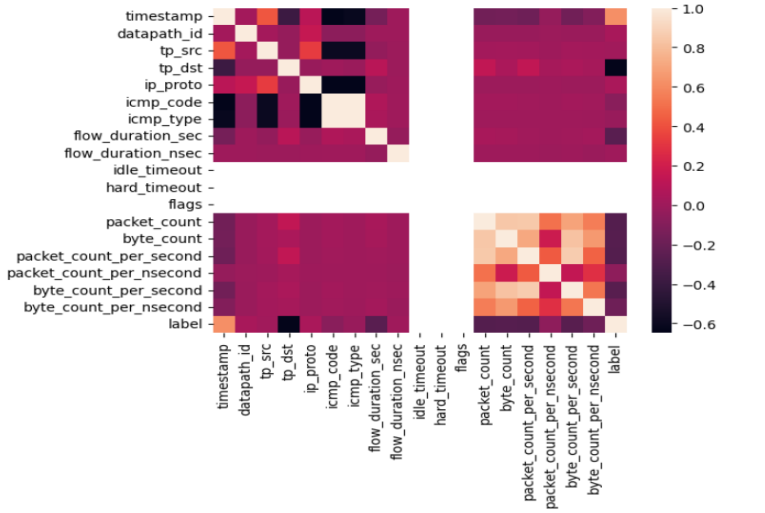


Figure 4.2.5 Correlation

CHAPTER 5

MACHINE LEARNING MODELS

Machine learning offers the potential to enhance DDoS detection by enabling automated, adaptive, and intelligent analysis of network traffic patterns. By learning from historical data, machine learning algorithms can identify abnormal traffic behaviors associated with DDoS attacks and differentiate them from normal network traffic.

5.1 Classification Algorithm for DDoS Detection

a) **Logistic Regression**: Logistic Regression is a linear classification algorithm that models the relationship between the input features and the probability of a binary outcome. It can be employed for DDoS detection by learning the parameters that best fit the training data and making predictions based on the calculated probabilities.

```
#LOGISTIC REGRESSION
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
classifierlr = LogisticRegression(solver='liblinear', random_state=0)
flow_modellr = classifierlr.fit(X_train, y_train)

y_predlr = flow_modellr.predict(X_test)


print("Confusion Matrix")
cmlr = confusion_matrix(y_test, y_predlr)
print(cmlr)

acclr = accuracy_score(y_test, y_predlr)

print("Success Accuracy = {0:.2f} %".format(acclr*100))
faillr = 1.0 - acclr
print("Fail Accuracy = {0:.2f} %".format(faillr*100))
```

```
Confusion Matrix
[[41995   232]
 [ 3925 28344]]
Success Accuracy = 94.42 %
Fail Accuracy = 5.58 %
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predlr))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	3145
1	0.99	1.00	1.00	446077
accuracy			0.99	449222
macro avg	0.50	0.50	0.50	449222
weighted avg	0.99	0.99	0.99	449222

Figure 5.1.1 Logistic Regression

b) **Random Forest:** Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. It is effective for DDoS detection due to its ability to handle high-dimensional data and identify complex patterns in the network traffic.

```
#RANDOM FOREST
classifierrf =RandomForestClassifier(n_estimators=10, criterion="entropy", random_state=42)
flow_modelrf = classifierrf.fit(X_train, y_train)

y_predrf = flow_modelrf.predict(X_test)


print("Confusion Matrix")
cmrf = confusion_matrix(y_test, y_predrf)
print(cmrf)

accrf = accuracy_score(y_test, y_predrf)

print("Success Accuracy = {0:.2f} %".format(accrf*100))
failrf = 1.0 - accrf
print("Fail Accuracy = {0:.2f} %".format(failrf*100))
```

```
Confusion Matrix
[[  3750      0]
 [     0 535316]]
Success Accuracy = 100.00 %
Fail Accuracy = 0.00 %
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predrf))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	76954
1	1.00	1.00	1.00	58493
accuracy			1.00	135447
macro avg	1.00	1.00	1.00	135447
weighted avg	1.00	1.00	1.00	135447

Figure 5.1.2 Random Forest

c) **Naive Bayes:** Naive Bayes is a probabilistic algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label. Naive Bayes can be effective for DDoS detection when features such as packet size, packet frequency, and traffic source are considered.

```
#Naive Bayes
classifiernb = GaussianNB()
flow_modelnb = classifiernb.fit(X_train, y_train)

y_prednb = flow_modelnb.predict(X_test)


print("Confusion Matrix")
cmnb = confusion_matrix(y_test, y_prednb)
print(cmnb)

accnb = accuracy_score(y_test, y_prednb)

print("Success Accuracy = {0:.2f} %".format(accnb*100))
failnb = 1.0 - accnb
print("Fail Accuracy = {0:.2f} %".format(failnb*100))
```

```
Confusion Matrix
[[42181    46]
 [14268 18001]]
Success Accuracy = 80.79 %
Fail Accuracy = 19.21 %
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_preddt))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	42227
1	1.00	1.00	1.00	32269
accuracy			1.00	74496
macro avg	1.00	1.00	1.00	74496
weighted avg	1.00	1.00	1.00	74496

Figure 5.1.3 Naive Bayes

d) **k-Nearest Neighbours (k-NN):** k-NN is a non-parametric algorithm that classifies new data points based on the majority class of their k nearest

neighbours. It can be applied to DDoS detection by measuring the similarity of network traffic patterns and classifying them accordingly.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
flow_model = classifier.fit(X_train, y_train)

y_pred = flow_model.predict(X_test)


print("Confusion Matrix")
cm = confusion_matrix(y_test, y_pred)
print(cm)

acc = accuracy_score(y_test, y_pred)

print("Success Accuracy = {0:.2f} %".format(acc*100))
fail = 1.0 - acc
print("Fail Accuracy = {0:.2f} %".format(fail*100))
```

