
**SCHOOL OF GRADUATE STUDIES**

# ENHANCING SECURITY IN SOFTWARE DEFINED NETWORKING USING DEEP LEARNING FOR DETECTION AND MITIGATION OF DISTRIBUTED DENIAL OF SERVICE ATTACKS

**MSc. THESIS**

**SIRAJ AHMED YASSIN**

**APRIL, 2024**

**WOLKITE, ETHIOPIA**

**Wolkite University**

**School of Graduate Studies**

# Enhancing Security in Software Defined Networking using Deep Learning for Detection and Mitigation of Distributed Denial of Service Attacks

**A Thesis Submitted to School of Graduate Studies, in Partial Fulfilment of the Requirements for the Degree of Master of Computer Science and Engineering (Specialization: Computer science)**

**Siraj Ahmed Yassin**

**Major Advisor: Mesfin Abebe (PH.D.)**

**APRIL, 2024**

**Wolkite, Ethiopia**

APPROVAL SHEET

**SCHOOL OF GRADUATE STUDIES**

**WOLKITE UNIVERSITY**

**Enhancing Security in Software Defined Networking using Deep Learning for Detection and Mitigation of Distributed Denial of Service Attacks**

Submitted by:

*Siraj Ahmed* _____ _____

Name of Student Signature Date

Approved by:

1. Dr. MESFIN ABEBE _____ 25/04/24

   Name of Major Advisor Signature Date

2. _____ _____ _____

   Name of Co-Advisor Signature Date

3. _____ _____

   _____

   Name of Chairman, DGC Signature Date

4. _____ _____ _____

   Name of Dean, SGS Signature Date

**WOLKITE UNIVERSITY**

**SCHOOL OF GRADUATE STUDIES**

We hereby certify that we have read and evaluated this Thesis titled " **Enhancing Security in Software Defined Networking using Deep Learning for Detection and Mitigation of Distributed Denial of Service Attacks** prepared under our guidance by Siraj Ahmed Yassin. We recommend that the Thesis shall be submitted as fulfilling the requirements for the award of a MSc. degree in Computer Science and Engineering.

| | | |
|---|---|---|
| DR. MESFIN ABEBE | | 27/05/24 |
| Major Advisor | Signature | Date |
| | | |
| Co-Advisor | Signature | Date |

As members of the Board of Examiners of the Master of Science Thesis open defense examination, we have read and evaluated this Thesis prepared by Siraj Ahmed Yassin and examined the candidate. we hereby certify that, the thesis is accepted for fulfilling the requirements for the award of the degree of Master of Science (M.Sc.) in Computer Science and Engineering.

| | | |
|---|---|---|
| 1. Baye Y. (Ph.D.) | | 05/27/2024 |
| Name of External Examiner | Signature | Date |
| 2. | | |
| Name of Internal Examiner | Signature | Date |
| 3. | | |
| Name of Chairman | Signature | Date |

Final approval and acceptance of the Thesis is contingent upon the submission of its final copy to the Council of Postgraduate Program (CPGS) through the candidate's department or school graduate committee (DGC or SGC).

# DECLARATION

By my signature below, I declare and affirm that this thesis is my own work. I have adhered to all ethical principles in the preparation, data collection, data analysis, and completion of this thesis. All academic content included in this thesis has been properly acknowledged through citation. I affirm that all sources used in this document have been cited and referenced. Every effort has been made to avoid plagiarism in the preparation of this thesis.

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master from the School of Graduate Studies at Wolkite University. The thesis will be deposited in the Wolkite University Library to be made available to borrowers under the library's rules. I solemnly declare that this thesis has not been submitted to any other institution for the award of any academic degree, diploma, or certificate.

Brief excerpts from this thesis may be used without special permission, provided that full and accurate credit is given to the source. Requests for permission to quote extensively from, or reproduce this thesis in whole or in part, may be granted by the Department of Graduate Studies if it deems the use of the material to be in the interest of academic integrity. For all other cases, permission must be obtained from the author.

_____ _____

*Siraj Ahmed Yassin* *Date*

# ACKNOWLEDGMENT

I would like to express my sincere gratitude to **Allah** for giving me the strength and guidance to complete this research journey successfully, also I would like to acknowledge the invaluable advice and direction of my advisor, **Dr. Mesfin Abebe**, who supervised my work and helped me improve my research skills. He directed me to think critically and creatively, and gave me constructive feedback and suggestions that enhanced the quality of this research.

I am grateful for the substantial support from **CloudLab.us**, which granted me access to extensive computing resources over several months. This was instrumental in conducting my resource-intensive experiments and achieving my research goals.

I am deeply thankful to my wife, **Semira Behiru**, who supported me every day and encouraged me when I felt tired after many sleepless nights and challenges. She is my source of motivation and inspiration, and I would like to thank my children, who have been sources of joy and inspiration for me throughout this research. Even though they may not understand what I was doing, their smiles and hugs always cheered me up and gave me the strength to carry on. I dedicate this work to both of them, hoping that they will grow up to be curious and passionate learners.

I am also grateful to Instructor, **Fuad Yimer**, who showed me the way forward and a general overview in my research process. He was available to answer my questions and guide me through the research process. Finally, I thank all those who, in their own unique ways, contributed to this research journey. Their kindness and support will always be remembered.

TABLE OF CONTENTS

# ABSTRACT

*The growing reliance on Software-Defined Networking (SDN) necessitates robust security solutions, particularly against the escalating threat of Distributed Denial-of-Service (DDoS) attacks. Accurately and efficiently detecting both known and novel DDoS attacks in SDN environments remains a significant challenge. This study proposes a novel deep learning approach for efficient and accurate DDoS attack detection and mitigation within SDN. The proposed method utilizes a two-stage model: Stage 1 involves a comparative analysis between optimized Convolutional Neural Networks (CNN), Convolutional Neural Networks with Bidirectional Long Short-Term Memory (CNN-BiLSTMs), and Convolutional Neural Networks with Bidirectional Long Short-Term Memory and Attention (CNN-BiLSTMAttns), where all models achieved near-perfect accuracy (99.99%), with the CNN emerging as the most resource-efficient option. Stage 2 evaluates unsupervised learning with tuned Autoencoders (AE) and Variational Autoencoders (VAE) for anomaly detection, with the AE outperforming the VAE at a 99.86% detection rate. Various thresholding techniques were assessed with the AE, including percentile, Interquartile Range (IQR), Cumulative Sum (CUSUM), Peak-to-Peak, Control Chart, and Z-score, with CUSUM achieving the highest precision (100%) while Control Chart and Z-score demonstrated lower effectiveness. This two-stage approach combines the efficiency of a CNN for known attacks with the anomaly detection capability of an AE for novel attacks, using CUSUM thresholding for optimal results, thereby enhancing the resilience of SDN networks against DDoS threats. This innovative two-stage deep learning approach enhances SDN resilience by efficiently detecting both known and evolving DDoS attacks. It combines a resource-efficient Convolutional Neural Network (CNN) for known threats with the anomaly detection capability of Autoencoders (AE) for novel attacks.*



**Key words:** Autoencoder, Convolutional Neural Network , Distributed Denial of Service, Deep Learning, Software Defined Networking ,Threshold

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AE | Autoencoder |
| ACK | Acknowledgement |
| API | Application Programming Interface |
| AUC | Area Under Curves |
| AUC-ROC | Area Under the Receiver Operating Characteristic Curve |
| BiLSTM | Bidirectional long short-term memory |
| CNN | Convolutional Neural Networks |
| CUSUM | Cumulative Sum |
| DDoS | Distributed Denial of Services |
| DoS | Denial of Services |
| FTP | File Transfer Protocol |
| DNS | Domain Name System |
| GRU | Gated recurrent unit |
| GUI | Graphical User Interface |
| HTTP | Hyper Text Transfer Protocol |
| HOIC | High Orbit Ion Cannon |
| ICMP | Internet Control Message Protocol |
| IDS | Intrusion Detection System |
| IP | Internet Protocol |
| IQR | Inter Quartile Range |
| LOIC | Low Orbit Ion Cannon |
| LSTM | Long Short-term Memory |
| NBI | North Bound Interface |
| NOS | Network Operating System |
| NTP | Network Time protocol |
| ODL | OpenDaylight |
| ONF | Open Network Foundation |
| ONOS | Open Network Operating System |
| REST | Representational State Transfer |
| RNN | Recurrent Neural Networks |

| | |
|---|---|
| SBI | South Bound Interface |
| SDN | Software defined Networking |
| SMOTE | Synthetic Minority Oversampling Technique |
| SMTP | Simple Mail Transfer Protocol |
| SNMP | Simple Network Monitoring Protocol |
| SYN | Synchronization |
| TCAM | Ternary content-addressable memory |
| TCP | Transport Control Protocol |
| UDP | User Datagram Protocol |
| VAE | Variational Autoencoder |

CHAPTER ONE

# 1. Introduction

## 1.1. Background of the Study

The way networks are designed, built, and managed has evolved over time. Traditional networks have been based on a distributed architecture, where control and data planes are tightly coupled, and the network devices (routers, switches, etc.) operate as independent entities. However, with the growth of modern applications and the increasing demand for network flexibility, scalability, and programmability, new network paradigms have emerged.

One of these new paradigms is Software-Defined Networking, which aims to overcome the limitations of traditional switching networks. This is done by separating the control and data planes, which were previously located inside switches and routers, and creating a more flexible and manageable environment. With Software-Defined Networking centralizes the control plane within a dedicated controller. By consolidating decision-making and management functions, this architectural shift simplifies policy enforcement and enables dynamic network configuration adjustments. The controller communicates with network switches and routers, ensuring efficient resource utilization and agility. Ultimately, SDN holds the promise of substantial enhancements in both network performance and operational efficiency[1].

SDN offers several benefits, including programmability, vendor neutrality, agility, centralized network supervision, and automation. Administrators can manage multiple network devices from a logically centralized controller, leading to easy separation between experimental and production network traffic. Researchers can conduct experiments on a production network without interfering with actual network traffic. Additionally, SDN is less expensive than traditional networks, as a single centralized controller can manage all devices, saving deployment time and management expenses [2].

Even though SDN architecture provides several advantages over traditional network architecture, yet it is frequently exposed to network vulnerabilities. Especially Distributed Denial of Service (DDoS), can be launched from compromised hosts connected to switches. DDoS attacks can easily overload the controller processing capacity and flood the switch flow-table[3].

Distributed Denial-of-Service (DDoS) attacks are a major concern in software-defined networking (SDN). These attacks involve flooding the network with traffic to the point of overload, rendering the network and its services unusable. The damage caused by such attacks can be significant, not only for the targeted network but also for other networks and services connected to it[2], [4].

Traditional methods of detecting and mitigating DDoS attacks are no longer sufficient due to the increasing sophistication of attackers. Fortunately, deep learning techniques have shown great promise in detecting and classifying malicious traffic patterns in real time. By analyzing network traffic data, these techniques can identify and flag suspicious traffic, allowing security measures to be implemented to mitigate the attack[2], [4].

In recent years, researchers have proposed various deep-learning approaches for DDoS detection and mitigation in SDN. One such approach is a hybrid deep learning model that combines different deep learning algorithms and features to improve the accuracy and speed of attack detection and mitigation. This model is effective in detecting DDoS attacks and outperforms traditional methods[2], [4].

Overall, it is important for SDN security professionals to remain vigilant in their efforts to protect networks from DDoS attacks. The use of deep learning techniques is a promising area for improving the detection and mitigation of such attacks, and ongoing research in this area is crucial for staying ahead of evolving threat landscapes[2], [4].

### 1.2. Motivation

- As the use of SDN becomes more prevalent in modern networks, the threat of DDoS attacks is also increasing. Therefore, developing effective and efficient DDoS detection and mitigation techniques in SDN is crucial for maintaining network security and reliability.
- DDoS attacks are becoming more sophisticated and difficult to detect using traditional techniques, highlighting the need for more advanced and intelligent defense mechanisms. Deep learning has shown promising results in detecting and mitigating DDoS attacks in SDN, making it an important area of research for improving network security.
- DDoS attacks can have serious consequences for organizations, including downtime, data breaches, and financial losses. By developing more effective and

efficient DDoS detection and mitigation techniques in SDN, organizations can better protect their assets and reduce the impact of these attacks.

- The use of SDN is becoming more prevalent in critical infrastructure, such as power grids and transportation systems. As these systems become more interconnected, the risk of large-scale DDoS attacks increases, making it essential to develop effective defenses against these threats

### 1.3. Problem Statement

Distributed Denial of Service (DDoS) attacks are a serious threat to Software Defined Networks (SDNs), as they can target different layers of the network architecture and cause severe disruptions. A DDoS attack on the control plane of an SDN can compromise the controller, which is the central entity that manages a large number of switches and applications. This can lead to the failure of the entire network. A DDoS attack on the data plane of an SDN can overload the OpenFlow switch's flow table memory, which stores the rules for packet forwarding. When the flow table is full, the switch cannot accept new flows or install new rules from the controller, resulting in packet loss. The switch also has to buffer the packets that do not match any flow table entry and send them to the controller via packet-in messages. If the rate of new flows is high, the switch's buffer can fill up and the switch has to send the whole packets instead of just the headers to the controller. This can increase the communication bandwidth consumption and delay the installation of new rules[5].

DDoS attacks have become more frequent and sophisticated over time. According to a recent report, In the third quarter of 2022, there was a steady rise in Distributed Denial of Service (DDoS) attacks, especially those carried out by skilled professionals. The count of smart attacks during this period doubled compared to the same time last year[6].

While this problem is critical, existing studies have not provided satisfactory solutions. One of the main limitations of previous studies is that they have used outdated or irrelevant datasets that do not reflect the dynamic and diverse nature of DDoS attacks in SDN environments. Another limitation is that they have not classified the types of attacks into multiple categories, which makes it difficult to identify and analyze the characteristics and impacts of different attacks. Moreover, most of the proposed approaches have not offered end-to-end solutions that can be implemented in SDN settings, especially in terms of the mitigation module application. Furthermore, some of the studies have used inappropriate

or suboptimal deep learning models that can introduce additional overhead and side effects on the SDN performance. And also previous mechanisms for detecting DDoS attacks may not be able to cope with the increasing volume and diversity of DDoS attacks, which are constantly evolving and adapting to new strategies. Finally, some of the studies have not validated their results on different datasets to ensure their generalizability and robustness.

## 1.4. Research Questions

1. What are the most suitable deep learning architectures and techniques for detecting and classifying Known DDoS attacks in SDN environments?
2. How can deep learning models be trained and adapted to detect unknown or novel DDoS attacks in SDN environments?
3. For unsupervised deep learning-based DDoS detection in SDN environments, which threshold calculation method achieves the best balance between attack detection rate and false alarm rate?

## 1.5. Research Objectives

### 1.5.1. General Objective

To enhance network security within a software-defined network (SDN) environment by effective DDoS attack detection and mitigation system using deep learning models.

### 1.5.2. Specific Objectives

To build and validate supervised deep learning models to detect Known DDoS attacks

To build and validate unsupervised deep learning models to detect unknown DDoS attacks

To optimize hyperparameters and training parameters

To find Best Thresholding Techniques for Unsupervised Model

To combine supervised and unsupervised approaches

To integrating the mode into the SDN network infrastructure

### 1.6. Significance of the Research

- Improving the security of SDN-based networks: By developing more accurate and efficient DDoS detection and mitigation techniques, this study could help improve the overall security of SDN-based networks, which are becoming increasingly prevalent in modern computing environments.
- Reducing the impact of DDoS attacks: DDoS attacks can cause significant disruption to network services, resulting in financial losses and damage to organizational efficiency By improving DDoS detection and mitigation capabilities, this research could help minimize the impact of such attacks.
- Advancing the field of deep learning: this research could contribute to the ongoing development of deep learning techniques, which are increasingly being applied to a range of real-world problems, from image recognition to natural language processing and anomaly detection
- Enabling more efficient network management: By developing more efficient DDoS detection and mitigation techniques, this research could help network administrators more effectively manage network traffic and resources, leading to improved network performance and reliability.
- Informing future research in the field: By identifying the most effective DDoS detection and mitigation techniques for SDN-based networks, this research could help inform future research in the field, potentially leading to further advances in network security and deep learning techniques.

### 1.7. Scope and Limitation

**Scope:** This Study focuses on identifying and mitigating volumetric (DDoS) attacks within Software-SDN environments. Volumetric DDoS attacks, such as flooding and amplification attacks, overwhelm a network with a massive influx of traffic, rendering it inaccessible for legitimate users. The advantages of SDN, with its centralized control plane and programmability, create unique opportunities for enhanced defense against volumetric DDoS attacks. This study aims to explore and evaluate advanced techniques specifically designed for SDN deployments.

**Limitation:** Due to the absence of a readily available real-world SDN environment, the proposed approach evaluates using simulations that emulate real-world volumetric DDoS attacks. These simulations assess the accuracy and timeliness of attack detection within the

SDN environment. While simulations offer a controlled environment, they might not fully capture the complexities and variations present in real-world networks

### 1.8. Research Contribution

This study has made the following output to the field of security within Software-Defined Networking (SDN) environments. The key findings are as follows:

1. Efficient Detection of Known DDoS Attacks: The study successfully developed a two-stage deep learning approach that initially employs a supervised learning model to identify well-known DDoS attack patterns. The optimized Convolutional Neural Network (CNN) architecture emerged as the most resource-efficient model, achieving an impressive 99.99% accuracy rate.

2. Identification of Unknown DDoS Attacks: In the second stage, the study focused on detecting novel attack types using unsupervised learning models. The Autoencoder (AE) was found to be slightly more efficient and accurate than the Variational Autoencoder (VAE), with a 99.86% efficiency rate.

Threshold Calculation Techniques: A variety of threshold calculation techniques were applied to the AE model to enhance its anomaly detection capability. The Cumulative Sum (CUSUM) method achieved a perfect precision of 100%, making it the most effective technique for minimizing false positives.

