DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN

Tewelde Gebremedhin Gebremeskel

A Thesis Submitted to the Department of Computer Science and Engineering,
School of Electrical Engineering and Computing

Office of Graduate Studies
Adama Science and Technology University

Aug 2022
Adama, Ethiopia

DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN

Tewelde Gebremedhin Gebremeskel

Adviser: Dr.Ketema Adere

A Thesis Submitted to the Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Office of Graduate Studies
Adama Science and Technology University

Aug 2022
Adama, Ethiopia

DECLARATION

I declare that this Master thesis entitled "**DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN"** is my work and has not been submitted to any university for a similar purpose. The references used in this proposal are duly recognized by proper citations.

Tewelde Gebremedhin Gebremeskel _____ _____

Name of student Signature Date

RECOMMENDATION

I, the advisor of this thesis, hereby certify that I have closely advised the student while developing this thesis and read the draft thesis entitled "**DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN**" prepared under my guidance by **Tewelde Gebremedhin**. Therefore, I recommend the submission of the thesis to the department for further review and evaluation.

Ketema Adere (Ph.D.)	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

APPROVAL PAGE

I hereby certify that the recommendation and suggestions given by the thesis review committee are appropriately incorporated into the final thesis entitled "**DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN**" by **Tewelde Gebremedhin**.

Ketema Adere (Ph.D.)	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

APPROVAL PAGE OF BOARD OF REVIEWERS

We, the undersigned, members of the Board of Reviewers of the thesis open defense by **Tewelde Gebremedhin** have read and evaluated his thesis entitled "**DDoS Attack Detection and Classification Using Entropy and Deep Learning Model for Multi-controller SDN**" and assessed the understanding of the thesis is accepted and we recommend the implementation of the thesis.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Reviewer 1	Signature	Date
_____	_____	_____
Reviewer 2	Signature	Date

Final approval and acceptance of the thesis is contingent upon the submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

First and foremost, I would like to thank God for giving me the strength and ability to undertake this research study. Without his countless blessing, everything would not have been possible.

I am grateful for my friends, whose constant love and support keep me motivated and confident. My achievements and success are because they believed in me.

I am grateful to my brother and my family, who have provided me with moral and emotional support. I am also grateful to my other family members who have supported me along the way.

I would like to express my deepest gratitude to my advisor, Ketema Adere (Ph.D.), whose sincerity and encouragement I will never forget. The door to Ketema Adere (Ph.D.) office was always open whenever I go for guidance and comments. His constant constructive ideas enabled me to gain good research experience.

I would also like to thank all of my friends who stayed this long process with me, always offering support and love. Also, I extend many thanks to ASTU staff members especially the ASTU ICT Directorate center staff for their kind support during my research study.

Thanks for all your encouragement!

TABLE CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

SDN	software-defined network
DDOS	Distributed denial of service
IP	internet protocol
ML	machine learning
SVM	support vector machine
FPR	false positive rate
TPR	true positive rate
TNR	True negative rate
FNR	False negative rate
RNN	recurrent neural network
MLP	Multilayer Perceptron
GRU	gated recurrent unit
LSTM	Long short-term memory
AE	autoencoder
BI-LSTM	Bidirectional Long short-term memory
DL	Deep learning
CNN	convolutional neural network
GPU	Graphical processing unit
API	application programming interface
VLAN	virtual local area network
TCP	Transmission Control Protocol
ICMP	Internet Control Message Protocol
MAC	media access control
RAM	random-access memory
CPU	central processing unit
TCAM	ternary content-addressable memory
SPF	single-point failure
RSMQ	*Redis Simple Message Queue*
KNN	K-Nearest Neighbour
HTTP	Hypertext Transfer Protocol
PCA	Principal component analysis
LDAP	Lightweight Directory Access Protocol

NIB	Network Information Base
VM	Virtual machine
IDLE	Integrated Development and Learning Environment
Dst IP	destination Internet protocol
Src IP	Source Internet Protocol
Pcap	packet capture
ROC	receiver operating characteristic curve
AUC	Area under the ROC Curve
ACC	accuracy
NAC	Network access control
ONOS	open network operating system

ABSTRACT

The data and control planes are separated by software defined network (SDN), which also addresses the challenge of deploying new services and offers various advantages for networking. But that didn't mean SDN solves every problem in networks like "security, and reliability remain as the issues yet to be addressed. In a such case SDN is more vulnerable to Distributed-Denial of Service (DDoS) attacks. The attacker can launch DDoS attacks to the controller in order to make the controller out of service. Most papers are focused on a single controller topology which leads to a single controller failure, use binary classification which leads to lack of detailed classification of the attack type. In this thesis, to address the DDoS security issue in multi-controller SDN, a system based on information entropy and deep learning has been proposed. The advantages of information entropy and deep learning are combined in this technique. This Two-level detection is used for network traffic to ensure high accuracy and reduced the workload of deep detection server controller at the same time. Firstly, suspicious traffic can be reviewed through information entropy detection by the controller. Then, fine-grained packet-based detection is performed by the Long Short-Term Memory (LSTM) to classify the attack in to different attack type category. Finally, the controller sends the updated traffic information to neighbor controllers. To avoid a single point of controller failure in our thesis, we use a multi-controller which is a logically centralized and physically distributed controller. To address lack of detail attack classification we use categorical classification. This type of classification allowed to make specific Attack classification in which what type of attack is coming to the controller. And also, we used the chi-square (x^2) test feature selection algorithm to reveal the most relevant features that scored the highest in the provided data set, only the most pertinent features were picked. The experiment finding proves that the proposed LSTM model achieved an accuracy of up to 99.42% using CIC-DDoS2019 dataset which has the potential to detect and classify the DDoS attack traffic effectively in the multi-controller SDN environment. As we have noticed from the related models that our proposed model has enhanced by 0.42% accuracy which is higher than Recurrent Neural Network -Autoencoder (RNN-AE) model on the CICDDoS2019 data set. Also, it improves by 0.44% accuracy which is higher than the Convolutional Neural Network (CNN) model on the ICICDDoS2017 data set.

***Keywords**: Software Defined Network, Distributed-Denial of Service, Deep learning, multi-controller, DDoS attack detection and classification.*

CHAPTER ONE

1. INTRODUCTION

1.1. Background of the Study

SDN is a new design that consists of : data, control, and application plane, with independent of one another(T. Wang et al., 2021). The data plane consists of switches and routers that forward network traffic; the control plane is comprised of POX, Floodlight, and Open Daylight controllers; The SDN controller will be unable to respond to normal traffic coming from the rest of the network during a DDoS attack, and SDN will lose its centralized control (Pomona, 2019). As a result, the key benefits of SDN, centralized network control, may be vulnerable by DDoS attacks. DDoS detection and classification solutions in SDN environments are very important to investigate to adapt SDN to data center more reliably and support the growth of future networks (R. Wang et al., 2015). Many researchers have brought a new SDN pattern to answer the demands of data centers, and it has gained significant attention. A single central entity with high processing capacity and good data management techniques would be required to respond flow requests from forwarding plane(Zakaria et al., 2017)(Scott-Hayward et al., 2013). The centralized controller may be overwhelmed with flow requests arriving at a fast rate, and risk network fluctuations degrading response times.

According to the authors,(Ahmad & Hussain, 2021)(Bannour et al., 2018) as the data plane devices and hosts grows, SDN controller will become a bottleneck. Scalability and reliability issues are handled by physically distributed SDN controllers. To reduce the consequences of attacks such as spoofing, DoS, tampering, and privilege escalation, a controller should be structured as a container for processes, with a permission structure for applications, and monitoring resources. Hackers, cyber terrorists have turned to DDoS attacks as a weapon of choice(Krishnan & Najeem, 2019). A victim Even of such attacks may become paralyzed fast, resulting in a significant loss of revenue. Despite the availability of a wide range of mitigation options, DDoS attacks continue to grow in occurrence, capacity, and complication. A new network paradigm is required to face today's difficult security challenges. The goal for DDoS attack is to use multi-source attacks to disrupt the target's services(Pandikumar et al., 2017).

A typical example of this type of attack is a flooding-based attack, in which a target is overrun by a lot of network traffic. DDoS attacks can overload servers, networks, and network equipment (routers, switches, and so on) with illegitimate traffic, in addition to paralyzing networks and services. Detecting, mitigating, and stopping DDoS attacks has elevated to a top priority due to an increase in the quantity and variety of DDoS attacks (Lawal & Nuray, 2018)(Sebbar et al., 2018). Due to recent advancements in SDN and its swift and widespread adoption in the network world, several academics have been actively interested in developing SDN-based network security solutions. Since its deployment in wide-scale networks, SDN-based solutions have gotten greater attention(Xu et al., 2014)(Mohsin & Hamad, 2022).

Most researchers are working on detecting and classifying DDOS attacks with a single controller using different mechanisms, and also, they focus on either the accuracy or efficiency but not both. but what can be done in data centers with multiple controllers? And how can we improve the accuracy and efficiency both at the same time? This will be the question next. There are multiple controllers in data centers that need to be protected from DDoS attacks. Each of these controllers has a different network traffic tolerance level. Spoofing the source is one approach to hiding the perpetrator's identity when a DDoS attack occurs(Fawcett et al., 2018)(Zubaydi et al., 2017). Fake source addresses are referred to as spoofing. There will consequently be a flood of incoming packets with different IP addresses. The switch table is not likely to contain these addresses. The controller consequently processes them. If the controller gets a high number of packets, malicious packets will use all of the controller's processing resources. The controller's resources are depleted as a result of high-rate attacks, and legitimate production traffic is restricted. The main benefit of employing a controller is that just the first packet's rules may be added and withdrawn from the table. This implies that each packet will only be processed once.

1.2. The Motivation of the Study

Nowadays SDN security and reliability is the most challenging part. Since a huge number of a packet are sent to the controller to make out of service it will be more difficult to detect the attack. When given a large amount of high-quality data to work with, deep learning performs best, and as the amount of data accessible increases, so does its effectiveness. MLP, RNN, LSTM, and GRU which are more popular in attack detection and classification. Based on recent studies, deep learning frameworks have been used to identify and categorize attacks. In(Elsayed et al., 2020) they propose RNN-autoencoder models to detect and classify the attack based on a binary method

using the DDoS2019 dataset. In this approach, the problem is that when the attack classification is done by using RNN-autoencoder it classifies into the attack and normal(generally) but did not describe which attack type what type arrived to the controller (specifically identify the attack type). The other proposed using entropy-based and deep learning(L. Wang & Liu, 2020). That is using CNN without feature selection method and it is in single controller architecture. So, the problem here is a single point of failure, lack of detailed attack type classification and leads to classification error and decrease training time this makes the model less accurate. And the other problem is most researchers are working on the efficiency or accuracy of DDoS attack detection and classifying but not both. Based on these problems we inspire to do this study to identify and categorize DDoS attacks in multiple controllers using an entropy-based and deep learning model.

1.3. Statement of the Problem

SDN paradigm gives an ability to plan efficient countermeasures against DDoS attacks, it also comes with the danger of vulnerabilities and malfunctions as a result of controller attacks. When a user sends a DDoS attack to a centralized controller, the controller may go down or be taken out of service since the forwarding plane cannot function without the controller. For the various SDN layers, it is crucial to offer a strong security result. Without incorporating a strong security outcome, SDN's appeal becomes a problem. A number of existing DDoS attack detection and classification methods do not combine information entropy and deep learning. Therefore, the first problem with these models expands the gap between the accuracy or efficiency of the model in the SDN multi-controller.

The second problem is researchers use all types of features in the given dataset for model training. This leads the model to increase redundant data, training time. And this makes the model reduce accuracy and increase classification error. The use of a centralized controller topology, which results in a single controller malfunction, is the third issue. Lastly, lack of a detailed classification of the attack type. From the baseline paper the researchers used binary classification (normal and attack) (Elsayed et al., 2020)(Mhamdi et al., 2020). This classification leads to a more generalized classification than specific. In such a case It is difficult to know what specific type of attack is coming to the controller. A number of the articles they've published so far have been centered on either the model's efficiency or the accuracy, rather than both at once. To address these challenges with better accuracy and efficiency this study combines information entropy and deep learning detection and classification for multi-controller SDN.

1.4. Research Questions

This study intended to answer the following three significant research question

RQ1: How to design a reliable and effective architecture for DDoS detecting and classification model in SDN multi-controller?

RQ2: How to improve the accuracy and performance of the designed architecture of DDoS detection and classification mechanism in multi-controller SDN?

RQ3: How efficient and accurate is our model for detecting and classification the DDoS attacks using a benchmark dataset by comparing the suggested deep learning models with baseline studies and other deep learning approaches?

1.5. Objectives of the Study

1.5.1. General Objective

Develop a model which has better efficiency and accuracy of DDoS attack detection and classification using entropy and deep learning model for multi-controller SDN.

1.5.2. Specific Objective

- To prepare the datasets for DDoS attack detection and classification.
- To Design a reliable and effective architecture for better efficiency and accuracy of DDoS attack detecting and classification model in SDN multi-controller.
- To implement the proposed models of DDoS detection and classification methods in multi-controller SDN
- To measure how accurate our model is in the detection and classification of the attack.

1.6. Contribution of the Study

- To make the SDN-based data center more reliable, secure, and available: in this case since we used multi-controller-based SDN if one controller failed the backup controller continues working.
- To decrease workload, less classification error on the controller: when we deploy entropy-based and deep learning using feature selection allows the controller to decrease workload and less classification error. This allows the controller easily classify the attack and work properly.

- To detect and classify the attack before the controller is down: when the attack comes to the controller the detection and classification are deployed using deep learning, in this case, the attack is classified to which type of attack is coming to which port. This makes it the network administrators easy to manage the network in and outflow.
- To increase the accuracy of detection and classification of the attack on the controller

1.7. Scope and Limitation of the Study

1.7.1. Scope of the Study

The thesis concentrates on detecting and classifying DDoS attacks on SDN multi-controller by using entropy-based and deep learning.

The following topics are covered by the thesis study's scope within the constraints of time and resources:

- This thesis is focused only to six DDOS attack behaviors (DrDoS_UDP, SYN, UDPLag, DrDoS-LDAP, Benign, and WebDDoS attacks) and not considered other attack behaviors on the dataset.
- RNN, GRU, LSTM, and MLP are experimented with, and selected the best accuracy model.
- Train-test the model using the CICDDoS2019 dataset.
- Logically centralized and physically distributed controllers' architecture is used to increase reliability and security.
- Adding chi-square(x^2) feature selection method for better accuracy and less attack classification error.
- Attack type classification is based on the categorical classification method.
- Focused on attack detection, and classification but not on mitigation.
- We didn't focus on the controllers' load balancing (we used existing POX controller load balancing for experimental reasons),

1.7.2. Limitations of the study

- The issue of high-performance devices capable of deep learning, which requires high-performance technology GPU significantly impacted the success of this thesis.

1.8. Organizations of the Thesis

This is how the thesis's content was created: The first chapter outlines the general study topic, the driving force behind it, the motivation, the issue, the goals, the significance, and the steps taken to get there. The literature review and associated works are discussed in Chapter 2. Chapter three discusses the methodology, and in chapter four, we try to explain how the research topics will be addressed. Along with the suggested architecture and methodology for the suggested solution, a detailed explanation of how it functions is also provided. The simulation setup and simulation results were discussed in Chapter 5. Sixth chapter will include the result and assessment of the suggested system. The study's conclusion, research contribution, and future work are all discussed in the last chapter.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORK

2.1. Introduction

This chapter provided a thorough explanation of the traditional network concept, an overview of SDN, the various types of attacks in SDN, DDOS attacks in SDN, methods for detecting and categorizing DDOS attacks, different kinds of statistical and classification algorithms, and related work on the security of the controllers. The most pertinent research articles were used to study, analyze, and compare the chosen research area. In order to develop, implement, and deploy SDN, it is needed to grasp several concepts, principles, and techniques. The research project discussed in this thesis is provided in the end.

2.2. Overview of Traditional Network

The basis of modern telecommunications technology is the computer network, as networks have become an essential component of our homes, workplaces, and educational institutions (Latah & Toker, 2020). Due to the vast volume of information and data kept in the cloud, network technology has advanced to today's high technologies, and network computing technology has entered our daily lives(Nugraha & Murthy, 2020). Even though traditional networks are widely used and adopted, they are complex and difficult to manage.

For instance, network administrators must use low-level and vendor-specific commands to individually configure each network device. The data and control plane are combined inside networking devices. This limits flexibility and holds up the innovation and growth of networking infrastructure. Furthermore, network environments must tolerate fault dynamics and react to load variations(Bouet, n.d.). Traditional networks are complicated structures consists of specialized devices such as firewalls, load balancers, and middleboxes.

For efficient packet delivery, the routing algorithm governs the next-hop router usually, it performs two important tasks. Routing algorithms are applied to incoming traffic for the first time, and the network topology is always updated(Sagar et al., 2019). Routing tables that are kept locally are used to handle control planes. Second, the router uses what is referred to as the data plane service to decide how to forward incoming traffic based on routing tables. Routing with traditional mechanisms puts a heavy burden on the router's processing power.

2.3. SDN Architecture

SDN seeks to overcome the difficulties by reconsidering network administration. Decoupling of data planes and control planes, centralized management of the entire network, and programmability are the three basic aspects of SDN. By exposing a centralized view and administration over an entire network infrastructure, additional functionality can be introduced as needed(Ravi et al., 2020). To enable the central control logic, SDN discovers and updates a global abstract network view. There are three main layers of SDN architecture(Yue et al., 2020).

2.3.1. Application Layer

Which contains network programs that can help the control layer with network configuration by introducing additional network features such as security, manageability, and forwarding schemes(Jayakumar, 2020)(Redekar & Chatterjee, 2017). The controllers can provide it with a distracted and global view for the network, that it can use to deliver suitable direction to control plane. A northbound application interface is the link between the application and control layers.

2.3.2. Control Layer

The control plane serves as a link among an SDN controller and an SDN data plane. SDN has the advantage of implementing the control plane in an open, vendor-neutral, and interoperable manner(Benzekki et al., 2016). All forwarding actions, ability advertising, statistics reporting, and event alerting are managed programmatically.

2.3.3. Infrastructure Layer

Switches, connections, and hosts are examples of forwarding devices found at the infrastructure layer. The switching device executes packet forwarding in the data plane in accordance with the controller's forwarding rules(Hu et al., 2018). Packet forwarding is the primary purpose of an SDN switching device's data plane. In particular, before forwarding a packet to the following hop, the switching device first checks for a matching forwarding rule when it receives the packet.

2.4. Overview of DDoS Attack

A Company's website, router, or other computing resources, such as RAM and CPU, are the targets of a DDoS attack by compromised computer systems. It prevents users of the targeted system from accessing the system(Mittal et al., 2022). Messages in bulk malicious packets cause the target's machine to slow down, preventing legitimate users from accessing the services they want. Many organizations, including nationalized and private banks, enterprises, and individuals linked to the internet, can be infected by a DDoS attack.

2.4.1. Causes of DDoS Attack

The likelihood of conducting a DDoS assault has significantly increased during the past few years. Although this attack had a criminal element, it's never been understood why. (Dallou & Al-duwairi, 2020). For instance, on the night before Christmas in 2014, the hacking organization Lizard Squad attacked Sony and Microsoft's gaming servers. One suspected member stated, "It was for fun!" during the investigation into the motive for the attack. It is frequently difficult for figuring out what motivates these attacks and why they occur. The fundamental reason is that some unknown external sources are in control of the calculating resources used in such attacks(Pei et al., 2019). The following is the overview of the attack's causes(Ahuja & Singal, 2019).

- When two people or groups are at odds, DDoS is a potent tool for impeding the infrastructure and applications of the other side.
- An angry person may become an attacker and carry out unwanted activities in response
- Through cyber warfare, a terrorist cell attempts to attack some of the complex zones to destroy the financial structure. This type of attack is motivated by politics or geopolitics.

2.5. Types of DDoS Attacks

DDoS attacks are meant to impede the network's resources, and they are started by numerous connected devices. Furthermore, (Alshra’A et al., 2021) the attackers challenge to overwhelm the target with fake packets for the malicious packets to stay served. We'll go through different forms of DDoS attacks in this section.

2.5.1. Volumetric Attack

The attacker wants to overwhelm the victim's network bandwidth by injecting a ton of traffic. By using a variety of amplification tactics, this kind of attack is simple to neutralize. The majority of

the time, reflection-based volumetric assaults simply send valid queries to a DNS or network time protocol (NTP) server while utilizing a fake source IP address to target a service. Bits/s are a unit of measurement for the severity of this attack. Examples of this attack include UDP floods and ICMP floods.

2.5.2. Protocol Attack

This attack takes use of the protocol stack's network and transport layer vulnerabilities. This attack is mostly focused on consuming server resources such as a load balancer and firewall. The size of the attack is restrained in packets per second. ICMP Smurf attack is an example mentioned.

2.5.3. Application Layer Attack

This attack takes use of the protocol stack's application layer's flaws. By leveraging server resources like processing power, memory, and bandwidth between two separate memories, it aims to unbalance the services offered to actual consumers. Examples of this technique include the DNS attack and the HTTP attack. We also list a few typical DDoS attack types in addition to the ones mentioned above. Flooding with UDP is an unreliable and session-less protocol. It's used to send datagrams, which are brief communications. The UDP flooding attack uses UDP packets to flood random ports on the network. During the attack, the attacker utilizes a faked source IP address. As a result, the victim receives a high volume of datagrams, causing a buffer overflow. flooding ICMP attack In the network, ICMP is frequently used to check whether a machine is responding or not. An ICMP echo request packet is sent to the system in question for this. The system will return an ICMP reply packet if it successfully receives the message(Alshra'A et al., 2021).

2.6. SDN as the Victim of DDoS Attack

SDN faces additional vulnerabilities as the control and data planes are decoupled. Since the emergence of SDN, DDoS attacks have been a major research area. It has been observed that SDN systems and DDoS attacks have a contradictory relationship. SDN's advantages allow it make simple to detect and respond. SDN is vulnerable to DDoS attacks because of its decoupled nature(Mehr & Ramamurthy, n.d.).

2.6.1. DDoS Attack on Forwarding Layer

The fundamental working concept of SDN is that each incoming flow's is sent to the SDN controller, and need to wait till the flow rule is returned before continuing. It enables the attacker

to inject unredacted fresh flows into the victim switch. Due to the expensive cost of standard switches, the TCAM is limited in size. Commercial switches, on the other hand, can contain thousands of rules at once.

2.6.2. DDoS Attack on Control Layer

The SPF is a difficult job for a centralized SDN. Despite the existence of a multi-controller SDN topology as an option, SPF remains a thoughtful threat to DDoS attacks. When the network is overburdened, packets are generated each time. If the runoff request arrives in the controller, the controller's resources will be exhausted after a certain amount of time. The entire flow will be forwarded to the controller if the buffer of the target switch becomes overflowing the packet header(Kansal & Dave, 2017). The bandwidth required to control the link will be exhausted by the new flows. The regular flow request, in this case, causes a congestion problem. Manipulation of the packet in messages may result in an unexpected side-channel attack.

2.6.3. DDoS Attack on the Application Layer

The attacker could target a specific SDN application. Targeting the application layer, on the other hand, can have an impact on northbound APIs. Because the OF isn't realized, an attack on one application can lead to the compromise of others. By generating excessive working threads, A harmful program can use up all of the system's resources, affecting the performance of other applications.

2.7. DDoS Attack Detection Techniques in SDN

DDoS detection strategies divided into the following categories: entropy method, ML, DL. here we explain the detection methods that we use in our research that is the entropy method and Deep learning model.

2.7.1. Entropy-based Solutions

Entropy is a statistical measure of an entity's randomness over a given period. High entropy denotes a high level of randomness in an attribute, and less calculation overhead. The method evaluates the entropy value of three things throughout the detection process: port address, IP address, and packet count. An attack might take place, for instance, if all packets are sent to the same host (or Dst IP). The detecting module operates in the network's edge switch to lessen the workload of the controller. The proposed entropy mechanism compared the entropy flow values

of Src and Dst IP addresses detected by the SDN controller to predefined threshold values that change adaptively based on network dynamics(Dallou & Al-duwairi, 2020).

This researcher surveys the different methods that we can use in DDOS detection and mitigation methods like entropy, Machine learning, and Statistical analysis(Jo & Park, 2018)(Yu et al., 2021). This researcher describes when and how this method can be used. ML techniques are applied on DDoS attacks on SDN controllers or switches by constructing the model automatically from the training set of data(J. Wang et al., 2017).

This research aims to propose a technique for detecting DDoS attacks at an early stage using a multi-controller approach(Mousavi & St-Hilaire, 2015)(Ahalawat et al., 2019). Not only detecting attacks, but it can also detect the attacking paths and recruit a mitigation process to protect network devices as soon as an attack is identified. The assault largely focuses on traffic attacks, bombarding the target with a large number of TCP, UDP, and ICMP packets.

proposes a distributed controller architecture in this paper for detecting DoS attacks in SDN networks and recovering attacked controllers without impacting the network's overall performance(Etaiwi et al., 2017) (Tayfour & Marsono, 2021).

2.7.2. Machine Learning-based Solutions

In the old network, ML techniques such as SVM, decision trees, random forests, and others were commonly employed for anomaly detection(Tonkal et al., 2021)(Cui et al., 2020). SDN-based anomaly detection systems can also benefit from these approaches.

To identify suspicious traffic provided by the detecting trigger mechanism, use a combined ML method based on K-Means and KNN(Tan et al., 2020) (Pei et al., 2019). When the network subjected to higher-scale traffic, however, the controller's workload grows, and DDoS detection efficiency decreases.

In this article, ML and entropy method in SDN are used for detecting DDoS attacks(Banitalebi & Mohammadreza, 2020). The researcher applied a Dynamic threshold rather than a static threshold. The researcher comes to a conclusion by noting that at this time, more attacks are discovered in a shorter amount of time. Selecting short durations causes the attack detection process to begin early, abuse CPU and network bandwidth resources, damage controller performance, and, as a result, permit attacks to spread over a longer length of time, having a detrimental impact on attack detection(H. Z. Wang et al., 2016) (Deepa et al., 2018).