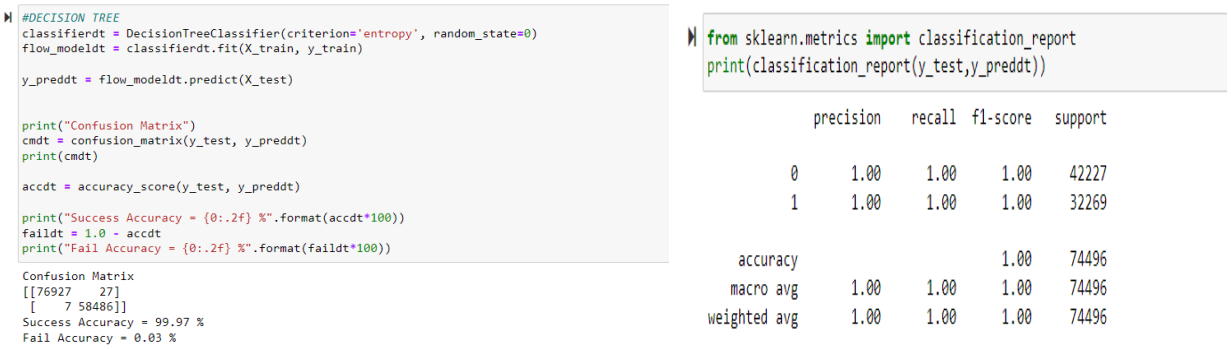
```
Confusion Matrix
[[42165    62]
 [  150 32119]]
Success Accuracy = 99.72 %
Fail Accuracy = 0.28 %
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1267
1	1.00	1.00	1.00	178422
accuracy			1.00	179689
macro avg	1.00	1.00	1.00	179689
weighted avg	1.00	1.00	1.00	179689

Figure 5.1.4 KNN

e) **Decision Tree:** Decision tree algorithms are commonly used for DDoS detection due to their ability to handle categorical and numerical data while providing interpretable and explainable models. CART is another popular decision tree algorithm. It can handle both classification and regression problems. CART can also perform tree pruning to avoid overfitting CART is another popular decision tree algorithm. It can handle both classification and regression problems. CART builds binary trees by recursively splitting data based on Gini impurity or entropy measures. It supports categorical and numerical attributes and handles missing values. CART can also perform tree pruning to avoid overfitting.



```
#DECISION TREE
classifierdt = DecisionTreeClassifier(criterion='entropy', random_state=0)
flow_modeldt = classifierdt.fit(X_train, y_train)

y_preddt = flow_modeldt.predict(X_test)


print("Confusion Matrix")
cmdt = confusion_matrix(y_test, y_preddt)
print(cmdt)

accdt = accuracy_score(y_test, y_preddt)

print("Success Accuracy = {0:.2f} %".format(accdt*100))
faildt = 1.0 - accdt
print("Fail Accuracy = {0:.2f} %".format(faildt*100))
```

```
Confusion Matrix
[[76927    27]
 [    7 58486]]
Success Accuracy = 99.97 %
Fail Accuracy = 0.03 %
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_preddt))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	42227
1	1.00	1.00	1.00	32269
accuracy			1.00	74496
macro avg	1.00	1.00	1.00	74496
weighted avg	1.00	1.00	1.00	74496

Figure 5.1.5 Decision Tree

5.2 MODEL TRAINING AND EVALUATION

a) **Splitting the Dataset:** Divide the pre-processed dataset into training and testing sets. The training set is used to train the machine learning model, while the testing set is used to evaluate the model's performance. It is important to maintain a balance between the two sets, ensuring they have a representative distribution of normal traffic and DDoS attacks.

b) **Model Training:** Train the selected machine learning model using the training dataset. The model learns to recognize patterns and features that distinguish normal traffic from DDoS attacks. Adjust the model's hyper parameters through techniques like cross-validation or grid search to optimize its performance.

c) **Model Evaluation:** Evaluate the trained model using the testing dataset. Measure its performance metrics, such as **accuracy, precision, recall, F1 score.** These metrics provide insights into how well the model performs in distinguishing normal traffic from DDoS attacks.

d) **Model Optimization:** If the model's performance is not satisfactory, you can further optimize it by adjusting hyper parameters, trying different feature selection techniques, or exploring different machine learning algorithms. This iterative process helps improve the model's accuracy and effectiveness.

DDoS attacks are dynamic and can evolve over time. Therefore, continuous monitoring and adaptation of the machine learning models are necessary to ensure robust and effective detection in real-world scenarios.

5.3 PERFORMANCE METRICS

a) **Accuracy**: Accuracy measures the proportion of correctly classified instances

(both normal and DDoS attacks) out of the total number of instances in the dataset. However, accuracy alone may not be sufficient when dealing with imbalanced datasets.

Accuracy = (Number of Correctly Classified Instances)/(Total Number of Instances)

b) **Precision:** Precision represents the ability of the model to correctly identify DDoS attacks among the instances it classified as attacks. Precision indicates the model's ability to avoid false positives

Precision = True Positives / (True Positives + False Positives)

c) **Recall (Sensitivity):** Recall measures the proportion of correctly classified DDoS attacks among all actual DDoS attacks. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

Recall = True Positives / (True Positives + False Negatives)

d) **F1 Score:** The F1 score combines precision and recall into a single metric, providing a balanced measure of the model's performance. It is the harmonic mean of precision and recall and is useful when the dataset is imbalanced.

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

The F1 score ranges between 0 and 1, with higher values indicating better performance.