5. Model integration: The combination of supervised and unsupervised models leverages the strengths of both to enhance SDN security.

### 1.9. Organization of the Thesis

The rest of the paper is structured as follows:

**Chapter 2: Literature Review and Related Works**

This chapter delves into existing literature relevant to the research. SDN concepts and definitions are explored, covering SDN planes, interfaces, working principles, and security challenges. Deep learning techniques are introduced, emphasizing their potential applications in network security. DDoS attacks and their impact on networks are examined, discussing common attack techniques and tools. Related work on DDoS attack detection and mitigation in SDN using deep learning is reviewed, identifying research gaps and strengths.

**Chapter 3: Research Methodology**

This chapter outlines the research methodology. The process of dataset collection and preprocessing is described, focusing on identifying critical DDoS attacks specific to SDN. The approach involves setting up an SDN simulation environment for data collection. The steps to generate and collect traffic data are detailed, followed by a description of the dataset itself. Additionally, the two-stage model is introduced: Stage 1 utilizes supervised deep learning for known DDoS detection, while Stage 2 employs unsupervised deep learning for detecting unknown DDoS attacks. Model selection and hyperparameter tuning are discussed, along with the enhancement of unsupervised DDoS detection using various thresholding techniques.

**Chapter 4: Proposed Approaches Design**

In this chapter, the design of the proposed two-stage model is presented. The architecture and functionality of each stage are described. Stage 1 focuses on supervised deep learning for known DDoS detection, while Stage 2 addresses the challenge of detecting unknown DDoS attacks using unsupervised deep learning. The selection process for both supervised and unsupervised DDoS detection models is discussed. Additionally, thresholding techniques are explored to enhance the performance of unsupervised DDoS detection. Finally, the mitigation approach within the SDN environment is outlined.

**Chapter 5: Experiment and Implementation**

This chapter details the experimental setup and implementation. Validation of dataset preprocessing procedures is provided, along with insights into the two-stage model training. Specifically, the experimental setup for both stages supervised Stage 1 and unsupervised Stage 2 DDoS detection models—is described. Hyperparameter tuning is discussed, along with experiments on threshold techniques using autoencoders. The integration of Stage 1 and Stage 2 is explored, and the proposed approach is deployed within the SDN environment, including algorithm details and flowcharts.

**Chapter 6: Result and Discussions**

In this chapter, the results of the proposed two-stage model are evaluated and interpreted. The evaluation outcomes for the supervised Stage 1 model are presented, and the performance of the unsupervised Stage 2 model is analyzed. Different thresholding techniques for the unsupervised DDoS detection model are explored. The results are summarized, emphasizing computational efficiency and the full spectrum of model capabilities. Finally, a discussion relates the findings back to the research objectives.

**Chapter 7: Conclusion and Future Works**

The final chapter summarizes key research findings and contributions. The significance of the proposed two-stage deep learning approach for DDoS detection and mitigation in SDN environments is highlighted. Additionally, the limitations of the study are discussed, and potential directions for future research are suggested.


Figure 1. Thesis Organization

CHAPTER TWO

# 2. Literature Review and Related Works

## 2.1. SDN Concepts and Definitions

Initially, the concept of SDN was introduced to describe the research work and ideas related to OpenFlow at Stanford University, USA. However, the networking industry has expanded the definition to include anything related to software, leading to confusion about what SDN actually entails. To address this issue, the authors of this paper aim to provide a clearer and less ambiguous definition of SDN[5].

SDN is a network architecture with four main pillars. Firstly, the control and data planes are separated, which means that the network devices responsible for packet forwarding are stripped of their control functions. They argue that forwarding decisions are based on flows, which are defined as a set of packet field values acting as a match criterion and a set of actions. All packets of a flow receive the same service policies at forwarding devices, enabling different types of devices to behave uniformly. This flow-based approach provides unprecedented flexibility, subject only to the capabilities of the implemented flow tables. The author states that the control logic is moved to an external entity called the SDN controller or NOS, which runs on commodity server technology and provides essential resources and abstractions for programming forwarding devices based on an abstract network view. The NOS serves a similar purpose to that of a traditional operating system. Lastly, the network is programmable through software applications running on top of the NOS, which interact with the underlying data plane devices. This programmability is a core feature of SDN and is considered its main value proposition [7].

### 2.1.1. SDN Planes

SDN is an innovative architecture that provides flexibility and programmability in modern data centers by separating control functionality from forwarding hardware. It is a three-layer architecture, consisting of a control plane (controller), data plane (forwarding plane), and application plane (management plane).

**Data plane(Forwarding Layer)** : the data planes of SDN is made up of OpenFlow-enabled switches, which handle traffic based on the instructions given by the controller. OpenFlow switches have a flow table containing three fields: match, actions, and counters. The match field includes different matching header fields such as Ethernet, IP, TCP, or UDP,

depending on the version of OpenFlow. Actions available in the action field include forwarding or dropping packets. The counter field serves a quality of service function by keep track the packet count for each flow. Upon receipt of packets, an OpenFlow-compatible device compares them against the established rules in the flow table. Should there be a match, the device then executes the corresponding action for that rule, which could include either forwarding or discarding the packet. Otherwise, it drops the packet or sends it to the controller using the southbound interface. The controller inserts or modifies the flow entries into the OpenFlow switch according to the program running on the application plane[2].

**Control plane(Control Layer):** The control plane of SDN works as a Network Operating System (NOS) that decides the functionality of SDN switches. It maintains a centralized view of the network and can examine network traffic more efficiently than traditional networks. The centralized controller has complete control over the entire network. Popular SDN controllers include NOX, POX, Ryu, Beacon, OpenDaylight, Floodlight, ONIX, and ONOS.[2] .There are many open source SDN controllers available, each with different features, performance, and capabilities. Some of the most popular open source SDN controllers are discussed in the following table[7].

Table 1. Opensource SDN Controller Comparison

| No | Controller | License | Architecture | Language | Documentation | Platform |
|---|---|---|---|---|---|---|
| 1 | ONOS | Apache 2.0 | Distributed Flat | Java | Good | Linux, macOS, Windows |
| 2 | Trema | GPL 2.0 | Centralized | C, Ruby | Fair | Linux |
| 3 | Ryu | Apache 2.0 | Centralized | Python | Good | Linux, macOS |
| 4 | RUNOS | Apache 2.0 | Distributed Flat | C++ | Fair | Linux |
| 5 | FloodLight | Apache 2.0 | Centralized | Java | Good | Linux, macOS, Windows |
| 6 | Maestro | LGPL 2.1 | Centralized | Java | Limited | Linux, macOS, Windows |
| 7 | MuL | GPL 2.0 | Centralized | C | Good | Linux |

| | | | | | | |
|---|---|---|---|---|---|---|
| **8** | ODL | EPL 1.0 | Distributed Flat | Java | Good | Linux, macOS, Windows |
| **9** | POX | Apache 2.0 | Centralized | Python | Limited | Linux, macOS, Windows |
| **10** | NOX | GPL 2.0 | Centralized | C++ | Limited | Linux |
| **11** | Beacon | GPL 2.0 | Centralized | Java | Fair | Linux, macOS, Windows |
| **12** | Faucet | Apache 2.0 | Centralized | Python | Good | Linux |

**Application plane(Application Layer):** The application plane of SDN comprises multiple applications such as load balancers, firewalls, traffic monitors, and others that run on the controller. These application determine how the forwarding device operates. Thus, SDN leverages the network infrastructure more effectively compared to traditional networks. Within SDN, a single device is capable of multiple functions, whereas in a standard network, each device is limited to a specific function because of its inflexible design[2].

SDN requires a southbound API to provide communication between the control functionality and forwarding hardware. Various southbound APIs are available, including ForCES, OpenFlow, and Netconf. The Open Network Foundation (ONF) defines OpenFlow as the standard southbound interface. Transport layer security is applied to the OpenFlow protocol to enable secure communication across the southbound interface, while it is not required, as advised by the ONF. The management plane interacts with the controller through the northbound API, which includes REST API, Frenetic, Pyretic, Procera, Netcore, among others. The standardization of the northbound interface is still a topic of discussion due to its dynamic nature[2].

### 2.1.2. SDN Interfaces

**Southbound interface:** The southbound interface is capable of connecting the control layer and infrastructure layer. The control layer sends some monitoring and managing messages to the infrastructure layer via the southbound interface, while the infrastructure layer packages some messages to report the current network status to the control layer. As mentioned above, besides specifying the components and basic functions of SDN enabled switch, the OpenFlow switch specification also introduces the OpenFlow protocol, a de-

facto southbound interface in SDN. In OpenFlow 1.3.0, there are three types of messages, which are (1) the controller-to-switch messages, (2) asynchronous messages, and (3) symmetric messages. Among these messages, the controller-to switch messages are provided to the controller to manage or monitor the switch. Only the controller has the ability to start and send these messages to the switch. Conversely, the switch is the only entity capable of initiating asynchronous messages that are delivered to the controller informing it of changes or events that have occurred within the switch. Different from these two kinds of messages, the symmetric messages can be initiated by the controller or switch, which are employed to achieve some special functions. Setting up communication between the controller and switch, ensuring the connection remains active, and allowing for future updates to the OpenFlow protocol.. The detailed descriptions of these three messages[8].

**Northbound interface:** The control layer provides an abstract network view and a programmable interface for the application layer via the northbound interface. Utilizing the northbound interface, the application layer is able to access the network's structure and comprehensive details about each switch, including information on flow tables, group tables, port statistics, queue configurations, meter statistics, and the role of the controller. Meanwhile, the application layer can also add, modify and delete flow entry, group entry, meter entry, and modify the behavior of ports in the switch an modify the controller role. A variety of northbound interfaces have been released by certain businesses and institutions. Examples include the APIs from NOX, POX, and RYU, the REST APIs which are commonly supported by numerous controllers, the tinyNBI known for its complete compatibility with every OpenFlow version, and the Frenetic which offers its own programming language. However, different from the southbound interface, there is no leading specification in the northbound interface currently[8].

**Westbound/Eastbound Interfaces:** The westbound/eastbound interfaces are used in SDN containing the multi-controller. In expansive network setups utilizing SDN, a single controller might struggle to handle the substantial network traffic or data flows. Therefore, these larger networks are segmented into more manageable smaller domains, each overseen by its own dedicated controller. Thus, communication between these separate controllers is necessary so that the global network view can be presented to the application plane; this communication is where westbound/eastbound interfaces are used[9].

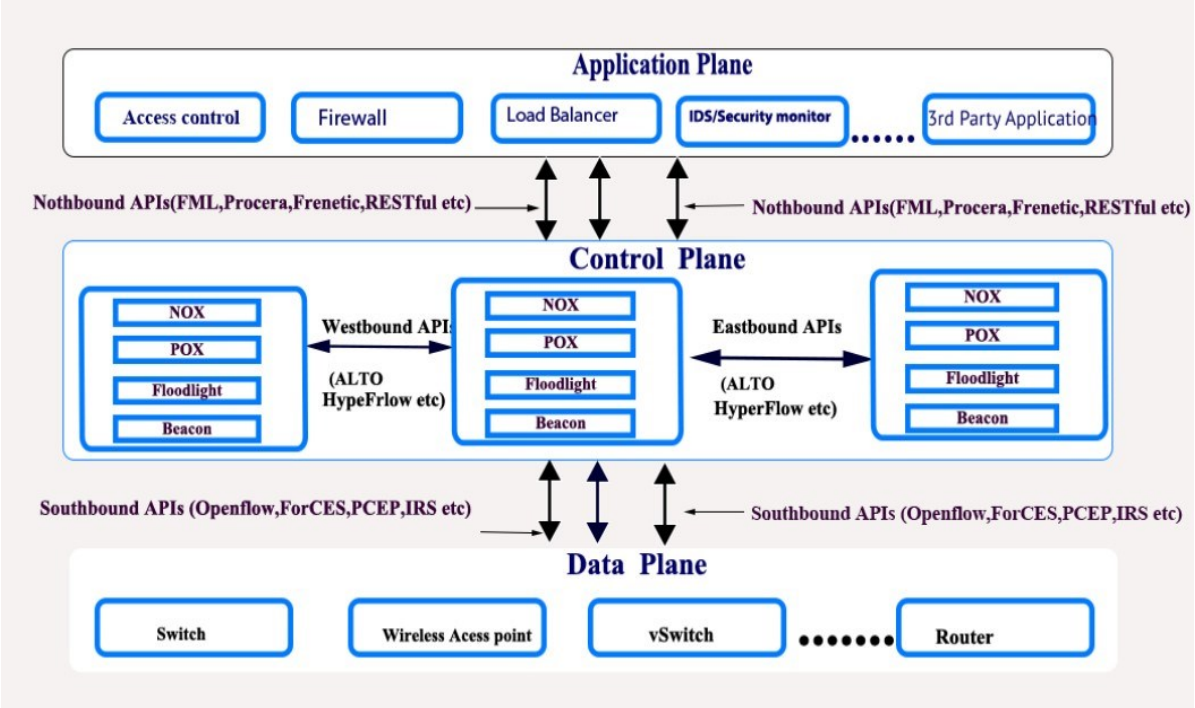



Figure 2. SDN Components And Architecture

2.1.3. Working Principles and Network Programmability of SDN

**OpenFlow protocol:** OpenFlow is a protocol managed by the Open Networking Foundation that enables programmability of the data forwarding plane in software-defined networks (SDN). The protocol uses a flow table to store a list of flow entries sorted by priority, which define how network traffic should be handled. Each flow entry consists of rules, priority, counters, actions, timeouts, cookies, and flags. When a packet arrives at an OpenFlow switch, it is matched against the rules in the flow table to determine how it should be processed. If a matching flow entry is found, the corresponding action is executed on the packet. If not, the packet is forwarded to the controller over a secure channel using OpenFlow messages. [10]

**OpenFlow message :** According to [11]**,** the OpenFlow protocol defines several message types, including the Hello Message, Features Request/Reply Messages, Flow Mod Message, Packet-Out Message, and Packet-In Message. The Hello Message is the first message exchanged between the OpenFlow switch and controller to establish a connection. The Features Request/Reply Messages are used to retrieve the switch features, such as the number of ports, supported protocols, and buffer sizes. The Flow Mod Message is used by

the controller to add, modify, or delete flow entries in the switch's flow table. The Packet-Out Message is used by the controller to send a packet to the switch for transmission, while the Packet-In Message is used by the switch to inform the controller about an incoming packet that does not match any of the flow table entries.

In a software-defined network, the communication between the source and destination involves several steps. As described by [10], the packet first reaches the OpenFlow switch when the source sends a packet to the destination. The OpenFlow switch then performs a lookup in its flow table to find a matching flow entry. If there is no matching flow entry in the flow table, the switch sends a Packet-In message to the controller, asking for instructions on how to handle the packet. The controller then receives the Packet-In message and sends a Flow Mod message to the switch with instructions on how to handle the packet. If there is a matching flow entry in the flow table, the switch forwards the packet to the next hop based on the instructions in the flow entry. The packet is forwarded from switch to switch until it reaches its destination, where an ACK packet is sent back to the source, following the same steps as described above, as noted by [9].

In summary, the OpenFlow architecture and message types, as well as the steps involved in communication between the source and destination in a software-defined network, are essential components of SDN that enable centralized network control and management. These concepts have been extensively studied and discussed in various works, [11]), [10], and [9]. Fig. 2 presents a step-by-step explanation of the workflow process of an OF switch in SDN. The flow tables of an OF switch are numbered sequentially, starting from zero. The first flow table is flow table 0, and the packet is first checked against the flow entries in this table. If no match is found, the packet is forwarded to the next flow table with a higher number.

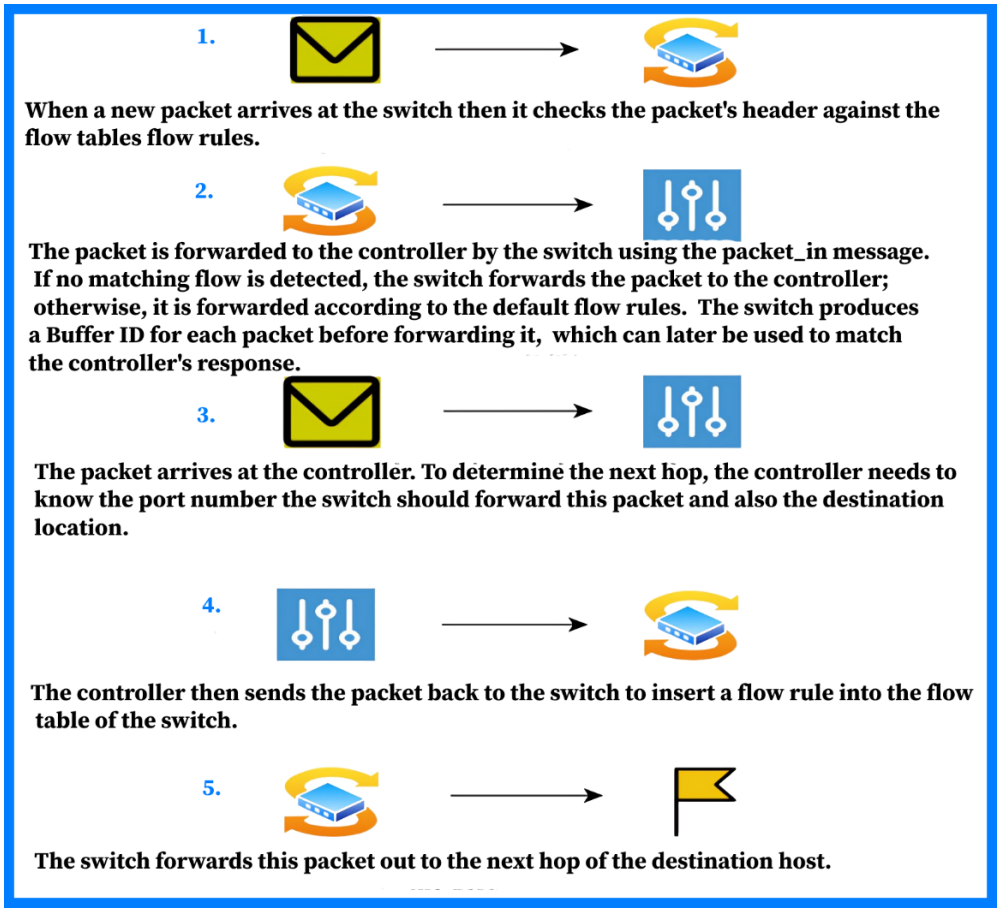



Figure 3. Workflow Process of an OpenFlow Switch in SDN

### 2.1.4. SDN Security Challenges

#### 2.1.4.1. Data Plane Attack

SDN networks' data plane, composed of simple OpenFlow-enabled switches, is susceptible to a range of security vulnerabilities stemming from its inherent characteristics and limited resources [1] [2] [3]. One concern is the manipulation of flow table rules, where adversaries can insert or modify rules to intercept or alter the network traffic, or to conduct Man-in-the-Middle Attacks (MITM), resulting in data integrity loss [1]. Another concern is the topology spoofing, where adversaries can exploit the switches' limited processing capabilities to create a fake network view for the controller, redirecting traffic to their machines or to a black hole [1] [2]. Moreover, these switches are also vulnerable to flow table overloading and buffer saturation attacks, which can exhaust the memory and processing resources of the switches and the controller, and prevent them from handling legitimate traffic [1] [3].