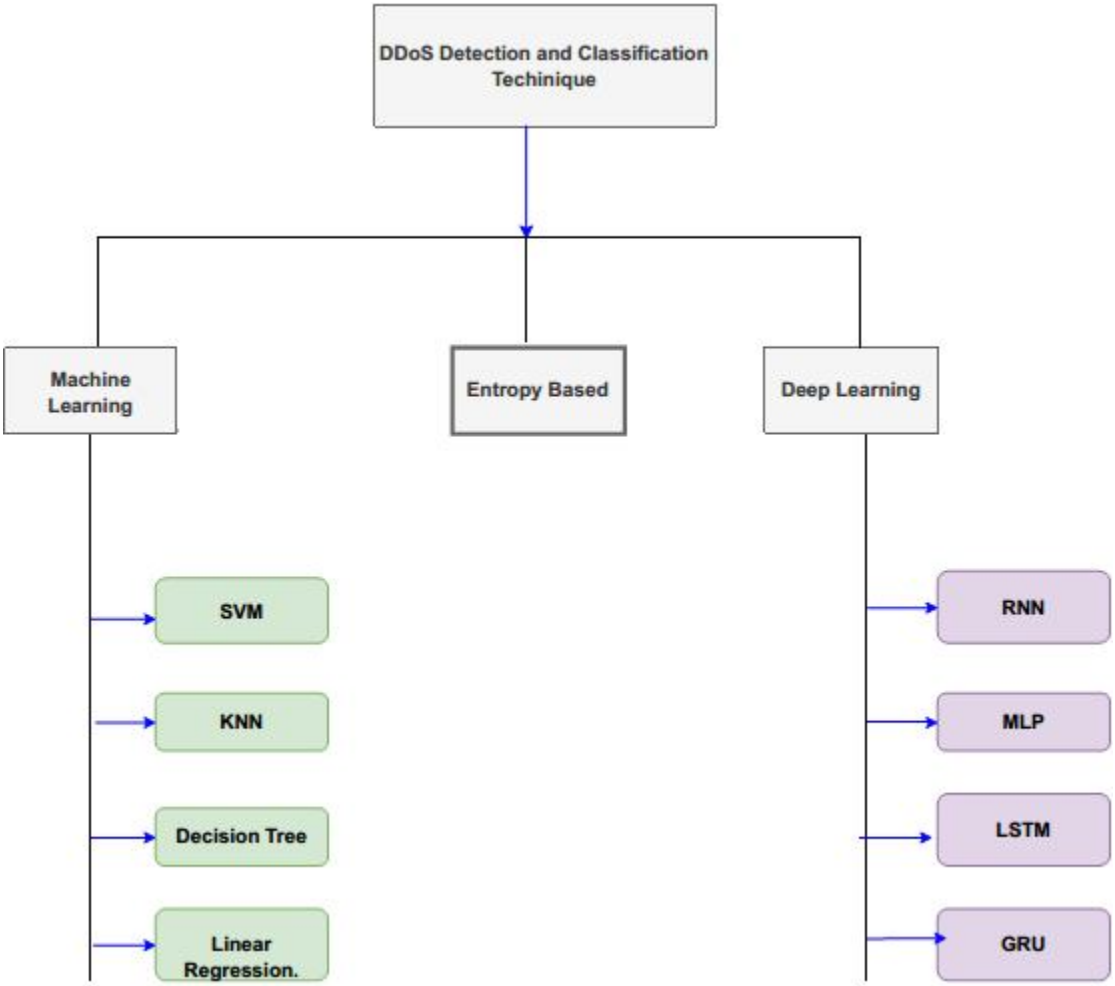

Figure 2.1.Technique used for DDoS attack detection and classification process

2.7.3. Deep Learning Based Solution

In order to solve problems utilizing unstructured data, DL model are mathematical models that resemble the human brain. These models are developed in the form of a neural network made up of neurons(Agrawal et al., 2022). The three main layers of a neural network are the input layer, hidden layer, and output layer. We shall deal with these neural networks, which are categorized as RNN, LSTM, MLP, GRU, etc., based on these kinds of data. DL can extract the raw features from given data set.

2.8. Types of the algorithm used in deep learning

2.8.1. LSTM

Long-term dependencies can be learned and memorized using LSTMs, a form of RNN. The default behavior is to remembrance past information for a long time. Over time, LSTMs keep track of info. Because they remember past inputs, they are valuable in time-series prediction. LSTMs have a chain-like structure with four interconnected layers that communicate distinctly in addition to time-series predictions, LSTMs are commonly used for voice recognition, music creation, and medication research. The LSTM is made to avoid long dependency issues and may retain extensive history data. The accuracy of detection can be increased by looking at the flow data over a longer time frame.

A memory block is made up of one or more memory cells with input and output gates that are adaptive multiplier gates. The data is saved in a memory block. The information flow into and out of the memory cell is controlled by the input and output gates. Based on the input and output gates, it updates the data across successive time steps. The forget gate, which erases the previous value at a particular time step, is present in addition to the input gate and the output gate.

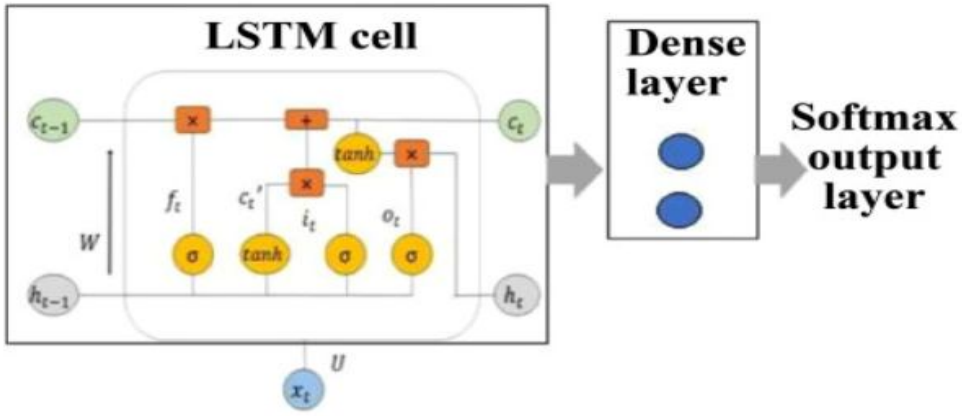

Figure 2.2. LSTM process(Nugraha & Murthy, 2020)

What Are LSTMs and How Do They Work? First, they forget about non-essential aspects of the previous state. Then they update the cell-state values on a case-by-case basis. Finally, the state of several parts of the cell's output. proposes that DDoS attacks be detected using a hybrid CNN-

LSTM model. This method's performance is evaluated using customized datasets(Nugraha & Murthy, 2020). The achieved results are pretty outstanding, with all performance indicators above 99%. The current study recommends using a hybrid DL model, such as a CNN-BiLSTM to estimate DDoS attacks(Alghazzawi et al., 2021). Only the maximum important features were chosen that scored the highest. According to the experiment results, the proposed CNN-BI-LSTM achieved an accuracy of up to 94.52 using the CICDDoS2019 data set. This study investigates deep learning-based models such as LSTM and CNN. The suggested model which has an accuracy of 89.63% outperforms like SVM (86.85 %) and Naive Bayes (82.61percent). Although KNN, a linear-based model, beat suggested model (reaching an accuracy of 99.4%) (Gadze et al., 2021). Deep learning and information entropy have been used to provide an approach for detecting DDoS attacks(L. Wang & Liu, 2020). To locate the attack, this method used a two-level detection mechanism. To determine which switch the malicious traffic entered the network through, the controller carried out the first step based on entropy. Based on a convolution neural network, fine-grained packet-based deep detection discriminated DDoS assault traffic.

2.8.2. RNN

RNN has connections that create directed cycles, the outputs from the LSTM can be used as inputs in the current phase. The LSTM's internal memory enables it to retain previous inputs, and its output becomes an input to the current phase. The researcher proposes DDoSNet, an IDS for DDoS attacks in SDN (Elsayed et al., 2020). The method uses DL to combine an RNN and an autoencoder. the model is evaluated using **CICDDoS2019** dataset. During the training day, many attack types—including UDP, SNMP, NetBIOS, LDAP, TFTP, NTP, SYN, WebDDoS, MSSQL, UDP-Lag, DNS, and SSDP DDoS attacks were covered.

2.8.3. MLP

MPLs are a good place to begin if you're interested in knowing something about DL technology. Activation functions are provided across numerous layers of perceptron’s in a feedforward neural network called an MLP. MLPs contain fully connected input and output layers. They have the same number of input and output layers, but they can have several hidden layers. The proposed Autoencoder (AE) component model offers an real feature extraction that automatically discover the most weighted feature sets without the need for human interaction(Wei et al., 2021).

And to avoid the performance overhead and bias the MLP section of the planned model uses the reduced feature created by AE as inputs, classifying the attacks into various DDoS attack categories. achieved through intensive and comprehensive testing on many parts of performance on the CICDDoS2019 dataset, demonstrating both a very high and robust accuracy rate, as well as an F1-score that approaches percent, outperforming other similar methods.

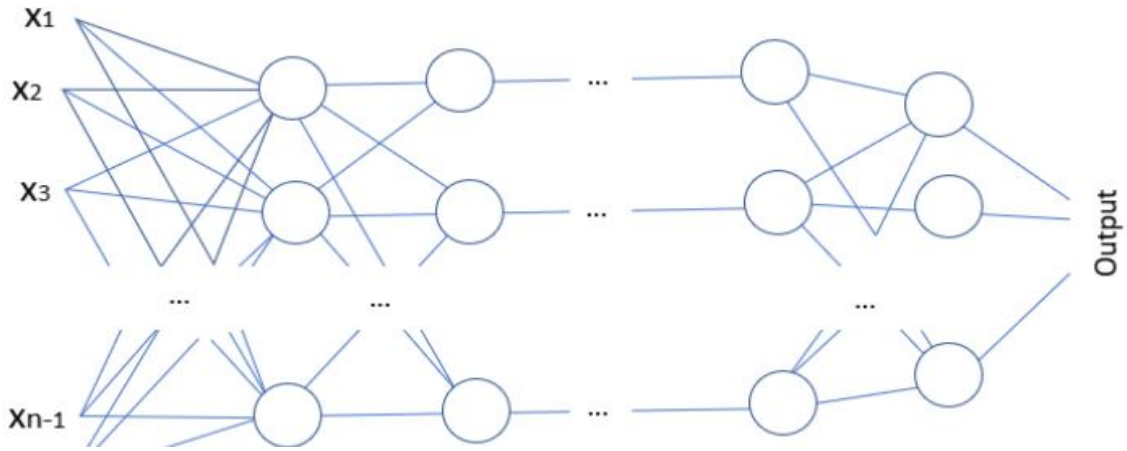

Figure 2. 1. MLP Design(Wei et al., 2021)

2.9. The Architecture of the SDN Control plane

A single centralized Controller, a physically distributed but logically centralized SDN controllers, or a hybrid SDN controller are all possible configurations for SDN(Zubaydi et al., 2017)(Paliwal et al., 2018). A single controller system might be unable to keep up with the network's expansion, though. With an increase in requests and a struggle to maintain the same level of performance, it is likely to become overloaded (a controller bottleneck). Large-scale real-world network installations have various needs, and a centralized SDN controller cannot accommodate them. Data centers and service provider networks are two examples of such large-scale networks with various scalability and reliability requirements(Bannour et al., 2018)(Mousavi & St-Hilaire, 2015). t logically centralized and physically distributed SDN controllers have the ability of multi-controller that have better reliability, scalability, and availability(Tadros et al., 2019).

2.10. Related Work

Several researchers studied that DDOS attack is the major attack that many attackers use to attack the network, especially in SDN controllers. This attack has affected the entire network if the controller is attacked by a DDOS attack. These researchers use different methods of DDOS attack identification and classification methods. Existing DDoS attack identification methods are the change of methods used in traditional networks(Sagar et al., 2019). Statistical mechanism can evaluate a huge amount of traffic quickly and at a low rate of calculation, nevertheless its correctness is dependent on threshold selection. ML and DL are the algorithms used to classify a large amount of data in the SDN controllers. Many researchers have proposed several approaches or mechanisms to address DDOS identification and categorization on single controller SDN. Most researchers focus on either efficiency or accuracy but not both at the same time. Most researchers have focused the detection and classification on a single controller, but not focused on multi-controller. The most important mechanism DDOS identification and categorization on SDN controller have been used by the following researchers. Here is the related work which is connected to the proposed papers to body of knowledge as found in the literature. Here the researchers proposed a solution to identify an attacks using the entropy based of the Dst IP address a lightweight technique that is proposed to detect these attacks early on and is intended for a single controller(Mousavi & St-hilaire, 2018), (Dallou & Al-duwairi, 2020). An attack on the entire network may go undetected. In this paper, the advantage of using entropy-based is evaluating a huge volume of data quickly with less-cost calculation which leads to better efficiency. But the problem here is its correctness depends on threshold selection and it will have some one side which leads to low accuracy in the attack detection and classification method.

To classify the traffic A strategy for detecting DDoS attacks using information entropy and DL is proposed. This technique employed a two-level detection strategy(L. Wang & Liu, 2020). The controller has run the initial entropy-based section to determine which switch the suspicious traffic entered the network from. CNN-based so well packet-based deep detection identified DDoS assault stream. The advantage here is the researcher uses the two-level detection method for better efficiency and accuracy at the same time. But here the problem is it depends on single controller architecture this leads to controller failure that faces a problem of reliability, availability, and scalability. And the researcher uses the binary classification method. This method has a lack of detailed attack description in which the attack is coming to the controller. The AE-MLP hybrid

technique, which contains two DL models, is proposed for active DDoS attack identification and categorization, according(Wei et al., 2021). The proposed AE component model offers an active feature removal that automatically finds the most appropriate feature sets without the need for human interaction. In order to reduce the performance overhead and bias associated with processing large feature sets with noise, the MLP section of the proposed model employs the condensed feature sets generated by the AE as inputs. The system then divides the attacks into several DDoS attack types based on thorough research on the CICDoS2019 dataset, achieving an F1-score that is close to 98% while outperforming other competing methods.

The Coordinator Controller was designed and implemented to allow multiple SDN organizational domains to collaborate by combining the controllers' northbound APIs (J. Wang et al., 2017). The researcher creates a multi-domain environment with three suppliers to validate the suggested architecture. The results show how highly the network state values services for end-to-end provisioning and consistency. The problem here is the entropy did not detect the attack accurately.

The paper aims to present a technique for detecting DDoS attacks at an early stage using a multi-controller implementation(Pandikumar et al., 2017). The proposed method detects the attack in SDN using the entropy method. the approach is not only detected attacks, but it can also identify the attacking paths to protect network devices as soon as an attack is identified.In a distributive SDN multi-controller platform, this paper proposes a collaborative approach for DDOS attack detection(Kavitha et al., 2019).

This research aims to mitigate an attack before the controller is impacted by huge number of packets that must be successfully transmitted to multi-controllers, as well as to classify packets and assign targets to controllers in a distributed multi-controller SDN architecture in order to detect malicious packets in the early phases of processing. The researcher proposes DDoSNet, RNN combined with an autoencoder in this method, which is based on DL. The new dataset CICDoS2019, which comprises a variety of DDoS attacks and fills in the gaps in previous datasets, is used to test the model(Elsayed et al., 2020).

In comparison to existing benchmarking methods, we obtain a significant improvement in attack classification. And then the model gives us a lot of confidence in protecting these networks, and it can identify all attacks and normal classification with 99% accuracy. To estimate DDoS attacks, the study indicated using a hybrid DL model, such as a CNN-BiLSTM. By selecting the features with the highest score, only the most important features were chosen.

2.11. Summary of the related work on this study

The following tables show earlier works on DDoS Detection and classification in SDN.

Table 2.1. Research-related work with their limitation and gaps

Authors	*Title*	*method used*	*controller architecture*	*Data set used*	*Gaps*
(Kavitha et al., 2019)	The Detection and Mitigation of DDOS Attacks in SDN using Distributed Controllers	Entropy-based	Pox multi-controller (3 controllers)	Scapy and Wireshark are used to generate traffic	▪ Its correctness depends on the threshold selection value. ▪ This allows to decrease attack detection accuracy
(L. Wang & Liu, 2020)	A DDoS Attack Detection Method Using Deep Learning in SDN and Information Entropy	information entropy and CNN	POX single controller	CICIDS2017	• Single point of controller failure. • Not reliable • Leads to controller performance overhead
(Gadze et al., 2021)	Deep Learning for DDOS Attack Detection and Mitigation on SDN Controllers	RNN_LSTM	Floodlight in a single controller	malicious traffic was generated using hping3	▪ Not effective in a large-scale network ▪ A single point of failure causes a full network failure
(Alghazzawi et al., 2021)	A Hybrid Deep Learning Model for DDoS Attack Detection with Improved Feature Selection	CNN-BI-LSTM	Single controller	CIC-DDoS2019	▪ limited attack type classification (binary classification) ▪ not categorical classification
(Elsayed et al., 2020)	DDoSNet: Detecting Network Attacks	RNN with autoencoder	Single controller	CICDDoS2019	▪ limited attack type classification (binary classification) ▪ High-controller performance overhead
(Pandikumar et al., 2017)	Early Detection of DDoS Attacks in a Multi-Controller Based SDN	based on entropy method	Multi-controller POX	Packet generation is done by Scapy	▪ Its correctness depends on the threshold selection ▪ This leads to decrease attack detection and classification accuracy

2.12. Baseline works

Both CNN (L. Wang & Liu, 2020)and RNN- autoencoder(Elsayed et al., 2020) take as the baseline for the experiment. This Baseline model compares various DDoS Detection and classification models which use different deep learning approaches based on different benchmark datasets. Here the problem is a single point of a controller failure, a lack of detailed attack type classification, and increasing error rate classification. In this proposed method the detection and classification are done in a single controller, also the classification is binary, CICDDoS2017 dataset is used for the model training. The results of the experiments show that this strategy is 98.98% accurate.

2.13. Summery

As we have seen in the literature review a number of researchers are using entropy-based method, deep learning, and machine learning algorithm as the detection and classification of different attack types. Now a day These detection and classification methods very familiar and a number of researchers applied them. In the related works we found a gap of single point of failure, controller overloading and using of binary classification this leads to lack of detailed information of the attack type. The other researcher's gap is using of either entropy based or deep learning not both of them. The gap in entropy is its accuracy depends on the threshold value this makes the model less accuracy. This entropy method is better in efficiency but the attack accuracy detection rate is less b/s in the entropy method we classify or detect the attack by the threshold or window size. in this case, the accuracy rate is very less so we need a methodology that helps us to give a high accuracy detection rate by using a deep learning algorithm. And also, a single controller failure will happen because the architecture is single controller. Based on this this thesis focused on DDoS attack detection and classification in multi-controller scenarios. In this case, we try to identify and classify the type of attacks that we use in this research those are UDP, LDAP, ICMP, and TCP sync. These attack types are extracted from the CIDDoS2019 Dataset.

CHAPTER THREE

3. RESEARCH METHODOLOGY

3.1. Chapter Overview

For the investigation of explanatory research questions, we try to explain the methods or models that we use in the proposed solution in the SDN multi-controller. And then we use simulation software to see in experiments and finally, we evaluate the model. This chapter explains the process of DDoS detection and classification, including the methodology employed. This comprises the methods used for data collection, processing, using equipment or materials, and choosing a method. The diagram below describes the procedures or methods from the packet and dataset to evaluation measures used in the proposed work to address the study question.

3.2. Procedure of the Study

The following steps are taken to complete the study's objective:

- **Literature Review**: Different literary works that are connected to this thesis work have been reviewed, and various notions have been embraced.
- **Formulating the problem**: based on the literature review we identify the problems to be solved in this thesis.
- **Design the architecture and develop model**: modeling the system and creating a suitable architecture for DDoS attack detection and classification in multi-controller SDN
- **Simulation:** Google Colab and Mininet will be used to test and experiment with the suggested model.
- **Result and analysis:** showing the result for the proposed model.
- **Evaluation of the proposed model**: The simulation's results will be validated, and performance will be evaluated.

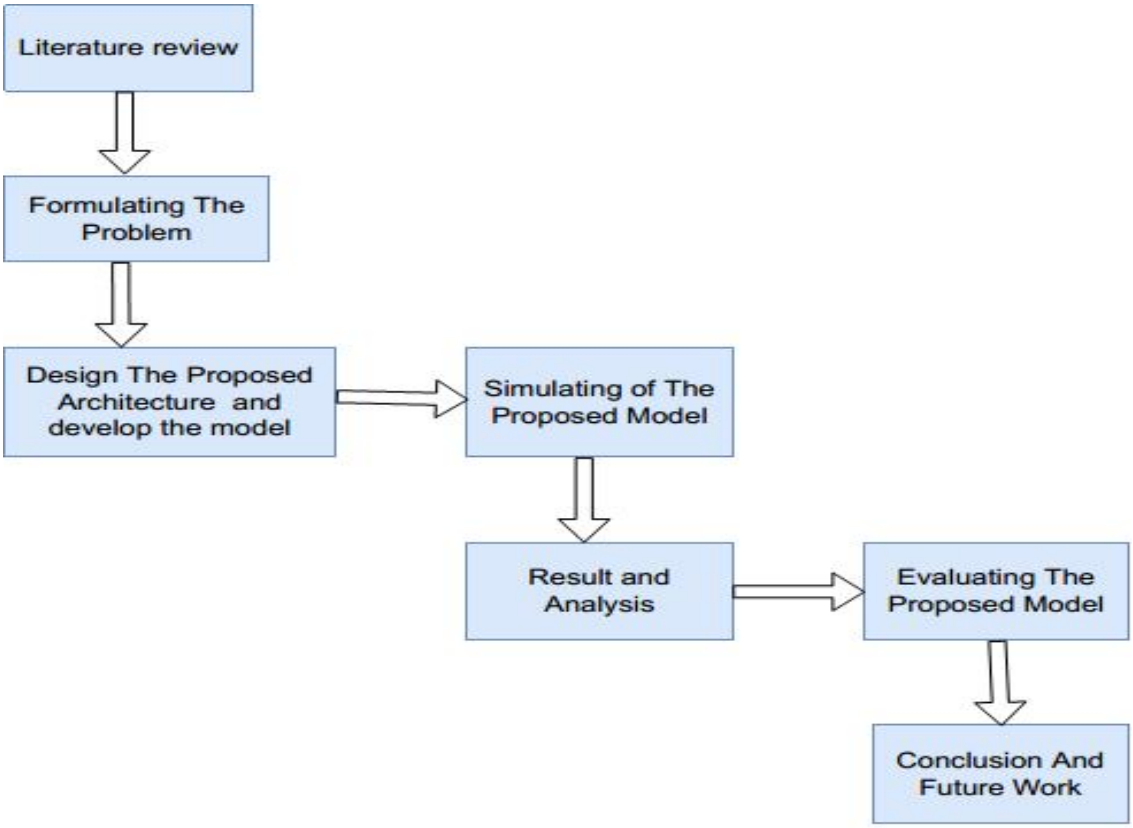

Figure 3. 1.The methodology of DDoS attack detection and classification in multi-controller SDN

3.3. Dataset Description

Data sets are used to prepare adequate training and testing datasets for the desired solution.

CIC-DDoS2019: DDoS attacks are a type of network security threat that aims to overload target networks with malicious traffic. Despite the development of numerous machine learning techniques for DDoS attack detection, one of the main issues is the development of a little processing overhead. (Banitalebi & Mohammadreza, 2020).

Reflection-based DDoS: These are attacks where the identity of the perpetrator is concealed by making use of trustworthy third-party components. Attackers send the packets to reflector servers with the source IP configured to the victim's IP in order to flood the target with reply packets. These attacks can be conducted utilizing application layer protocols that make use of transport layer protocols, such as TCP, UDP, or a combination of the two.

Exploitation-based Attacks: These attacks conceal the attacker's identity by utilizing a legitimate third-party component. The same attacks can also be carried out using transport layer protocols like TCP and UDP in place of application layer protocols. flood SYN.

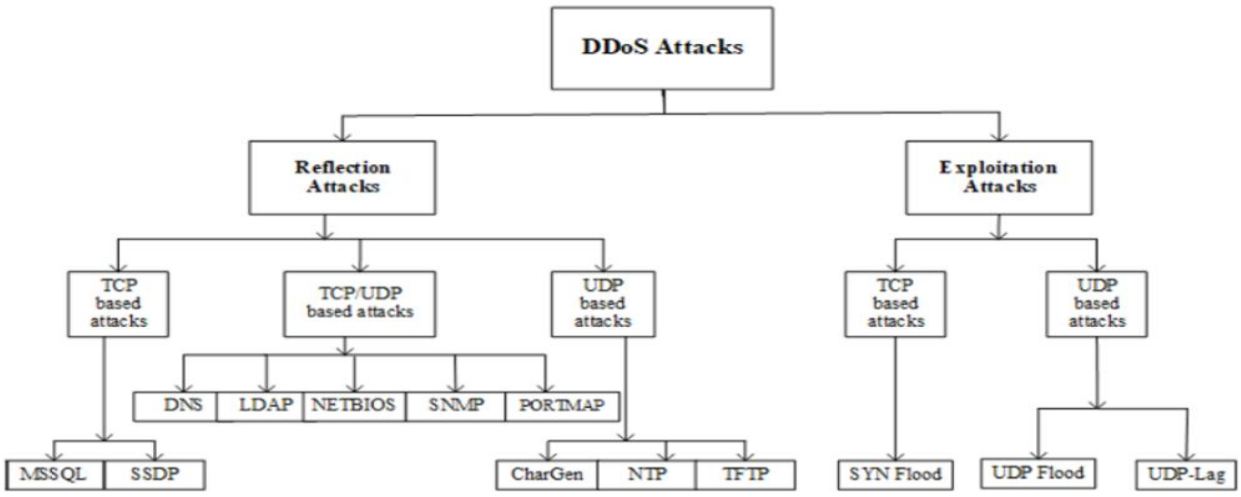

Figure 3. 2.DDoS attack types(Elsayed et al., 2020)

3.4. Dataset Preprocessing Techniques

We directly prepare the data so that it is suitable for the training model. More than 80 features from the CICDDoS2019 dataset are available in.csv format thanks to CICFlowMeter. Before the module training, we preprocess the data using a few procedures.

1. **Removing Socket Features:** All socket features, including source and destination IP, source and destination port, similar HTTP, and unnamed, are removed. We must train the model using packet characteristics since these features differ between networks. Furthermore, the same IP address may belong to both the intrusive party and regular users(Elsayed et al., 2020). Due to the model's potential bias toward the socket information, training the DL model with socket information may result in an overfitting issue.
2. **Cleaning the Data:** We must eliminate all of the missing (nan) and infinity values from the original data because they are very prevalent.
3. **Normalize the Input Data:** Different numerical values are present in the feature data. Directly training the model with the original data can lead to classification errors, and the model then takes a long time to train.
4. **Encoding the Labeled Data**: In order to categorize the input traffic into different sorts of attacks, we trained our categorical classification model. The types of attacks we use for detection and categorization include DrDoS LDAP, DrDoS UDP, UDP-lag, Syn, BENIGN, and WebDDoS. The label feature must be labeled into categorical classification with 0 and 1.