Table 5.3.1 ML Performance Metrics

	LOGISTIC REGRESSION	**RANDOM FOREST**	**KNN**	**DECISSION TREE**	**NAÏVE BAYES**
ACCURACY	94.42%	100%	99.7%	99.97%	80.79%
PRECISION	0.99	1	0.99	0.99	0.99
RECALL	0.87	1	0.99	0.99	0.55
F1 SCORE	0.93	1	0.99	0.99	0.71

CHAPTER 6

DEEP LEARNING MODELS

6.1 Deep Learning Algorithms for DDOS Detection:

CNN (Convolution Neural Network): Convolutional Neural Networks (CNNs) have been successfully applied to DDoS attack detection due to their ability to capture spatial and temporal patterns in data. Here's how CNNs can be used in the context of DDoS attack detection:

1. Convolutional Layers: The convolutional layers in a CNN consist of filters or kernels that convolve with the input data to capture local patterns and features. In the case of DDoS attack detection, these filters can capture spatial patterns in network traffic, such as specific packet sizes or rate fluctuations. The convolutional layers perform feature extraction by applying these filters across the input data, generating feature maps.
2. Pooling Layers: After the convolutional layers, pooling layers are often used to down sample the feature maps. Pooling reduces the spatial dimensionality of the feature maps while preserving important features. Common pooling techniques include max pooling or average pooling. The down sampling helps the CNN focus on the most informative features and reduces the computational complexity.
3. Activation Functions and Regularization: Activation functions such as ReLU (Rectified Linear Unit) are commonly used to introduce non-linearity into the CNN model. Non-linear activation functions help the model learn complex relationships between features. Additionally, regularization techniques such as dropout or batch normalization can be applied to prevent over fitting and improve the generalization of the model.

By leveraging the spatial and temporal patterns in network traffic data, CNNs can learn discriminative features for detecting DDoS attacks. However, it's important to have a diverse and representative dataset, carefully pre-process the data, and fine-tune the CNN architecture and hyper parameters to achieve optimal performance in DDoS attack detection.

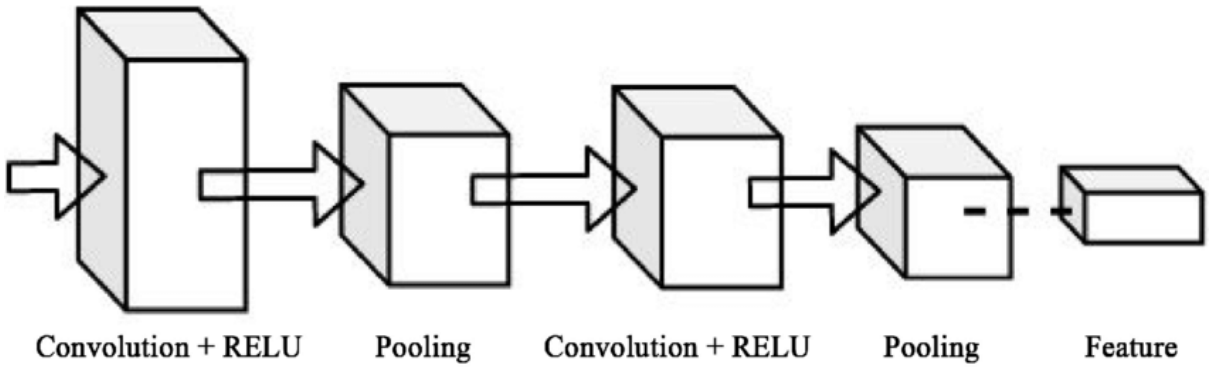


Figure 6.1.1 Convolution Neural Network

ANN (Artificial Neural Networks): Artificial Neural Networks (ANNs), also known as Multilayer Perceptron's (MLPs), can be used for DDoS attack detection. ANNs are a type of feedforward neural network that can learn complex patterns and make predictions based on input data. Here's how ANNs can be applied to DDoS attack detection:

1. Architecture: ANNs typically consist of an input layer, one or more hidden layers, and an output layer. The number of hidden layers and the number of neurons in each layer can vary depending on the complexity of the problem and the available resources. The hidden layers allow the ANN to learn hierarchical representations of the input features, capturing relevant patterns and relationships.
2. Activation Functions: Activation functions introduce non-linearity into the ANN, enabling it to learn complex relationships between the input features. Common activation functions used in ANNs include sigmoid, tan h, and ReLU (Rectified Linear Unit). These functions determine the output of each neuron in the network based on its weighted inputs.

ANNs can effectively capture non-linear relationships and patterns in network traffic data,

making them suitable for DDoS attack detection. However, it's essential to have a well-prepared dataset, handle class imbalance, and carefully tune the architecture and hyper parameters of the ANN to achieve accurate and reliable detection results.