The infrastructure layer of SDN consists of dumb OpenFlow switches that forward data from source to destination. However, this layer is vulnerable to attacks by adversaries who

can gain unauthorized access and launch resource exhaustion attacks by sending malicious traffic to the switches. These attacks can also exploit the limited memory resources of the switches and plant false rules in the flow table, causing performance degradation and traffic manipulation [2].

Distinguishing between legitimate and malicious flow entries poses a significant challenge, as OpenFlow switches lack the ability to differentiate on their own. This lack of intrinsic defense mechanisms places a heavy reliance on the SDN controller, which can compromise the entire infrastructure layer if attacked by an adversary [2] [3]. Moreover, the data plane is vulnerable to saturation attacks due to the constraint on flow table entries caused by memory limitations, and to outages due to the interdependence between the data plane and control plane [3].

In summary, SDN data plane attacks exploit the vulnerabilities of OpenFlow-enabled switches, affecting flow rule integrity, switch performance, and overall network security. These challenges underscore the need for robust control plane security and innovative defenses to protect the integrity and availability of the data plane [1] [2] [3].

2.1.4.2. Control Plane Attack

The controller is the core component of the SDN network, as it is responsible for making the forwarding decisions for the switches in the data plane. However, this also makes it the most attractive target for attackers, who can exploit its vulnerabilities and compromise the entire network. Some of the security challenges that face the controller are:

**Attacks on the northbound and southbound interfaces:** The controller communicates with the applications in the management plane through the northbound interface, and with the switches in the data plane through the southbound interface. These interfaces can be exploited by attackers to gain unauthorized access to the controller or to send false or malicious information to it. For example, an attacker can use a vulnerable application to install fake flow rules on the switches, or use a compromised switch to send false topology information to the controller [1] [2] [3] [5].

**Attacks on the controller resources:** The controller has limited computational and memory resources, which can be overwhelmed by a large number of new flows or packet-in messages from the switches. A new flow is a packet that does not match any existing flow rule on the switch, and a packet-in message is a message that the switch sends to the

controller to request a flow rule for a new flow. An attacker can send a large number of packets with different header fields that do not match any existing flow rules on the switches, causing them to generate many packet-in messages to the controller. This can consume the memory and processing resources of the controller, or its bandwidth, and prevent it from handling legitimate traffic or functions. This can cause congestion or denial-of-service attacks on the controller [1] [2] [3] [4] [5].

**Attacks on the controller scalability:** The controller needs to implement flow rules for every new flow in the data plane, which can quickly become a bottleneck in high-speed networks with many switches and hosts. This can reduce the performance and reliability of the network, and make it more susceptible to distributed denial-of-service attacks [3] [4] [5].

**Attacks on the controller trust:** The controller trusts the information received from the switches and the applications, which can be manipulated by attackers to create a fake network view or to misroute traffic. For example, an attacker can spoof the switch identity or link status to redirect traffic to a black hole or a man-in-the-middle attack [1] [2].

2.1.4.3. Application Plane Attack

The application plane of SDN networks faces security challenges due to its lack of authentication and access control mechanisms [1] [3]. This layer, responsible for implementing network policies and interacting with the controller, is susceptible to malicious applications, resource exhaustion, and policy conflicts [1]. Attackers can exploit this vulnerability by running malicious programs on the controller, potentially compromising the entire network [1]. Resource-intensive requests sent by malicious applications can overwhelm the controller's computational and network resources [1]. Policy conflicts arise when contradictory instructions are inserted by various applications on the controller [1].

The application layer is located at the top in SDN architecture, engaging in tasks like security enforcement and network management [2]. Adversaries aiming to exploit this layer can target specific applications to drain their resources or focus on the Northbound interface to disrupt communication between the controller and the application layer [2]. Unauthorized access to SDN applications can lead to policy violations across the network [2].

2.1.4.4. Interfaces Attack

**Southbound interface:** The OpenFlow is a widely used southbound protocol. However, it does not require a secure connection between the OpenFlow switch and the controller, as TLS level security is optional but not mandatory. This makes it easy for attackers to intercept the rules that the controller sends to the OpenFlow switch through the southbound interface. Therefore, an attacker can alter existing flow rules or insert malicious flow entries to divert packets, which can have disastrous effects on the whole network. Furthermore, the limited bandwidth of the communication channel makes it prone to bandwidth saturation attack. To overload the bandwidth of the communication channel, a malicious node modifies the values of packet header fields so that these packets do not match with the existing flow rules and the switch has to forward these packets to the controller, causing congestion on the communication channel[1].

**Northbound interface:** The applications use the northbound interface to gather network statistics or to instruct the controller for network management. Due to the absence of standardization, this interface is exposed to multiple security threats such as tampering, DoS, and eavesdropping[1].

**East-west bound interface:** In multi-controller architecture, controllers communicate with each other through the east-west bound interface. Due to the lack of TLS adoption, the attacker can easily obtain the information to compromise the controller[1].

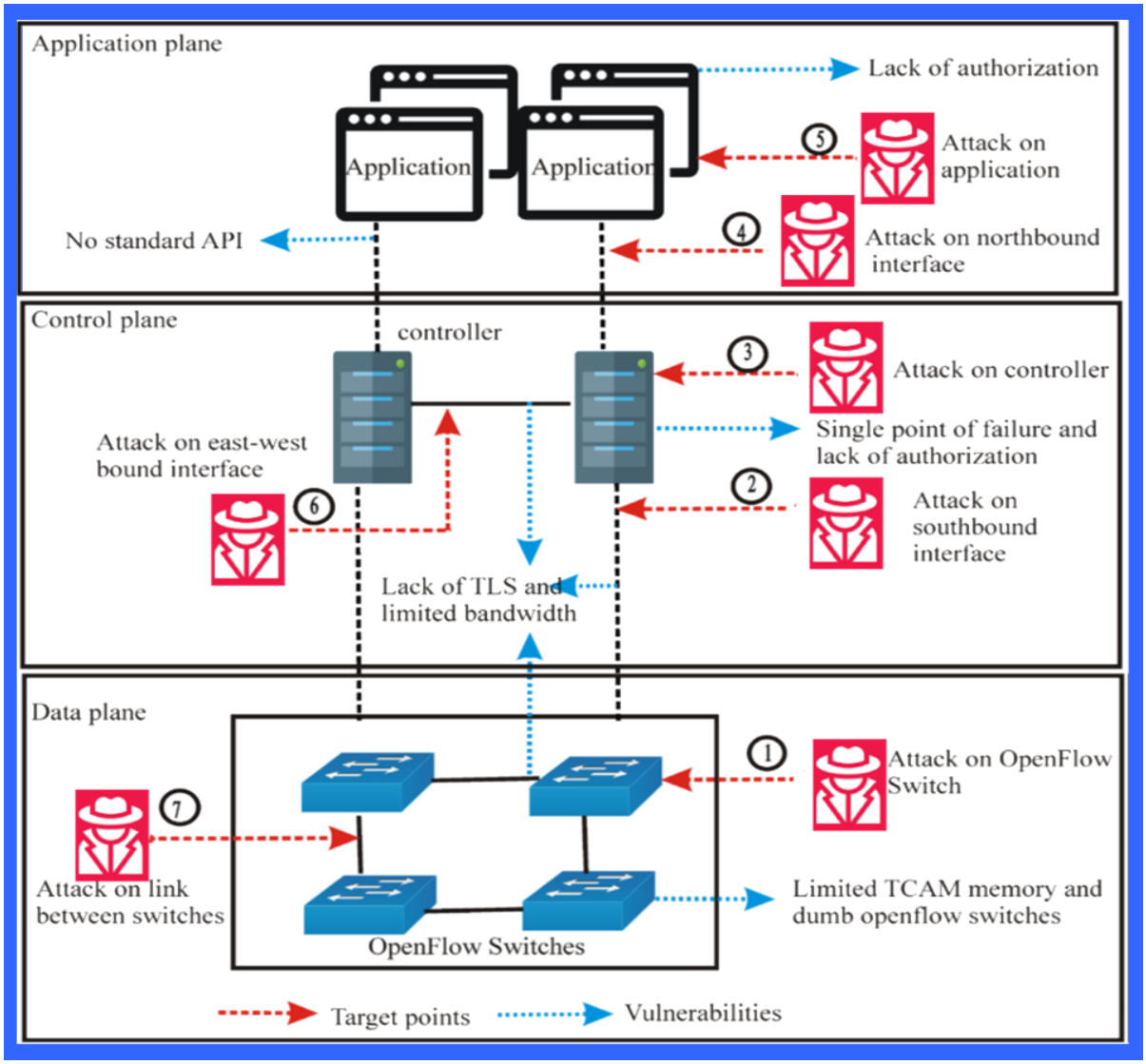



Figure 4. Security Vulnerability of SDN

### 2.2. Deep Learning Techniques and Potentials in Network Security

The fusion of deep learning and network security is a powerful way to combat the evolving cyber threats. Deep learning can automatically extract complex patterns and features from data, which can improve network security applications [6]. This combination offers a new paradigm shift, enabling advanced intrusion detection, anomaly recognition, and threat mitigation.

One of the main applications of deep learning in network security is network attack detection. Deep learning methods can detect subtle patterns that indicate attacks [6]. For example, intrusion detection systems (IDS) with deep learning frameworks can identify network intrusions and unauthorized activities more accurately. Neural networks such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long

Short-Term Memory (LSTM) networks can capture sequential patterns in network traffic data [7]. By learning the normal behavior of network traffic, deep learning models can quickly identify deviations, signaling potential attacks.

Deep learning also addresses computer security challenges in various dimensions. One of them is malware detection and classification. Deep learning models, especially CNNs, can extract complicated features from malware binaries, enabling fast and precise malware detection [7]. This also applies to the detection of zero-day vulnerabilities and unseen attacks, which are hard for conventional security mechanisms to identify. The adaptive nature of deep learning algorithms allows them to evolve and adapt to new threats, enhancing the overall security.

Another aspect is user authentication and access control based on behavioral patterns. The analysis of user behaviors, through techniques like Recurrent Neural Networks (RNNs), helps to identify unauthorized access attempts and compromised accounts [8]. Deep learning-driven user authentication systems can effectively identify anomalies in behavior, such as unusual login times or interaction sequences, reducing false positives and improving security.

From a broader perspective, the integration of neural network and deep learning methods leads to comprehensive cybersecurity paradigms. Deep cybersecurity involves using deep learning models to orchestrate advanced threat detection, incident response, and predictive analysis [8]. This approach enables organizations to anticipate and counteract potential breaches before they become actual threats.

To sum up, the combination of deep learning and network security brings a new era of advanced threat detection, anomaly recognition, and cyber defense mechanisms. The application of deep learning in network security, informed by research from [6], [7], and [8], has shown remarkable potential in identifying network attacks, detecting malware, and enhancing user authentication systems. As the cybersecurity landscape continually evolves, deep learning's adaptability and proficiency in uncovering hidden patterns in complex data make it a formidable tool in the ongoing battle against cyber threats.

## 2.3. DDoS Attacks and Impact on Networks

A Distributed Denial-of-Service (DDoS) attack is a malicious attempt to disrupt the normal functioning of a network or a service by overwhelming it with a large amount of traffic from multiple sources . The main objective of a DDoS attack is to exhaust the network resources, such as bandwidth, memory, CPU, or disk space, and prevent legitimate users from accessing the network or the service . DDoS attacks can cause severe damage to the network infrastructure and the service providers, such as financial losses, reputation damage, customer dissatisfaction, legal liabilities, and operational disruptions[8][10][12].

### 2.3.1. Common DDoS Attack Techniques

DDoS attackers use various techniques to launch and execute their attacks. Some of the common techniques and tools are:

**Botnets**: A botnet is a network of compromised devices (also called bots or zombies) that are remotely controlled by an attacker (also called botmaster) through a command and control (C&C) server . The attacker can use the botnet to generate and send traffic to the target network or service from multiple sources. Botnets can be composed of different types of devices, such as computers , smartphones , routers , IoT devices , etc. Some examples of botnets are Mirai , Satori , Reaper , etc. .

**Amplification**: Amplification is a technique that exploits the asymmetric nature of some network protocols that allow an attacker to send a small request with a spoofed source IP address (the target IP address) to a server that responds with a large response to the target IP address . This way, the attacker can amplify the traffic volume by several times without consuming much bandwidth. Some examples of amplification protocols are DNS , NTP , SNMP , etc. .

**Reflection**: Reflection is a technique that uses legitimate servers as intermediaries to send traffic to the target network or service. The attacker sends a request with a spoofed source IP address (the target IP address) to a server that responds with a normal response to the target IP address. This way, the attacker can hide its identity and location from the target network or service. Some examples of reflection protocols are DNS , NTP , SNMP , etc .

**Spoofing**: Spoofing is a technique that involves changing or falsifying the source IP address of packets or requests sent by an attacker to make them appear as if they come from legitimate sources . This way, the attacker can bypass the network filters or firewalls that

rely on the source IP address to identify and block malicious traffic. Spoofing can also make it difficult to trace back the origin of the attack .

**Tunneling**: Tunneling is a technique that uses encrypted or encapsulated protocols to hide the malicious traffic from the network monitoring or detection systems. The attacker can use tunneling to send traffic through a secure or trusted channel , such as VPN, SSH, SSL, etc., that cannot be easily inspected or analyzed by the network devices or tools. Tunneling can also help the attacker to evade the network policies or restrictions that limit the access or usage of certain protocols or ports .

2.3.2. Common DDoS Attack Tools

**LOIC**: LOIC is an open source software with a GUI interface that Praetox Technologies developed to launch DDoS attacks such as TCP SYN and UDP attacks that can deplete the resources of a target server . The LOIC DDoS attack requires the source URL or IP address, the port address, and the attacking service (TCP or UDP). After entering all the necessary information in the LOIC GUI field, LOIC begins to send a large amount of traffic to the target server. The LOIC tool creates TCP and UDP DDoS flooding attacks to test the processor capacity of the target server[13] [8].

**HOIC**: HOIC is a widely used tool for launching DDoS attacks that can hit up to 256 websites at the same time. Unlike LOIC, HOIC only generates HTTP traffic. It also has an integrated scripting system that takes .hoic files as input and allows the user to increase the intensity of the attack[14].

**Slowloris** : is a powerful tool that creates and maintains many connections to the targeted web server for as long as possible. It uses these connections to send partial HTTP requests continuously. This makes the servers under attack keep the connections open, expecting the incomplete attack requests to finish[15].

**RUDY**: The Rudy tool splits the data from the form into many small pieces, each with only one byte of information. It then sends these pieces at irregular intervals, keeping the web server from ending the connection and making it wait for the whole data to arrive. This way, the Rudy tool can create a few thousand requests in a matter of minutes, which overwhelms the web server and stops it from serving other users. This results in a denial of service attack. Any website that has a web form that takes http POST requests, such as login details, feedback forms, or web mail, can be vulnerable to this kind of attack[16].

**GoldenEye**: is a tool that tests the resilience of HTTP/S servers against DDoS attacks by using persistent socket connections and caching. It can also analyze the behavior and goals of malware by changing the system environment dynamically and observing the malware's reactions. GoldenEye is fast and efficient, and it is written in python with a timer function that works on any operating system[15].

**Hping and Hping3:** is a network tool that can send custom TCP/IP packets. It can handle TCP, UDP, ICMP, and RAW-IP protocols, and it can modify the packets it sends. This tool can adjust the size, number, and fragmentation of packets to overload the target and evade or assault firewalls. Volumetric attack (bandwidth attack) employed (ICMP and UDP flooding) against the victim. Volumetric DDoS attacks aim to exhaust the internal network capacity with extremely large amounts of malicious traffic. These attacks try to use up the bandwidth of the target server[17]**.**

### 2.4. SDN As Target of DDoS Attack

Software-defined networking (SDN) is a new paradigm that aims to provide more flexibility, scalability, and programmability to the network management and operation. SDN decouples the control plane from the data plane, and centralizes the network intelligence and decision making in a software entity called SDN controller. The SDN controller communicates with the network devices, such as switches and routers, through a standardized interface called southbound interface, such as OpenFlow. The SDN controller also exposes a northbound interface to the network applications and services, such as routing, load balancing, security, etc., that can request and control the network resources and behaviors [2].