3.5. Feature Selection

We focus on choosing the best features to predict DDoS assaults rather than selecting all features from the source data. The best features from the raw data can be chosen using a variety of methods, such PCA, decision trees, the Random Forest Regressor, and chi-squared (x^2) tests. An x^2 test examines whether or not the correlation between the predictor and target variables affects the frequency of specific classes and traits.

3.6. Parameter Setting

To increase the likelihood of getting an appropriate classification while back testing the proposed model, the hyperparameters must be set up correctly and changed during creating the deep learning model. Higher accuracy and a reduced chance of overfitting the data are two benefits of using the suggested ideal hyperparameters. The ideal value for a hyperparameter is determined by evaluating the deep learning model by back testing it against the test data.

3.7. Model Training

3.7.1. Define and Compile the Model

After feature selection, the suggested framework Keras' input format is supported by DL model, which is created utilizing the Regularization, recurrent, activation, and dense layer shapes of the input layer. Consequently, the model needs to be compiled after the model and network are created. Once it has been created, the model can define the metrics, optimizers, and loss function. At this stage, the loss function is defined and bound by the model compilation. For the next stage of model training, the evaluation metrics are necessary.

3.7.2. Train the Model

Training and testing will align with the build model once the network is complete. The model will evaluate how well the dataset fitted in this step. How probable the proposed model to fit the data, allowing it to produce very accurate categorization of future attacks? The model will set and fit callbacks to enable dynamic learning rate adjustments and early stopping by continuously tracking changes in loss and iteration over a predefined number of epochs throughout the given dataset. The returned history item records a history of the metric and loss values throughout training.

3.8. The Hardware and Software used in the Simulation

3.8.1. Software Tools Used

1. **Mininet:** Mininet 2.2.0, a network emulator, is used in this thesis to simulate the network and establish a network of virtual hosts, switches, and controllers. For highly flexible custom forwarding and SDN, Mininet switches offer OpenFlow, while Mininet hosts run Linux network software.

2. **POX Controller:** POX is a Python-based open-source OpenFlow SDN controller. It's designed to let you develop and prototype new network applications quickly. Simple OpenFlow devices can be changed into hubs, switches, load balancers, and firewalls via the POX controller. We employed a flat architecture controller with three controllers in this paper.
3. **Python2:** Python 2.7 is a scripting language that is reactive and designed to be simple to learn. We use Python 2.7 to configure the Pox controller and to develop the model using entropy-based and deep learning techniques in this research.
4. **Google Colab:** Colab is a free Jupyter notebook environment that runs in the cloud. The main benefit is that there is no setup needed. You may make, upload, and share notebooks. It is possible to import and export Google Drive notebooks. GitHub notebooks can be imported and published. Import external datasets, for example, A free cloud service with a free GPU is available.
5. **Scapy**: Scapy used for packet generation, scanning, sniffing, attacking, and forging, it is an extremely potent tool.
6. ***Keras:*** A high-level Python library for deep learning that may be used on top of TensorFlow is called keras, which is small and simple to learn.It is more user-friendly and easier to build a model. It also can be run on CPU and GPU.
7. ***Mendeley Desktop:*** is a simple tool for managing references that serves as a social network for academics to find relevant publications.
8. ***Microsoft Office Packages:*** It uses to write the study document in a proper format.

3.8.2. The Hardware Tools

We use hardware tools to implement this study appropriately. Hardware tools needed for this study list in the table below.

No.	*Device Name*	*Used in the study*
1	Hard Disk	For storage thesis files.
2	GPU	The process of training large datasets on the CPU takes time, so we used GPU for this study to speed up the training and computation.
3	RAM	It is used with the GPU to accelerate the training process.

Table 3.1. Hardware tools used

3.9. Performance Evaluation Methods

The proposed method's performance is evaluated using the following

A. **Accuracy (ACC):** Is measured by the total number of data samples that were successfully classified as well as the proportion of queries for which the model was able to predict the correct label.

B. Precision: Evaluates the accuracy of the right predictions by dividing the total positives by the total positives.

C. Recall: Recall can be considered a metric of the completeness of classifiers. Many False Negatives are indicated by a low recall. The recall serves as a measure of how well our model detects True Positives.

D. F1-measure: Precision and recall in relation to a particular positive class are combined to create the F1 score.

3.10. Summary

In this thesis, we propose a DDoS attack detection method based on information entropy and DL in multi-controller SDN. To get the result that we want we use the latest data set that is CICDDoS2019. This dataset contains 11 attack types and Bengin but due to the limitation of a good performance of the computer and GPU. we only select 6 types of attacks that is DrDoS_LDAP, DrDoS_UDP, UDP-lag, Syn, BENIGN, and WebDDoS are the type of attacks that we use in the detection and classification in the model. to develop a good model the dataset must be pre-processing. We use feature selection to get a model that has good accuracy. A good accuracy means we can detect and classify the attack correctly. to evaluate our model we use accuracy, recall, f1-score, TPR, FPR metrics, and Precision

CHAPTER FOUR

4. PROPOSED SOLUTION FOR DDOS ATTACK DETECTION AND CLASSIFICATION FOR MULTI-CONTROLLER SDN

4.1. Chapter Overview

In this chapter we proposed the solution to the problem. The main role of the study is DDoS detection and classification in multi-controller SDN which is Implemented with Three POX controllers. and the model performance is evaluated through accuracy, recall, f1-measure, and Precision. In the previous paper, (L. Wang & Liu, 2020) the researchers uses a different mechanism to detect and classify the attack with better accuracy and efficiency in a single controller SDN. But here our work is in multi-controller SDN and we select a model which have a better efficiency and accuracy model that detect and classify DDOS attack. In these proposed mechanisms we use Entropy-based and Deep learning. We focus on the attack behaviors of LDAP, UDP, and SYN flood attacks that target the controller.

4.2. Proposed Model Architecture

Entropy-based and Deep learning is the proposed model for effectively and accurately classifying the attacks. For network traffic, two-level detection is used to ensure better accuracy and reduced computational complexity at the same time. The controller performs an initial category based on information entropy in order to get better efficiency, and the deep detection server provides a packet-based deep section in order to get better accuracy and fine granularity. In this baseline paper(L. Wang & Liu, 2020), the researcher uses Binary classification to categorize the traffic into attack and normal. The first gap is the classification is more generalized classification than specific. In a such case It is difficult to know what specific type of attack is coming to the controller. The second gap is the researcher use all type of features in the dataset. This makes the model high redundant data, high misleading data, and high training time. This all makes the model reduce accuracy and increase classification error. The third gap is the researcher use only a single controller topology which makes a single point of failure. In the proposed model, we use the feature selection method that is chi-square(x^2) to select 20 features from the given dataset. The chi-square (x2) test feature selection algorithm reveals the most weighted features and accomplish an effective classification. This will increase classification accuracy and reduce classification error. The features of the classifiers used after the feature selection phase. And also, in this thesis we consider a more detailed description of what type of attack it is. So, in this case, we use categorical classification which is more specific type of attack classification. Finally, we use a multi-controller to avoid SPF.

Based on the proposed diagram below we address the following gaps with the proposed model architecture solution.

- The irrelevant attribute and high training time addressed using feature selection algorithm.
- The Binary classification (attack and normal) which is lack of detailed classification of the attack type. This proposed model addresses by adding Categorical classification.
- Using centralized controller topology which leads to the SPF. In this proposed model is addressed by multiple controller detection.

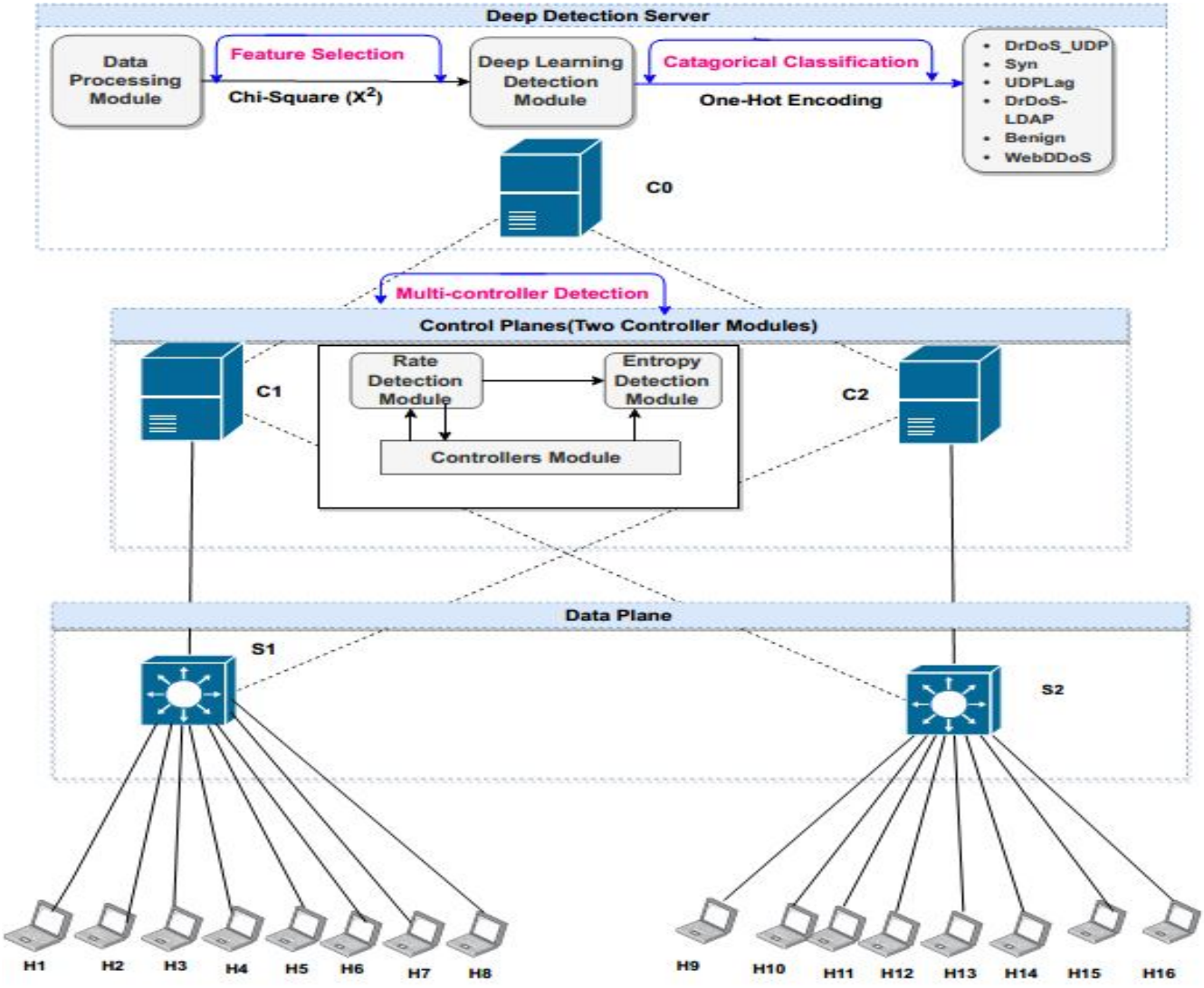

Figure 4. 1. The proposed architecture for DDoS attack detection and classification method

4.2.1. Entropy-based Method (Controller Detection Design)

Entropy is one of the most DDoS attack detection methods and defense techniques used by many researchers. It's used to figure out how random particular attributes in network packet headers are distributed. For those advantages, entropy becomes a common method used for classifying DDoS attacks in network security. Entropy is calculated using probability distributions for several network features including source and destination IP, and port numbers(L. Wang & Liu, 2020). Anomalies are detected using predetermined criteria on changes in the entropy values. The PACKET IN messages rate detection, port entropy detection, and control module are included in the first section of the overall approach. Filtering suspicious traffic is the controller's responsibility in order to increase recall. The controller itself implements the control module. Here in our multi-controller architecture, the control module is implemented by two controllers, and finally, detection and classification occur by one deep detection server.

(1) Rate Detection Module: An attacker can launch a DDoS in an SDN system by sending spoof packets that don't match any switch flow entries. The switch will request the controller for the processing method by sending PACKET IN notifications. As a result, the controller's packet delivery rate increases quickly and dramatically. The likelihood of an attack on the existing network rises when it reaches the normal threshold. The following step is to determine if the abnormal act is caused by network attack.

(2) Entropy Detection Module: Normally, packets are directed to a number of hosts on a network, but during a DDoS attack, a huge number of attack packets are directed to the same IP address. Based on the data size or the number of packets in the flow table entry with the same IP address, the assault can be assessed and discovered. The Open switch updates the flow table entry with all the packet information when a packet from a client arrives at the switch. Next, the controller rule set checks the packet fields (i.e., packet sizes, IP address, port, and window size). The packets are forwarded to the designated destination port as soon as the condition is met. The switch recognizes the rule mismatches and sends the packets to the controller when numerous packets arrive with a repeated IP address. The controller figures out the entropy that the switches provide. Entropy is calculated based on the threshold value that is discovered in each suspect host when compared to the rule set. The controller determines that there has been a DDOS attack. In the meantime, the controller sends the packets based on the high entropy value to the deep detection server module for additional attack detection and classification. Then the attack detection and classification information are sent to the other controllers which are connected in a distributed setup. To prevent the controllers and other forwarding components, attacks must be discovered as soon as possible. The current study took into account the first 50 incoming packets for detection as compared to the prior study, which only focused on the first 50 packets for attack detection.

4.2.2. Deep Detection Server Design (Deep Learning)

In this research, we offer a mechanism for detecting DDoS attacks in multi-controller SDNs based on information entropy and DL. To begin, the controller can review suspicious (untrusted packets) traffic by detecting information entropy. The Deep Learning model then uses detailed packet-based detection to classify the attack into different attack types. This technique brings together the advantages of information entropy and DL. Finally, the controller distributes the change to all neighborhood controllers.

Two-level detection and classification is used for network traffic to ensure better accuracy and minimal computing complexity. To ensure great efficiency, the controller runs a primary section based on information entropy. The packet-based uses deep detection to ensure fine granularity and better accuracy. The detection includes the data processing section and the deep learning detection section. Data processing and deep learning detection are both included in the detection process. In the first stage, the traffic will be changed into the acceptable input shape. The outcomes of the detection and classification processes are output in the second section using the deep learning approach.

(1) **Data Processing Module:** before feeding the input data to the training model we need to prepare the data like normalization, encoding, feature extraction, and feature selection.

(2) **Feature Selection:** Choosing the most crucial features to feed into deep learning algorithms is one of the key aspects of feature engineering(Niu et al., 2018). By removing redundant or unnecessary features and reducing the set of features to those that are most relevant to the deep learning model, feature selection techniques are used to reduce the number of input features. This feature selection makes our model higher accurate in detection and classification, reduces overfitting, and reduces running time and less error rate.

(3) **Deep Learning Detection Module:** here in this module, we applied four deep learning models and compare the best accuracy and low error rate. These algorithms are MLP, RNN, LSTM, and GRU. To avoid overfitting and increase the model's accuracy for generalization, the dropout layer is introduced before the output layer. It allows neurons to be turned on or off according on a parameter's probability.

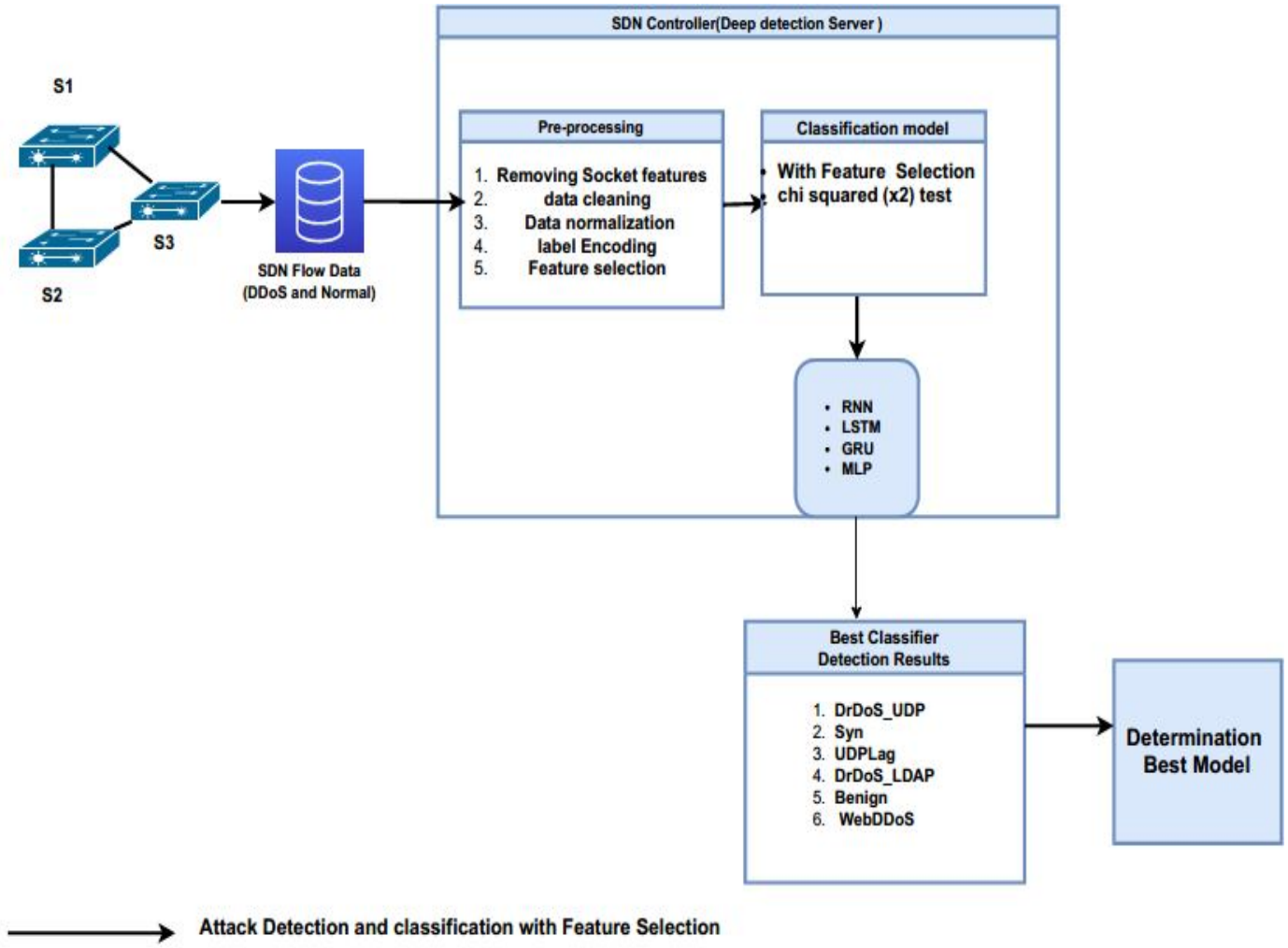

Figure 4.1 The process of DDoS attack detection and classification in the deep detection server with feature selection.

4.2.3. Multi-Controller Architecture

We employ a physically distributed and logically centralized controller architecture in this thesis. By being logically centralized, we can benefit from the idea of a multi-controller design while always assuming that there is only one controller. All of the controllers in a logically centralized design are charged equally with the same tasks. Because of network synchronization, they are always aware of any changes in the network and can instantaneously communicate the same information. The network information is kept in the NIB (Network Information Base), and each controller's state is synchronized by writing to and reading from the NIB's contents. Based on the packet threshold level, controllers immediately identify malicious behavior and determine whether to forward the packets or send them to the deep detection server for attack classification. In order to update the databases of the other controllers in the distributed controller connection domain, this analysis will be communicated to the controller database and updated there as well.

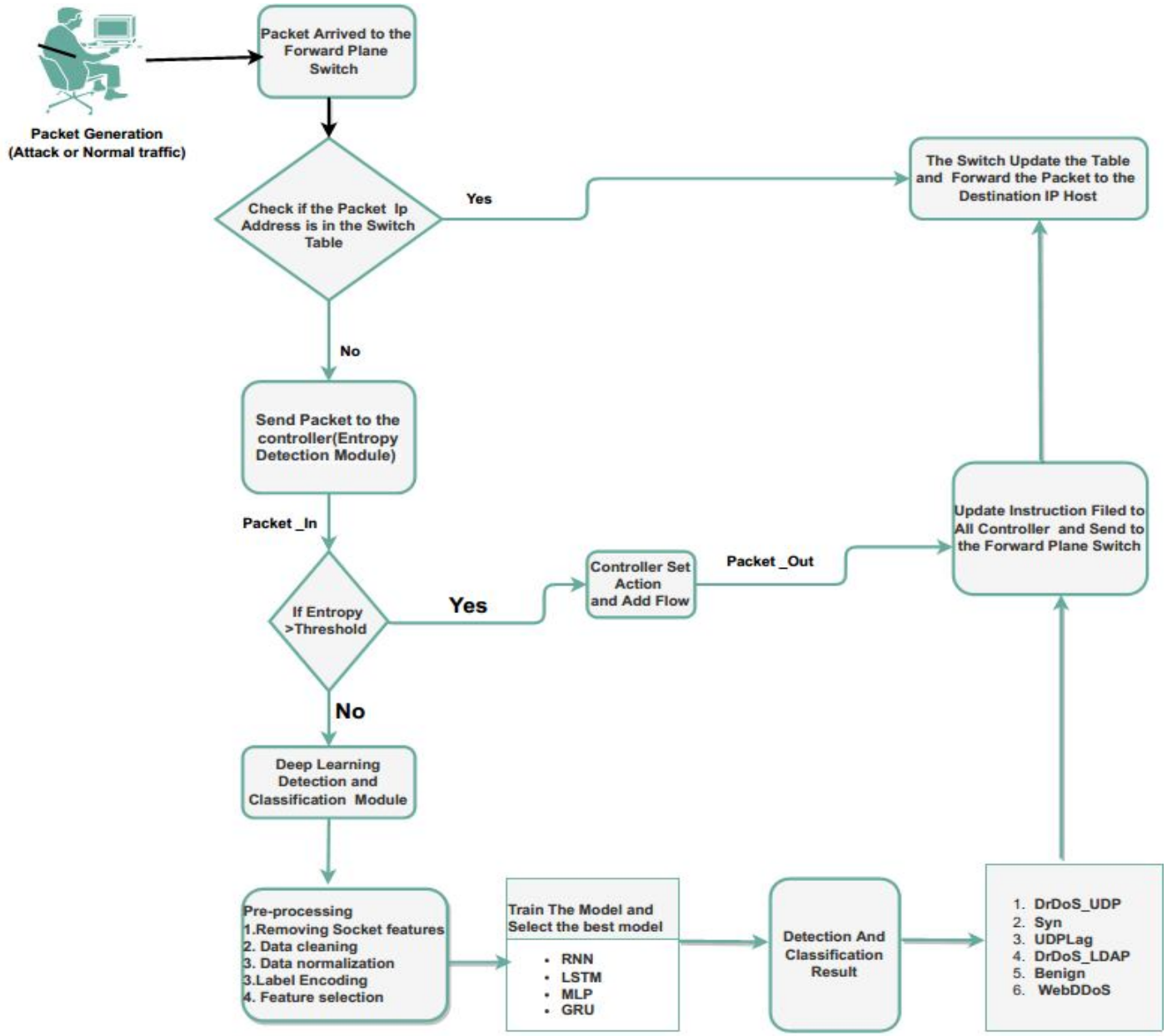

Figure 4. 2. The workflow of proposed model architecture

4.3. Proposed Deep LSTM Architecture

LSTM deep learning model is effective because they can learn a hierarchy of basic to complex features thanks to the stacking of many layers. To address categorization problems, there are numerous algorithms available on the market. One of these, the neural network, is well known for accurately describe the ability. But it requires a lot of computation time. The values of the hyperparameters and the level of preparation are the only factors that affect how effective the LSTM model is. The settings for the model are shown below.

Proposed operational summary of DDoS classification LSTM model

1. Read the data
2. Combine the different attack type into single .csv file
3. Remove socket feature
4. Data cleaning
5. Normalize the data and scale the data set
6. Create the train test split
7. Label Encoding
8. Create the model
9. Pass the data set to the model to build the sequence
10. Build the model
11. for each sequence of the model do:
    - The model will start inserting the sequence to the LSTM input layer with 3 neurons
    - add dropout weight of 0.01 Thus will reduce overfitting to improve the model performance
    - Then pass it to an LSTM layer with 2 neurons
    - Add dropout to drop unwanted neurons with a dropout weight of 0.01
    - Pass it to the output dense layer to make the classification result of 6 attack types

4.3.1. Network Architecture Layers

A. Hidden Layer

It is essential to take into account the dropout, the number of LSTM cells that should be employed in each layer, and the number of hidden layers while developing an LSTM model. This study has found effective models, however there is no right or wrong method to choose the quantity of hidden layers or the number of cells within each layer. This figure relies on how the LSTM model is implemented because there will be different numbers of cells and layers, and it is possible to have the same number of cells in each layer to achieve the best structure.

B. Dense Layer

A dense layer is a layer in a densely connected network, connecting every cell to every other cell in the layer above it. Building their model of hidden layers, followed by numerous dense layers, is how effective models are observed when using layers.

4.4. Optimal Hyperparameter

Some hyperparameters must be properly set up and updated when creating the LSTM model in order to get a trustworthy estimate when the suggested model is checked once more. Empiric tests on the proposed model and data might be conducted to determine the appropriate hyperparameters that would increase accuracy and reduce the possibility of over-fitting the results. According to several studies which were thought to be highly useful, the LSTM model was created using default hyperparameters that better suit this application when undertaking an empiric evaluation. After that, look at each hyperparameter individually to try to determine its ideal value. The ideal value for a hyperparameter is determined by assessing the LSTM model by back-testing it using the test results.

4.4.1. Regularization Technique

The learning algorithm becomes less sensitive to the training data and methods thanks to regularization, which gives it consistency. Because they lack a reference and don't know the true function to use to compare the proposed approximate function with it, the best strategy will be to build a complicated model that overfits (exceeds expectations in how well it matches the training data) and regularizes it so that it would have a reasonable generalization error.

A. Dropout

An effective method for reducing over-fitting is called a dropout. It involves randomly picking cells within a layer according to the probability selected, and changing those cells' output values to 0. To determine the ideal level of dropout, an empirical analysis was conducted, and it was then expanded to include all hidden layers. The LSTM model was created and trained before the dropout was set to various values with a constant difference between consecutive dropout values.

B. Early Stopping

As soon as the recorded metric stops improving, early stopping is Stop training. This strategy seeks to regularize and simplify the cost function so that the generalization error is reduced. A validation error is reported after each iteration, which is how it operates. A copy of the parameters will be saved and used again until the optimization method is finished if the validation error decreases. When it comes to calculating time and resources, it's an excellent strategy. The purpose of training is to reduce loss; hence, the metric to track is loss, and the mode is minimum. The model updates at the end of each epoch. If the loss is no longer decreasing, the Fit() training loop will confirm that it takes patience and min delta into account.