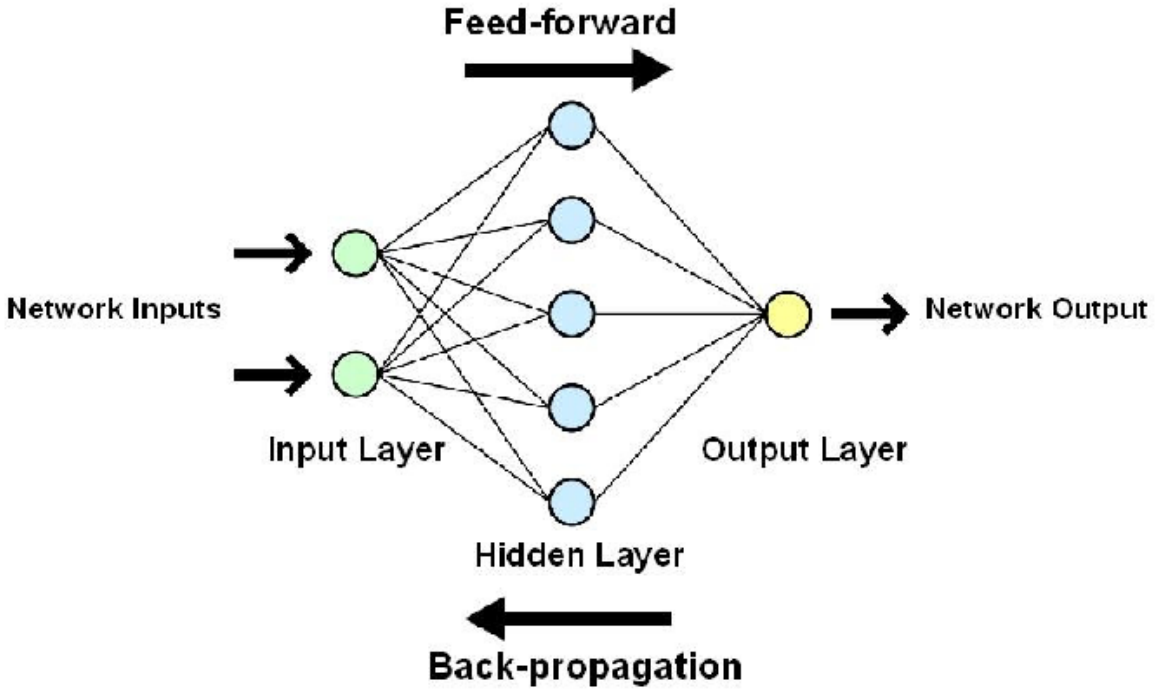


Figure 6.1.2 Artificial Neural Network

6.2 Model Training And Evaluation

a) **Training Process:** Iterate through the training data by feeding batches of input data into the model. For each batch, perform a forward pass through the model to obtain predictions, calculate the loss using the chosen loss function, and then perform backpropagation to update the model's weights. Repeat this process for a specified number of epochs or until the model converges.

b) **Hyper parameter Tuning:** Experiment with different hyper parameters, including learning rate, batch size, number of layers, number of neurons, activation functions, regularization techniques (such as dropout or L2 regularization), and others. Adjust

these hyper parameters to find the optimal combination that maximizes the model's performance.

c) Validation and Evaluation: Periodically evaluate the model's performance on the validation set during training to monitor its progress and prevent overfitting. Once training is complete, evaluate the model's performance on a separate testing dataset to assess its generalization ability. Calculate performance metrics such as accuracy, precision, recall, F1 score, and AUC-ROC to measure the model's effectiveness.

d) Epoch: In the context of deep learning, an epoch refers to a complete pass through the entire training dataset during the training process. During each epoch, the model processes all the training samples and adjusts its weights and biases based on the computed errors or losses.

CNN:

```
epochs, batch_size = 110, 200
n_steps, n_features = x_train.shape[1], x_train.shape[2]


model = Sequential()

model.add(Conv1D(filters=128, kernel_size=3, padding='valid', activation='relu', input_shape=(n_steps,n_features)))
model.add(Conv1D(filters=128, kernel_size=3, padding='valid', activation='relu'))

model.add(MaxPooling1D(pool_size=1))


model.add(Flatten())

model.add(Dense(150, activation='relu'))

model.add(Dropout(0.75))

model.add(Dense(9, activation='softmax'))
model.compile( optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'])

model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size)
```

Figure 6.2.1 CNN Model

The provided code snippet represents a neural network model built using the Keras library. It is a convolutional neural network (CNN) model with the following architecture:

- Input layer: The input shape of the network is determined by the n_steps (number of time steps) and n_features (number of features) variables derived from the shape of the training data.
- Convolutional layers: Two convolutional layers are added with 128 filters each, a kernel size of 3, and a ReLU activation function. The first layer takes the input shape, and the subsequent layers automatically infer the shape.
- MaxPooling layer: A MaxPooling layer with a pool size of 1 is added to reduce the dimensionality of the data.
- Flatten layer: Flattens the input data into a one-dimensional vector.
- Dense layers: A fully connected Dense layer with 150 units and a ReLU activation function is added. Dropout regularization with a rate of 0.75 is applied to reduce overfitting.
- Output layer: A Dense layer with 9 units (assuming it represents a classification problem with 9 classes) and a softmax activation function is added to obtain the predicted probabilities for each class.
- Model compilation: The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and sparse categorical accuracy metric.
- Model training: The model is trained using the fit function with the provided training data (x_train and y_train). The number of epochs and batch size used for training is determined by the epochs and batch_size variables, respectively.

ANN:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(X, Y, test_size= 0.2, random_state=0)
from sklearn.preprocessing import StandardScaler  # scaling of the data

scaler_X = StandardScaler()
x_train_scaled = scaler_X.fit_transform(x_train) # preprocessed training data
x_test_scaled = scaler_X.fit_transform(x_test) # preprocessed testing data
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    keras.layers.Dense(64, input_shape=(21,), activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(2, activation='sigmoid')])
```

Figure 6.2.2 ANN Model

The updated code snippet includes additional steps for data preprocessing and splitting the dataset into training and testing sets. It also An artificial neural network architecture.

- Importing libraries: The train_test_split function from the sklearn.model_selection module is imported for splitting the data, and the StandardScaler class from sklearn.preprocessing is imported for data scaling.

- Splitting the dataset: The train_test_split function is used to split the input data X and target labels Y into training and testing sets. The test_size parameter is set to 0.2, indicating that 20% of the data will be used for testing, while 80% will be used for training. The random_state parameter is set to 0 to ensure reproducibility.

- Data scaling: Two instances of StandardScaler are created, one for scaling the training data (x_train) and another for scaling the testing data (x_test). The fit_transform method is used to perform the scaling operation, which standardizes the features by subtracting the mean and scaling to unit variance.

- Neural network model: The model is defined using the Sequential class from tensorflow.keras. It consists of three dense (fully connected) layers. The first layer has 64 units, takes an input shape of (21,), and uses the ReLU activation function.

The second layer has 32 units and also uses the ReLU activation function. The final layer has 2 units and uses the sigmoid activation function, suitable for binary classification tasks.

6.3 PERFORMANCE METRICS

PARAMETERS	ANN		CNN	
EPOCHS	10		110	
ACCURACY	EPOCH 1	99.95%	EPOCH 1	99.68%
	EPOCH 10	100%	EPOCH 10	100%
AVERAGE TIME TAKEN	45 s		29 s	

Table 6.3.1 DL Performance Metrics

CHAPTER 7

IMPLEMENTATION AND EXPERIMENTAL SETUP

7.1 SDN Environment Setup

This Section involves the steps and procedure to install all the packages and software required to implement the project. The Platform setup is done on Ubuntu 20.04.1 LTS operating system. The project demonstration and simulation was using the following tools:

- Openflow Protocol For SDN
- Ryu Controller
- Mininet
- Hping3
- Iperf

Before moving forward with packages installation make sure you have the updated ubuntu OS and linux libraries with pyhton 3.8 installed.

Openflow protocol For SDN or OpenVswitch has to be installed as it is the standard protocol for software defined network.