SDN offers many benefits and opportunities for network security, such as global network visibility, dynamic policy enforcement, fine-grained traffic control, etc. . However, SDN also introduces new vulnerabilities and challenges for network security, especially for DDoS attack detection and mitigation. Some of the main vulnerabilities and challenges are:

**Centralized control plane:** The SDN controller is a single point of failure and a single point of attack for the entire network. If the SDN controller is compromised or overloaded by DDoS attacks, it can cause severe damage to the network functionality and performance .

**Southbound interface:** The SBI is a critical communication channel between the SDN controller and the network devices. If the SBI is disrupted or manipulated by DDoS attacks, it can affect the network configuration and operation .

**Northbound interface:** The NBI is an open interface that allows various network applications and services to access and control the network resources and behaviors. If the NBI is exploited or abused by DDoS attacks, it can cause conflicts or inconsistencies among different applications and services .

**Resource constraints:** The SDN devices, such as switches and routers, have limited resources, such as memory, CPU, or flow table entries, that can be easily exhausted by DDoS attacks. This can degrade the network performance and quality of service .

**Novel attack vectors:** The SDN architecture and features enable new types of DDoS attacks that are specific to SDN environments , such as control plane saturation attack , data plane saturation attack , application layer saturation attack , etc. .

Therefore , there is a need for effective and efficient DDoS attack detection and mitigation solutions in SDN environments that can cope with these vulnerabilities and challenges .

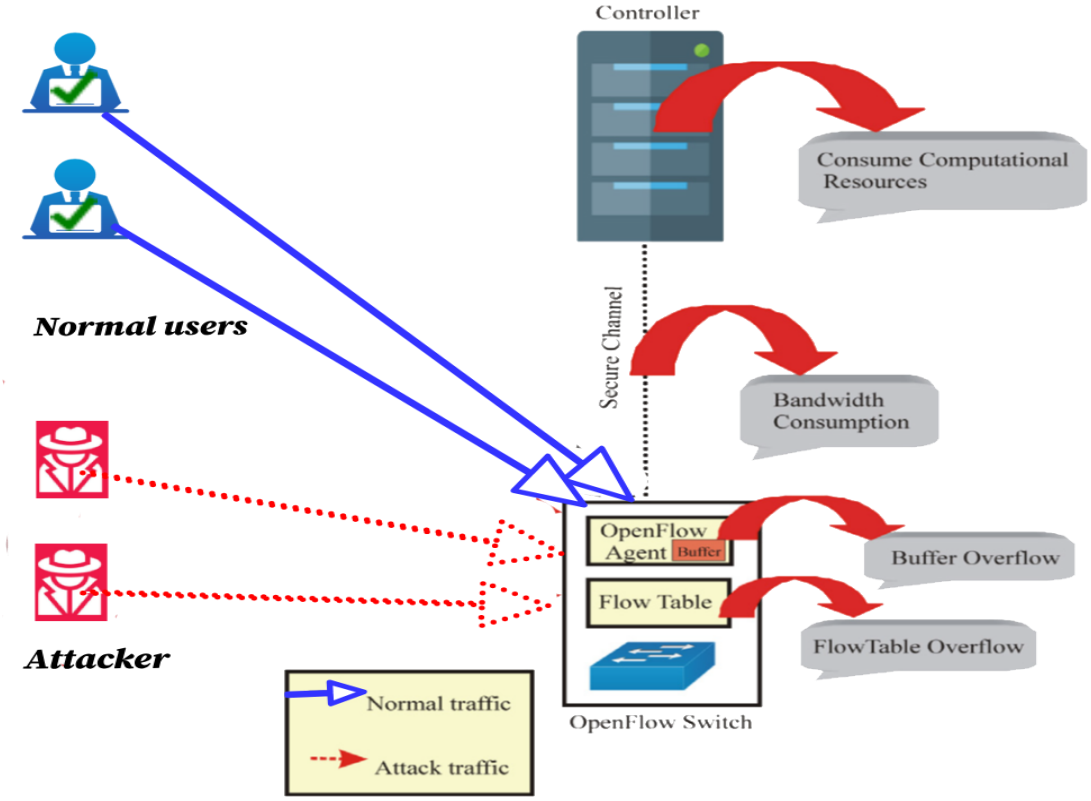



Figure 5. SDN as Target of DDoS Attack

### 2.5. DDoS Attack Detection and Mitigation in SDN: Related work

DDoS attacks are a serious threat to the security and performance of SDNs, and detecting and defending against them is an important and urgent research problem. Many researchers have proposed different methods and techniques to address this problem, using various tools and approaches, such as statistics, rules, policies, and machine learning. Among these methods, deep learning, which is a subset of machine learning, has shown promising results. This study discusses the latest and most relevant deep learning models for detecting and defending against DDoS attacks in SDNs.

The paper [18] proposes a hybrid deep learning algorithm that combines three types of deep learning models: a 1D convolutional neural network (CNN), a gated recurrent unit (GRU), and a dense neural network (DNN). The CNN extracts features from the network traffic data, the GRU captures the temporal dependencies of the data, and the DNN classifies the data into normal or attack categories. The proposed system also uses a feedback mechanism to update the SDN controller with the detection results and apply appropriate countermeasures to mitigate the attacks. The paper claims that the proposed system achieves high accuracy rates of 99.81% and 99.88% on two different datasets, and outperforms several existing methods in terms of detection accuracy, false positive rate, false negative rate, and detection time.

However, the proposed system lacks information on how to apply the proposed system in a real SDN environment. It only describes the implementation details of the hybrid deep learning algorithm and the feedback mechanism, without explaining how to integrate them with the SDN controller and the switches. It also does not discuss the computational complexity and scalability of the proposed system, which could limit its practical applicability. Moreover, the proposed system cannot detect novel types of DDoS attacks.

The other study[19] proposes a deep learning technique that uses new features extracted from traffic statistics to detect DDoS attacks in both the control plane and the data plane of SDNs. The technique employs two AE-BGRU models, one for each plane, to classify the traffic as normal or attack based on different features. The technique also monitors the switch's average arrival bit rate with an unknown destination address to detect abnormal traffic patterns. Moreover, the technique mitigates the DDoS effects by updating the user's trust value based on the traffic classification and blocking suspicious senders according to the trust value. However, the study does not provide enough details about the dataset that

was used to train and test the model, such as the specific types of attacks, their diversity, and the size and balance of the dataset. This could limit the model's generalizability and robustness to real-world scenarios with different attack types or data distributions. Furthermore, the study does not explore other possible features that could be relevant for DDoS detection in SDN, which may affect the model's accuracy and performance. Additionally there is a problem Independent deep learning models for control and data planes in the proposed approach raise concerns about potential synchronization issues and conflicting attack detections, hindering effective mitigation.

The paper[20] proposes an optimized artificial intelligence model one dimensional-convolutional neural network (1D-CNN) to detect and mitigate DDoS attacks in SDN environment. The model is trained with labeled network traffic data and tuned with non-dominated sorting genetic algorithm II (NSGA-II) to achieve the best accuracy with minimum training time. And this study reports that the model achieves significantly improved detection accuracy of 99.99% compared to other machine learning models. The paper also shows that the NSGA-II enhances the model accuracy with an improvement rate of 9.5%, 8%, 5.4%, and 2.6% when it is compared to logistic regression, random forest, support vector machine, and k-nearest neighbor optimized models respectively. The paper claims that the model can effectively detect and mitigate sophisticated DDoS attacks in SDN environments. However, the paper does not evaluate the model on other datasets or scenarios, which may limit the generalizability and robustness of the model to different types of attacks or data distributions. Even the papers claim it mitigate DDoS attack but it did not. And also, the paper does not evaluate the approach's ability to detect unknown or novel DDoS attacks.

The other study [21] presents a supervised learning model for detecting flooding DDoS attacks in SDN by monitoring the fluctuation of flows. The model uses the number of flows, packets, and bytes per time slot as features and compares them with a trained predictor. The paper shows that the model outperforms four other machine learning techniques in terms of accuracy, precision, recall, F1-score, and false alarm rates. However, the model uses only a single feature, the number of packets/bytes, to detect attacks. While the authors find that this is sufficient for their purposes, it is possible that using more features could improve performance. And also, the proposed approach is not effective to detect evolving attack

types. Moreover, the model could be enhanced by using more advanced machine learning techniques, such as Deep learning.

The study [22] proposes a deep learning-based approach that uses recurrent neural networks (RNNs) to detect distributed denial of service (DDoS) attacks on software-defined networking (SDN) controllers. The authors claim that their approach can overcome the limitations of traditional methods, such as high false positives, low accuracy, and inability to detect novel attacks. The approach consists of three stages: data preprocessing, cross-feature selection, and detection. However, the approach has some drawbacks. It relies on a fixed threshold to classify the network traffic as normal or DDoS, which may not be optimal for different scenarios or datasets. It also does not propose any mitigation policy to deal with the detected DDoS attacks, which reduces its practical value. Moreover, compared to other literature on the problem, the approach achieves low accuracy.

And the study [23] proposed model that combines Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) to detect Distributed Denial of Service (DDoS) attacks in Software-Defined Networking (SDN) environments. The system aims to improve the accuracy and performance of DDoS detection by using SHapley Additive exPlanations (SHAP) for feature selection and Bayesian optimization for hyperparameter tuning. The system is evaluated on two datasets, CICDDoS-2019 and InSDN, and achieves high accuracy rates compared to existing methods. However, the system does not provide any mitigation or prevention strategies for DDoS attacks, which are essential for enhancing the security and resilience of SDN environments. And also, the authors claim that they update the model periodically with the production data, but they do not specify how often they do so, what criteria they use to trigger the update.

The study [24] introduces an adversarial DBN-LSTM method for DDoS detection and defense in SDN settings. It integrates the predictive power of Deep Belief Networks (DBN) with the sequential data processing strength of Long Short-Term Memory (LSTM) networks, enhanced by the adversarial robustness of Generative Adversarial Networks (GANs). The method's primary advantage is its high efficiency in detecting prevalent DDoS attacks, validated by tests on the CICDDoS 2019 dataset. The combination of DBN and LSTM provides a deep learning model adept at identifying complex attack patterns and retaining crucial data points for accurate predictions. Despite its effectiveness, the model's complexity demands significant computational resources, which may affect scalability and

real-time deployment. It also focuses less on mitigation and prevention strategies, which are vital for comprehensive DDoS defense. The paper does not detail the model's update process with production data, a factor critical for maintaining long-term accuracy and relevance.

Table 2. Related Work Summary

| Related | Proposed | Dataset | Strength | Limitation | Year |
|---|---|---|---|---|---|
| [18] | A hybrid deep learning algorithm that combines CNN,GRU and DNN | Custom CICDDoS 2019 | High DDoS detection accuracy (reported as 99.81% and 99.88. and shows potential for low-rate attack detection | The model adds complexity costs, doesn't fully assess evolving DDoS patterns, and lacks real-world testing. | 2023 |
| [19] | It uses AEBGRU models for DDoS detection in SDN planes, leveraging traffic data and trust mitigation. | NSLKDD | DDoS detection with real-time adjustments. Combines traffic analysis, trust management, and unique feature extraction for high attack pattern sensitivity. | Limited data and potential for missed features raise concerns about real-world performance. Separate control/data models create synchronization challenges.. | 2024 |
| [20] | AI system utilizes a 1D-CNN and NSGA-II for DDoS detection in SDNs, using | Custom | optimized by NSGA-II for superior DDoS detection accuracy in SDNs over classic models | Its untested generalizability on diverse datasets and its unproven effectiveness | 2023 |

| | | | | | |
|---|---|---|---|---|---|
| | Mininet and the Ryu controller. | | | against novel or unknown DDoS attacks. | |
| [21] | Supervised learning to detect DDoS attacks in SDNs by analyzing network traffic patterns for irregularities. | 1999 DARPA and InSDN | Efficient detection of DDoS attacks in SDNs using a minimal feature set, which simplifies the model and reduces computational demands | The model's limited adaptability to new attack types and its potential for improved performance through the use of additional features and advanced techniques. | 2022 |
| [22] | By utilizing RNN It involves data preprocessing, cross-feature selection, and detection phases, employing a benchmark dataset for validation | Custom | Overcome the limitations of traditional methods, such as high false positives, low accuracy, | Fixed threshold for traffic classification not be universally effective, and the absence of a mitigation strategy. | 2023 |
| [23] | Integrates MLP and CNN to detect DDoS attacks in SDN, | CICDDoS -2019, InSDN | It demonstrates high accuracy in DDoS detection on CICDDoS-2019 | It Lacks DDoS mitigation strategies and lacks clarity on | 2023 |

| | | | | | |
|---|---|---|---|---|---|
| | utilizing SHAP for feature selection and Bayesian optimization for tuning. | | and InSDN datasets, outperforming existing methods. | the frequency and criteria for model updates with production data. | |
| [24] | Utilizes adversarial DBN-LSTM for DDoS detection in SDN, enhancing sensitivity and feature extraction with GANs. | CICDDoS 2019 | The DBN-LSTM model improve DDoS attack detection accuracy by being trained with adversarial examples. | Require significant data and resources; defense against new DDoS strategies not fully assessed | 2023 |
| [25] | (DDNN) model to analyze network traffic data and identify patterns indicative of DDoS attacks. | CICDDoS 2019 | Feature-Rich Model: The DNN model considers a large number of feature values, leading to accurate classification. | The lack of details about the dataset makes it difficult to assess the model's generalizability to real-world scenarios with potentially unseen attack patterns | 2023 |

### 2.5.1. Summary of Related works gaps

Existing DDoS detection methods proposed by the literature (studies [18]-[24]) in SDN suffer from various limitations. Firstly, their reliance on irrelevant or inadequate datasets hinders their real-world applicability. Secondly, resource inefficiency poses scalability challenges. Thirdly, the lack of thorough investigation into different thresholding techniques impacts their adaptability. Fourthly, models trained solely on known attack types cannot guarantee effective detection of novel or unknown attacks. Lastly, the absence of real-world testing and reliance on limited features limits their overall effectiveness. Our proposed research aims to address these gaps and enhance DDoS detection reliability in SDNs.

CHAPTER THREE

# 3. Research Methodology

The research employed various methodologies to achieve the general and specific objectives of the study. These methodologies, centered around the application of deep learning techniques, aim to significantly enhance DDoS attack detection and mitigation capabilities within SDN infrastructures. This research methodology is designed as a systematic seven-step process.

**Review of Literature:** The research begins with a thorough literature review, focusing on SDN-based DDoS and deep learning solutions. This step is crucial for identifying gaps in the current state of knowledge and setting a foundation for the study.

**Defining Problems:** After the literature review, specific problems within the domain of SDN-based DDoS attacks are defined. These problems guide the direction of the research and ensure that the study addresses relevant and significant issues.

**Set Questions and Objectives:** With the problems defined, the next step is to formulate research questions and establish clear objectives. This phase sets the stage for a targeted investigation, providing a roadmap for the research.

**Two Stage Deep Learning Solution:** The fourth step involves proposing a novel two-stage deep learning solution. This solution leverages both supervised and unsupervised learning methods to enhance attack detection and mitigation.

**Dataset Collection and Analysis:** In this stage, the research focuses on selecting appropriate attack types, constructing an SDN environment, and gathering new datasets. These datasets, along with public benchmark datasets, are used for training and validation. Data preprocessing, including cleaning and balancing, is performed to ensure the integrity of the model.

**Implementation:** The sixth step is the practical application of the proposed deep learning solution. This includes training the model and deploying a mitigation strategy within the SDN infrastructure.

**Results and Discussion**: The final step involves evaluating the model using appropriate metrics, demonstrating the feasibility of the deep learning solution, and discussing the

effectiveness of the mitigation strategy. The results are interpreted to provide insights into the challenges and potential of using deep learning for SDN-based DDoS attack mitigation.

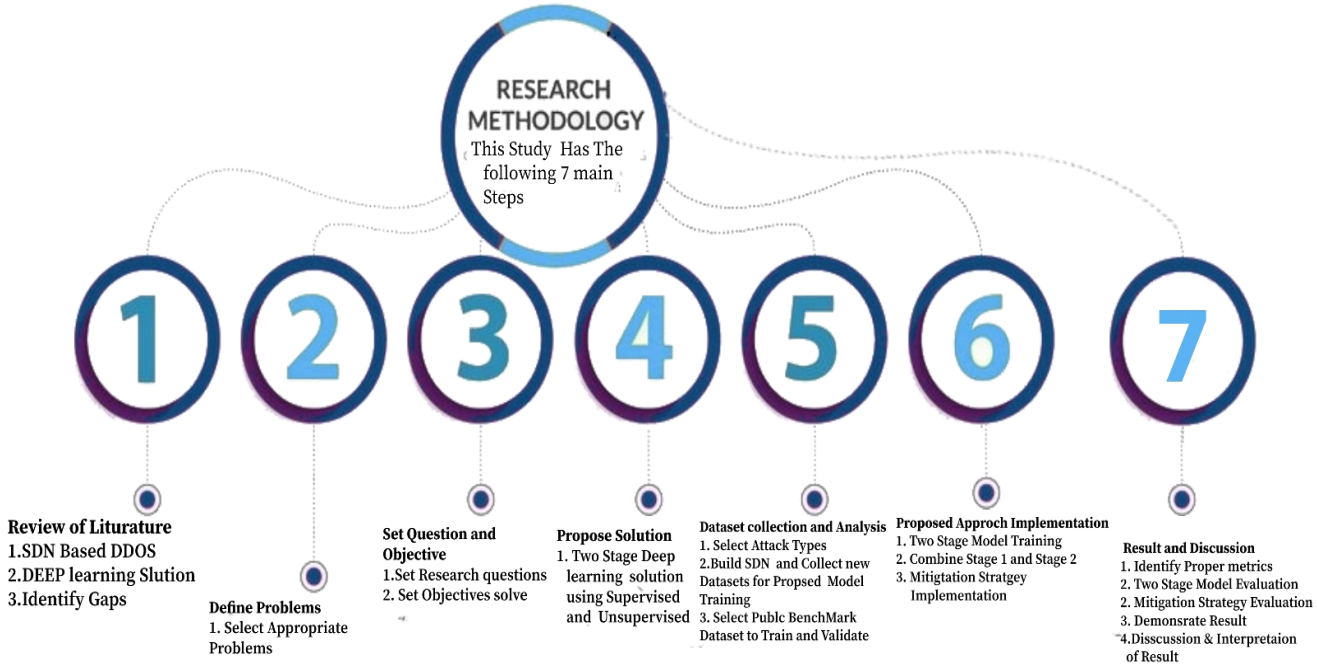



Figure 6. Systematic Research Methodology

### 3.1. Dataset Collection and Preprocessing

#### 3.1.1. Identifying Critical DDoS Attacks on SDN

Following an in-depth examination of various scholarly articles and the latest trends in Distributed Denial of Service (DDoS) attacks, it has been observed that certain types of attacks have emerged as the most serious threats. These include TCP SYN Flooding, UDP Flooding, ICMP Ping Flooding, Slowloris Attack, LAND Attack, DNS Amplification, and NTP Amplification attacks. These specific types of DDoS attacks have been identified as the most severe due to their potential to cause significant disruption and damage to network systems.