Model once it is discovered to stop decreasing. The training will be stopped after the Stop() function returns true.

4.4.2. Number of Epoch

The epoch is the point at which all of the training data has been delivered over the network; consequently, one epoch is a transmission of all of the training data over the network. The training data is divided into a batch size, with a batch size of 1000, before being dispersed throughout the network. The epoch continues until all samples have been transmitted via the network, at which point one epoch has ended.

4.4.3. Activation Function

ReLU: Even if the input is not row, the rectified linear unit produces 0 if the input is less than 0. If the input is larger than 0, the output will be identical to the input. The ReLU's operation is more similar to that of the hypothetical biological neurons. The hidden layer of NN uses this activation function, which is the most common. Max(0,z)max is a deceptively straightforward formula (0,z). Despite its name and outward look, it is not linear and, while performing better than Sigmoid, offers the same advantages.

Softmax: In general, we employ the neural network function at the last layer that computes the probabilities distribution of the event over 'n' distinct events. The function's ability to handle several classes is its key benefit. The SoftMax activation function is more frequently utilized in categorical classification and is used in experiments on the output layer of the model.

4.4.4. LSTM Loss Function

During fast learning preparation, the loss function determines the separation between an LSTM model's output and the desired output. The validation data set provided by the user yields the best results and prevents over-fitting by stopping the model during training since the performance of the training data is compared to the validation data at each epoch. It might be over-fitting if the training loss decreases but the validation gain grows simultaneously.

4.4.5. Optimizer

Due to Adam's great efficiency and quick convergence compared to the other potential optimizers, it was recommended that it be used by default for the LSTM model. The suggested model set the learning rate to 0.001 while utilizing the Adam optimizer. Adam is an adaptive learning method; therefore, it measures each user's learning rates for different variables. Adam adjusts the learning rate for each neural network weight using the first and second gradient estimates because its name is taken from the adaptive moment calculation.

CHAPTER FIVE

5. IMPLEMENTATION OF THE PROPOSED MODELS

5.1. Chapter Overview

In this section, we discussed the implementation of the proposed model that is entropy-based and deep learning-based. We also discussed the configuration of the working environment in Mininet and the deep learning models in collab environments implantation

5.2. Environment Setup

Application Software and Integrated Development Environment

Colab: Using the TPU, GPU, RAM, and vast storage space to upload the data set for training the results until a 12-hour session, and it is a cloud-based Jupyter notebook can speed up computing.

Jupyter Notebook: is the most popular and suggested integrated development environment (IDE) for researchers to update Python code, visualize certain diagrams, train and test charts, and process progress. The Jupyter notebook with a 6.1.5 version, installed via anaconda installation, is used to carry out the suggested method.

IDLE: Is a Learning Environment used for writing Python programs, IDLE offers a fully functional text editor. It makes writing Python code simple for programmers. In this experiment, we used idle 3.9(64-bit).

Mininet: The network emulator used for this experiment is called Mininet, and it creates a network of virtual hosts, switches, controllers, and links. It is a common network emulator that may be applied to SDN. Also, on a single laptop or computer, Mininet offers research, learning, and testing that contributes in the development of a network. In this experiment, we used Mininet 2.2.2.

POX controller: Pox is mostly used for experimentation because it is quick, easy to use, and built to serve as a platform for a custom controller. It is an upgraded version of NOX, which it replaced. Both run on Python. With topology discovery, POX runs on Windows, Mac OS, and Linux. Pox is an Opensource. supports python language and OpenFlow V1.0.

Scapy: Used for packets generation, scanning, sniffing, attacking, and forging, it is an amazingly powerful tool.

Python: Python 2.7 was chosen as the programming language to carry out the suggested approach.

Keras: Keras, which loads the pre-trained model, is version 2.8.0, Enables quick and simple prototyping. supports both convolutional and recurrent networks, as well as a combination of the two. runs smoothly on both the CPU and GPU.

5.3. Perform experiments On Entropy-based Method

Entropy determines how random an event is to occur relative to all other events. There is no flow matching it in the switch table, so the packet must be transmitted to the controller to be evaluated for a new flow. Lower values of entropy will be regarded as attacks based on the tests conducted in this thesis, which helped us determine a threshold for entropy. The threshold can be changed whenever the network setup varies. The source address of each packet is fresh when it reaches the controller. They are passed on to the controller because they do not appear in the switch's database. The controller will establish a flow in the switch for each new incoming connection so that the remaining arriving packets will be sent directly to their destination without further processing.

When every packet is going to just one host in this scenario, maximum entropy happens. When every packet in a window is going to the same host, minimum entropy is reached. In this experiment, we also explain if one controller fails the other controller continues working. Switches will search for the second controller added to their configuration if they can't maintain communication with the first controller. Losing the controller to a DDoS attack is one of the scenarios that was suggested. The backup controller will receive regular status reports from the deep server controller letting it know it is still alive. The second controller will assume control and begin operating the network normally if the first controller enters an unknown state or is rendered unreachable.

The experiment covers both normal and attack traffic run scenarios.

- **Normal Traffic**: To determine the threshold for typical traffic, all switches are running regular traffic with packets created at random and sent to all hosts (normal).
- **Attack Traffic:** Two hosts are used to conduct the attack traffic and Manual attacks were carried out.

Following the initial configuration of the test environment, Mininet should be running on the virtual box and be available in a ssh window. Here is the virtual window running in Mininet and the Mininet running in SSH. We have Mininet and pox directory. Inside the pox directory, we have l3.editing.py and detection.py files that detect the attack. in the Mininet ssh, we have launchAttack.py and launch_traffic.py.

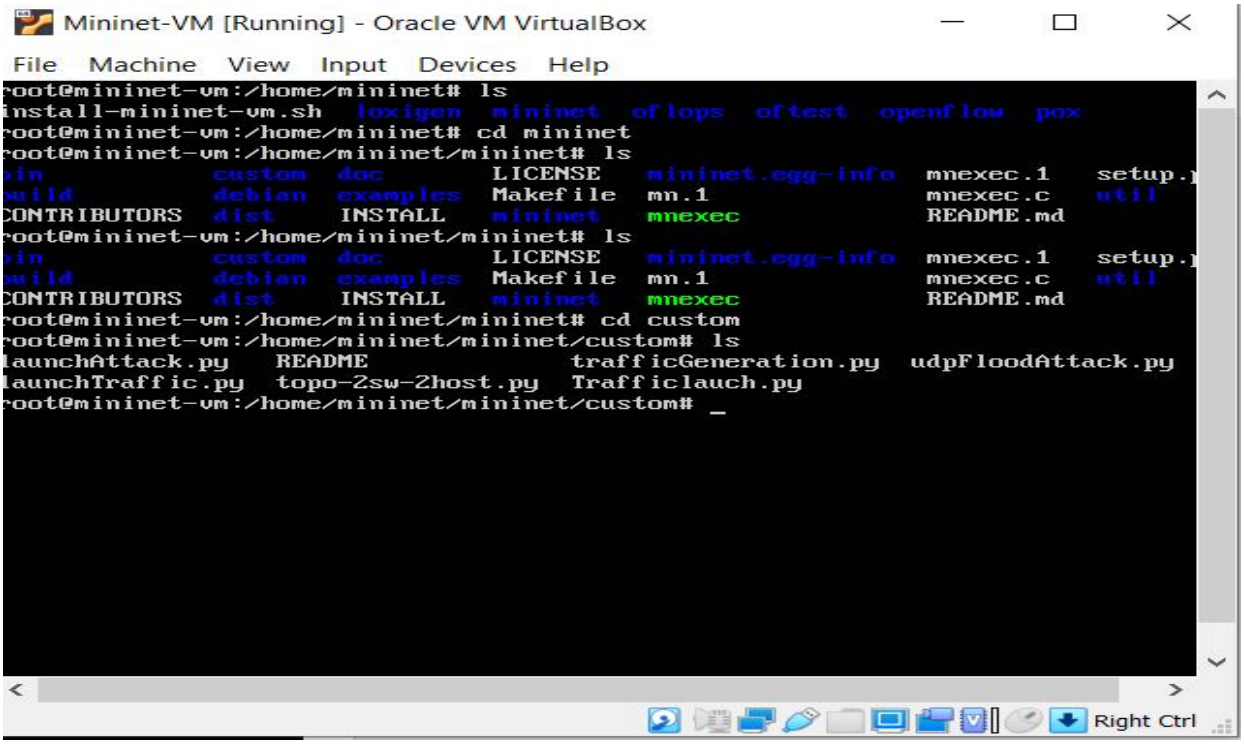

Figure 5.1.Virtual box window running mininet

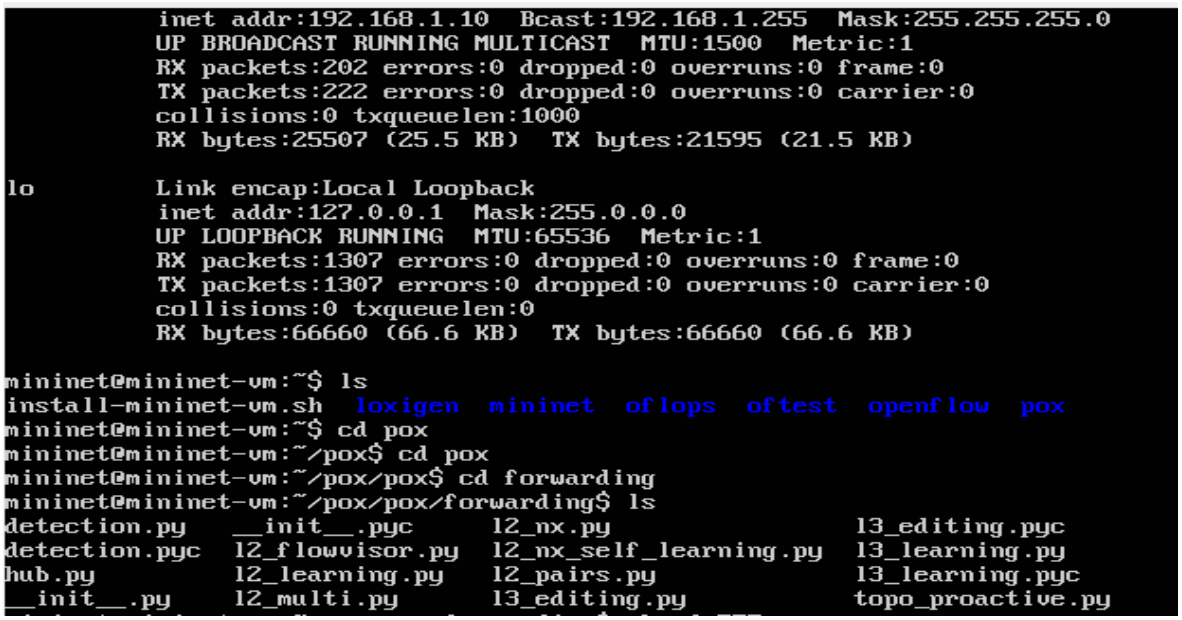

Figure 5.2. SSH mininet running on window operating system

The POX controller is connected to the Mininet topology in the figure 5.3 below. This pox controller supports OpenFlow.of_01. Then topology is created with tree structure. In this case, **Ovsk** indicates the open switch and tree topology with a given controller IP and port number

and uses OpenFlow.of_1. With this command, a network is created and links, hosts, switches, and controllers are added. The address for a loopback is 127.0.0.1.

$ python./pox.py forwarding. l3_editing

$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --

controller=remote,ip=127.0.0.1,port=6633

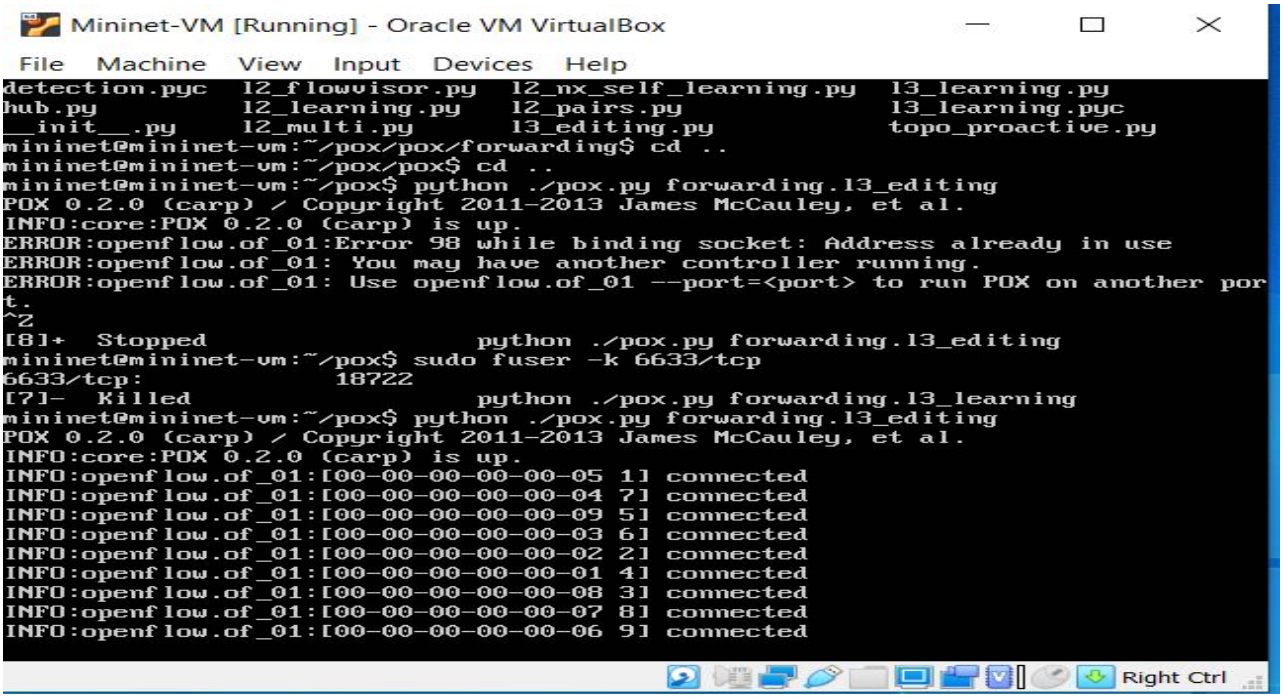

Figure 5.3 Connecting the pox controller with mininet topology

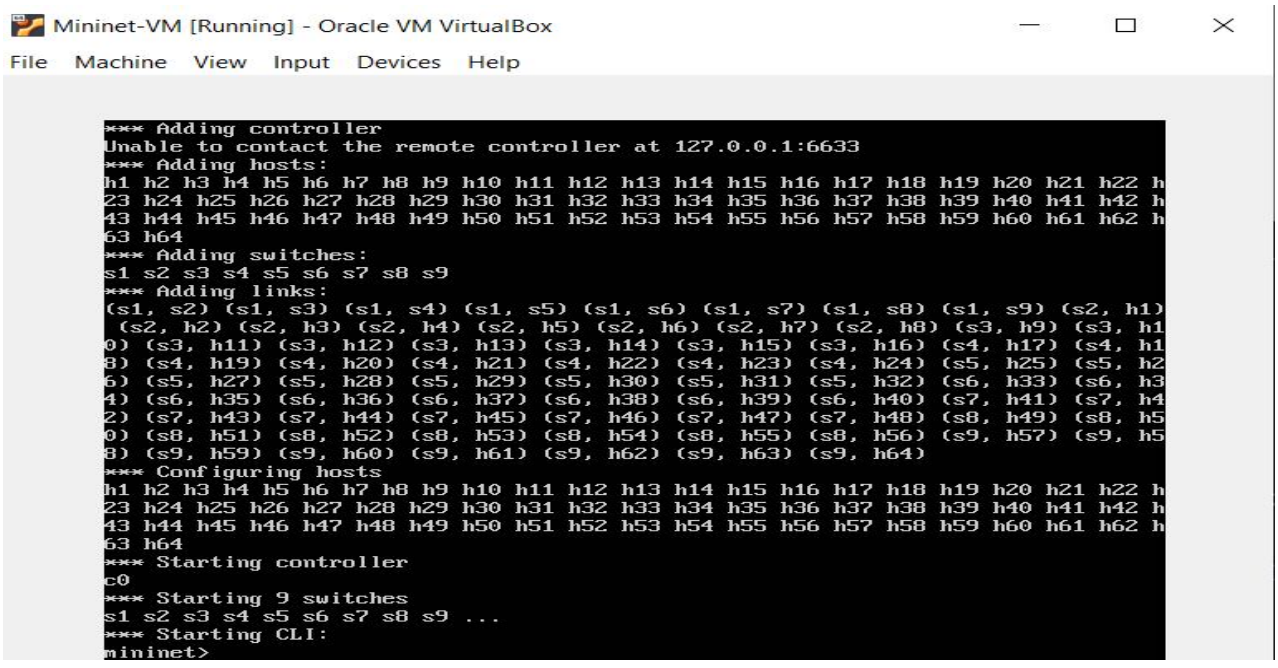

Figure 5.4. Run Pox controller and create Mininet topology

In the ssh window, type the following command to open Xterm windows:

mininet> Xterm h1 h2 h3 h64

And run the following command in the ssh window running Mininet

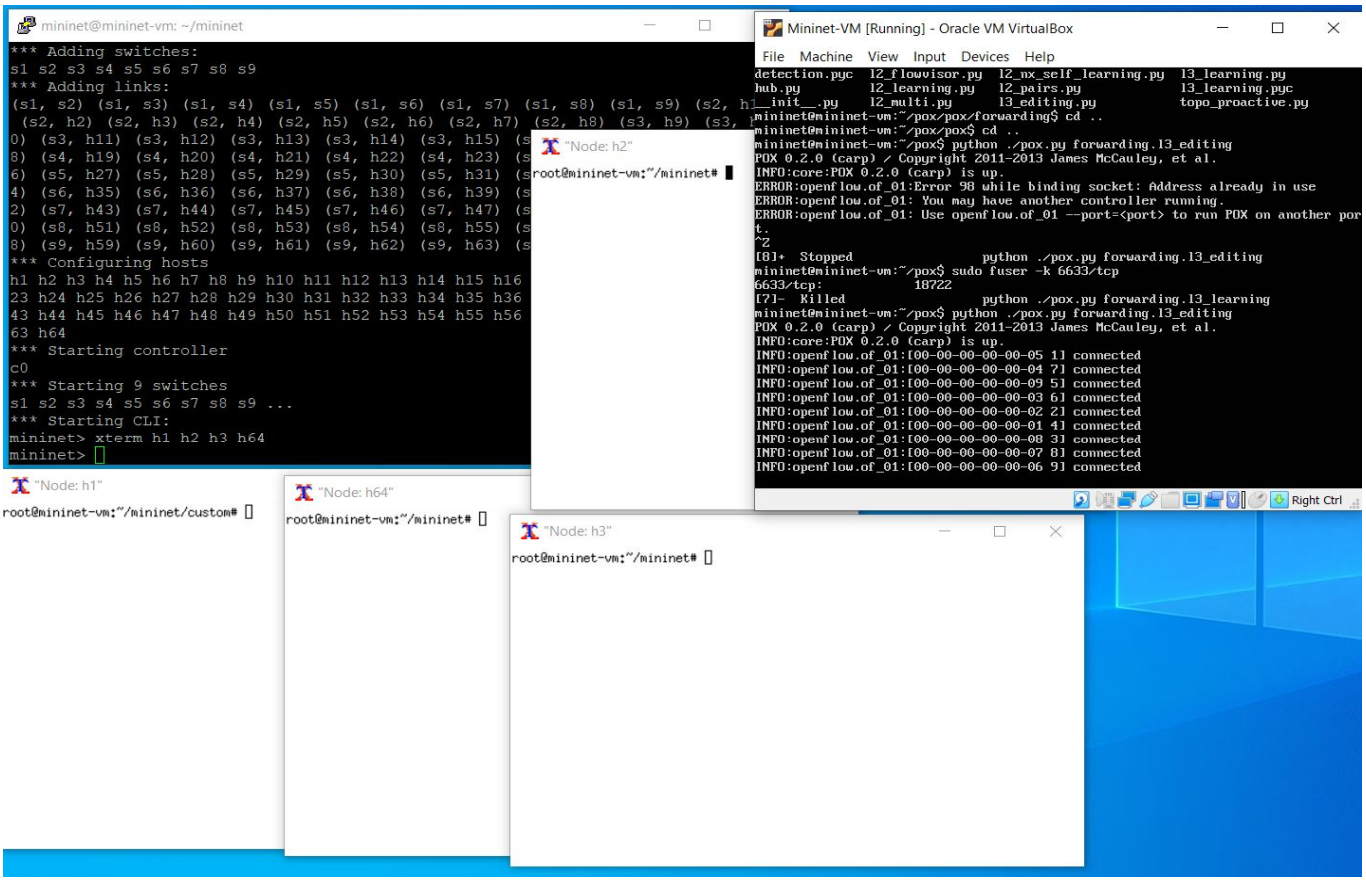

Figure 5.5. Open xterm windows after running the above command

As seen in the above diagram, we create a topology that connects the pox controller to a mininet topology that has 64 hosts connected to the controllers. We are now able to communicate the topology with the pox controller and are prepared to produce normal and ATTACK traffic for the controllers.

5.3.1. Normal Traffic Packet Generation

All hosts in Mininet receive incremental IP addresses starting at 10.0.0.1 on the ward. While all other hosts and switches are operating normally, we randomly selected one host in a switch to transmit attack packets to another host.

To run the normal traffic code, use the following command:

 # python./launchTraffic.py -s 2 -e 65

The prefixes -s and -e here denote the start and end parameters, respectively. The inputs provided to the Gendest function, which produces the Dst IP address, are 2 and 65.

Figure 5.6. Normal traffic change generated by host 1

In figure 5.6 A list of IP addresses to which packets are being delivered serves as the basis for calculating entropy. The first three octets of 10.0.0 are the same in all of the destination IP addresses. The launchTraffic.py file randomizes the fourth octet. In our mininet topology, these IP addresses correspond to the 64 hosts. A dictionary is kept up to date for every 50 packets delivered. Both the destination IP addresses and the number of times a packet has been sent to each address are included in the dictionary. Finally, entropy is calculated as follows:

Entropy = - (count/50) * log10(count/50),

where 50 is the number of packets that were counted when the dictionary was used.

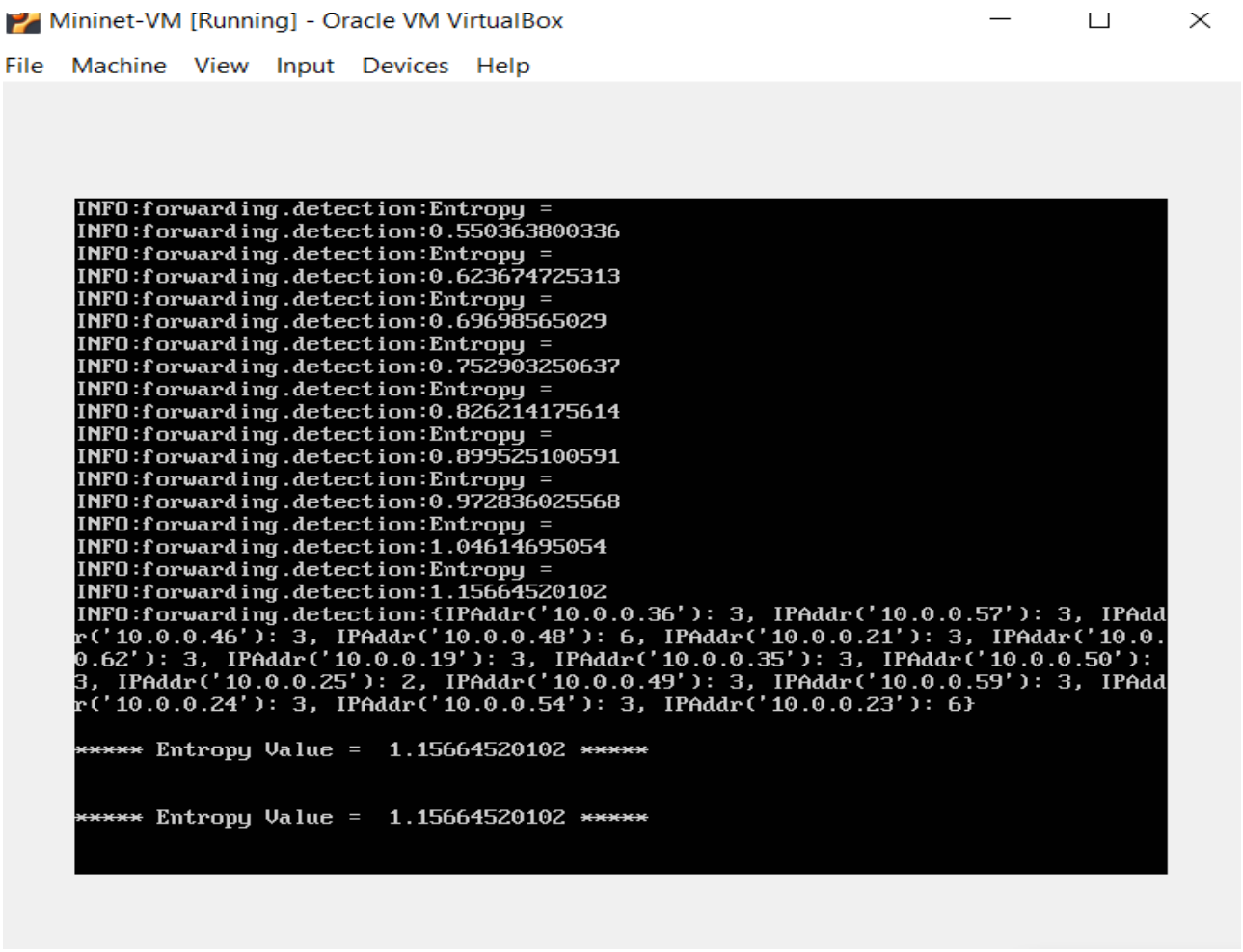

Figure 5.7. Entropy calculated based on listed Ip address by generating normal traffic

Now, as seen above figure 5.7, we should be able to observe the pox controller outputting a list of entropy values. As a result of the loss of a switch, we select for an entropy value of 1.00 rather than 1.15 to prevent false positives and negatives.