```
shwetha@shwetha-VirtualBox:~/ryu/ryu$ cd
shwetha@shwetha-VirtualBox:~$ ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.17.3
DB Schema 8.3.0
shwetha@shwetha-VirtualBox:~$
```

Figure 7.1.1 OpenVSwitch Version

Ryu Controller has to be installed to see the process of detection and mitigation stages. To install ryu controller you need to install PIP of python to install python packages, as ryu is a python based controller it has to be installed using

```
shwetha@shwetha-VirtualBox:~$ sudo apt install python3-pip
[sudo] password for shwetha:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.2).
The following packages were automatically installed and are no longer required:
  libflashrom1 libftdi1-2 libllvm13
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 150 not upgraded.
shwetha@shwetha-VirtualBox:~$
```

Figure 7.1.2 Pip install

```
shwetha@shwetha-VirtualBox:~$ sudo pip3 install ryu
Requirement already satisfied: ryu in /usr/local/lib/python3.10/dist-packages (4
.34)
Requirement already satisfied: eventlet!=0.18.3,!=0.20.1,!=0.21.0,!=0.23.0,>=0.1
8.2 in /usr/local/lib/python3.10/dist-packages (from ryu) (0.33.3)
Requirement already satisfied: netaddr in /usr/local/lib/python3.10/dist-package
s (from ryu) (0.8.0)
Requirement already satisfied: oslo.config>=2.5.0 in /usr/local/lib/python3.10/d
ist-packages (from ryu) (9.1.0)
Requirement already satisfied: tinyrpc in /usr/local/lib/python3.10/dist-package
s (from ryu) (1.1.6)
Requirement already satisfied: msgpack>=0.3.0 in /usr/local/lib/python3.10/dist-
packages (from ryu) (1.0.4)
Requirement already satisfied: webob>=1.2 in /usr/local/lib/python3.10/dist-pack
ages (from ryu) (1.8.7)
```

Figure 7.1.3 Ryu install

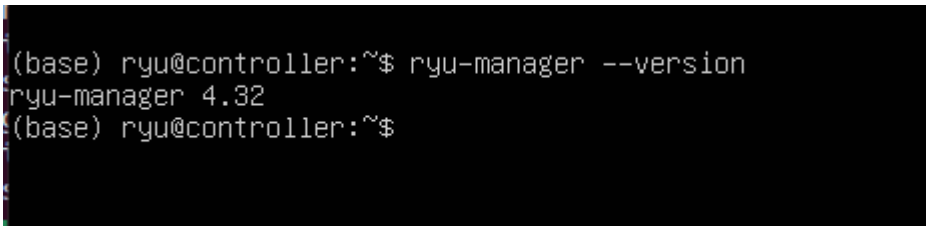


Figure 7.1.4 Ryu version Check

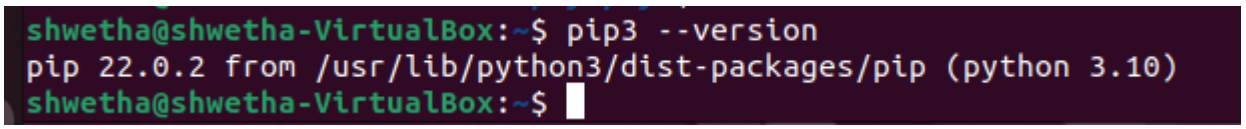


Figure 7.1.5 Pip version check

Mininet is a network simulator and creates virtual network topology for software defined networks.

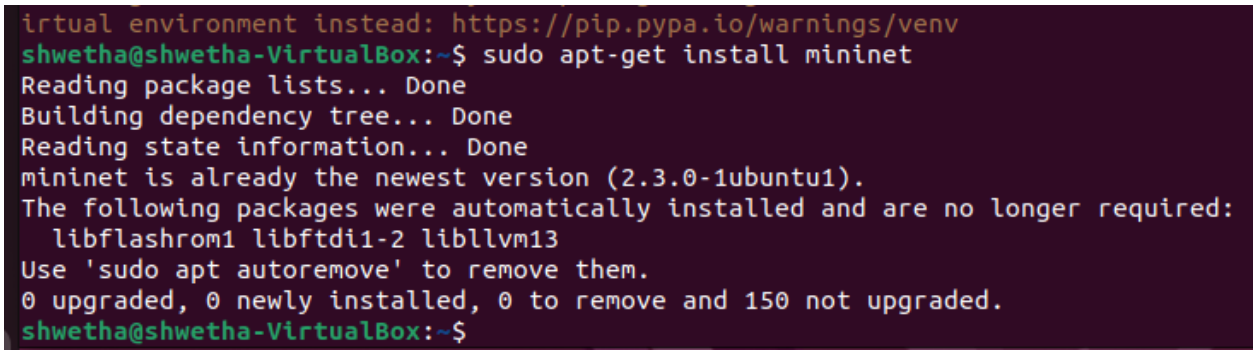


Figure 7.1.6 Mininet install

Ryu packages has to be installed using pip as the ryu controller needs Python packages to run in this environment

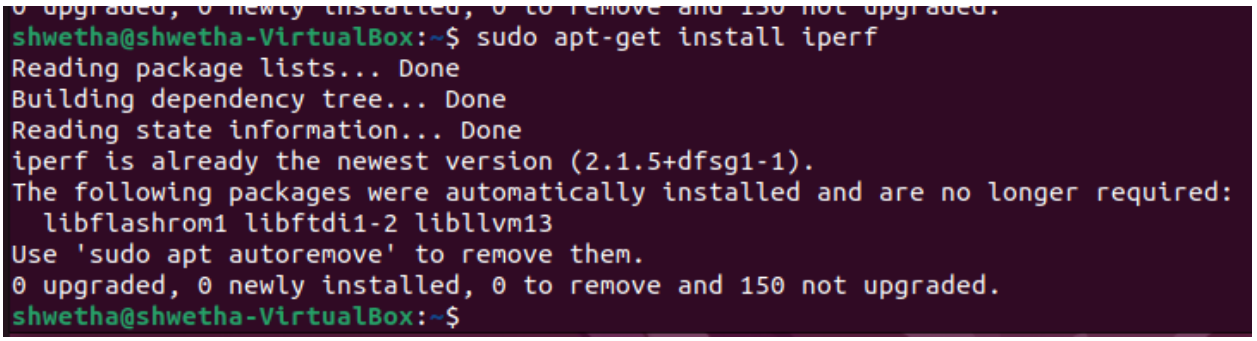


Figure 7.1.7 Iperf install

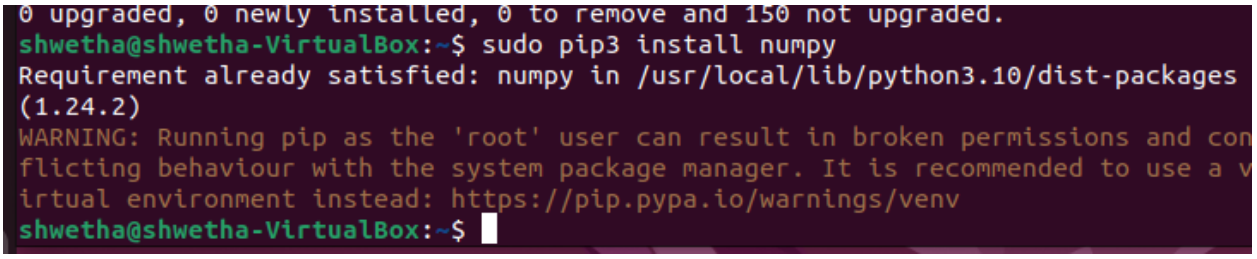


Figure 7.1.8 Numpy Intall

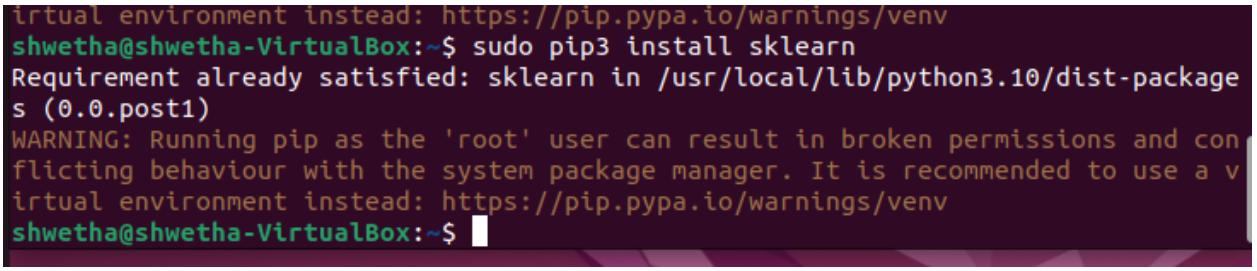


Fig 7.1.9 Sklearn install

These are all the tools and packages required for the project to run and simulation to work. In the next sections we will see how to run attacks and obtain results.