**ICMP flood attack** In this scenario, a significant amount of ICMP packet requests are directed towards the targeted individual. When the transmitted data exceeds 65535 bytes, the target's host computer ceases to function. An ICMP flood attack could also have serious implications for an SDN. In this type of attack, the attacker sends a flood of ICMP requests to a target server. In an SDN, this could cause the network controller to become overwhelmed with connection requests. Since the controller is responsible for managing all

network connections, this could lead to a denial of service where legitimate network requests cannot be processed[26], [27].

**UPD flood attack:** In SDN architecture, when a new flow arrives, the SDN switch sends a packet-in message to the SDN controller. The controller then decides where the packet should go. Once the switch learns the decision, it does not need to query the controller again. However, an adversary can abuse this system by constantly sending packets to the SDN switch with randomly filled destination addresses. This forces the controller to continuously process UDP address-resolution-protocol (ARP) packets, which can be thought of as a UDP flooding attack. During a UDP flooding attack, the host controlled by the attacker sends a large amount of UDP traffic to a random communication port on the target machine. This forces the machine to check applications listening on these communication ports and continuously reply, which can lead to system failure or full bandwidth occupation of the target machine[27], [28].

**TCP SYN DDoS attack:** is a type of Distributed Denial of Service (DDoS) attack that exploits the TCP three-way handshake mechanism. The attacker sends a large number of TCP SYN packets to the target server, using spoofed or random source IP addresses. The server allocates resources for each SYN packet and sends back a SYN-ACK packet, waiting for the final ACK packet from the client. However, the ACK packet never arrives, as the attacker either ignores the SYN-ACK packet or sends a RST packet to reset the connection. This causes the server to keep the half-open connections in a backlog queue, eventually exhausting its resources and preventing legitimate clients from establishing connections. Software Defined network is vulnerable to TCP SYN DDoS attacks, as the SDN controller is a critical point of failure and a potential target for attackers. Moreover, the SDN switches forward the attack packets to the controller, overloading it with control messages and reducing its performance[29], [30].

**Slowloris attack**: is a type of low and slow DDoS attack that aims to make a server unresponsive by opening and maintaining many partial HTTP connections to the target. This exhausts the server’s resources and prevents legitimate clients from accessing the service. In a software defined network (SDN) environment, Slowloris can also affect the performance of the SDN controller, which is responsible for managing the network flows[31].

**DNS amplification attack:** is a type of reflection-based DDoS attack that exploits the vulnerability of the DNS protocol to amplify the traffic sent to the victim. In this attack, the attacker sends spoofed DNS queries with the victim's IP address as the source to public DNS servers, which then reply with large DNS responses to the victim, overwhelming its network bandwidth and resources. DNS amplification attack is a serious threat to the Internet, as it can generate traffic up to several hundred times larger than the original query, and can target any IP address without prior knowledge. DNS amplification attack is also difficult to trace and stop, as the attacker can use multiple DNS servers and change the query type and content[32], [33], [34].

**NTP amplification attack:** is a type of DDoS attack that exploits the Network Time Protocol (NTP) to generate large amounts of traffic and overwhelm a victim's network. It works by sending spoofed requests to NTP servers with the source IP address of the target, causing the servers to send back amplified responses to the target. This can cause SDN to suffer from network congestion, packet loss, and service degradation for the victim[35].

### 3.1.2. SDN Simulation For Data Collection

The simulations were effectively carried out using the Mininet emulator and Ryu controller. Mininet, a network emulator based on Linux, enables network engineers to construct a realistic virtual Software-Defined Network (SDN). It generates virtual hosts, switches, and links, supports the OpenFlow protocol, and runs actual kernel and application codes on a machine. Interaction with the simulated network is facilitated through a Command Line Interface (CLI) and an API. Mininet's capabilities are further enhanced by Miniedit, which allows for the creation and configuration of network topologies, as well as the creation, interaction, and customization of prototypes for SDN applications. Mininet is recognized for its ease of learning and cost-effectiveness as a network testbed. Its installation process is straightforward, achievable through simple commands or by using a pre-built virtual machine. Additionally, it offers a Python API, supports integration with real-world environments, and integrates seamlessly with well-known SDN controllers. Dataset preprocessing is an integral part of this setup, ensuring that the data used in simulations is representative and accurately configured for the SDN environment.

In order to gather the necessary dataset, a comprehensive simulation was conducted. This simulation was structured around a network of six OpenFlow switches, each of which was

interconnected. The network was further expanded with the inclusion of 18 hosts, with each switch being directly connected to three of these hosts.

The simulation was executed in a controlled environment provided by CloudLab.us. This platform offered the necessary resources and infrastructure to ensure the successful completion of the simulation.

The network emulator, Mininet, was run on a machine with Ubuntu 22.04 as its operating system. This machine was equipped with substantial computational resources, including 192GB of RAM and Two Intel Xeon Silver 4114 10-core CPUs running at 2.20 GHz, to ensure the smooth running of the simulation and the accurate collection of data. This emulator contain 6 OpenFlow switch and each OpenFlow switch connected with 3 host

Similarly, the controller for the network was also run on a machine with Ubuntu 20.04. This machine, like the one running Mininet, was well-resourced with 192GB of RAM and and Two Intel Xeon Silver 4114 10-core CPUs running at 2.20 GHz. This ensured that the controller could effectively manage the network and contribute to the successful collection of the dataset. The following figure shows Environment for traffic generation and collect traffic as Dataset.

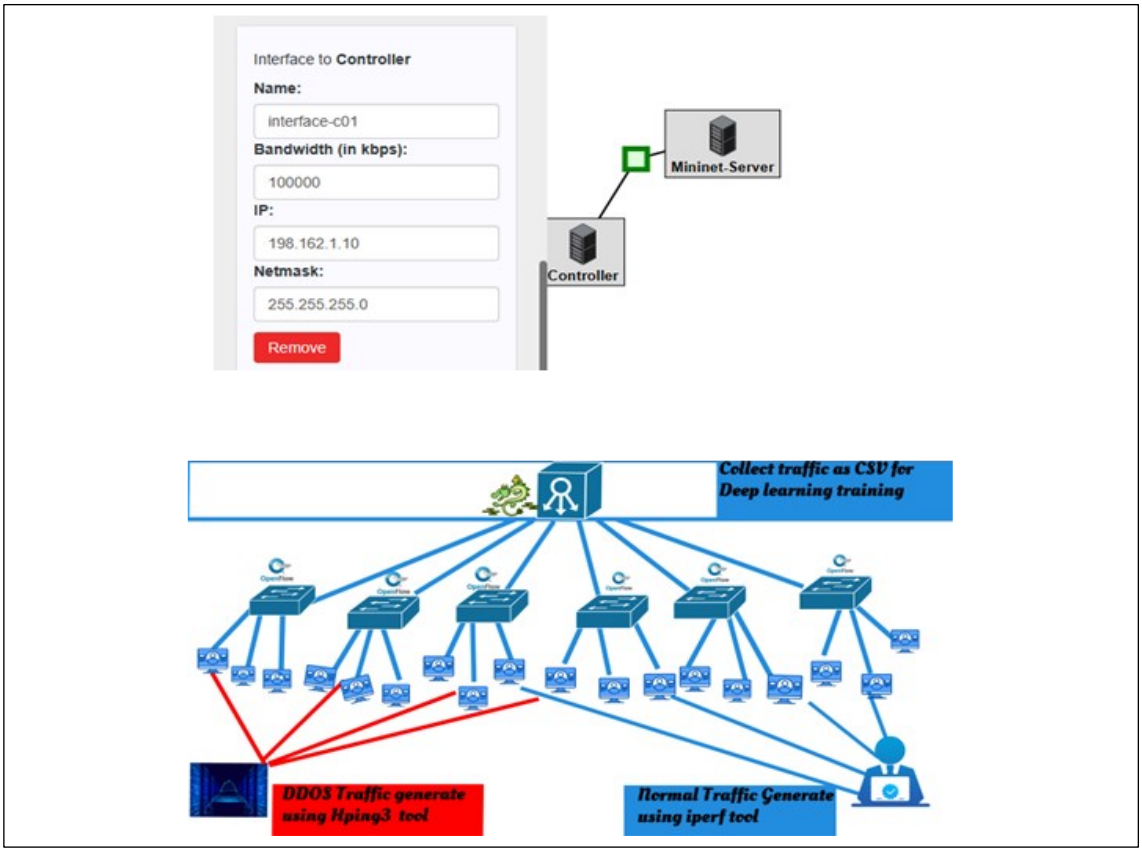



Figure 7. Data Collection SDN Simulation

One of the main objectives of this experiment is to generate realistic traffic patterns that can be used to test the performance and security of the SDN environment. To achieve this, two types of traffic are generated: normal traffic and DDoS traffic.

To generate normal traffic, various communication patterns have been executed between hosts that mimic real-world communication scenarios, such as browsing, streaming, downloading, etc. These patterns can be achieved by having hosts send requests to each other using common protocols, such as HTTP, FTP, SMTP, etc. The frequency and size of the requests can be varied to create different traffic loads and patterns**.**

To generate DDoS traffic, a high volume of traffic from multiple hosts should be generated using various tools, such as ‘‘iperf’’ and ‘‘hping3.’’ These tools can create different types of packets and send them at a high rate to the target. A custom script was used to simulate attack traffic for ICMP, UDP, and TCP protocols, by specifying the source and destination IP addresses, ports, packet size, and rate. During the DDoS traffic generation, the experiment generated main types of DDoS attacks, such as UDP flood, TCP SYN flood, ICMP ping flood, LAN attack, Slowloris attack, NTP amplification, and DNS amplification attack. These attacks aim to exhaust the network bandwidth, server resources, or application layer services of the target. ***(the specific commands used for this are included in the Appendix E)***

3.1.4. Collect Traffic

Monitoring and Logging: Another important objective of this experiment is to monitor and log the network behavior and performance under different traffic conditions. To do this, a standalone application runs inside the Ryu controller, which is the software component that controls the SDN environment. The application monitors the global topology construction, which is the process of discovering and connecting the network devices, such as switches, routers, hosts, etc. The application also logs different OpenFlow switch flows statistics on a regular basis, such as the number of packets, bytes, duration, actions, etc. The OpenFlow switch flows are the rules that define how the switches forward the packets based on their header fields, such as source and destination IP addresses, ports, protocols, etc. The OpenFlow switch flow details and requests to the SDN controller were written into a comma-separated values (CSV) file, which can be used for further analysis and visualization. The CSV file contains information such as the timestamp, switch ID, flow ID, match fields, actions, packet count, byte count, duration, etc

Algorithm 1: Traffic Collection for Dataset

---

**Input**: Network traffic data from SDN switches

**Output:** Traffic data records stored in a CSV file

**1:** Start the traffic monitoring thread on the SDN controller.

2: Continuously monitor the network at predefined intervals.

3: **for** each Datapath (switch) in the SDN network **do**

4: Send a request to the Datapath to retrieve flow statistics.

5: Receive the flow statistics reply from the Datapath.

6: **for** each flow entry in the flow statistics **do**

**7:** Extract traffic features.

8: Calculate packet other features.

9: Generate a unique flow identifier based on the traffic features.

10: Write the extracted data and calculated metrics to a **CSV file.**

**11: end for**

**12: end for**

13: Repeat the process at the next interval.

---

3.1.5. Dataset Description

As the Appendix E shows, the selected features are relevant and efficient for DDoS detection, as they focus on attack signatures that capture essential characteristics of network traffic. They also provide comprehensive attack coverage, as they address diverse attack types, such as high-volume, protocol-specific, short-lived, long-lived, and port-specific attacks. Moreover, they enable flow-based detection, which allows granular analysis and context-aware behavior of individual flows. Finally, these columns are important for machine learning, as they are often identified as highly relevant features for building effective models that can distinguish between normal and DDoS traffic**.** ***(Detail of Each Feature Appendix E)***

### 3.1.6. Dataset Preprocessing

**Data cleaning:** is a crucial preprocessing step in data analysis. This process involves meticulously reviewing the dataset to correct or remove any data that is duplicated, corrupted, improperly formatted, or incomplete. The integrity of a dataset is paramount, as it directly impacts the accuracy and reliability of any subsequent analysis. Ensuring clean data means that the algorithms applied later can work with the best possible information, leading to more valid results.

**Data Transformation**: The transformation of data is another vital step in preparing data for analysis, particularly when working with Deep Learning (DL) algorithms. This process entails changing the data from one format or structure into another. A typical example of this would be the conversion of string data into numerical data, which is often necessary for DL algorithms to process the information effectively. This step ensures that the data is in the right form and structure for the algorithms to perform optimally.

**Data Balancing:** In the dataset preparation phase, data balancing is a critical step to ensure that machine learning models are not biased towards more frequently occurring classes. Initially, the dataset showed a significant imbalance with a high number of instances for certain types of DDoS attack traffic compared to benign traffic. Specifically, there were 729,334 instances of Slowloris Attack, 637,700 of TCP SYN Flood, 617,528 of DNS Amplification, 613,062 of ICMP Ping Flood, 599,248 of UDP Flood, 515,000 of NTP Amplification, 2,152 of Land DDoS Attack, and 128,966 of Benign traffic. To address this challenge, resampling techniques and synthetic data generation methods like the Synthetic Minority Over-sampling Technique (SMOTE) were utilized.

### 3.1.7. Benchmarking Public Datasets

**CICDDoS2019 Dataset:** The CICDDoS2019 dataset is a comprehensive collection designed for the evaluation of Distributed Denial of Service (DDoS) attack detection systems. Developed by the Canadian Institute for Cybersecurity, it addresses the shortcomings of previous datasets by including a wide range of DDoS attacks, both benign and malicious, that reflect real-world data. The dataset contains network traffic analysis results labeled with timestamps, source and destination IPs, ports, protocols, attack types and other features, making it a valuable resource for developing and testing DDoS detection[36].

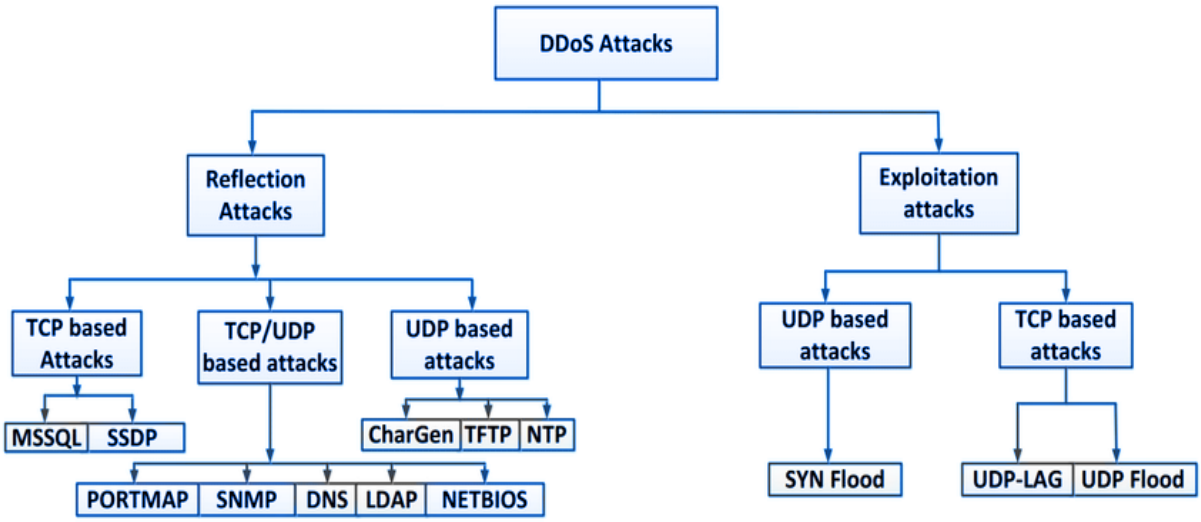



Figure 8. CICDDoS2019 Dataset Classification

**InSDN Dataset:** The InSDN dataset, a novel intrusion dataset designed specifically for Software-Defined Networks (SDN) environments, addresses the lack of publicly available data for anomaly detection systems in these networks [37]. This comprehensive dataset includes benign traffic alongside various attack categories that target different elements of the SDN platform (reference the paper again for specific details). It categorizes network traffic into different classes to facilitate intrusion detection system development. One class represents normal traffic, which is regular network activity without any malicious intent. The other classes encompass various attack categories, such as Denial-of-Service (DoS) attacks that disrupt services by flooding the network, Remote to Local (R2L) attacks that exploit vulnerabilities for unauthorized access, User to Root (U2R) attacks that escalate user privileges, and Probe attacks that scan networks for weaknesses. By understanding these different traffic types, researchers can design intrusion detection systems that can effectively safeguard SDN environments.