5.3.2. Attack on the host

In this case, one of the other hosts has become the target because we generated the attack traffic from two hosts. In this case, we generate an attack that is launchAttack.py to host 1 and host 2 targeting host 64. By the following command

$ python./launchAttack.py 10.0.0.64

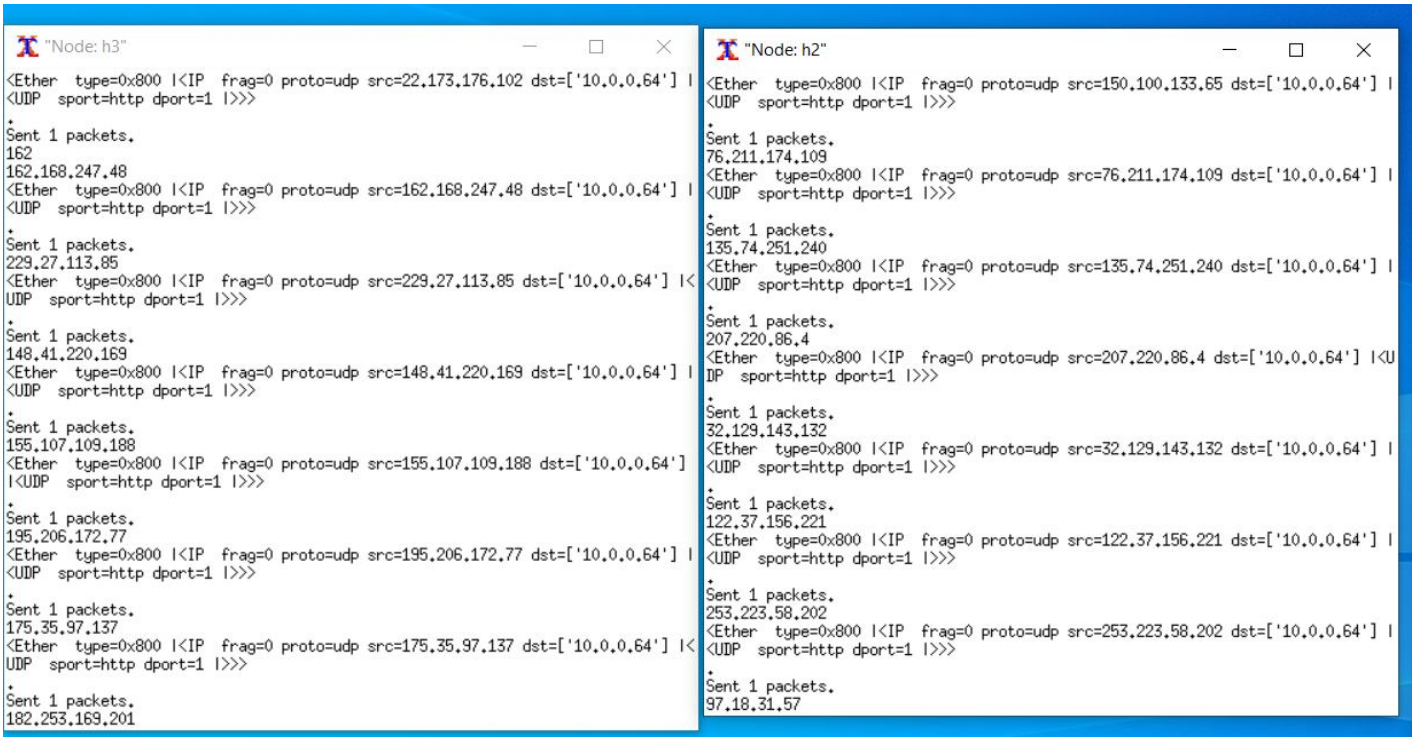

Figure 5.8. DDoS attack traffic generation on host 1 and host 2

As the figure 5.8 below clearly shows, when a network attack is identified, the entropy value does not remain as stable as it would in a normal situation, which causes a sudden change in packet flow to happen as is expected. This implies that the number of packets will grow, which will cause the entropy value to decrease. The amount of entropy drops below the threshold, which in this instance is equal to 1.

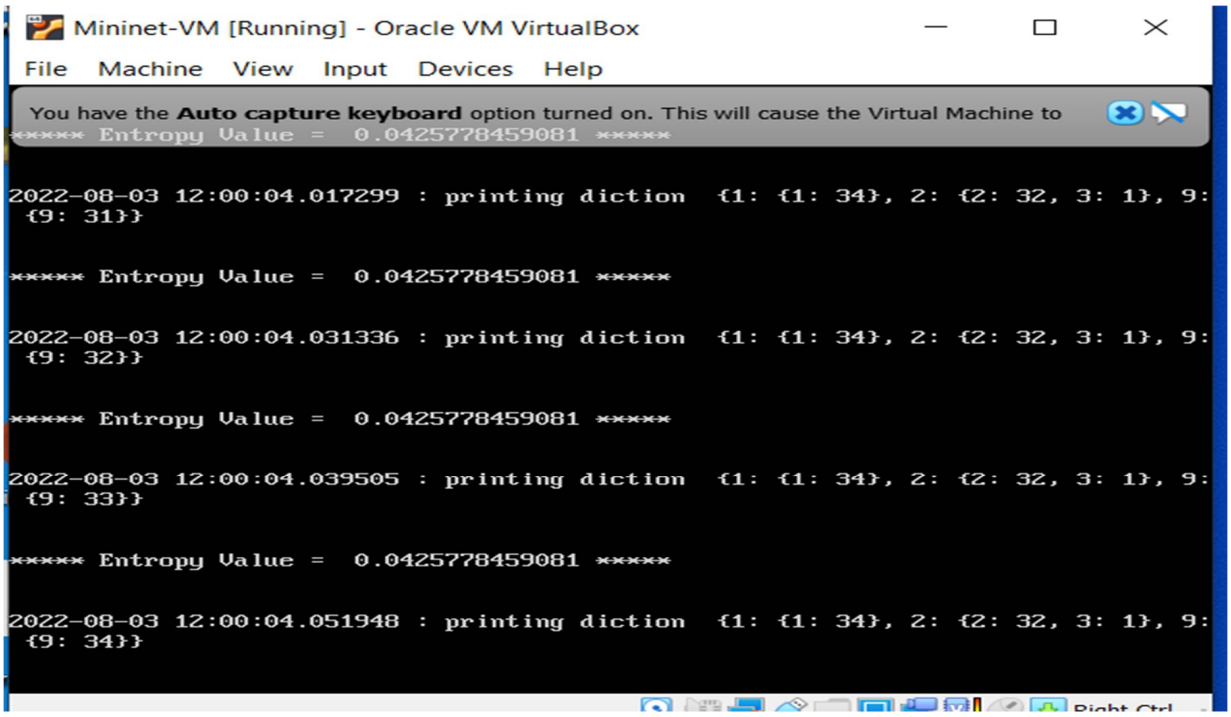

Figure 5.9. Entropy value when an attack is generated from the host1 and host 2

5.4. Deep Learning (Deep Detection Server) Simulation

5.4.1. Dataset preparation

DDoS2019 dataset is used in this experimentation. This includes 12 categories of the most recent common DDoS assaults as well as normal traffic (benign) DDoS attacks. This DDoS is based on real-world facts (PCAPs). And with labeled flows based on attack vectors (.CSV files). These DDoS attack types dataset have a size of **20.7 GB** and it is quite difficult to get the type of device to process this type of huge size b/s it needs a high GPU. So, the option that we decide is taking 6 types of attack types. In this experiment, we select 81 features and 300,000 Total rows for each attack type and we get more than 1 million rows in the dataset. And the attack type that we use in this experiment is LDAP, UDP, UDP_lag, SYN, WEBDDoS, and BENIGN, Preprocessing the dataset in order to reduce overfitting and make it more suitable for training deep learning models is always the first step. We merge the different .csv attack types within a single .csv file.

The implementation environment that we use in this experiment is Google Colab with GPU. The deep learning algorithm that we used in this implementation is MLP, LSTM, RNN, and GRU and we select the algorithm that has the best accuracy. Our dataset contains categorical and numerical values. In The dataset, we have input variables that have 88 features and one target variable (Label). In the input variable, we need to preprocess like removing socket features, cleaning the data, and standardization of input data. After merging the 6 attack types in the target variable (Label) we formed categorical data that have 6 attack types. So here we need to encode the label by one-hot encoding. Here is the source code that we use to select 300,000 rows for each attack type and merge the six types of the attack type.

```
[1] import pandas as pd
    import numpy as np
    number_of_samples = 300000
```

```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] DrDoS_UDP_data_2_per = pd.read_csv('/content/drive/MyDrive/DDos_Dataset/DrDoS_UDP.csv',  nrows = number_of_samples)
    Syn_data_4_per = pd.read_csv('/content/drive/MyDrive/DDos_Dataset/Syn.csv',  nrows = number_of_samples)
    UDPLag_data_2_0_per = pd.read_csv('/content/drive/MyDrive/DDos_Dataset/UDPLag.csv',  nrows = number_of_samples)
    DrDoS_LDAP_data_2_0_per = pd.read_csv('/content/drive/MyDrive/DDos_Dataset/DrDoS_LDAP.csv',  nrows = number_of_samples)
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (85) have mixed types.Sp
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
[ ] # Merge all the Dataset to make one data
    data = pd.concat([DrDoS_LDAP_data_2_0_per, DrDoS_UDP_data_2_per, Syn_data_4_per, UDPLag_data_2_0_per], ignore_index = True)
```

Figure 5.10. Merging in the six attack types in one

5.4.2. Dataset pre-processing

In this experiment, we get the data ready so that it can be used directly to train the model. The CICDDoS2019 dataset is presented in a flow-based format, with CICFlowMeter extracting over 80 features. Before the module training, we go through a few procedures to prepare the data.

5.4.2.1. Removing socket features

From the 88 features dataset, we remove the socket features like unnamed, source IP, Destination IP, destination Port, source port, and similar HTTP. These attributes differ from network to network; hence we must train the model using packet features. Moreover, an invader and a regular user may share the same IP address. Because the DL model will be biased toward the socket data after being trained with it, overfitting may happen from using socket data. After eliminating the unnecessary features, we ended up with a total of 81 features for the model input.

```
[9]  # Drop Unnamed:0, Unnamed:0.1 columns
     #data = data.drop(['Unnamed: 0','Flow ID', ' Source IP', ' Source Port', ' Destination IP',
            #' Destination Port', ' Timestamp', 'Flow Bytes/s',
            #' Flow Packets/s', 'SimillarHTTP'], axis = 1)

     #data = data.drop(['Unnamed: 0','Flow ID', ' Source IP', ' Source Port', ' Destination IP',
            #' Destination Port', ' Timestamp', 'Flow Bytes/s', ' Flow Packets/s', 'SimillarHTTP'], axis = 1)

     data = data.drop(['Unnamed: 0', ' Source IP', ' Source Port', ' Destination IP',
             ' Destination Port', 'SimillarHTTP'], axis = 1)
```

Figure 5.11. Removing the socket features code

5.4.2.2. Cleaning the data

We delete all of the 103442 missing (nan) and infinite values from the original data sets.

```
[11] data_real = data.replace(np.inf, np.nan)

[12] data_real.isnull().sum().sum()

     103442

[13] data_df = data_real.dropna(axis=0)

[14] data_df.isnull().sum().sum()

     0
```

Figure 5.12. Cleaning the data code

5.4.2.3. Standardization of the data

The numerical values of the feature data vary. The model takes a while to train because using the original data to train it directly can lead to classification errors. In this example, we'll use a standard scaler as our standardization approach. This method corrects the distribution to have a mean of zero and a standard deviation of one by subtracting the mean (a procedure known as centering) and dividing by the standard deviation for each input variable.

In this case, standard scaler methods are used to scale numerical data. It is applied to feature scaling. Data preprocessing includes the phase of feature scaling. Because of this, we used Standard Scalar to scale the feature's magnitude within a given range. In general, the data we obtain from the real world differ greatly from one another, and this directly affects the performance of the model. Scaling the data before processing it is therefore always a desirable idea.

```
[53]  from sklearn.preprocessing import StandardScaler
      ss_20 = StandardScaler()
      X_train_std_20 = ss_20.fit_transform(X_train_20)
      X_test_std_20 = ss_20.fit_transform(X_test_20)

[54]  X_train_std_20.shape
      (654729, 20)

[55]  y_train_20.shape
      (654729,)

[56]  X_test_std_20.shape
      (280599, 20)

[57]  y_test_20.shape
      (280599,)
```

Figure 5.13. Standardization of the input data code

5.4.2.4. Encoding the labeled data

In order to categorize the input traffic into LDAP, UDP, UDP lag, SYN, WEBDDoS, and BENIGN, we trained our model for category classification. The categorical labels had to be replaced because deep models can only handle float or numeric values to convert this categorical data in the label to numerical data, we use One-hot Encoding. The attack label had to be converted for this categorical value (i.e., benign and the five attack types). For example, [1, 0, 0, 0, 0, 0] indicates UDP, [0, 1, 0, 0, 0, 0] indicates BENGIN attack type, [0, 0, 1, 0, 0, 0] indicates LDAP attack, [0, 0, 1, 0, 0, 0] indicates SYN attack type, [0, 0, 0, 0, 1, 0] indicates UDP-lag attack type, and [0, 0, 0, 0, 0, 1] indicates WEBDDoS attack type. We used a 6-bit feature vector to indicate different with the addition of 6 more feature vectors, our total number of features increased to 87. (i.e., 81 representing the original features plus 6 features for an attack label).

```
[59] y_train_MLP_20 = np.array(y_train_20)
     y_test_MLP_20 = np.array(y_test_20)

     y_train_MLP_onehot_20 = to_categorical(y_train_MLP_20)
     y_test_MLP_onehot_20 = to_categorical(y_test_MLP_20)

     X_train_20_MLP = np.array(X_train_std_20)
     X_test_20_MLP = np.array(X_test_std_20)
```

```
y_train_MLP_onehot_20
```

```
array([[0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       ...,
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0., 0.]], dtype=float32)
```

Figure 5.14. Encode the label data code

5.5. Feature selection

We have more than 80 characteristics in this DDoS dataset, but instead of selecting every feature from the source data, we focus on finding the right features to detect DDoS attacks. The best features from the raw data can be chosen using PCA, decision trees, random forest regressors, and chi-squared (x2) tests (Gadze et al., 2021). An x2 test was utilized in this study to rank and choose features, and it produced encouraging outcomes. An x2 test examines whether the frequencies of specific classes and attributes depend on the correlation between predictor and target variables or are independent of it. It was figured out in the following manner.

When the dependent variable is assessed at a nominal level, the Chi-square statistic is a non-parametric (distribution free) technique used to analyze group differences. Like all non-parametric statistics, the Chi-square is reliable regardless of how the data are distributed. It does not call for homoscedasticity in the data or equality of variances among research groups. It enables examination of numerous group studies as well as dichotomous independent variables. We choose Chi-square because of its flexibility in handling data from both two group and multiple group studies, robustness with respect to data distribution, ease of computation, detailed information that can be deduced from the test, and use in studies for which parametric assumptions cannot be met.

$$\Upsilon_c^2 = \sum \frac{(Ai - Bi)^2}{Bi}$$

Figure 5.15. Chi-squared (x2) feature selection method formula (Gadze et al., 2021)

where A stands for the observed value, C for the degree of freedom, and B for the projected observation for the ith class. The x2 test was used to the original data set to choose the most pertinent features that were strongly correlated with the target variables. The more optimal features have a stronger link with the target characteristic, therefore they were chosen using the Python-based Sklearn program, to create our learning model, we used 20 features from the original CICDDoS2019 dataset.

Average Packet Size,Min Packet Length,Packet Length Mean, ACK Flag Count, Avg Fwd Segment Size, Fwd Packet Length Min, Max Packet Length, Protocol, Fwd IAT Total, Fwd Header Length.1, Flow Duration,Fwd Packet Length Max, Fwd Packet Length Mean, Fwd Header Length, min_seg_size_forward,Flow ID, Flow Bytes/s, Total Fwd Packets, Init_Win_bytes_backwad,Timestamp. The top 11 features, which were chosen based on their relationship to the attribute values, are listed in part in the table below. The features with the highest scores are listed in order of their rankings, which show a strong connection to and dependency on the target class.

F.No	Optimal features	Rank scores
1	Timestamp	0.3
2	Min Packet Length	0.08
3	Fwd Packet Length Min	0.078
4	Protocol	0.075
5	ACK Flag Count	0.074
6	Average Packet Size	0.073
7	Packet Length Mean	0.07
8	Fwd Packet Length Mean	0.067
9	Avg Fwd Segment Size	0.06
10	Fwd Packet Length Max	0.057
11	Max Packet Length	0.05

Table 5.1 Optimal feature selection samples

5.6. Model Building

In this thesis we used multiple deep learning algorithms to test the DDoS2019 dataset then and we select the algorithm that has the best accuracy. Then finally we compare our best model algorithm with other researchers who use different deep learning algorithms with this dataset. These deep learning algorithms are:

5.6.1. RNN

At any input stage, the RNN takes into account the earlier computations along with the current occurrences. Such techniques for training the model can preserve all data information with the least amount of loss. When compared to other RNN approaches, the standard RNN is easier to use and requires less training time(Elsayed et al., 2020). This model is capable of fully learning a vector representation of the input data. In the given dataset pre-processing is the first phase. in this case removing the socket feature, cleaning the data, normalizing the input data, and encoding label data we then going to train our model. Using the Holdout cross-validation approach, we divided the data into train and test groups. Instead of using the k-fold cross-validation technique in this thesis, we employed the train-test split technique. Although cross-validation is a popular technique for classifier evaluation, we do not employ it in this circumstance.

```
batch_size = 1000

# Initialize the network
model_SimpleRNN = Sequential()
model_SimpleRNN.add(SimpleRNN(3,input_dim=20, return_sequences=True, a
ctivation='relu'))
model_SimpleRNN.add(Dropout(0.01))
model_SimpleRNN.add(SimpleRNN(3,input_dim=20, return_sequences=False))
model_SimpleRNN.add(Dropout(0.01))
model_SimpleRNN.add(Dense(6))
model_SimpleRNN.add(Activation('softmax'))

monitor = EarlyStopping(monitor='val_loss', min_delta=1e3, patience=5,
 verbose=1, mode='auto', restore_best_weights=True)

model_SimpleRNN.compile(loss='categorical_crossentropy',optimizer='ada
m',metrics=['accuracy'])
history_RNN=model_SimpleRNN.fit(X_train_SimpleRNN_reshape, y_train_one
hot_SimpleRNN, validation_data=(X_test_SimpleRNN_reshape, y_test_one_h
ot_SimpleRNN),batch_size=batch_size, epochs=30,callbacks=[monitor])
```

Figure 5.16. Define RNN network layer

Using category classification, the SoftMax layer in this experiment divides the input data into six different attack types. We used categorical-cross-entropy in the loss function along with the Adam optimizer (which has a learning rate of 0.001), "Relu" activation in all of the different layers, and kernel regularizer = 12. In order to reduce overfitting when we trained the model,

we also used a dropout (0.01). Additionally, we train our model across a number of 30 epochs with 1000 batches.

```
model_SimpleRNN.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_31 (SimpleRNN)   (None, None, 3)           72

 dropout_82 (Dropout)        (None, None, 3)           0

 simple_rnn_32 (SimpleRNN)   (None, 3)                 21

 dropout_83 (Dropout)        (None, 3)                 0

 dense_46 (Dense)            (None, 6)                 24

 activation_40 (Activation)  (None, 6)                 0

=================================================================
Total params: 117
Trainable params: 117
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.17. RNN model summary

5.6.2. MLP

An improvement to the feed-forward neural network is MLP. The input layer, output layer, and hidden layer make up this structure. The real computational engine of the MLP consists of an arbitrary number of hidden layers that are placed between the input and output layers. In an MLP, data moves forward from the input to the output layer similarly to a feed-forward network. With the help of the back propagation learning algorithm.

```
batch_size = 800

# 1. define the network
model = Sequential()
model.add(Dense(500,input_dim=20,activation='relu'))
model.add(Dropout(0.03))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.03))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.03))
#model.add(Dense(100,activation='relu'))
#model.add(Dense(200,activation='relu'))
#model.add(Dropout(0.02))
model.add(Dense(6))
model.add(Activation('softmax'))monitor = EarlyStopping(monitor='val_loss',
 min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)

# try using different optimizers and different optimizer configs
model.compile(loss='categorical_crossentropy',optimizer='adadelta',metrics=['ac
curacy'])
history_mlp= model.fit(X_train_20_MLP, y_train_MLP_onehot_20, validation_data=(
X_test_20_MLP, y_test_MLP_onehot_20),batch_size=batch_size, epochs=40,callbacks
=[monitor])
```

Figure 5.18. Define the MLP network layer

Using category classification, the SoftMax layer in this experiment divides the input data into six different attack kinds. We used categorical-cross entropy in the loss function along with the Adadelta optimizer (which has a learning rate of 0.001) and Relu activation in all of the different layers. In order to reduce overfitting when we trained the model, we additionally utilized a dropout (0.03). Additionally, we train our model using a batch size of 800 and many epochs of 40.

```
model.summary()

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_7 (Dense)              (None, 500)               10500

 dropout_9 (Dropout)          (None, 500)               0

 dense_8 (Dense)              (None, 1000)              501000

 dropout_10 (Dropout)         (None, 1000)              0

 dense_9 (Dense)              (None, 1000)              1001000

 dropout_11 (Dropout)         (None, 1000)              0

 dense_10 (Dense)             (None, 6)                 6006

 activation_4 (Activation)    (None, 6)                 0

=================================================================
Total params: 1,518,506
Trainable params: 1,518,506
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.19. MLP model Summary

5.6.3. GRU

A more recent variety of recurrent neural network that resembles an LSTM is the GRU. The GRUs gave up using the cell state and switched to communicating in the concealed state. Only two gates—one for reset and the other for an update—are present. Because GRUs have fewer tensor operations than LSTMs, they can be trained more quickly. Which one is superior cannot be determined with certainty. Engineers and researchers typically test both to see which is more effective for their use case. Using category classification, the SoftMax layer in this experiment divides the input data into six different attack kinds. We used a dropout (0.01) to reduce overfitting when we train the model, we employed categorical-cross entropy with Adam optimizer (with a learning rate of 0.001). Additionally, we train our model through a number of 20-epoch iterations with 1000-batch sizes.

```
batch_size = 1000

# Initialize the network
model_GRU = Sequential()
model_GRU.add(GRU(2,input_dim=20, return_sequences=True))
model_GRU.add(Dropout(0.01))
model_GRU.add(GRU(1,input_dim=20, return_sequences=False))
model_GRU.add(Dropout(0.01))
model_GRU.add(Dense(6))
model_GRU.add(Activation('softmax'))

monitor = EarlyStopping(monitor='val_loss', min_delta=1e3, patience=5, verbose=1
, mode='auto', restore_best_weights=True)

model_GRU.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['acc
uracy'])
history_GRU= model_GRU.fit(X_train_GRU_reshape, y_train_onehot_GRU, validation_d
ata=(X_test_GRU_reshape, y_test_one_hot_GRU),batch_size=batch_size, epochs=20,ca
llbacks=[monitor])
```

Figure 5.20. Define GRU Network Layer

```
[ ]  model_GRU.summary()

Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_24 (GRU)                (None, None, 2)           144

 dropout_100 (Dropout)       (None, None, 2)           0

 gru_25 (GRU)                (None, 1)                 15

 dropout_101 (Dropout)       (None, 1)                 0

 dense_55 (Dense)            (None, 6)                 12

 activation_49 (Activation)  (None, 6)                 0

=================================================================
Total params: 171
Trainable params: 171
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.21. GRU model Summary

5.6.4. LSTM

In this experiment, Initialization of Sequential function the **SoftMax layer** takes the input and classifies the data into six different attack types using **categorical classification**. In the loss function, we used **categorical-cross-entropy** with **Adam optimizer (**have a learning rate of **0.001)** and **"Tanh"** activation in all different layers We also used a **dropout(0.05)** this Dropout Layer used for prevent overfitting when we train the model. We also train our model with several epochs of **23,** Prediction of DDoS Attacks using **the SoftMax** function, and batch size of **1000**.

```
batch_size = 1000

# Initialize the network
model_LSTM = Sequential()
model_LSTM.add(LSTM(3,input_dim=20, return_sequences=True))
model_LSTM.add(Dropout(0.01))
model_LSTM.add(LSTM(2,input_dim=20, return_sequences=False))
model_LSTM.add(Dropout(0.001))
model_LSTM.add(Dense(6))
model_LSTM.add(Activation('softmax'))

monitor = EarlyStopping(monitor='val_loss', min_delta=1e3, patience=5, verbose=
1, mode='auto', restore_best_weights=True)

model_LSTM.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['a
ccuracy'])
history_LSTM= model_LSTM.fit(X_train_lstm_reshape, y_train_onehot_lstm, validat
ion_data=(X_test_lstm_reshape, y_test_one_hot_lstm),batch_size=batch_size, epoc
hs=20,callbacks=[monitor])
```

Figure 5.22. Define the LSTM model layer

5.7. Model testing and evaluation

Cross-Validation is a resampling technique with the fundamental idea of splitting the dataset into 2 parts- training data and test data. Train data is used to train the model and the unseen test data is used for prediction. Several cross-validation methods are used in machine learning. Examples like the **Hold out cross-validation method**, **Leave One Out Cross-Validation, K-fold cross-validation**, etc. As a rule, the proportion of training data has to be larger than the test data. One of the major advantages of the **Hold out cross-validation** method is that it is computationally inexpensive compared to other cross-validation techniques. So, in this paper, we use the **Holdout cross-validation method** b/s It is the simplest evaluation method and it is computationally inexpensive compared to other cross-validation techniques. In this dataset, we specify the random state =42 to avoid high variance, test size=30, And training size=70.