7.2 Integration of Machine Learning Framework

Integrating ML models into the Ryu controller for DDoS detection in SDN environments:

- Data Collection: Ryu controller collects network traffic data from SDN switches or network devices.
- Pre-processing: Collected data is transformed into a suitable format for ML analysis.
- Training Phase: ML model is trained using labeled datasets of normal and attack traffic samples.
- Integration with Ryu Controller: Trained ML model is integrated into the Ryu controller framework.
- Real-Time DDoS Detection: Ryu controller monitors network traffic and feeds it to the ML model for classification.
- Response and Mitigation: Upon detecting a DDoS attack, the Ryu controller triggers appropriate mitigation actions.
- Adaptation and Model Updating: ML model is periodically retrained to adapt to evolving attack techniques.

Flow training and prediction:

The flow_training() method is called during the initialization of the SimpleMonitor13 class. It loads the training dataset from a CSV file ("dataset_bp.csv") and preprocesses it. The features and labels are extracted from the dataset, and a K-Nearest Neighbors (KNN) classifier is trained using the features and labels. The accuracy of the model is evaluated using the test dataset. The flow_predict() method is invoked from the monitoring loop to predict whether the current traffic is legitimate or part of a DDoS attack. It loads the prediction dataset from the "predictdataset_bp.csv" file, preprocesses it, and predicts the labels using the trained model. Based on the predicted labels, it determines if a DDoS

attack is in progress and prints relevant information, such as the victim host. Finally, it clears the prediction dataset file for the next iteration. This ryu application integrates machine learning into the Ryu controller to monitor and detect DDoS attacks by training a classifier using flow statistics and predicting the legitimacy of traffic in real-time. It showcases the implementation of a basic DDoS detection mechanism in an SDN environment.

7.3 DETECTION AND MITIGATION OF DDOS ATTACK

Steps followed in the detection and mitigation process:

- STEP 1: Initiating Ryu Controller

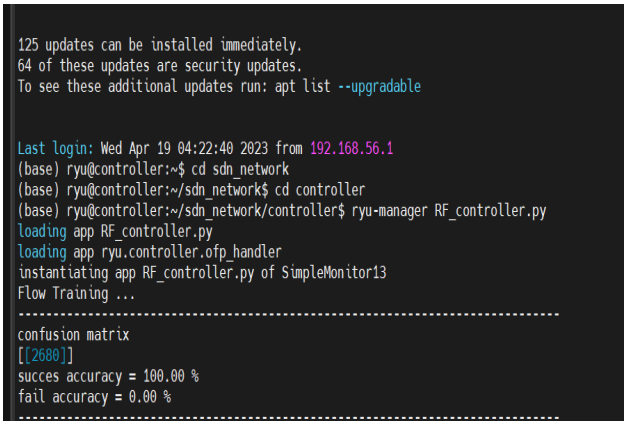


Figure 7.3.1 controller random forest

Figure 7.3.2 controller KNN

- STEP 2: Creating the topology in mininet

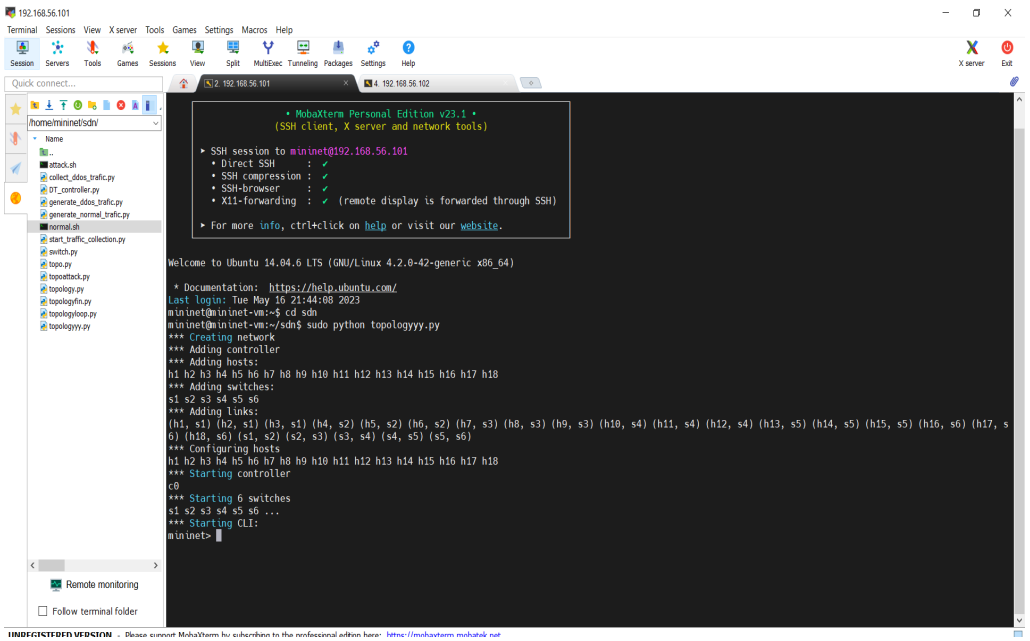


Figure 7.3.3 Custom topology

- STEP 3: Once controller initiated and topology built we need to Open the host we wish and feed the traffic.