CHAPTER FOUR

# 4. Proposed Approach Design

This study presents an innovative two-stage approach detailed in Figure 7. Combining the strengths of supervised and unsupervised learning methodologies, the approach strengthens the SDN against both known and Unknown DDoS attack patterns.

## 4.1. Stage 1: Supervised Deep Leaning Known DDoS Detection

The first stage of the approach is a supervised multi-label classifier that uses a deep learning model to detect and classify DDoS attacks. This detector and classifier trained on and diverse dataset that covers various types and intensities of DDoS attack scenarios. The classifier can accurately identify and categorize known DDoS attack patterns based on the features and labels of the data. This stage acts as a preventive shield, filtering out DDoS traffic class from normal network behavior.

This stage has the benefit of precision in known DDoS attack identification: the supervised multi-label classifier excels in precisely recognizing and categorizing known DDoS attack patterns. By leveraging labeled data, it offers a proactive defense against established threats. It also has resource efficiency by focusing exclusively on the identification of malicious traffic, which optimizes resource utilization and ensures a targeted and efficient response to known DDoS attacks.

## 4.3. Stage 2: Unsupervised Deep learning Unknown DDoS Detection

Following the initial classification, the second stage introduces an unsupervised unknown DDoS detection mechanism. Leveraging an advanced Autoencoder architecture, this stage scrutinizes network traffic predicted as normal by the first classifier. Operating without prior knowledge of labeled DDoS class, the unsupervised approach of this stage is adept at detecting subtle deviations indicative of novel or evolving DDoS attack patterns. This layer of adaptability provides a robust defense against emerging threats.

This stage has benefit on Sensitivity to Novel DDoS Attack Patterns: The unsupervised DDoS detection stage demonstrates heightened sensitivity to novel or unknown DDoS attack patterns by scrutinizing ostensibly normal traffic. Dynamic Adaptability: Operating in an unsupervised manner, this stage dynamically adapts to emerging threats, ensuring the security framework remains resilient in the face of evolving DDoS attack landscapes.

The advantage of this Stage lies in its ability to identify subtle and previously unknown patterns that may indicate anomalies. By training the model on normal traffic, it develops a baseline of what constitutes typical network behavior. When the model encounters data that deviates from this baseline, it can flag these instances as potential anomalies. This is particularly useful for detecting new types of attacks or unusual activities that have not been seen before.

Testing the model on a combination of benign and anomalous traffic allows for a realistic assessment of its detection capabilities. It ensures that the model is not only good at recognizing normal behavior but also effective at identifying and differentiating between benign variations and genuine threats. This balanced testing approach helps in fine-tuning the model's sensitivity to anomalies, reducing the likelihood of false positives (benign activities mistaken as threats) and false negatives (actual threats not detected), which are critical metrics in the performance of any anomaly detection system. Overall, this method enhances the model's practical utility in real-world cybersecurity applications where the nature of network traffic is constantly evolving.

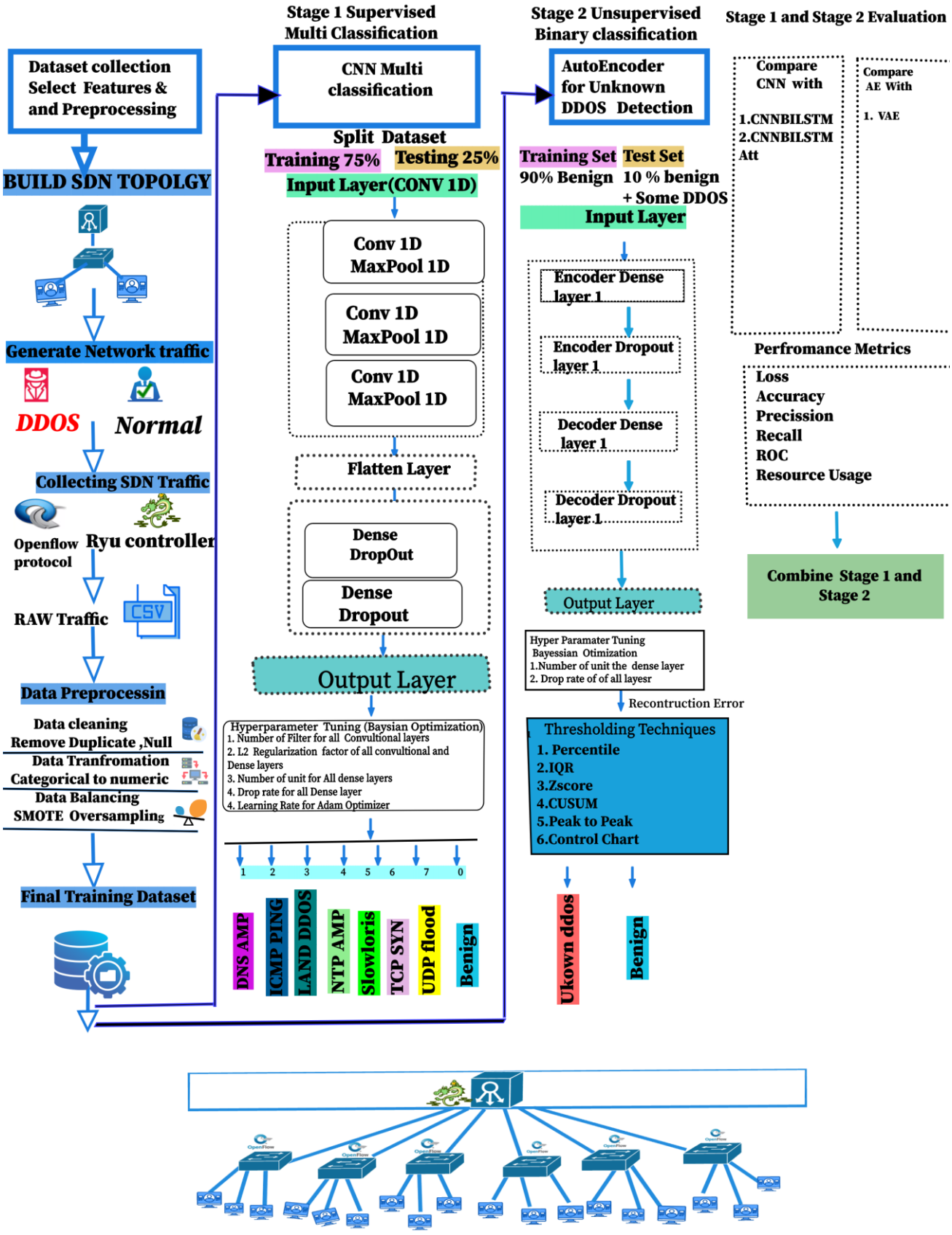



Figure 9. Proposed Approach Design

### 4.4. Combining Stage 1 and Stage 2 Deep Learning:

This two-stage approach leverages two models for enhanced DDoS detection. The first stage utilizes a deep convolutional neural network (CNN) for multi-class classification. This CNN analyzes the test data, assigning each sample to a specific class, including a potential "benign" class for normal traffic.

The second stage focuses on refining unknown DDoS detection within "benign" samples identified by the CNN. Here, an Autoencoder attempts to reconstruct the input data. By comparing the original and reconstructed data, the Autoencoder identifies significant discrepancies that might indicate unknown DDoS attacks, potentially missed by the initial classification. These discrepancies are quantified using a specific threshold, leading to the final classification of each sample as either normal or "unknown DDoS."

### 4.5. Stage 1 Supervised DDoS Detection Model Selection

Three deep learning models based on CNN, CNNBiLSTM, CNNBiLSTMAtt architectures were selected for this stage. These models are widely used and effective for various sequence learning tasks, such as natural language processing, speech recognition, and time series analysis[38], [39] . These models were chosen because they can capture the temporal and spatial dependencies in the network traffic data, and learn complex and nonlinear patterns for DDoS detection. The rationale behind choosing each model is briefly described as follows**:**

**CNN:** Convolutional neural networks (CNNs) are composed of multiple layers of neurons that apply convolution operations to the input data. CNNs can extract local and global features from the data, and reduce the dimensionality and complexity of the data. CNNs are suitable for DDoS detection because they can handle high-dimensional and heterogeneous network traffic data, and learn discriminative features for normal and DDoS traffic[40], [41]**.**

**CNN-BiLSTM:** The CNN-BiLSTM hybrid neural network integrates the features of a convolutional neural network (CNN) and a bidirectional long short-term memory neural network (BiLSTM). In this model, the CNN captures local spatial patterns in the input data, while the BiLSTM processes sequences bidirectionally, capturing temporal dependencies. For well log prediction, the CNN extracts relevant features from the log data, and the

BiLSTM models the sequential nature of the logs. By combining these two components, the CNN-BiLSTM aims to enhance the accuracy of well log predictions[42]

**CNN-BiLSTM with Attention:** The CNN-BiLSTMAtt with attention model extends the hybrid architecture by incorporating an attention mechanism. Attention allows the model to dynamically focus on specific parts of the input sequence when making predictions. In the context of well logs, attention helps weigh the importance of different log features. By attending to relevant features, the model can make more informed predictions, especially when dealing with complex and noisy well log data. The combination of CNNs, BiLSTMs, and attention mechanisms in the CNN-BiLSTM with attention model aims to improve well log prediction performance[42].

### 4.6. Stage 2 Unsupervised DDoS Detection Model Selection

Two models were selected for this stage: Autoencoders (AEs) and variational Autoencoders (VAEs). These models are chosen because they have the potential to detect unknown anomalies or Distributed Denial-of-Service (DDoS) attacks within network traffic data using an unsupervised learning approach[43].

**Autoencoder :** is a type of neural network that can learn a compressed representation of the input data, and then reconstruct the original data from the compressed representation. It consists of two parts: an encoder that maps the input to a latent code, and a decoder that maps the latent code back to the output. Autoencoder can be used for dimensionality reduction, denoising, feature extraction, and generative modeling[44].

**Variational Autoencoder (VAE)**: is a variant of Autoencoder that introduces a probabilistic aspect to the latent code. Instead of encoding the input to a single point in the latent space, VAE encodes the input to a distribution over the latent space, and then samples a point from that distribution to decode. This allows VAE to capture the uncertainty and variability of the data, and also to generate new data by sampling from the prior distribution over the latent space. VAE can be seen as a combination of Autoencoder and variational inference, where the encoder acts as an inference model and the decoder acts as a generative model. VAE has many applications in image synthesis, text generation, anomaly detection, and more[44].

4.6.1. Autoencoder Model Thresholding Techniques

A key element in employing Autoencoders for anomaly detection is the choice of a suitable threshold. This threshold is established by examining the reconstruction error distribution for data considered normal. Data points that yield a reconstruction error surpassing this threshold are identified as outliers or anomalies[46]. This study applies different threshold calculation techniques and systematically uses Bayesian optimization to calculate and find the best threshold values. These techniques include Percentile, Z-Score, CUSUM, IQR, Peak-to-Peak, and Control Chart methods.

### 4.7. Mitigation Approach

By using a **pre-trained two stage deep learning model** to predict the behavior of the flow. The deep learning model is a program that learns from data and makes predictions based on patterns. If the prediction is "DDoS /unknown class", which means that the flow is known DDoS or unknown DDoS. the prediction information DDoS attack is provide to for user. The port is the point where the flow enters or leaves the switch. Then start process also identifies which switch port is related to the suspicious flow. The SDN controller then executes the action to block the port of the switch. Action to block the port. The port is the point where the flow enters or leaves the switch.

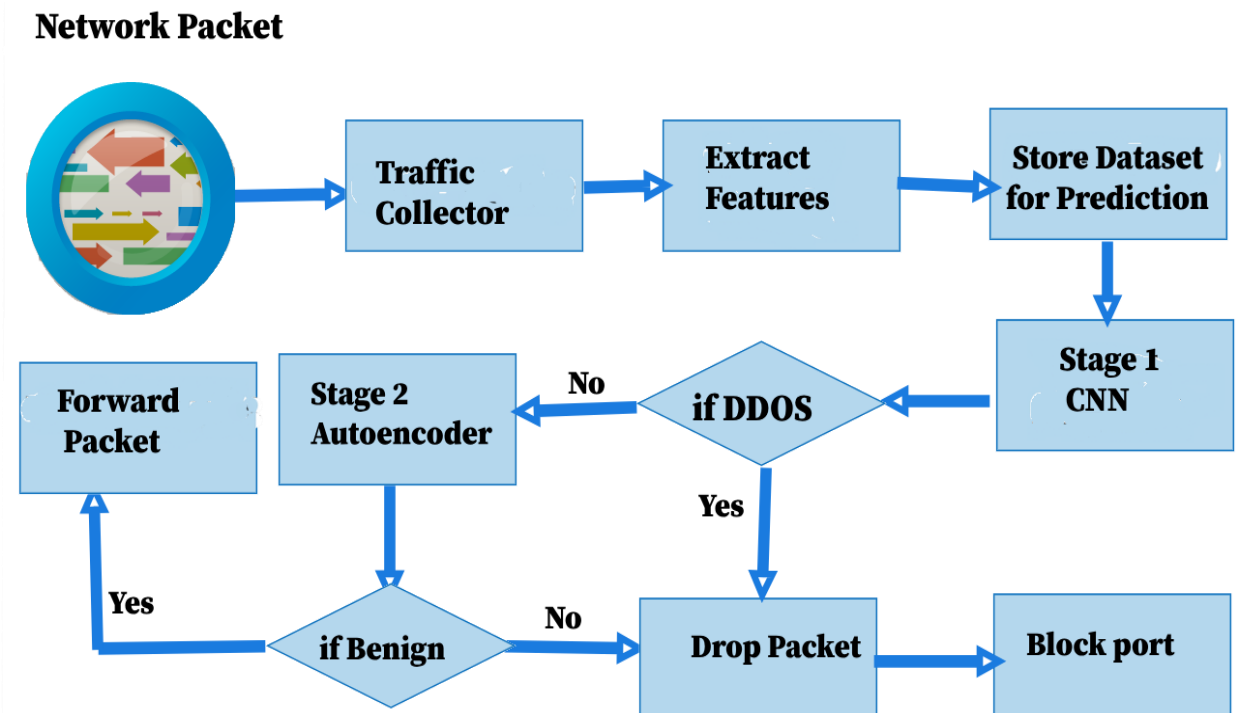



Figure 10. Mitigation Approach

CHAPTER FIVE

# 5. Experiment And Implementation

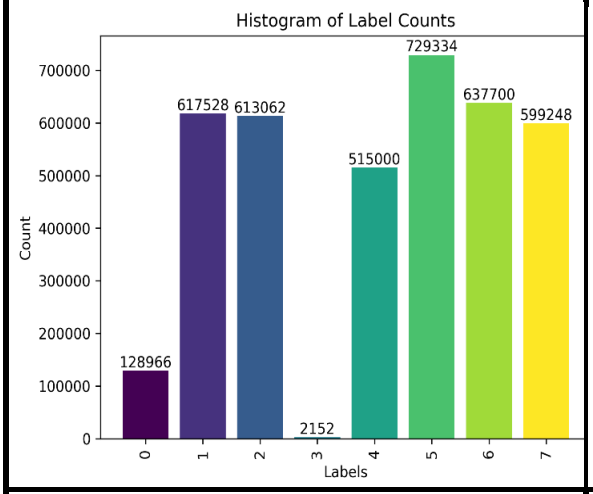
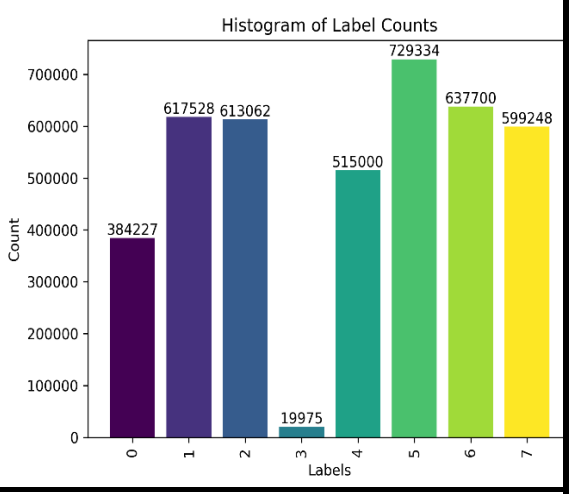
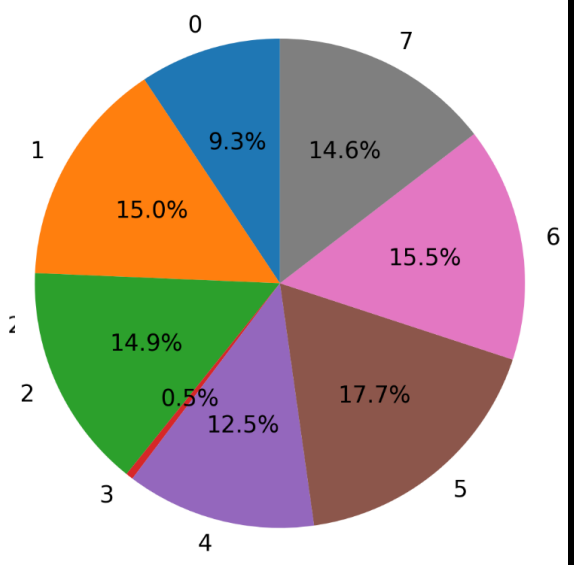
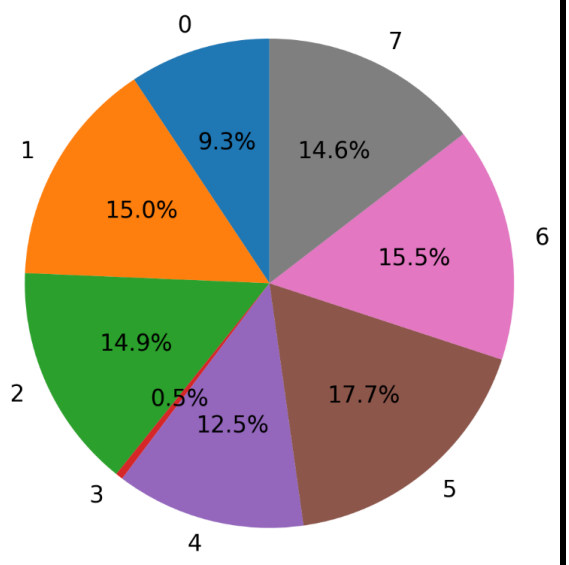
## 5.1. Dataset Preprocessing Experiment

**Data cleaning:** is a crucial preprocessing step in data analysis. This process involves meticulously reviewing the dataset to correct or remove any data that is duplicated, corrupted, improperly formatted, or incomplete. The integrity of a dataset is paramount, as it directly impacts the accuracy and reliability of any subsequent analysis. Ensuring clean data means that the algorithms applied later can work with the best possible information, leading to more valid results.