```
from sklearn.model_selection import train_test_split
X_train_20, X_test_20, y_train_20, y_test_20 = train_test_split(data_new_20features_X, dat
a_y_trans, test_size = 0.30, random_state = 42)
```

CHAPTER SIX

6. RESULT, EVALUATION, AND DISCUSSION

In this chapter, the result, and discussion of the proposed models for DDoS attack detection and Classification using Entropy-based and deep learning are discussed. In this case, we propose and test with multiple deep learning models. The four algorithms that we use are RNN, MLP, GRU, and LSTM. And here based on the result(accuracy) we compare the models and select the best model that has high accuracy than the other deep learning algorithms. After we select the best model, we evaluate our best model with baseline paper work. And finally, we show the result by comparing the proposed model with the baseline model we chose to compare.

6.1. Result of the Entropy based Experiment

The figure below shows that when normal traffic is flowing to the controller then the controller is expected to do nothing except calculate the entropy value. The controller can identify whether or not the packet is an attack using this entropy value. The controller in the figure below does nothing since the entropy value is stable and there is no sudden fluctuation that would lead one to conclude an attack is taking place.

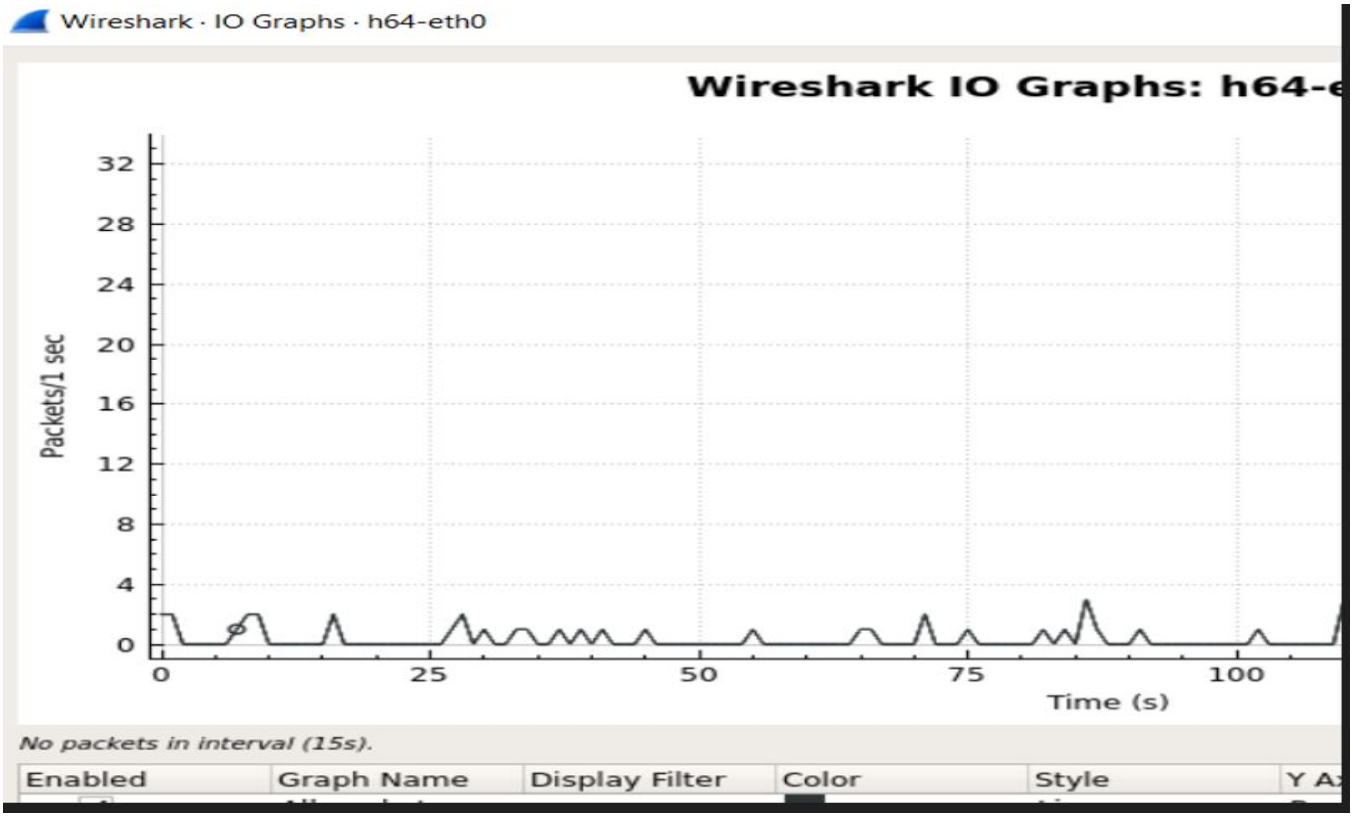

Figure 6.1. Shows IO graphs for normal traffic change that is generated from host one

As seen in Figure 6.2 below, when a network attack is identified, the entropy value does not remain as stable as it would in a normal scenario. Below the threshold value, which in this situation is equal to 1, the entropy value decreases. The dramatic change in packet flow occurs as expected, as the figure clearly demonstrates. This implies that there will be more packets overall. This packet increment implies that an attack is highly likely.

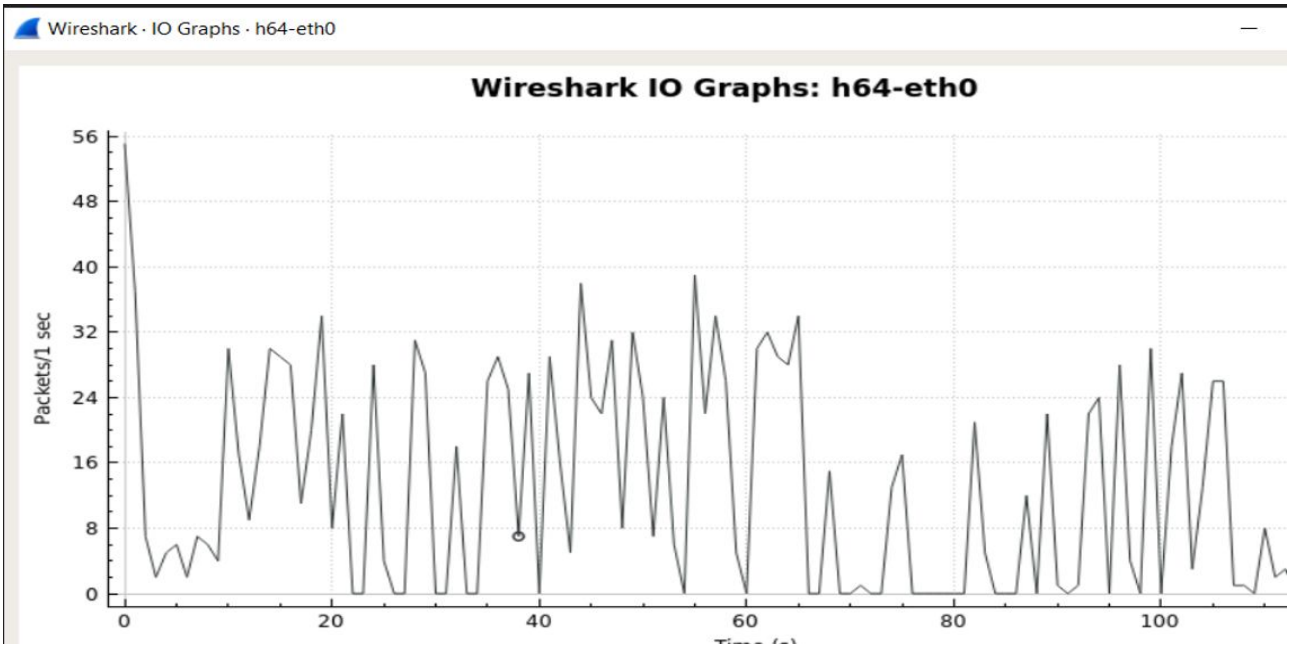

Figure 6.2. IO graph of DDoS Attack results of host 64

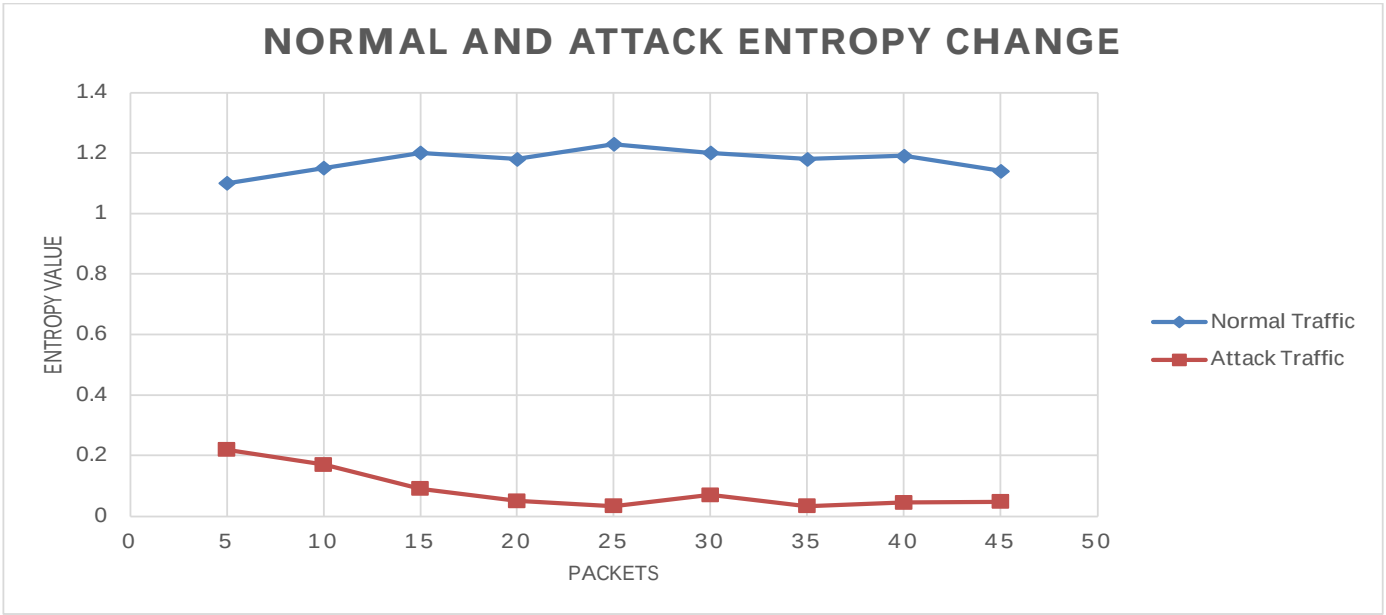

Figure 6.3. Normal and attack traffic Entropy change result

Based on Figures 5.6 by generating normal traffic we calculate the entropy value which is 1.15. to avoid false positives and negatives we take the entropy value 1. This entropy value will be a threshold for detecting the attack. In Figure 5.8 when traffic is generated from hosts 1 and 2 the entropy value decreases to 0.04. this decrement of entropy value increases the probability of an attack. Based on Figure 6.3, which is shown above, compares the outcomes of regular traffic, and attack traffic entropy value variations. As a result, as shown from the graphic, as regular traffic is generated, its entropy value changes and displays as growing from the set threshold. However, when a controller detects attack traffic, the entropy value decreases and falls below the threshold level. here we can conclude that the attack detection with entropy-based has good efficiency but not good accuracy. To increase the accuracy the controller, forward the traffic which has less than the threshold value into the deep detection server which is running a DL algorithm then the deep detection server classifies the attack and classifies it into different attack types.

6.2. Results of the Deep learning models

Here we made 4 experiments with different deep learning algorithms with the same dataset that is DDoS2019. These algorithms are RNN, MLP, GRU, and LSTM. We made an experiment on each algorithm finally we take the optimal one that has better accuracy.

6.2.1. Experiment 1: RNN model

In this experiment, the input is taken by the SoftMax layer and classifies the data into six different attack types using categorical classification. In the loss function, we used categorical-cross-entropy with Adam optimizer (have a learning rate of 0.001) and 'Relu' activation in all different layers and with kernel_regularizer = 12 We also used a dropout (0.01) to minimize model overfitting. We also train our model with several epochs of 30 and batch sizes of 1000.

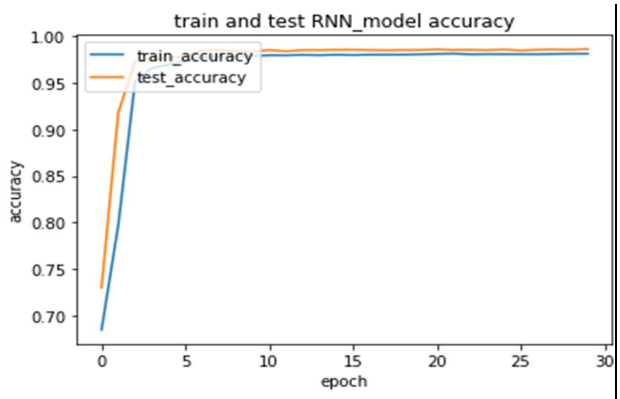

Figure 6.4 RNN Model performance on the train and test accuracy Vs epoch

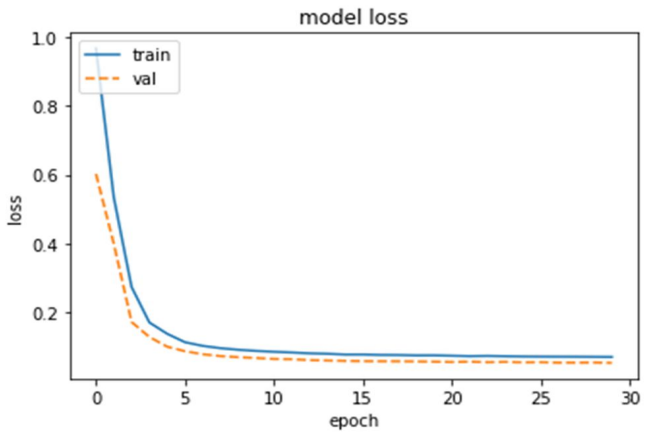

Figure 24 RNN Model performance on train and test loss Vs epoch

The diagram above shows that the RNN model trains up to 30 epochs, the training loss has reduced from 0.9674 to 0.0692, and the testing loss reduced from 0.6023 to 0.0521. Also, the training accuracy has improved from 0.6847 to 0.9811 and the testing accuracy enhanced from 0.7300 to 0.9861.

6.2.2. Experiment 2: GRU model

In this case, the SoftMax layer takes the input and classifies the data into six different attack types using categorical classification. In the loss function, we used categorical-cross-entropy with Adam optimizer (have a learning rate of 0.001) and used a dropout (0.01) to minimize overfitting when we train the model. We also train our model with several epochs of 18 and batch sizes of 1000.

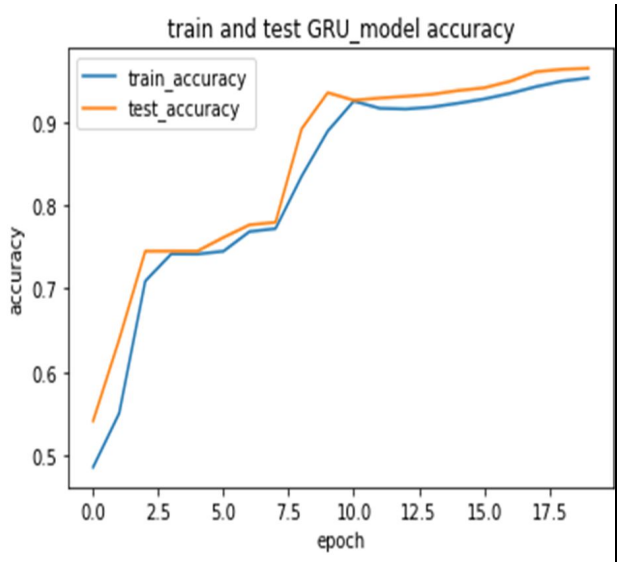

Figure 6.5. **GRU** model performance on a train and test accuracy Vs epoch

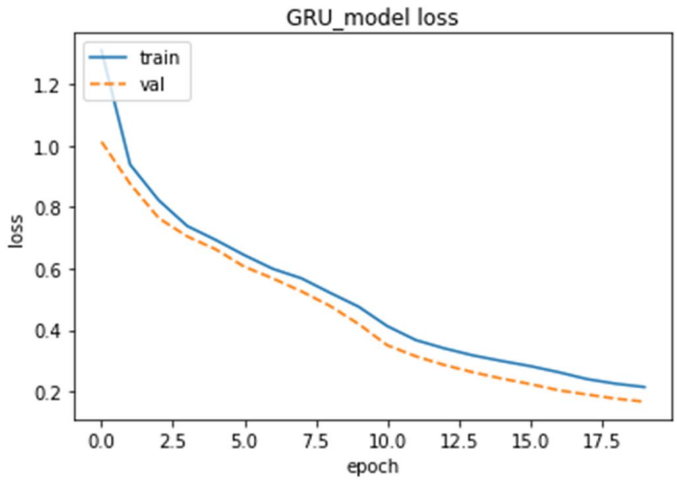

Figure 6.6. **GRU** model performance on train and test loss Vs epoch

The diagram above shows that the GRU model trains up to 18 epochs, the training loss has reduced from 1.3092 to 0.2140, and the testing loss reduced from 1.0126 to 0.1664. Also, the training accuracy has improved from 0.4858 to 0.9526 and the testing accuracy enhanced from 0.5413 to 0.9642.

6.2.3. Experiment 3: LSTM model

In this experiment, Initialization of Sequential function the **SoftMax layer** takes the input and classifies the data into six different attack types using **categorical classification**. In the loss function we used **categorical-cross-entropy** with **Adam optimizer (**have a learning rate of **0.001)** and **tanh** activation in all different layers We also used a **dropout (0.01)** this Dropout Layer was used for preventing overfitting when we train the model. We also train our model with several epochs of **23,** Prediction of DDoS Attacks using **the SoftMax** function, and batch size of **1000**.

Algorithm for LSTM framework

1. Given the following data: 20-features, optimizer, loss, metrics
2. Initiate model from sequential class;
3. Add LSTM layer: LSTM (3, input_dim=20)
4. Having weights from previous layer:

- Add Dropout (0.01)
- Add LSTM layer: LSTM (2, input_dim=20)
- Add Dropout (0.01)
- Having weights from the previous layer:
- Add Dense (6)
- add Activation (SoftMax)

Hyperparameter	Values
Batch size	1000
Input Feature	20
Dense	6
Early stopping	Keras callback early stopping with monitor of loss
dropout	0.01
Activation function	SoftMax layer
Learning rate	0.001
Output layer neurons	6
Hidden layer neurons	3,2
Optimizer	Adam
Epoch	23

Table 6.1. Hyperparameters setup for LSTM

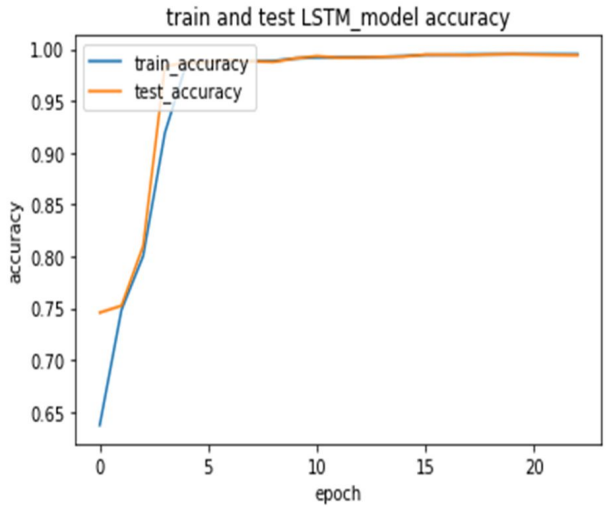

Figure 6.7. LSTM model performance on the train and test accuracy Vs epoch

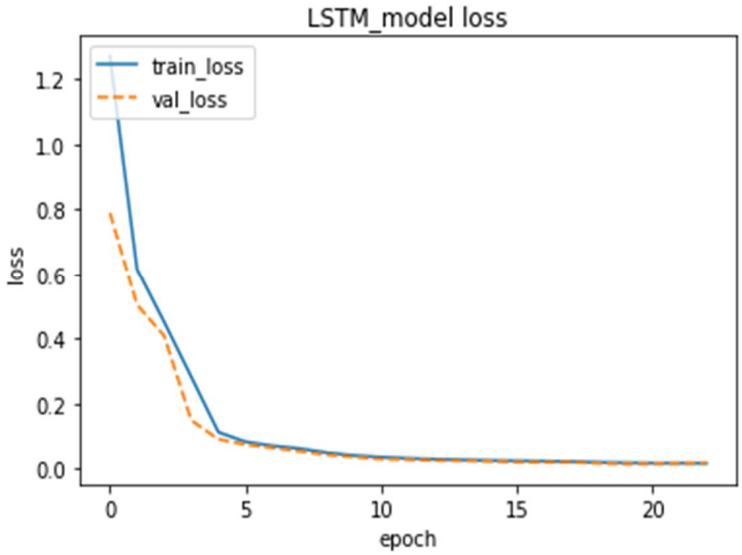

Figure 6.8. LSTM model performance on train and test loss Vs epoch

The diagram above shows that the LSTM model trains up to 23 epochs, the training loss has reduced from 1.2709 to 0.0178, and the testing loss reduced from 0.7896 to 0.0184. Also, the training accuracy has improved from 0.6372 to 0.9942 and the testing accuracy enhanced from 0.7458 to 0.9943.

6.2.4. Experiment 4: MLP model

Using category classification, the SoftMax layer in this experiment divides the input data into six different attack types. We used categorical-cross entropy in the loss function together with the Adadelta optimizer (which has a learning rate of 0.001) and Relu activation in all of the different layers. To reduce model overfitting, we additionally utilized a dropout (0.03). Additionally, we train our model using a batch size of 800 and some epochs of 40.

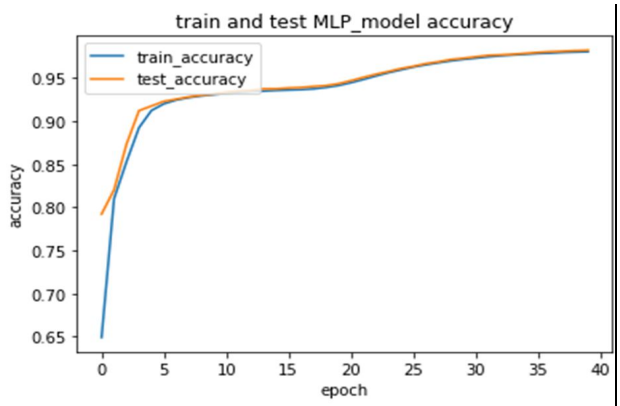

Figure 6.9. MLP model performance on the train and test accuracy Vs epoch

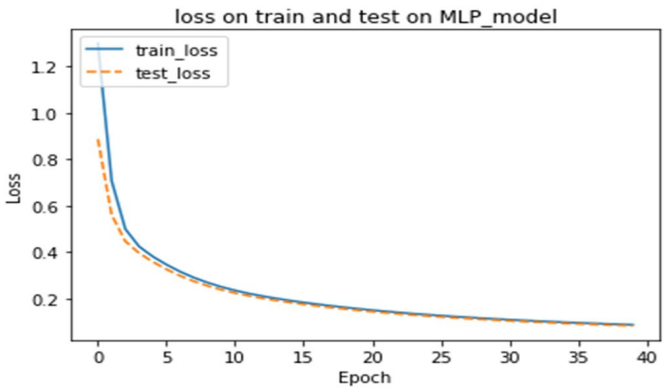

Figure 6.10. MLP model performance on train and test loss Vs epoch

The diagram above shows that the MLP model trains up to 40 epochs, the training loss has reduced from 1.3105 to 0.0822, and the testing loss reduced from 0.8991 to 0.0786. Also, the training accuracy has improved from 0.7151 to 0.9819 and the testing accuracy enhanced from 0.7899 to 0.9833. So, we can conclude that the LSTM model beats the other model's b/s it reduces training _loss by 0.0280 and testing _loss by 0.0193 which is lower than the other models and also improved the training accuracy by 0.99311 and testing accuracy by 0.9957.

6.3. Evaluation

6.3.1. Evaluation of our proposed model

In the above table, we compare six DDoS attack types accuracy, precision, recall, and F-score for four deep learning models. In this case, we made an experiment on four deep learning algorithms. these are RNN, GRU, MLP and LSTM. Based on the evaluation matrices like accuracy, recall, F-score, precision, training_loss, val_loss, training accuracy, and val_accuracy of our proposed models. We conclude that the LSTM model is better for the classification of the given dataset into 6 different attack types. In this experiment of the LSTM model, we get an accuracy of 99.42 which is better than the other three deep learning algorithms. And also, LSTM has reduced the training_loss = 0.028 and the Testing_loss = 0.0193 which is a lower loss rate than the other algorithms. Lastly, we also compare the accuracy which is also better than the others that have Training accuracy = 0.9931 and Testing _accuracy of 0.9942. based on the experiment LSTM have also better precision, recall, and F-score as we described in the table. So, based on the four experiments we can conclude that LSTM have a better classification accuracy than the others.

Algorithm Type	**Attack type**	**Performance matrices**			
		Accuracy (%)	**Precision**	**Recall**	**F1-score**
RNN (with Feature selection)	BENIGN	98.6	0.00	0.00	0.00
	DrDoS_LDAP		1.00	1.00	1.00
	DrDoS_UDP		0.99	1.00	1.00
	SYN		0.97	0.99	0.98
	UDP-lag		0.97	0.98	0.97
	WebDDoS		0.00	0.00	0.00
GRU (with Feature selection)	BENIGN	96.4	0.00	0.00	0.00
	DrDoS_LDAP		1.00	1.00	1.00
	DrDoS_UDP		0.90	0.99	0.94
	SYN		1.00	0.97	0.99
	UDP-lag		0.98	0.91	0.94
	WebDDoS		0.00	0.00	0.00
	BENIGN	98.3	0.96	0.34	0.50

MLP (with Feature selection)	DrDoS_LDAP		0.99	1.00	0.99
	DrDoS_UDP		0.98	0.98	0.98
	SYN		1.00	0.98	0.99
	UDP-lag		0.97	0.98	0.98
	WebDDoS		0.00	0.00	0.00
LSTM with Feature selection (Proposed solution)	BENIGN	99.42	0.78	0.63	0.70
	DrDoS_LDAP		1.00	1.00	1.00
	DrDoS_UDP		0.99	1.00	1.00
	SYN		1.00	0.99	1.00
	UDP-lag		1.00	0.99	0.99
	WebDDoS		0.00	0.00	0.00

Table 6.2. Comparison selective our four proposed Method

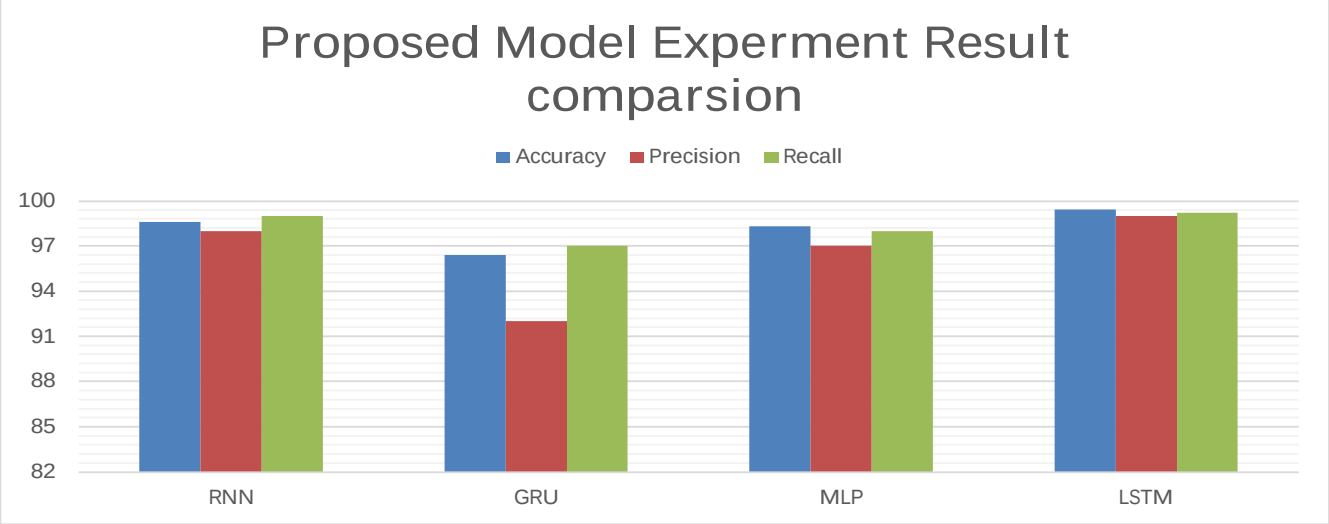

Figure 6.11. Proposed models Experiment results comparison

6.3.2. Performance-based on AUC-ROC metric

The ROC curve measure how accurately the model operates. The ROC curve indicates the relation between two parameters: classes for True and False. The separability between false positive and true positive rates is measured by the area underneath the ROC Curve (AUC). The figures below show that our LSTM_model gives an AUC of 99% for BENGIN, 99% for WEBDDOS, and 1.00 for DrDoS_LDAP, DrDoS_UDP, SYN, and UDP-lag. which means that our proposed model can separate 99% and 1.00 of each DDoS attack separated with positive and negative classes successfully.

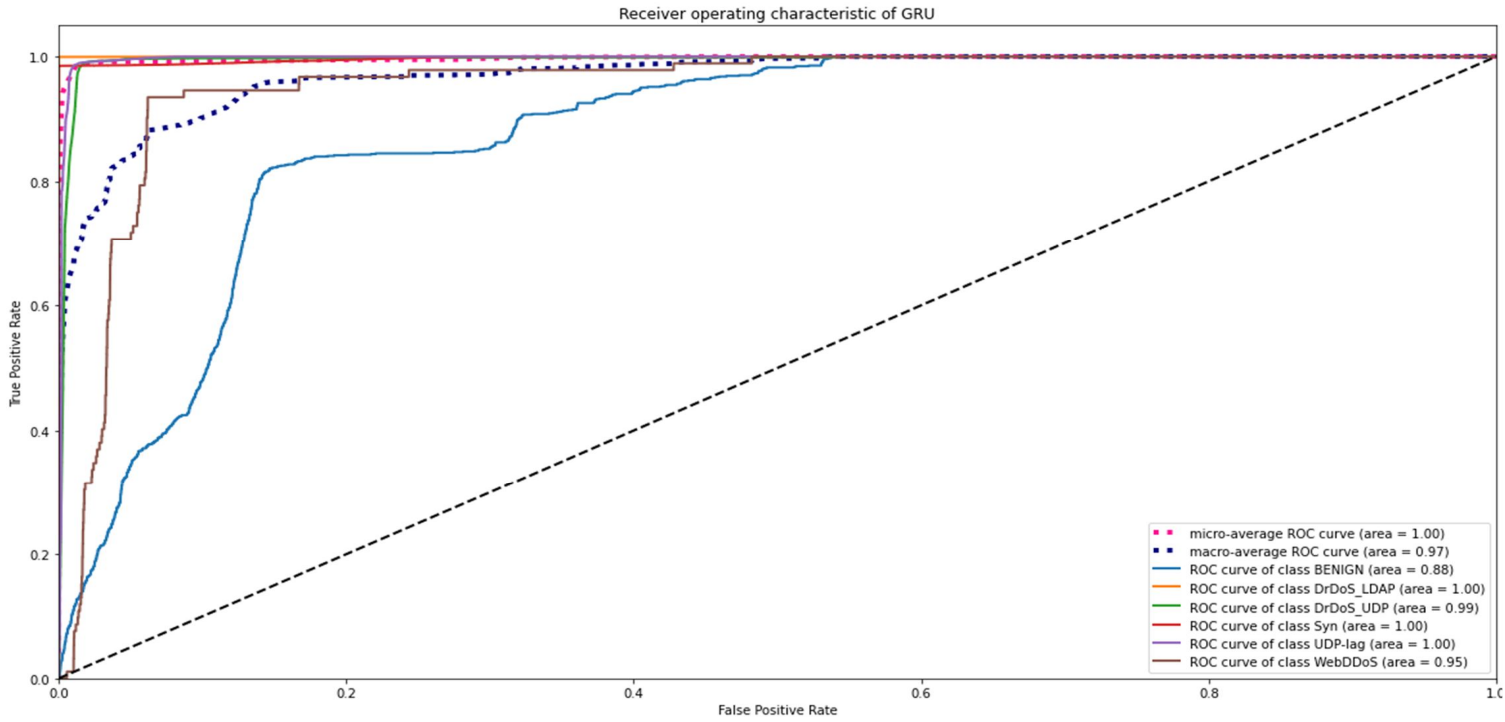

Figure 6.12 ROC for GRU model

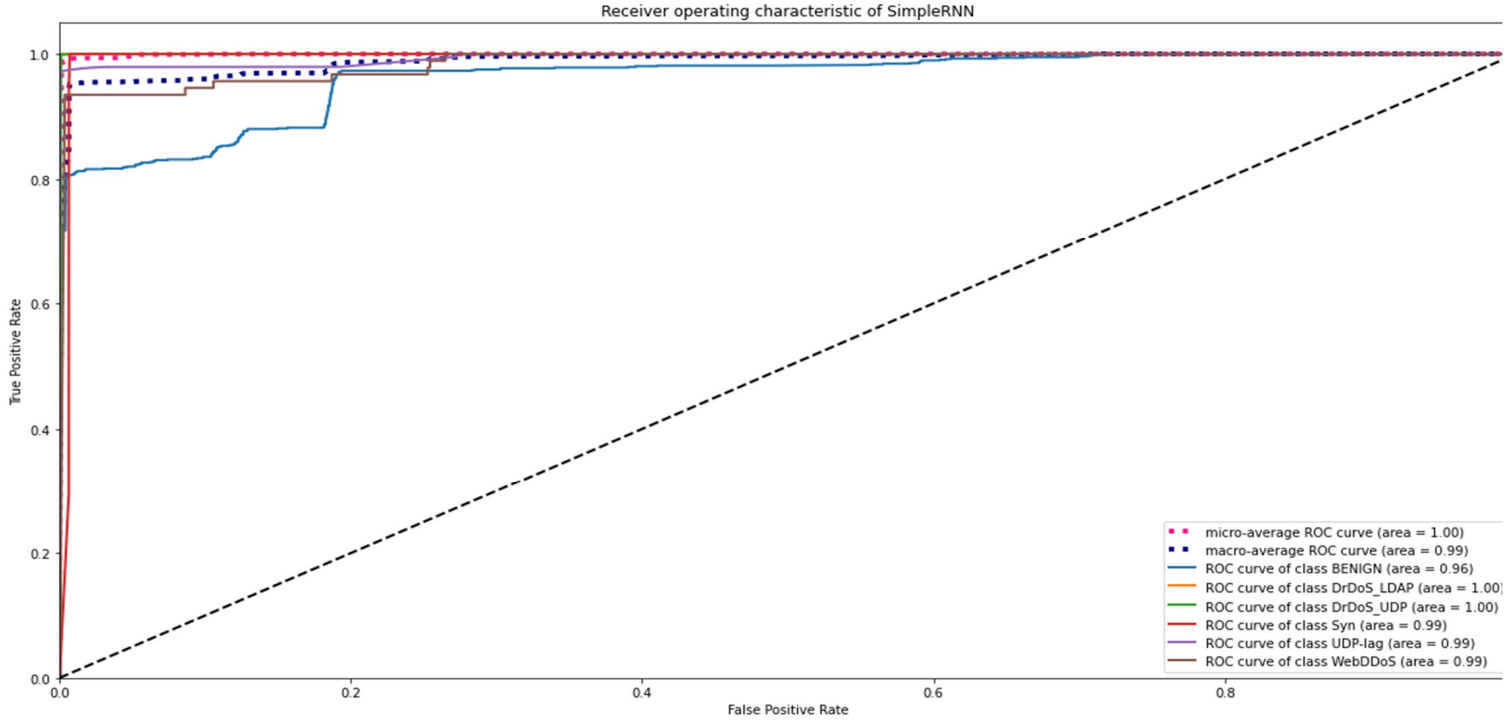

Figure 6.13 ROC for RNN model

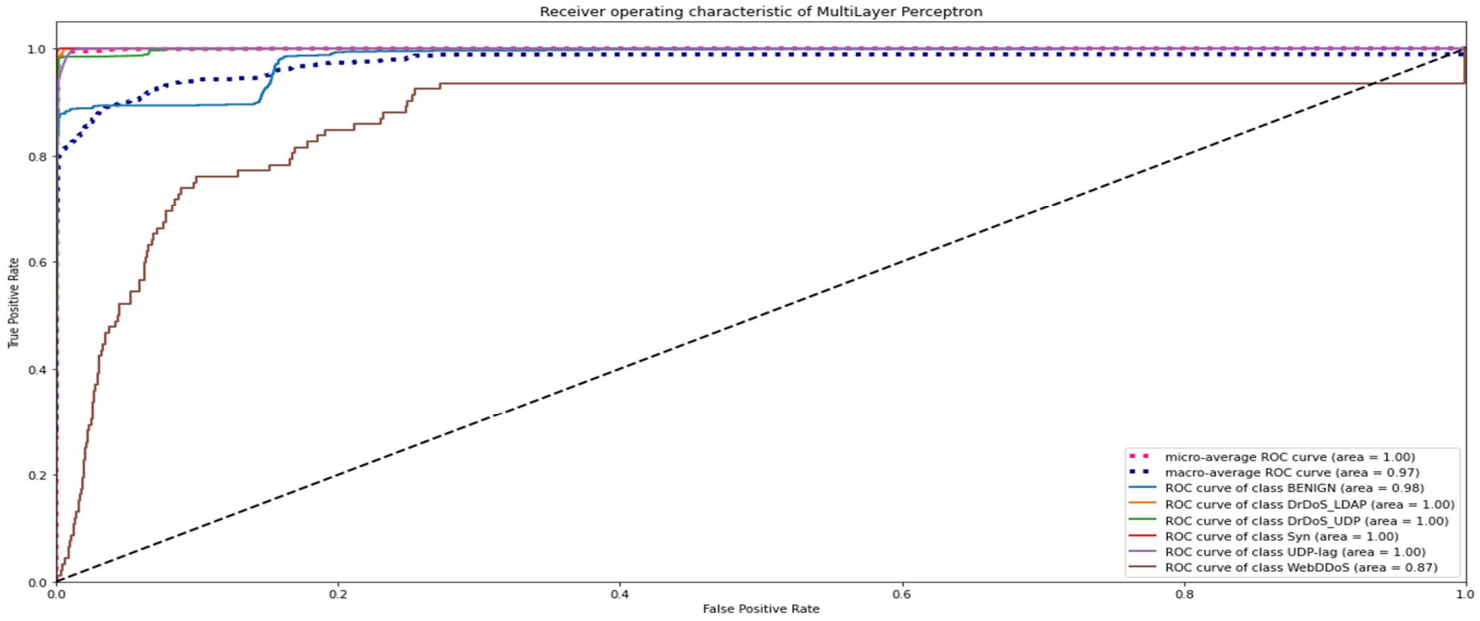

Figure 6.14 ROC for MLP model

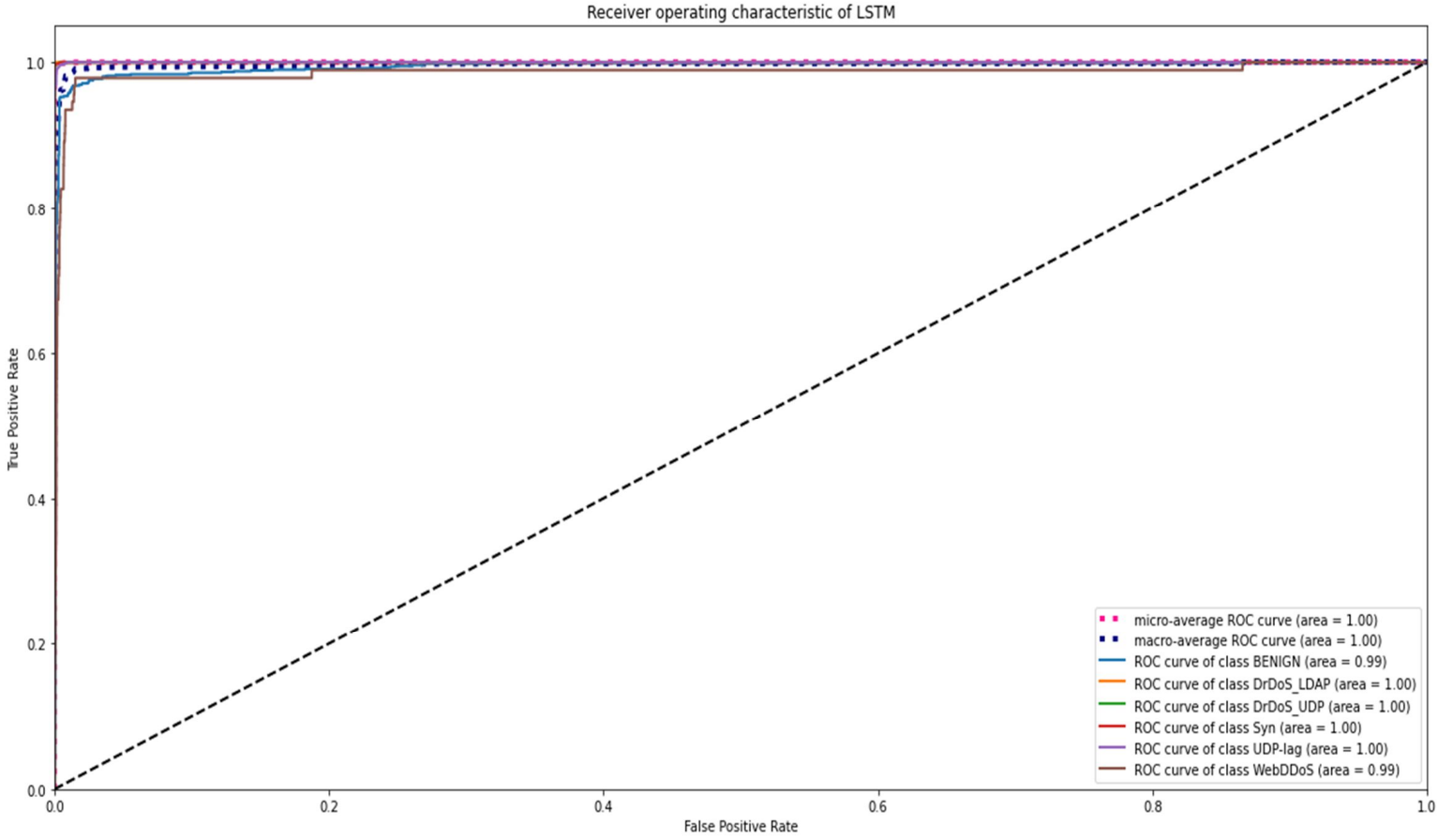

Figure 6.15 ROC for LSTM model

6.3.3. Experiment Result comparison with other related work models

paper	Techniques	Domain	Dataset	Performance			
				Accuracy	**Precision**	**Recall**	**F1-score**
(Elsayed et al., 2020)	(RNN) with AE	IDS	CICDDoS2019	99	99	99	99
(Wei et al., 2021)	MLP with AE	-	CICDDoS2019	98.34	97.91	98.48	98.18
(Alghazzawi et al., 2021)	CNN + BiLSTM		CICDDoS2019	94.52	94.74	92.04	93.44
(Gadze et al., 2021)	RNN LSTM	SDN	CICDDoS2019	89.63	-	-	-
(L. Wang & Liu, 2020)	CNN	SDM	CICDDoS2017	98.98	98.99	98.96	98.97
Our Proposed model	**LSTM With feature selection**	Multi-controller SDN	CICDDoS2019	99.421	99.40	99.464	99.429

Table 6.3 Compared our proposed model with related models

In the above table, we select baseline papers and related papers with the same dataset. So, from the table, we can achieve that our proposed model which is LSTM with feature selection has high accuracy than the other baseline papers and related works.

6.4. Result Discussions on the proposed model

Our experiment conducted outcome shows using the CICDDoS2019 data-set, the entropy-based and deep learning model obtained better results in terms of efficiency and accuracy in detecting and classifying DDoS attacks. In entropy-based detection identify the probability of the attack and send it to the deep learning module. This increases the efficiency of the controller and decreases the deep detection server from overloading. In this deep learning model experiment, we use a feature selection method that is the chi-squared (x2) technique which helps us to focus on the essential and high-weighted features. So, in the experiment RNN, MLP, LSTM, and GRU approach had used. Before training the model, we preprocess the data-set and after that, we use the chi-squared (x2) feature selection method to get high weighted features and then feed it to RNN, MLP, LSTM, and GRU deep learning model. the standard RNN is easier to use and needs less training time.

GRU outperforms LSTM and requires less memory because it has fewer training parameters. While LSTM is capable of learning long-term sequences on a larger sample and is more accurate. However, the outcomes of the detection and classification of DDoS attack using the feature selection method is shown in Table 6.2 when evaluating the qualitative results, we got acceptable results. When compared with the overall feature information in the data-set it leads to the fact that the detection and classification accuracy increase and the error rate decrease of the model using the given data-set. The results of the Detection and classification of our proposed deep learning models explained in Table 6.2 shows that LSTM model has higher accuracy than RNN, MLP, and GRU. Another thing we have noticed is that our proposed model has brought significant improvements compared to baseline and related work models. It enhances by accuracy of 0.421%, which is higher than the RNN-AE model on the CICDDoS2019 data set. Also, it improves by accuracy of 0.44%, which is higher than the CNN model on the ICICDDoS2017 data set.

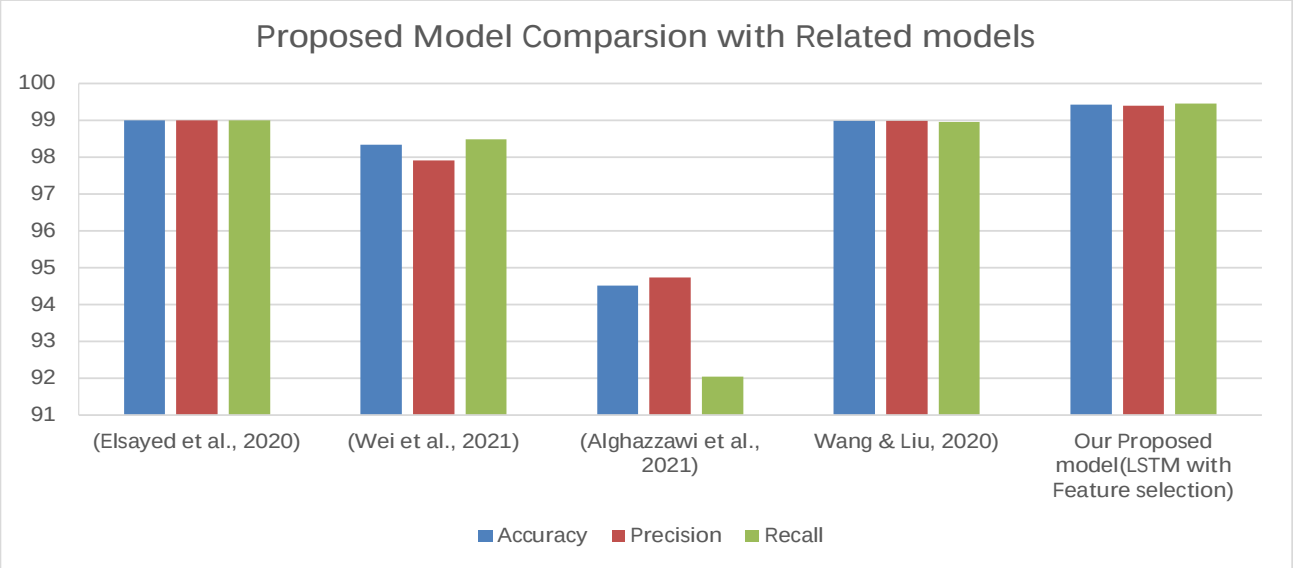

Figure 6.16 The comparison of our proposed model with across different related models

6.5. Research Question Discussion

The research question raises in the first chapter. According to their order in this section, those questions answer as follows:

- How to propose a reliable and effective architecture for better efficiency and accuracy DDoS detecting and classification model in SDN multi-controller?

To increase reliability, we use multi-controller SDN architecture. And to increase efficiency and accuracy we used entropy-based and deep learning models. This entropy based helps us to increase efficiency and deep learning helps us to increase accuracy. The attack comes from the data plane forward to the entropy module controller then this module calculates the probability

of being an attack then if it has a high probability the controller forward to the deep learning module for better accuracy. This entropy based allows the controller to prevent overloading which will increase efficiency and accuracy. These two methods together can improve the efficiency and accuracy of the model.

- How to improve the accuracy and performance of the designed architecture of DDoS attack detection and classification mechanism for multi-controller SDN?

The feature selection method (chi-squared (x2)) provides better accuracy by selecting only essential or high-weighted features from the dataset. Therefore, the LSTM model with the feature selection technique has high accuracy and low error classification. Then this LSTM model was deployed in the deep detection server in our controller architecture. Also, it classifies the attack by categorical classification to specifically know which attack type comes to the controller.

- How effective and accurate is our model for detecting and classification the DDoS attacks using a benchmark dataset by comparing the suggested deep learning models with baseline studies and other deep learning approaches?

In this experiment, we proposed four deep learning models which are RNN, LSTM, GRU, and MLP models with DDoS2019. Experimental outcomes show that feature selection with the LSTM model reached a better DDoS attack detection and classification accuracy. This LSTM model only learns essential or high-weighted features. LSTM model improved by 1.12% which is higher than MLP, 1.82% higher than RNN Model, and 3.02% higher than the GRU model. When we compare LSTM proposed model with baseline model. It enhances by 0.421%, which is higher than the RNN-AE model on the CICDDoS2019 data set. Also, it improves by 0.44%, which is higher than the CNN model on the ICICDDoS2017 data set.

CHAPTER SEVEN

7. CONCLUSION AND FUTURE WORK

7.1. Conclusion