  Commands used to generate traffic are:

  - ❖ Ping (host ip addr)
  - ❖ Hping3 (dedicated host ip addr from where we need to generate traffic) –icmp –rand-source –flood (Generate Attack Traffic)

- STEP 4: Open the host terminal and feed traffic in the below image host h1 h3 h5 and h17 are opened .

  - ❖ In h17 general command ping 10.0.0.15 given which meanes packets transmission and connectivity is initiated between h17 and h15 □Ryu controller predicts there is no traffic flow and it results **"TRAFFIC IS LEGITIMATE".**

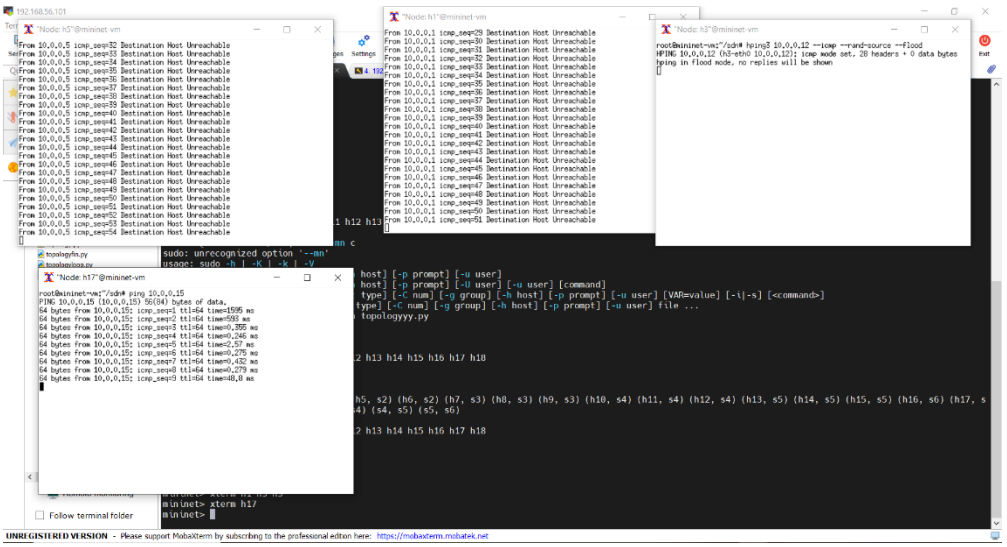


Figure 7.3.4 Attack Generation

- ❖ In h3 traffic is flooded from h12 Ryu controller predicts there is a attack which is DDOS and it results **“DDOS ATTACK IN PROGRESS” -“VICTIM”- “MITIGATION IN PROGRESS”.**
- ❖ Once attack is detected and victim found the port blocking process starts which is mitigation process, controller won’t allow any packets transferred between the ports where traffic is detected.

- STEP 5: Finally all the port which predicts attack are closed and new host h7 opened and initiated packet transfer results shows the **“TRAFFIC IS LEGITIMATE”**

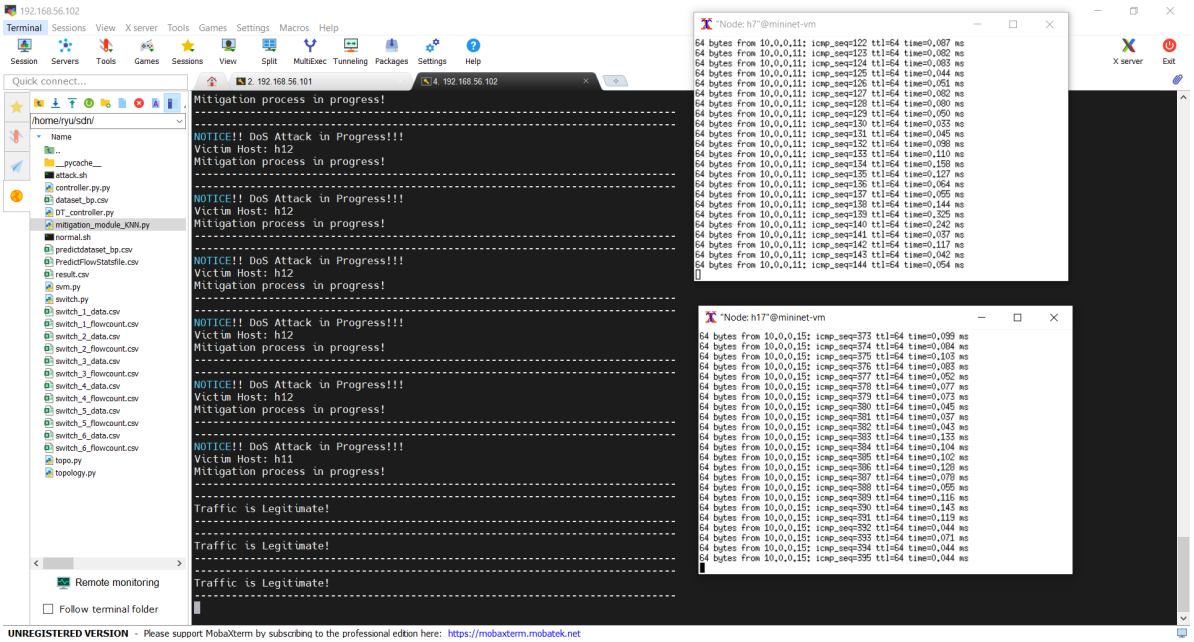


Figure 7.3.5 Attack Detection & Mitigation

CHAPTER 8

DISCUSSION AND CONCLUSION

8.1 Result

- Network traffic is controlled in a centralized manner by the controller which remotely directs the traffic between the hosts.
- Accuracy of the Machine learning and deep learning algorithms have been compared with the base paper and the results have been successfully replicated.
- The attack is detected in the SDN network using appropriate ML techniques and then the port where the host is attacked is blocked and the destination host is made unreachable as part of the mitigation.

8.2 Conclusion

In conclusion, the application of machine learning (ML) techniques for detecting Distributed Denial of Service (DDoS) attacks in Software-Defined Networking (SDN) environments holds significant promise. ML-based DDoS detection in SDN offers improved accuracy, real-time responsiveness, and adaptability, addressing the evolving nature of DDoS threats.

By leveraging ML algorithms, SDN controllers can analyze network traffic data, identify patterns, and distinguish between normal and malicious traffic. ML-based DDoS detection approaches have demonstrated the ability to effectively detect and mitigate DDoS attacks with reduced false positives and enhanced accuracy. The use of advanced ML algorithms, such as deep learning architectures, transfer learning, and ensemble techniques, has the potential to further enhance detection performance.