**Data Transformation**: The transformation of data is another vital step in preparing data This step ensures that the data is in the right form and structure for the algorithms to perform optimally. To simplify analysis, attack types were converted numerically: 0=Benign, 1=DNS Amplification, 2=ICMP Ping Flood, 3=Land DDoS, 4=NTP Amplification, 5=Slowloris, 7=TCP SYN Flood

**Data Balancing:** SMOTE was particularly used to improve the representation of the minority classes, which in this case were Benign and LAND DDoS attacks. After the application of SMOTE, the dataset achieved a more balanced distribution, with the number of instances for Slowloris Attack remaining at 729,334, TCP SYN Flood at 637,700, DNS Amplification at 617,528, ICMP Ping Flood at 613,062, UDP Flood at 599,248, NTP Amplification at 515,000, Land DDoS Attack increasing to 19,975, and Benign traffic significantly rising to 384,227 instances. This balanced dataset is crucial for training machine learning models that can generalize well to new, unseen data. The following tables display the dataset before and after cleaning and balancing.

Table 3. Dataset Before and After Cleaning and Balancing

| Dataset Distribution Before Cleaning and Balancing | Dataset Distribution After Cleaning and Balancing |
|---|---|
|  |  |
|  |  |

### 5.2. Experiment of Two stage Model Training

In this implementation, supervised and unsupervised models are trained as stage 1 and stage 2 respectively. The second stage is trained by using only benign dataset and tested with both benign and DDoS dataset, because the second stage focuses on perfectly training only normal dataset so that it can detect unknown DDoS attacks in the future.

5.2.1. Experiment Setup Two stage Model Training

The experiment was conducted on a powerful PowerEdge C4130 node of the Clemson cluster on cloudlab.us. This high-performance server is designed for accelerated computing and high-density deployments, making it ideal for demanding workloads. The C4130 boasts impressive specifications, including a powerful 2 x 2.50 GHz Intel Xeon CPU E5-2680 v3 processor with 48 logical processors, a substantial 256 GiB of DDR4 RAM for smooth operation, and dual 1 TB hard drives for storage. Additionally, a pair of 12 GiB Tesla K40m GPUs provides significant graphics processing power, perfect for tasks requiring intensive graphical computations.***(See Experimental Setup Topology Detail Appendix G)***

Table 4. Two stage model training Environment

| **Hardware** | **Size** | **Type** |
|---|---|---|
| **RAM** | 256 GiB | DDR4 Synchronous Registered (Buffered) |
| **CPU** | 2 x 2.50 GHz | Intel® Xeon® CPU E5-2680 v3 |
| **vCPU** | 48 | Logical processors |
| **Cache** | 2 x 30 MiB L3, 2 x 3 MiB L2, 2 x 768 KiB L1 | Memory cache |
| **ROM** | 64 KiB | BIOS |
| **NIC** | 4 x 10 GbE | Ethernet Controller X710 for 10GbE SFP+ |
| **GPU** | 2 x 12 GiB | GK110BGL [Tesla K40m] |
| **Disk** | 2 x 1 TB | 7.2K RPM 3G SATA HDDs |

5.2.2. Stage 1 Experiment Detail

The first-stage supervised model is trained using a dataset split into 75% for training and 25% for testing, as illustrated in Figure To ensure a fair comparison and select the best model, a CNN is trained with hyperparameter tuning using Bayesian optimization and Keras Tuner. Subsequently, the two remaining CNN variants, CNN-BiLSTM and CNN-BiLSTMAtt, are trained using the same architecture and hyperparameters. Finally, various metrics are calculated and analyzed to evaluate the performance of each model.***(Sample Code of this See in Appendix A)***

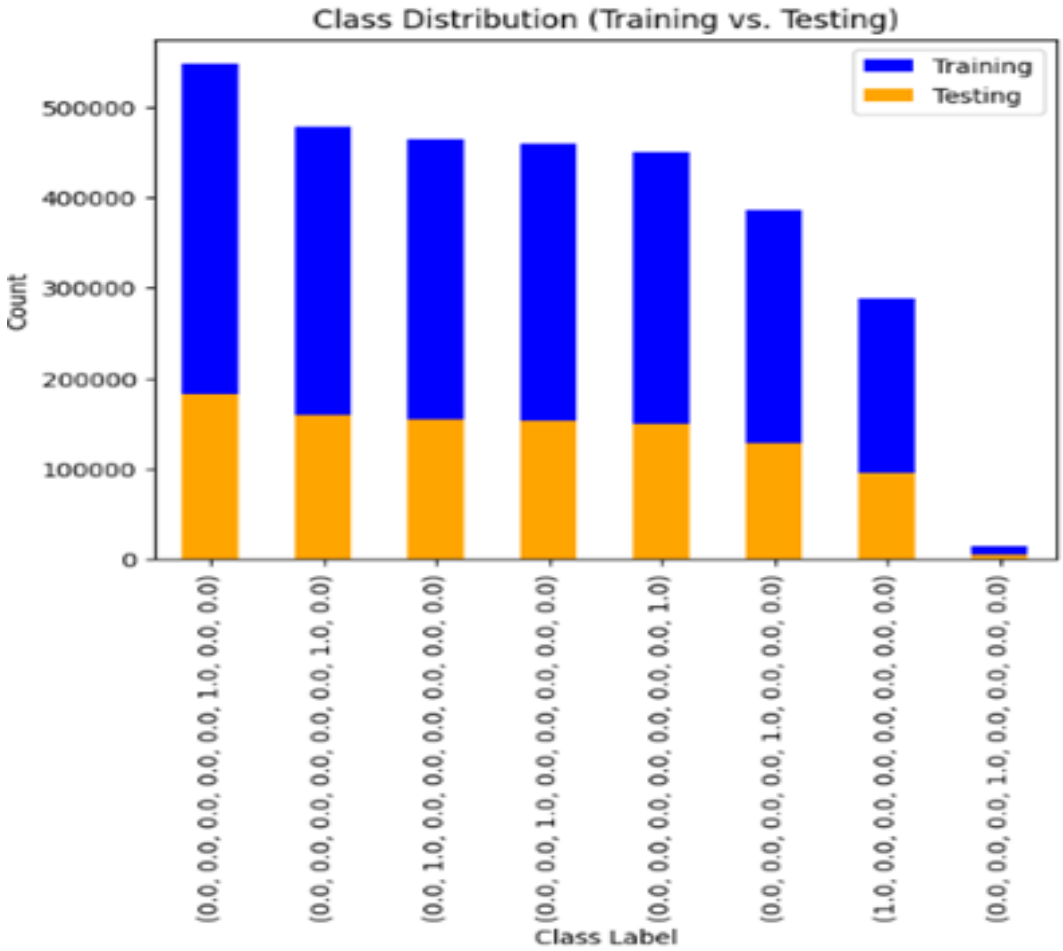



Figure 11. Stage 1 Training nd Testing Data Split

Table 5. Stage 1 Hyperparameter Tuning

| No | Hyperparameter | Search Space | | Optimized |
|---|---|---|---|---|
| | | Minimum | Maximum | |
| 1 | Number of filters in convolutional layer 1 | 64 | 256 | 96 |
| 2 | L2 regularization factor for convolutional layer 1 | 0.000001 | 0.01 | 0.0016168 |
| 3 | Number of filters in convolutional layer 2 | 64 | 256 | 96 |
| 4 | L2 regularization factor for convolutional layer 2 | 0.000001 | 0.01 | 0.0000048 |
| 5 | Number of filters in convolutional layer 3 | 64 | 256 | 128 |
| 6 | L2 regularization factor for convolutional layer 3 | 0.000001 | 0.01 | 0.0000025 |
| 7 | Number of units in dense layer 1 | 64 | 512 | 64 |
| 8 | L2 regularization factor for dense layer 1 | 0.000001 | 0.01 | 0.0000094 |
| 9 | Dropout rate for dense layer 1 | 0.3 | 0.7 | 0.5 |
| 10 | Number of units in dense layer 2 | 32 | 256 | 224 |
| 11 | L2 regularization factor for dense layer 2 | 0.000001 | 0.01 | 0.0044082 |
| 12 | Dropout rate for dense layer 2 | 0.3 | 0.7 | 0.5 |
| 13 | Learning rate for Adam optimizer | 0.0001 | 0.01 | 0.004224 |

5.2.3. Stage 2 Experiment Detail

The second stage is trained by using only benign dataset and tested with both benign and DDoS dataset, because the second stage focuses on perfectly training only normal dataset so that it can detect unknown DDoS attacks in the future.

To compare and select the unsupervised model fairly, a fully connected convolutional neural network Autoencoder (AE) model is trained with hyperparameter tuning using Bayesian optimization and Keras Tuner. Then, a variational Autoencoder (VAE) model is

trained with the same architecture and hyperparameters. Finally, different metrics are calculated and generated to evaluate the performance of both models. *(See Appendix B)*

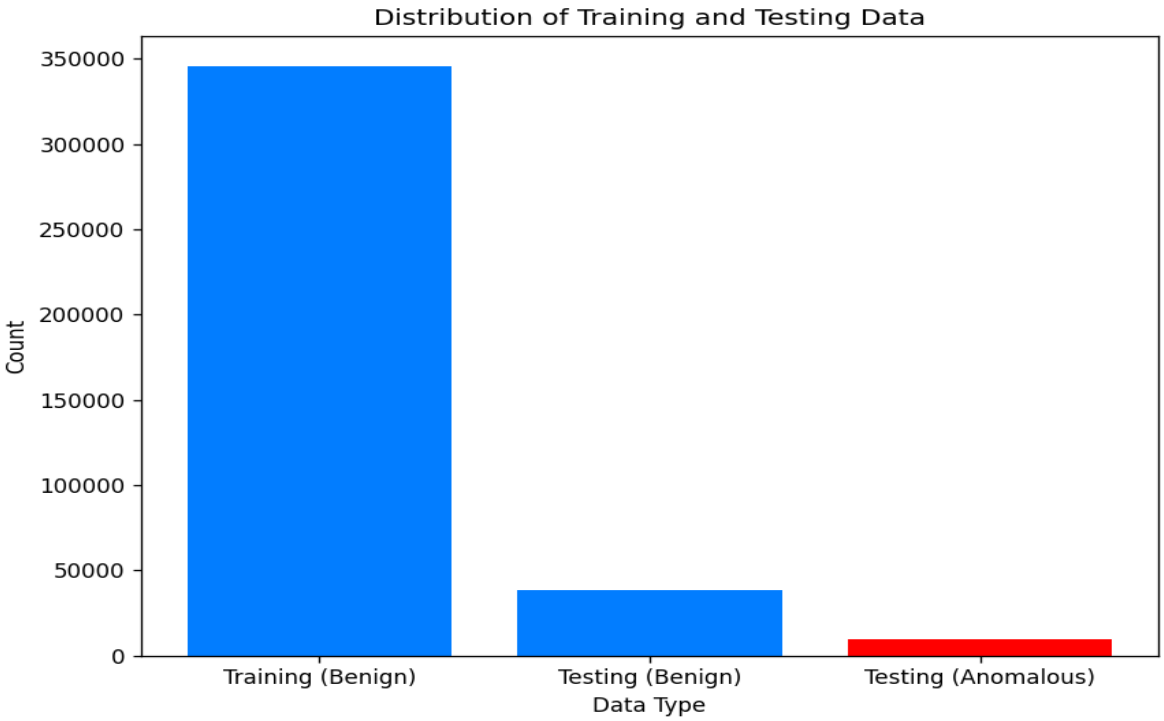



Figure 12. Stage 2 Training and Testing Split

Table 6. Stage 2 Hyperparameter Tuning

| No | Hyperparameter | Search Space | | Optimized |
|---|---|---|---|---|
| | | Minimum | Maximum | |
| 1 | Number of Encoder units in the first layer | 32 | 256 | 128 |
| 2 | Encoder dropout rates for the first layer | 0.2 | 0.5 | 0.3 |
| 3 | Number of Decoder units in the second layer | 16 | 128 | 16 |
| 4 | Decoder dropout rates for the first layer | 0.2 | 0.5 | 0.2 |
| 5 | Learning rate for Adam optimized | 0.0001 | 0.01 | 0.01 |

### 5.2.4. Models Hyperparameter Building Experiment

Hyperparameter tuning is one of the most challenging and time-consuming tasks in machine learning. Hyperparameters are the external configuration variables that control the learning process of the model, such as the number of units, the dropout rate, and the learning rate. Finding the optimal values for these variables can have a significant impact on the performance and accuracy of the model. However, the search space of hyperparameters is often large, complex, and non-convex, making it difficult to explore efficiently and effectively[45], [46].

Bayesian optimization with Keras Tuner is a powerful and practical solution for hyperparameter tuning. Bayesian optimization uses a probabilistic model, such as a Gaussian process, to capture the relationship between the hyperparameters and the objective function, which is usually the validation accuracy or loss. Bayesian optimization iteratively selects the next set of hyperparameters to evaluate, based on the acquisition function, which balances the trade-off between exploration and exploitation. Keras Tuner provides a simple and flexible interface to perform Bayesian optimization, and supports various types of hyperparameters, such as categorical, numerical, and conditional. Keras Tuner also allows to monitor and visualize the tuning process, and to save and resume the search state[45], [46].

In this study, it is proposed to use Bayesian optimization with Keras Tuner to tune the hyperparameters of both deep Convolutional neural network model for DDoS detection and Auto encoder model for unknown DDoS detections in network traffic data. Bayesian optimization is chosen because it can efficiently and effectively explore the large, complex, and non-convex search space of hyperparameters, and handle the expensive and noisy evaluations of the objective function. Keras Tuner is used because it is an easy-to-use, scalable, and flexible framework that integrates with the TensorFlow and Keras APIs, and offers various features and functionalities to perform and monitor Bayesian optimization.

### 5.2.5. Experiment on Threshold Techniques in Autoencoder

This Experiment applies different threshold calculation techniques and systematically uses Bayesian optimization to calculate and find the best threshold values. These techniques include Percentile, Z-Score, CUSUM, IQR, Peak-to-Peak, and Control Chart methods.

Table 7. Autoencoder Thresholding Techniques

| No | Thresholding Techniques | Range Between | Best Threshold point |
|---|---|---|---|
| **1** | Percentile | 50,99 | 79.358 |
| **2** | Z-Score | 0,5 | 0.0013 |
| **3** | CUSUM | 0.1 ,1.0 | 0.171 |
| **4** | IQR | 0.5 2.0 | 0.258 |
| **5** | Peak-to-Peak | 0.999 | 0.111 |
| **6** | Control Chart | 1,10 | 0.821 |

5.2.6 Experiment of Combining Stage 1 and Stage 2

Both stages are trained and evaluated independently and finally merged together.

The first stage utilizes a deep convolutional neural network (CNN) for multi-class classification. This CNN analyzes the test data, assigning each sample to a specific class, including a potential "benign" class for normal traffic.

The second stage focuses on refining unknown DDoS detection within "benign" samples identified by the CNN. Here, an Autoencoder attempts to reconstruct the input data. By comparing the original and reconstructed data, the Autoencoder identifies significant discrepancies that might indicate unknown DDoS attacks, potentially missed by the initial classification.

Algorithm 2: Stage 1 and Stage 2 Models Combined

---

**Input:** Test data for DDoS Detection

**Output**: Classification as 'Benign' or 'DDoS'

1: Input the test data into the first-stage model.

2: Predict the traffic type using the first stage model.

3: **if** the first Stage model predicts the traffic as 'Benign' **then**

4: Input the benign traffic data into the second-stage model.

5: Predict if the traffic is truly 'Benign' or 'DDoS' using the the Second Stage model.

6: Output the final model prediction.

7: **else**

8: Output the traffic as 'DDoS'.

9: **end if**

### 5.2.7 Experiment Deployment of Proposed Approach in SDN

The deployment of a two-stage deep learning model is carried out within a simulated SDN environment. This process involves constructing a simulation environment by selecting a suitable controller, such as Ryu, and using the emulator Mininet. The live simulation begins with the initialization of the SDN controller, followed by stages of monitoring and feature extraction. These stages are crucial for the effective identification and mitigation of DDoS traffic.

#### 5.2.7.1 Detection and Mitigation Procedure Experiment

The first step is to initialize the SDN controller, which is a device that manages the network of switches. A thread that will run the monitoring process, which is a function that checks the status of the switches, is also created and started.

The monitoring process requests flow statistics from each switch in the network at regular intervals. Flow statistics are data that show how the network traffic is flowing through the switches. Upon receiving the flow statistics from a switch, the SDN controller calls another function that extracts relevant features from the data. Features are characteristics that help understand the flow behavior.