The networking industry and academia have concluded that distributed controller designs are necessary for the future of SDN because centralized systems cannot meet the demands of efficiency, scalability, and availability. And also, DDoS attack detection and classification in multi-controller SDN have significant benefits to the new SDN-based designing Data centers. **Entropy-based** and **Deep learning** is the proposed model for effectively and accurately classifying the attacks. For network traffic, two-level detection is used to ensure greater accuracy and reduced computational complexity at the same time. To ensure great efficiency, the controller runs an initial section based on information entropy. To ensure fine granularity and better accuracy, the deep detection server is employed for the packet-based deep section. The baseline model has three separate gaps addressed by this proposed model. **The first** gap is an irrelevant attribute, classification errors, and huge training time. the researcher uses all types of features in the dataset. this makes the model have misleading data and high training time. it is better to use feature selection to minimize redundant data, misleading data, and less training time. This makes the model increase accuracy and minimizes the error rate. To identify the most weighted features by feature selection and carry out a successful classification, we apply the chi-square (x2) test feature selection technique.

Secondly, the baseline model is limited to Binary classification (attack and normal) which is lack of detailed classification of the attack type. This problem is addressed by categorical classification. This categorical classification allowed to make specific Attack Descriptions in which what type of attack is coming to the controller. **Thirdly**, the baseline model is focused on a centralized controller topology which leads to the SPF. In this situation, the requirements for availability, scalability, and security are not met by this architecture. To avoid a SPF or to increase efficiency, scalability, and availability. in our research, we use a multi-controller which is a logically centralized and physically distributed controller.

Finally, by incorporating the entropy-based and deep learning method we get a model that have a better efficiency and accuracy of DDoS attack detection and classification in multi-controller SDN. In the deep learning attack classification case, we use RNN, LSTM, MLP, and GRU models with a feature selection method. Based on the experiment result from RNN which has an Accuracy of 98.6%, MLP which has an Accuracy of 98.3%, GRU which has an Accuracy of 96.4%, and LSTM which has an Accuracy of 99.42%. The experiment showed that LSTM with feature selection has high accuracy than the other proposed models which is 99.42%. Compare the proposed model with baseline models on different benchmarks dataset with accuracy. Compared with the bassline work without feature selection CNN with

CICDDoS2017 data set which has an accuracy of 98.98%, and RNN-autoencoder with CICDDoS2019 dataset which has an accuracy of 99%. The experiment result shows the effectiveness and accuracy of the proposed model with feature selection have higher accuracy than the baseline papers without feature selection which is 99.42%. This work concludes that logically centralized and physically distributed architecture allows for increased reliability, efficiency, and availability, and also by using the feature selection method we decrease the error classification rate, by using categorical classification we will get a detailed attack type description finally we will have an efficient and accurate classification of the attack types.

7.2. Future Work

To improve a multi-controller network, network researchers and designers will need to address many issues that distributed architectures encounter, such as developing an effective communication process, coming up with a suitable network design, or incorporating new applications into the northbound interface that supports multiple controllers. In this thesis we have proposed securing the controller from DDoS attacks in a logically centralized physically distributed controller. And in this topology, we use the default POX controller load balancing just for experimental purposes. Future multi-controller structures would benefit further from the implementation of logically distributed controllers, which would allow them to combine the benefits of single- and multiple-controller-based systems. The task of controlling the entire environment falls on the controllers in multi-controller structures, whereas it would not fall on a controller in a single controller-based system. And we suggest that if a method for finding a means to find a way to balance the load amongst controllers could be realized, it would improve the performance and security of multi-controller-based systems even further.

In this paper, we used a single data set that is CICDDoS2019. To investigate how different traffic data sets are used in the future, we intend to evaluate how well our proposed model performs across more datasets. And we also used a single feature selection technique, the chi-squared measure, to identify important features. Researching different feature selection methods like autoencoder instead of the chi-squared test. In this thesis, we used a categorical classification for only 6 of the 12 attack types found in the CICDDoS2019 dataset. But each attack class needs to be categorized separately. We plan to enlarge our research categorical classification for the rest of attack type in the dataset. To create a heterogeneous dataset that correctly represents actual internet traffic, we will also simulate the SDN network with a range of parameters and attack traffics. Lastly, we plan working on measuring the effect of computation complexity on the controllers.

REFERENCES

Agrawal, A., Singh, R., Khari, M., Vimal, S., & Lim, S. (2022). Autoencoder for Design of Mitigation Model for DDOS Attacks via M-DBNN. *Wireless Communications and Mobile Computing*, *2022*. https://doi.org/10.1155/2022/9855022

Ahalawat, A., Dash, S. S., Panda, A., & Babu, K. S. (2019). Entropy Based DDoS Detection and Mitigation in OpenFlow Enabled SDN. *Proceedings - International Conference on Vision Towards Emerging Trends in Communication and Networking, ViTECoN 2019*. https://doi.org/10.1109/ViTECoN.2019.8899721

Ahmad, S., & Hussain, A. (2021). Scalability , Consistency , Reliability and Security in SDN Controllers : A Survey of Diverse SDN Controllers. In *Journal of Network and Systems Management* (Vol. 4). Springer US. https://doi.org/10.1007/s10922-020-09575-4

Ahuja, N., & Singal, G. (2019). DDOS Attack Detection Prevention in SDN using OpenFlow Statistics. *Proceedings of the 2019 IEEE 9th International Conference on Advanced Computing, IACC 2019*, 147–152. https://doi.org/10.1109/IACC48062.2019.8971596

Alghazzawi, D., Bamasaq, O., Ullah, H., & Asghar, M. Z. (2021). Efficient detection of DDoS attacks using a hybrid deep learning model with improved feature selection. *Applied Sciences (Switzerland)*, *11*(24). https://doi.org/10.3390/app112411634

Alshra'A, A. S., Farhat, A., & Seitz, J. (2021). Deep Learning Algorithms for Detecting Denial of Service Attacks in Software-Defined Networks. *Procedia Computer Science*, *191*(2019), 254–263. https://doi.org/10.1016/j.procs.2021.07.032

Banitalebi, A., & Mohammadreza, D. (2020). The DDoS attacks detection through machine learning and statistical methods in SDN. In *The Journal of Supercomputing* (Issue 0123456789). Springer US. https://doi.org/10.1007/s11227-020-03323-w

Bannour, F., Souihi, S., & Mellouk, A. (2018). Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Communications Surveys and Tutorials*, *20*(1), 333–354. https://doi.org/10.1109/COMST.2017.2782482

Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. *Security and Communication Networks*, *9*(18), 5803–5833. https://doi.org/10.1002/sec.1737

Bouet, M. (n.d.). *DISCO : Distributed Multi-domain SDN Controllers*.

Cui, J., Zhang, J., He, J., Zhong, H., & Lu, Y. (2020). DDoS detection and defense mechanism for SDN controllers with K-Means. *Proceedings - 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing, UCC 2020*, 394–401. https://doi.org/10.1109/UCC48980.2020.00062

Dallou, J., & Al-duwairi, B. (2020). *ADAPTIVE ENTROPY-BASED DETECTION AND MITIGATION OF DDOS ADAPTIVE ENTROPY-BASED DETECTION AND MITIGATION OF DDOS*. *September*.

Deepa, V., Muthamil Sudar, K., & Deepalakshmi, P. (2018). Detection of DDoS attack on

SDN control plane using hybrid machine learning techniques. *Proceedings of the International Conference on Smart Systems and Inventive Technology, ICSSIT 2018*, *Icssit*, 299–303. https://doi.org/10.1109/ICSSIT.2018.8748836

Elsayed, M. S., Le-Khac, N. A., Dev, S., & Jurcut, A. D. (2020). DDoSNet: A Deep-Learning Model for Detecting Network Attacks. *Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020*, 391–396. https://doi.org/10.1109/WoWMoM49955.2020.00072

Etaiwi, W., Biltawi, M., & Almajali, S. (2017). Securing distributed SDN controllers against dos attacks. *Proceedings - 2017 International Conference on New Trends in Computing Sciences, ICTCS 2017*, *2018-Janua*, 203–206. https://doi.org/10.1109/ICTCS.2017.52

Fawcett, L., Scott-Hayward, S., Broadbent, M., Wright, A., & Race, N. (2018). Tennison: A distributed SDN framework for scalable network security. *IEEE Journal on Selected Areas in Communications*, *36*(12), 2805–2818. https://doi.org/10.1109/JSAC.2018.2871313

Gadze, J. D., Bamfo-asante, A. A., Agyemang, J. O., & Nunoo-mensah, H. (2021). *An Investigation into the Application of Deep Learning in the Detection and Mitigation of DDOS Attack on SDN Controllers*. 1–22.

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018). Multi-controller Based Software-Defined Networking: A Survey. *IEEE Access*, *6*(c), 15980–15996. https://doi.org/10.1109/ACCESS.2018.2814738

Jayakumar, S. S. S. K. V. (2020). A Study on Various Attacks and Detection Methodologies in Software Defined Networks. *Wireless Personal Communications*, *114*(1), 675–697. https://doi.org/10.1007/s11277-020-07387-y

Jo, M. M., & Park, Y. (2018). *[1] M. M. Jo and Y. Park, “cte d Au tho r P roo f Un co rre rre Un roo f co d Au tho,” pp. 1–13, 2018, doi: 10.3233/JIFS-169833.cte d Au tho r P roo f Un co rre rre Un roo f co d Au tho*. 1–13. https://doi.org/10.3233/JIFS-169833

Kansal, V., & Dave, M. (2017). *Proactive DDoS Attack Detection and Isolation*.

Kavitha, D., Ramalakshmi, R., & Murugeswari, R. (2019). The Detection and Mitigation of Distributed Denial-of-Service (DDOS) Attacks in Software Defined Networks using Distributed Controllers. *2019 International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development, INCCES 2019*, *i*. https://doi.org/10.1109/INCCES47820.2019.9167698

Krishnan, P., & Najeem, J. S. (2019). A review of security, threats and mitigation approaches for SDN architecture. *International Journal of Innovative Technology and Exploring Engineering*, *8*(5), 389–393.

Latah, M., & Toker, L. (2020). Minimizing false positive rate for DoS attack detection: A hybrid SDN-based approach. *ICT Express*, *6*(2), 125–127. https://doi.org/10.1016/j.icte.2019.11.002

Lawal, B. H., & Nuray, A. T. (2018). Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN). *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018*, 1–4. https://doi.org/10.1109/SIU.2018.8404674

Mehr, S. Y., & Ramamurthy, B. (n.d.). *An SVM Based DDoS Attack Detection Method for Ryu SDN Controller*. 72–73.

Mhamdi, L., McLernon, D., El-Moussa, F., Raza Zaidi, S. A., Ghogho, M., & Tang, T. (2020). A Deep Learning Approach Combining Autoencoder with One-class SVM for DDoS Attack Detection in SDNs. *2020 8th International Conference on Communications and Networking, ComNet2020 - Proceedings*. https://doi.org/10.1109/ComNet47917.2020.9306073

Mittal, M., Kumar, K., & Behal, S. (2022). Deep learning approaches for detecting DDoS attacks: a systematic review. *Soft Computing*. https://doi.org/10.1007/s00500-021-06608-1

Mohsin, M. A., & Hamad, A. H. (2022). Implementation of Entropy-Based DDoS Attack Detection Method in Different SDN Topologies. *American Academic Scientific Research Journal for Engineering*, *86*, 63–76. http://asrjetsjournal.org/

Mousavi, S. M., & St-hilaire, M. (2018). Early Detection of DDoS Attacks Against Software. *Journal of Network and Systems Management*, *26*(3), 573–591. https://doi.org/10.1007/s10922-017-9432-1

Mousavi, S. M., & St-Hilaire, M. (2015). Early detection of DDoS attacks against SDN controllers. *2015 International Conference on Computing, Networking and Communications, ICNC 2015*, 77–81. https://doi.org/10.1109/ICCNC.2015.7069319

Niu, K., Jiao, H., Gao, Z., Jia, G., Yang, G., & Cheng, C. (2018). A developed feature selection method for classification based on united information gain. *2017 IEEE SmartWorld Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017 - *, 1–4. https://doi.org/10.1109/UIC-ATC.2017.8397477

Nugraha, B., & Murthy, R. N. (2020). Deep Learning-based Slow DDoS Attack Detection in SDN-based Networks. *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2020 - Proceedings*, 51–56. https://doi.org/10.1109/NFV-SDN50289.2020.9289894

Paliwal, M., Shrimankar, D., & Tembhurne, O. (2018). Controllers in SDN: A review report. *IEEE Access*, *6*(March 2011), 36256–36270. https://doi.org/10.1109/ACCESS.2018.2846236

Pandikumar, T., Atkilt, F., Hassen, C. A., & Tech Student, M. (2017). Early Detection of

DDoS Attacks in a Multi-Controller Based SDN. *International Journal of Engineering Science and Computing*, *7*(6), 13422–13429. http://ijesc.org/

Pei, J., Chen, Y., & Ji, W. (2019). *A DDoS Attack Detection Method Based on Machine Learning A DDoS Attack Detection Method Based on Machine Learning*. https://doi.org/10.1088/1742-6596/1237/3/032040

Ravi, N., Shalinie, S. M., & Danyson Jose Theres, D. (2020). BALANCE: Link Flooding Attack Detection and Mitigation via Hybrid-SDN. *IEEE Transactions on Network and Service Management*, *17*(3), 1715–1729. https://doi.org/10.1109/TNSM.2020.2997734

Redekar, P., & Chatterjee, M. (2017). *Hybrid Technique for DDoS Attack Detection*. *8*(3), 377–379.

Sagar, K., Sanjaya, S., Panda, K., & Sahoo, S. (2019). Toward secure software - defined networks against distributed denial of service attack. In *The Journal of Supercomputing* (Vol. 75, Issue 8). Springer US. https://doi.org/10.1007/s11227-019-02767-z

Scott-Hayward, S., O'Callaghan, G., & Sezer, S. (2013). SDN security: A survey. *SDN4FNS 2013 - 2013 Workshop on Software Defined Networks for Future Networks and Services*. https://doi.org/10.1109/SDN4FNS.2013.6702553

Sebbar, A., Boulmalf, M., Dafir Ech-Cherif El Kettani, M., & Badd, Y. (2018). Detection MITM Attack in Multi-SDN Controller. *Colloquium in Information Science and Technology, CIST*, *2018-Octob*, 583–587. https://doi.org/10.1109/CIST.2018.8596479

Tadros, C. N., Mokhtar, B., & Rizk, M. R. M. (2019). Logically Centralized-Physically Distributed Software Defined Network Controller Architecture. *2018 IEEE Global Conference on Internet of Things, GCIoT 2018*, *December*. https://doi.org/10.1109/GCIoT.2018.8620166

Tan, L., Pan, Y., Wu, J., Zhou, J., & Jiang, H. (2020). *A New Framework for DDoS Attack Detection and Defense in SDN Environment*. *XX*. https://doi.org/10.1109/ACCESS.2020.3021435

Tayfour, O. E., & Marsono, M. N. (2021). Collaborative detection and mitigation of DDoS in software-defined networks. *Journal of Supercomputing*, *77*(11), 13166–13190. https://doi.org/10.1007/s11227-021-03782-9

Tonkal, Ö., Polat, H., Başaran, E., Cömert, Z., & Kocaoğlu, R. (2021). Machine learning approach equipped with neighbourhood component analysis for ddos attack detection in software-defined networking. *Electronics (Switzerland)*, *10*(11). https://doi.org/10.3390/electronics10111227

Wang, H. Z., Zhang, P., Xiong, L., Liu, X., & Hu, C. C. (2016). A secure and high-performancemulti-controller architecture for software-defined networking. *Frontiers of Information Technology and Electronic Engineering*, *17*(7), 634–646. https://doi.org/10.1631/FITEE.1500321

Wang, J., Shou, G., Hu, Y., & Guo, Z. (2017). A multi-domain SDN scalability architecture

implementation based on the coordinate controller. *Proceedings - 2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2016*, 494–499. https://doi.org/10.1109/CyberC.2016.100

Wang, L., & Liu, Y. (2020). *A DDoS Attack Detection Method Based on Information Entropy and Deep Learning in SDN*. *Itnec*, 1084–1088.

Wang, R., Jia, Z., & Ju, L. (2015). An entropy-based distributed DDoS detection mechanism in software-defined networking. *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*, *1*, 310–317. https://doi.org/10.1109/Trustcom.2015.389

Wang, T., Feng, Y., & Sakurai, K. (2021). Improving the Two-stage Detection of Cyberattacks in SDN Environment Using Dynamic Thresholding. *Proceedings of the 2021 15th International Conference on Ubiquitous Information Management and Communication, IMCOM 2021*. https://doi.org/10.1109/IMCOM51814.2021.9377395

Wei, Y., Jang-Jaccard, J., Sabrina, F., Singh, A., Xu, W., & Camtepe, S. (2021). AE-MLP: A Hybrid Deep Learning Approach for DDoS Detection and Classification. *IEEE Access*, *9*, 146810–146821. https://doi.org/10.1109/ACCESS.2021.3123791

Xu, Y., Yan, Y., Dai, Z., & Wang, X. (2014). A management model for SDN-based data center networks. *Proceedings - IEEE INFOCOM*, 113–114. https://doi.org/10.1109/INFCOMW.2014.6849181

Yu, S., Zhang, J., Liu, J., Zhang, X., Li, Y., & Xu, T. (2021). A cooperative DDoS attack detection scheme based on entropy and ensemble learning in SDN. *Eurasip Journal on Wireless Communications and Networking*, *2021*(1). https://doi.org/10.1186/s13638-021-01957-9

Yue, M., Wang, H., Liu, L., & Wu, Z. (2020). Detecting DoS Attacks Based on Multi-Features in SDN. *IEEE Access*, *8*, 104688–104700. https://doi.org/10.1109/ACCESS.2020.2999668

Zakaria, N., Jawwad, B., & Khaled, A. S. (2017). *DDoS Attack Detection and Mitigation Using SDN : Methods , Practices , and DDoS Attack Detection and Mitigation Using SDN : Methods , Practices , and Solutions*. *February*. https://doi.org/10.1007/s13369-017-2414-5

Zubaydi, H. D., Anbar, M., & Wey, C. Y. (2017). Review on Detection Techniques against DDoS Attacks on a Software-Defined Networking Controller. *Proceedings - 2017 Palestinian International Conference on Information and Communication Technology, PICICT 2017,* 10–16. https://doi.org/10.1109/PICICT.2017.26

APPENDIXES

Appendix A: Sample code for attack generation from host