Future enhancements in ML-based DDoS detection in SDN may focus on several key areas. These include refining feature selection and extraction algorithms to capture more relevant characteristics of DDoS attacks, exploring deep learning architectures for more comprehensive pattern recognition, and developing real-time detection mechanisms to minimize the impact of attacks. Additionally, the integration of transfer learning techniques, ensemble methods, and adversarial attack detection can bolster the resilience of DDoS detection systems in SDN.

Furthermore, the development of autonomous and self-learning systems that dynamically adapt to evolving attack strategies is a crucial direction for future research. Such systems can leverage reinforcement learning techniques and continuously update their detection strategies based on real-time feedback.

Overall, ML-based DDoS detection in SDN holds great potential in strengthening the security and resilience of networks. By leveraging the power of machine learning algorithms, SDN can proactively detect and mitigate DDoS attacks, ensuring the availability and performance of network services. With ongoing advancements in ML techniques and the evolving landscape of DDoS threats, the future of DDoS detection in SDN looks promising, enabling more robust and effective defense mechanisms for network operators and administrators.

REFERENCES

1. J. Al-Jarrah, et al. "SDN-Based DDoS Attack Detection Using Machine Learning Techniques." (2020) IEEE Access, Vol. 8, 2020.

2. J. Han, et al. "Distributed Denial of Service (DDoS) Detection in SDN using Deep Learning." (2020) IEEE Access, Vol. 8, 2020.

3. Mahmoud Said El Sayed , Nhien-An Le-Khac , Senior Member, IEEE,
Marianne A. Azer , and Anca D. Jurcut, et al. "A Flow-Based Anomaly Detection Approach With Feature Selection Method Against DDoS Attacks in SDNs" (2020), IEEE Transactions On Cognitive Communications And Networking, Vol. 8, No. 4, December 2022

4. L. Liu, et al. "A Deep Learning Approach for DDoS Attack Detection in Software Defined Networks." (2020) IEEE Access, Vol. 8, 2020.

5. M. Saad, et al. "DDoS Attack Detection in SDN Environment Using Deep Learning." 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2020.

6. X. Yuan, et al. "Distributed Denial of Service Attack Detection Using a Hybrid Feature Selection and Deep Learning Approach in Software-Defined Networks." (2020) Future Internet, Vol. 12, Issue 10, 2020