```
import sys
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange
import time
def sourceIPgen():

  not_valid = [10,127,254,255,1,2,169,172,192]

  first = randrange(1,256)

  while first in not_valid:
    first = randrange(1,256)
    print first
  ip = ".".join([str(first),str(randrange(1,256)), str(randrange(1,256)),str(randrange(1,256))])
  print ip
  return ip

def main():
  for i in range (1,5):
    mymain()
    time.sleep (10)
#send the generated IPs
def mymain():
#getting the ip address to send attack packets
  dstIP = sys.argv[1:]
  print dstIP
  src_port = 80
  dst_port = 1

# open interface eth0 to send packets
  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

  for i in xrange(0,500):
# form the packet
    packets = Ether()/IP(dst=dstIP,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
    print(repr(packets))

# send packet with the defined interval (seconds)
    sendp( packets,iface=interface.rstrip(),inter=0.025)
#main

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

Appendix B: Sample code for normal traffic generation from host one

```
import sys
import getopt
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange


def sourceIPgen():
    not_valid = [10,127,254,1,2,169,172,192]

    first = randrange(1,256)

    while first in not_valid:
        first = randrange(1,256)

    ip = ".".join([str(first),str(randrange(1,256)),str(randrange(1,256)),str(randrange(1,256))])

    return ip


def gendest(start, end):

    first = 10
    second =0; third =0;
    ip = ".".join([str(first),str(second),str(third),str(randrange(start,end))])
   # print start
   # print end
    return ip

#if __name__ == '__main__':
  #main()

def main(argv):
   # global start
   # global end
    print argv
    try:
        opts, args = getopt.getopt(sys.argv[1:],'s:e:',['start=','end='])
    except getopt.GetoptError:
        sys.exit(2)
    for opt, arg in opts:
        if opt =='-s':
            start = int(arg)
        elif opt =='-e':
            end = int(arg)
    if start == '':
        sys.exit()
    if end == '':
        sys.exit()


    interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

    for i in xrange(1000):
        packets = Ether()/IP(dst=gendest(start, end),src=sourceIPgen())/UDP(dport=80,sport=2)
        print(repr(packets))

        sendp( packets,iface=interface.rstrip(),inter=0.1)

if __name__ == '__main__':
  main(sys.argv)
```

Appendix C: Sample code for entropy detection code from the host

```
import math

from pox.core import core

log = core.getLogger()

class Entropy(object):
        count = 0
        entDic = {}
        ipList = []
        dstEnt = []
        value = 1

        def statcolect(self, element):

                #print "Self values"
                #print "count is " + str(self.count)
                #print "Length of IP list is"
                #print len(self.ipList)
                #print "**********"
                l = 0
                self.count +=1
                self.ipList.append(element)
                if self.count == 50:
                        for i in self.ipList:
                                l +=1
                                if i not in self.entDic:
                                        self.entDic[i] =0
                                self.entDic[i] +=1
                        self.entropy(self.entDic)
                        log.info(self.entDic)
                        self.entDic = {}
                        self.ipList = []
                        l = 0
                        self.count = 0

        def entropy (self, lists):
                #print "Entropy called"
                l = 50
                elist = []
                for k,p in lists.items():
                        '''
                        log.info("p is")
                        log.info(p)
                        log.info("P is obtained from")
                        log.info(k)
                        log.info("l is")
                        log.info(l)
                        '''
                        c = p/float(l)
                        #log.info("Value of c is ")
                        #log.info(c)
                        c = abs(c)
                        elist.append(-c * math.log(c, 10))
                        log.info('Entropy = ')
                        log.info(sum(elist))
                        #log.info("****")
                        self.dstEnt.append(sum(elist))
                if(len(self.dstEnt)) == 80:
                        print self.dstEnt
                        self.dstEnt = []
                self.value = sum(elist)

        def __init__(self):
                pass
```

Appendix D: Sample code for L3. learning code from the host

```
A stupid L3 switch

For each switch:
1) Keep a table that maps IP addresses to MAC addresses and switch ports.
   Stock this table using information from ARP and IP packets.
2) When you see an ARP query, try to answer it using information in the table
   from step 1.  If the info in the table is old, just flood the query.
3) Flood all other ARPs.
4) When you see an IP packet, if you know the destination port (because it's
   in the table from step 1), install a flow for it.
"""
import datetime
from pox.core import core
import pox

from pox.lib.packet.ethernet import ethernet, ETHER_BROADCAST
from pox.lib.packet.ipv4 import ipv4
from pox.lib.packet.arp import arp
from pox.lib.addresses import IPAddr, EthAddr
from pox.lib.util import str_to_bool, dpid_to_str
from pox.lib.recoco import Timer

import pox.openflow.libopenflow_01 as of

from pox.lib.revent import *
import itertools
import time
#editing

from .detection import Entropy
diction = {}
ent_obj = Entropy()
set_Timer = False
defendDDOS=False
#blockPort=""
#end of editing
log = core.getLogger()
# Timeout for flows
FLOW_IDLE_TIMEOUT = 10

# Timeout for ARP entries
ARP_TIMEOUT = 60 * 2

# Maximum number of packet to buffer on a switch for an unknown IP
MAX_BUFFERED_PER_IP = 5

# Maximum time to hang on to a buffer for an unknown IP in seconds
MAX_BUFFER_TIME = 5


class Entry (object):
  """
  Not strictly an ARP entry.
  We use the port to determine which port to forward traffic out of.
  We use the MAC to answer ARP replies.
  We use the timeout so that if an entry is older than ARP_TIMEOUT, we
   flood the ARP request rather than try to answer it ourselves.
  """
  def __init__ (self, port, mac):
    self.timeout = time.time() + ARP_TIMEOUT
    self.port = port
    self.mac = mac

  def __eq__ (self, other):
    if type(other) == tuple:
      return (self.port,self.mac)==other
    else:
      return (self.port,self.mac)==(other.port,other.mac)
  def __ne__ (self, other):
    return not self.__eq__(other)

  def isExpired (self):
    if self.port == of.OFPP_NONE: return False
    return time.time() > self.timeout


def dpid_to_mac (dpid):
  return EthAddr("%012x" % (dpid & 0xffffffffffff,))


class l3_switch (EventMixin):
  def __init__ (self, fakeways = [], arp_for_unknowns = False, wide = False):
    # These are "fake gateways" -- we'll answer ARPs for them with MAC
    # of the switch they're connected to.
    self.fakeways = set(fakeways)

    # If True, we create "wide" matches.  Otherwise, we create "narrow"
    # (exact) matches.
    self.wide = wide

    # If this is true and we see a packet for an unknown
    # host, we'll ARP for it.
    self.arp_for_unknowns = arp_for_unknowns
```

```
global defendDDOS
global blockPort
timerSet =False
global diction
def preventing():
  global diction
  global set_Timer
  if not set_Timer:
    set_Timer =True
  #Timer(1, _timer_func(), recurring=True)


  #print"\n\n*********new packetIN************"
  if len(diction) == 0:
    print("Enpty diction ",str(event.connection.dpid), str(event.port))
    diction[event.connection.dpid] = {}
    diction[event.connection.dpid][event.port] = 1
  elif event.connection.dpid not in diction:
    diction[event.connection.dpid] = {}
    diction[event.connection.dpid][event.port] = 1
    #print "ERROR"
  else:
    if event.connection.dpid in diction:
  # temp = diction[event.connection.dpid]
  #print(temp)
  #print "error check " , str(diction[event.connection.dpid][event.port])
      if event.port in diction[event.connection.dpid]:
        temp_count=0
        temp_count =diction[event.connection.dpid][event.port]
        temp_count = temp_count+1
        diction[event.connection.dpid][event.port]=temp_count
        #print "printting dpid port number and its packet count: ",  str(event.connection.dpid), str(diction[event.connection.
      else:
        diction[event.connection.dpid][event.port] = 1

  print "\n",datetime.datetime.now(), ": printing diction ",str(diction),"\n"


def _timer_func ():
  global diction
  global set_Timer
  if set_Timer==True:
    #print datetime.datetime.now(),": calling timer fucntion now!!!!!"
    for k,v in diction.iteritems():
      for i,j in v.iteritems():
        if j >=50:
          print "_____________________________________________________________________________________________________"
          print "\n",datetime.datetime.now(),"*******    DDOS DETECTED    ********"
          print "\n",str(diction)
          print "\n",datetime.datetime.now(),": BLOCKED PORT NUMBER  : ", str(i), " OF SWITCH ID: ", str(k)
          print "\n_____________________________________________________________________________________________________"

          #self.dropDDOS ()
          dpid = k
          msg = of.ofp_packet_out(in_port=i)
          #msg.priority=42
          #msg.in_port = event.port
          #po = of.ofp_packet_out(buffer_id = buffer_id, in_port = in_port)
          core.openflow.sendToDPID(dpid,msg)


  diction={}

if not packet.parsed:
  log.warning("%i %i ignoring unparsed packet", dpid, inport)
  return

if dpid not in self.arpTable:
  # New switch -- create an empty table
  self.arpTable[dpid] = {}
  for fake in self.fakeways:
    self.arpTable[dpid][IPAddr(fake)] = Entry(of.OFPP_NONE,
     dpid_to_mac(dpid))

if packet.type == ethernet.LLDP_TYPE:
  # Ignore LLDP packets
  return

if isinstance(packet.next, ipv4):
  log.debug("%i %i IP %s => %s", dpid,inport,
            packet.next.srcip,packet.next.dstip)
  ent_obj.statcolect(event.parsed.next.dstip)#editing
  print "\n***** Entropy Value = ",str(ent_obj.value),"*****\n"
  if ent_obj.value <0.5:
    preventing()
    if timerSet is not True:
      Timer(2, _timer_func, recurring=True)
      timerSet=False
  else:
    timerSet=False
```

Appendix E: Sample CICDDoS2019 dataset

Init_Win_	act_data_	min_seg_	Active Me	Active Std	Active Ma	Active Mi	Idle Mean	Idle Std	Idle Max	Idle Min	SimillarHT	Inbound	Label
-1	0	0	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP
-1	1	-1	0	0	0	0	0	0	0	0	0	1	DrDoS_LDAP

Appendix F: Sample code for feature selection and categorical classification

▾ Feature Selection

+ Code + Text

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier(random_state=42)
model.fit(data_X, data_y_trans)
```

```
[ ] model.feature_importances_
```

```
[ ] feature_importance_std = pd.Series(model.feature_importances_, index=data_X.columns)
    feature_importance_std.nlargest(20).plot(kind='bar', title='Standardised Dataset Feature Selection using ExtraTreesClassifier')
```

```
[ ] data_X.shape
```

```
[ ] #data_new_20features_X = data_X[[' Timestamp', ' Source Port', ' Min Packet Length', ' Fwd Packet Length Min', 'Flow ID', ' Packet Le
    data_new_20features_X = data_X[[' Average Packet Size', ' Min Packet Length',' Packet Length Mean', ' ACK Flag Count', ' Avg Fwd Segm
```

```
[ ] y_train_MLP_20 = np.array(y_train_20)
    y_test_MLP_20 = np.array(y_test_20)

    y_train_MLP_onehot_20 = to_categorical(y_train_MLP_20)
    y_test_MLP_onehot_20 = to_categorical(y_test_MLP_20)

    X_train_20_MLP = np.array(X_train_std_20)
    X_test_20_MLP = np.array(X_test_std_20)
```

```
[ ]
    y_train_MLP_onehot_20
```

```
y_test_MLP_onehot_20
```

Appendix G: Sample code for the LSTM model

```
X_test_std_20
```

```
X_train_lstm_20.shape[0]
```

```
X_train_lstm_reshape = np.reshape(X_train_std_20, (X_train_lstm_20.shape[0], 1, X_train_lstm_20.shape[1]))
X_test_lstm_reshape = np.reshape(X_test_std_20, (X_test_lstm_20.shape[0], 1, X_test_lstm_20.shape[1]))
```

```
batch_size = 1000

# Initialize the network
model_LSTM = Sequential()
model_LSTM.add(LSTM(3,input_dim=20, return_sequences=True))
model_LSTM.add(Dropout(0.001))
model_LSTM.add(LSTM(2,input_dim=20, return_sequences=False))
model_LSTM.add(Dropout(0.001))
model_LSTM.add(Dense(6))
model_LSTM.add(Activation('softmax'))
```

```
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',
      restore_best_weights=True)
```

```
model_LSTM.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
history_LSTM= model_LSTM.fit(X_train_lstm_reshape, y_train_onehot_lstm, validation_data=(X_test_lstm_reshape, y_test_one_ho
```