The extracted features are then passed to the prediction process, which is a function that uses a **pre-trained two stage deep learning model** to predict the behavior of the flow. The deep learning model is a program that learns from data and makes predictions based on patterns. If the prediction is “DDoS /unknown class”, which means that the flow is known DDoS or unknown DDoS. the prediction information DDoS attack is provide to for user. The port is the point where the flow enters or leaves the switch. Then start process also identifies which switch port is related to the suspicious flow. The SDN controller then executes the action to block the port of the switch. Action to block the port. The port is the point where the flow enters or leaves the switch. The prediction process also identifies which switch port is related to the suspicious flow. The SDN controller then executes the action to block the port of the switch.

Algorithm 3: DDoS Attack Mitigation in SDN environment

---

**Input:** Network traffic data from SDN switches

**Output:** Mitigation action based on traffic analysis

1: Initialize the monitoring system and load the two-stage prediction model.

2: Continuously monitor the network traffic at predefined intervals.

3: **for** each datapath (switch) in the SDN network **do**

4: Send a request to retrieve flow statistics.

5: **for** each flow entry in the flow statistics **do**

6: Extract traffic features and compute traffic metrics.

7: Write the traffic data to a CSV file for prediction.

8: **end for**

9: **end for**

10: Load the traffic data from the CSV file.

11: Preprocess the traffic data for the first-stage CNN model.

12: Use the CNN model to predict the traffic type.

13: **if** the CNN model predicts a significant percentage of traffic as DDoS **then**

14: Log the detection of a potential DDoS attack.

15: Initiate the mitigation process immediately.

16: **else**

17: **if** the CNN model predicts the traffic as 'Benign' **then**

18: Preprocess the benign traffic data for the second-stage AE model.

19: Use the AE model to confirm if the traffic is truly 'Benign' or 'Anomalous'.

20: **if** the AE model predicts the traffic as 'Anomalous' **then**

21: Log the detection of a potential DDoS attack.

22: Initiate the mitigation process.

23: **end if**

24: **end if**

25: **end if**

26: Clear the CSV file for the next round of prediction.

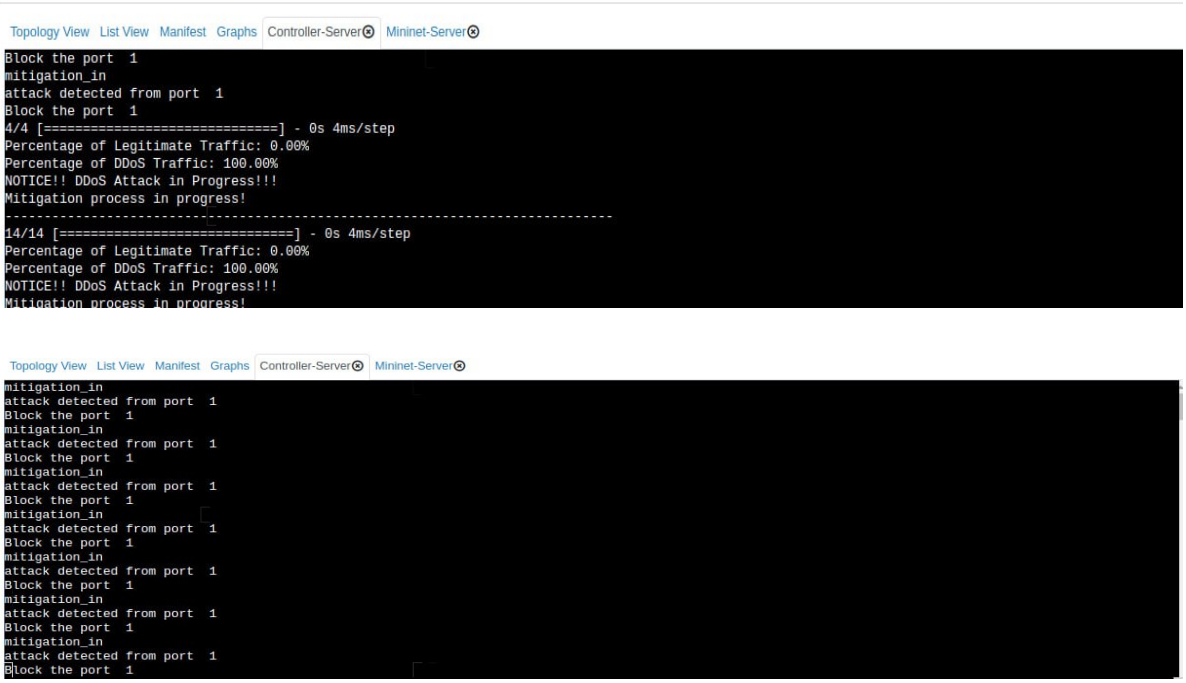



Figure 13. Mitigation Procedure

CHAPTER SIX

# 6. Result and Discussions

## 6. 1. Evaluation of Stage 1 Supervised DDoS Detection Model

The Stage 1 supervised DDoS Detection Model is evaluated using a variety of metrics to thoroughly assess its performance. This comprehensive evaluation framework ensures that every essential aspect of the model's performance is examined, from its predictive accuracy and reliability to its operational efficiency and resource utilization. By examining this suite of metrics, a clear picture of the model's strengths is obtained, ensuring it is equipped for real-world applications.

**Loss Function:** The loss function is a fundamental concept in machine learning and serves as the guiding metric for a model's training process. It is essentially a method to measure the "cost" of inaccuracies in the model's predictions. When a model makes a prediction during training, the loss function assesses how far off that prediction is from the actual value. This assessment is quantified as a numerical value, representing the error or "loss." The primary objective during training is to minimize this loss, which is achieved by adjusting the model's internal parameters through optimization algorithms.

**Categorical Cross-Entropy:** Categorical cross-entropy is a specific type of loss function best suited for classification problems involving multiple classes. It measures the "distance" between two probability distributions - the actual distribution (the true labels) and the predicted distribution (the model's outputs). The term "entropy" is borrowed from information theory, where it quantifies the amount of uncertainty or surprise associated with a particular distribution.

$$\text{Categorical Cross-Entropy Loss} = -\sum_{c=1}^{m} yi, c \,.\log\left(\hat{y}_{i,c}\right) \quad \text{.....Equation 1}$$

where ( m ) is the number of classes, ($y_{i,c)}$ is a binary indicator of whether class label (c) is the correct classification for observation (i), and ($\hat{y}_{i,c}$) is the predicted probability of observation (i) being of class (c).

**Accuracy:** Accuracy is a widely used performance metric in machine learning, particularly for classification problems. It provides a simple and intuitive measure of a model's overall effectiveness by indicating the proportion of correct predictions out of the total number of predictions made.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total number of Predictions}} \qquad \text{.....Equation 2}$$

**Recall Score**: Also known as sensitivity, this metric helps us understand the model's ability to find all the relevant cases within the data. It's especially important in scenarios like DDoS detection where missing an attack can be costly:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \qquad \text{.....Equation 3}$$

**Precision Score**: This tells us how many of the positive predictions made by the model were actually correct. A high precision score means that when the model predicts an attack, it's likely to be right:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \qquad \text{.....Equation 4}$$

**ROC AUC Score:** this calculated from roc_auc_score function. Is calculates ROC AUC for multi-class problems using the One-vs-One (OVO) approach. This method avoids directly comparing all classes at once. Instead, it treats classification as a series of one-on-one contests. For each class pair, it builds a separate ROC curve, analyzing how well the model differentiates those specific classes. The AUC is then calculated for each individual ROC curve, reflecting the model's ability to distinguish between that particular pair. Finally, the roc_auc_score function averages these AUC scores across all class pairs, providing a single ROC AUC score that summarizes the model's overall performance in handling multi-class classification.

$$\text{ROC AUC Score} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \text{AUC}_{ij} \qquad \text{.....Equation 5}$$

Where $\text{AUC}_{ij}$ is the area under the ROC curve for the class pair ( (i, j) ), and ( n ) is the total number of classes.

**F1 Measure**: This is the harmonic mean of precision and recall and is a single metric that combines both. It's useful when it required to balance the trade-off between precision and recall:

$$\text{F1 Score} = 2x\frac{\text{Precision X Recall}}{\text{Precision} + \text{Recall}} \quad \text{.....Equation 6}$$

**Time :** All models are provided with an identical set of test data to predict and to measure the time required to complete the prediction. This duration, which the model takes to make predictions, is measured in seconds**.**

$$t_{\text{prediction}} = t_{\text{end}} - t_{\text{start}} \quad \text{.....Equation 7}$$

Where $t_{\text{start}}$ the timestamp before the prediction starts and $t_{\text{end}}$ the timestamp after the prediction ends..

**RAM Usage:** All models are provided with an identical set of test data for predictions. The RAM usage, measured in megabytes (MB), is monitored during the prediction process for each model**.** which is important for understanding its impact on system resources.

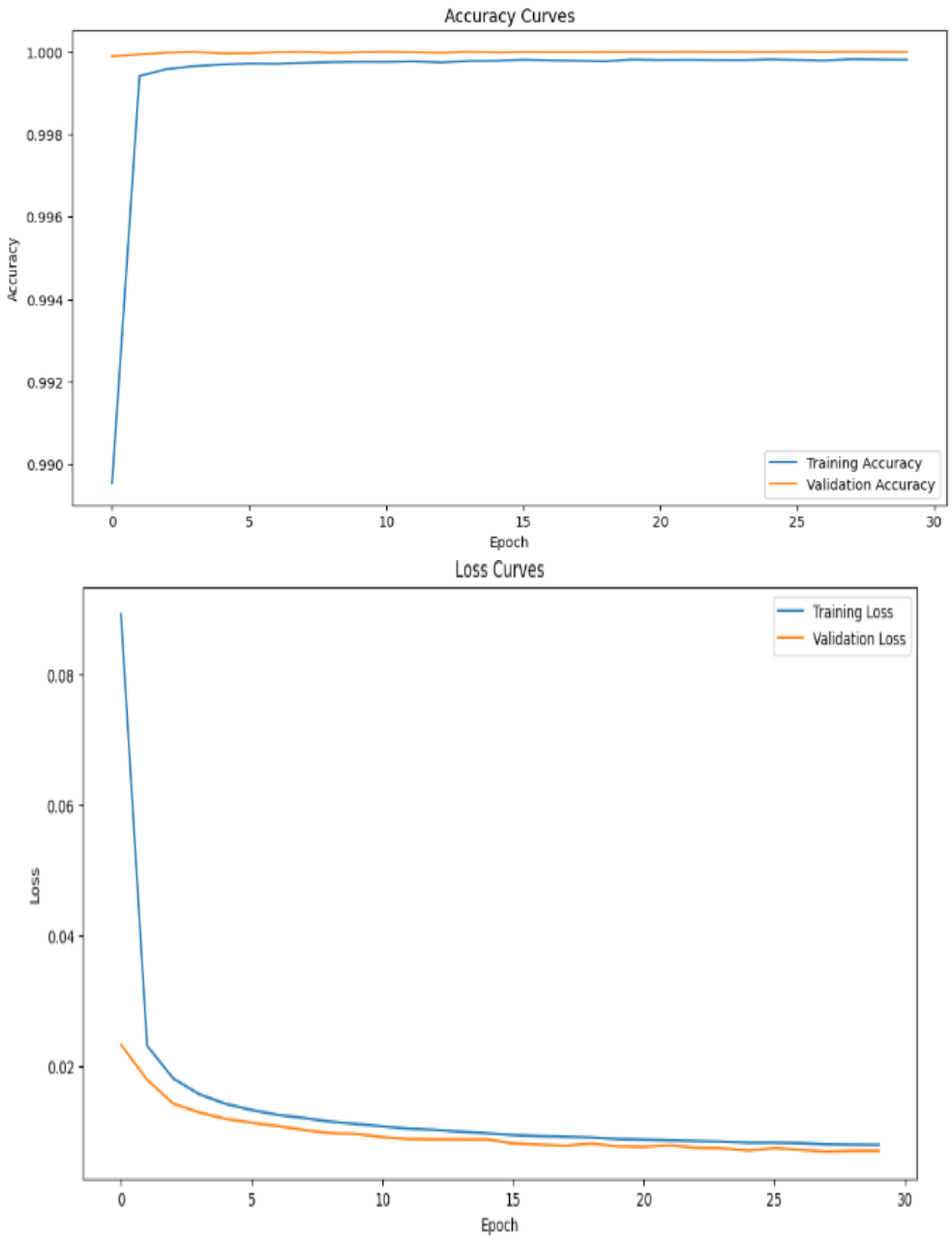
**CPU Usage (%):** Each model is provided with an identical set of test data for predictions. The CPU usage percentage indicates the amount of the total CPU power utilized by the model during the prediction process.

The psutil (Process and System Utilities) library is a cross-platform tool for retrieving information about running processes and system utilization in Python. It's widely used for system monitoring, profiling, limiting process resources, and managing running processes. This study use this toll for measuring CPU and RAM Usage.

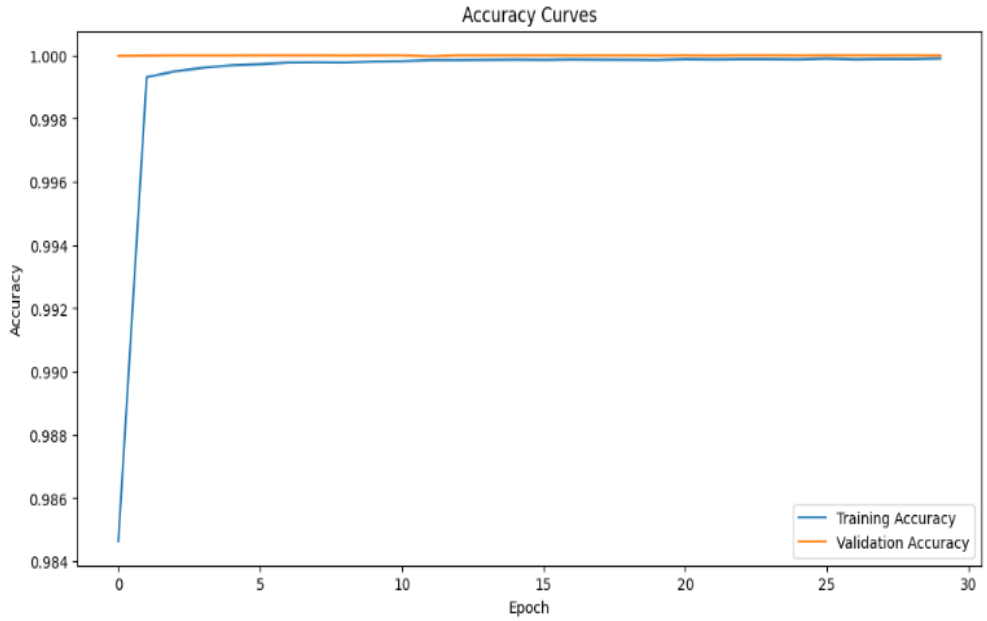
Table 8. Stage 1 Models Comparison.

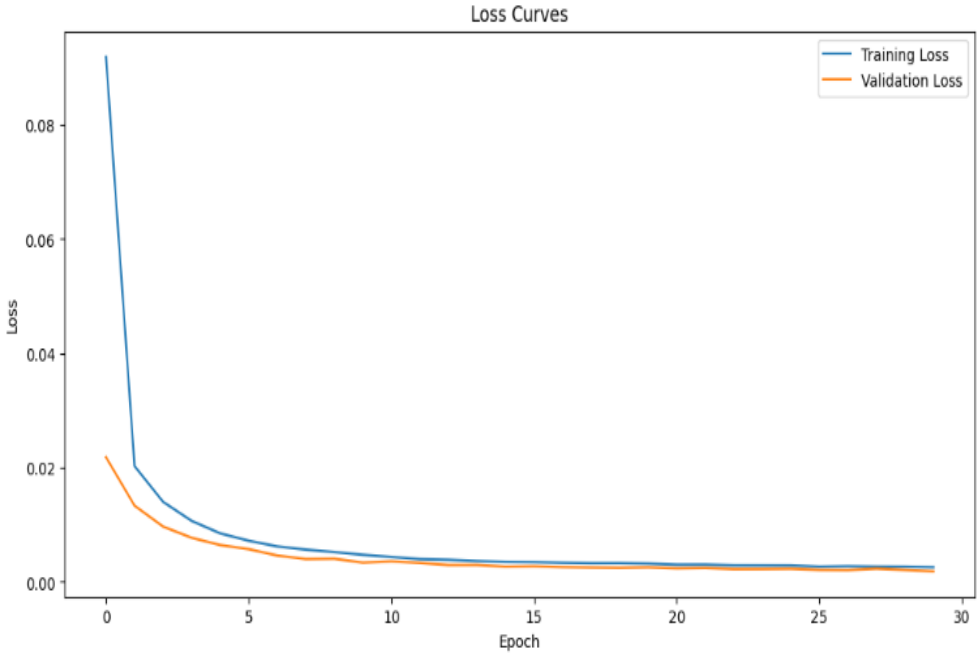
| Performance Metrics | CNN | CNNBiLSTM | CNNBiLSTM-Attention |
|---|---|---|---|
| Loss | 0.0069 | 0.0018 | 0.0058 |
| Accuracy | 0.9999 | 0.9999 | 0.9999 |
| Recall Score | 0.9999 | 0.9999 | 0.9999 |
| Precision Score | 0.9999 | 0.9999 | 0.9999 |
| ROC Score | 0.9999 | 0.9999 | 0.9999 |
| F1 Measure | 0.9999 | 0.9999 | 0.9999 |
| Time(S) | 80 | 134 | 121 |
| RAM(MB) | 98 | 135 | 105 |
| CPU% | 4.0 | 6.10 | 9.7 |

Table 9. Stage 1 Performance Comparison With Loss and Accuracy curves

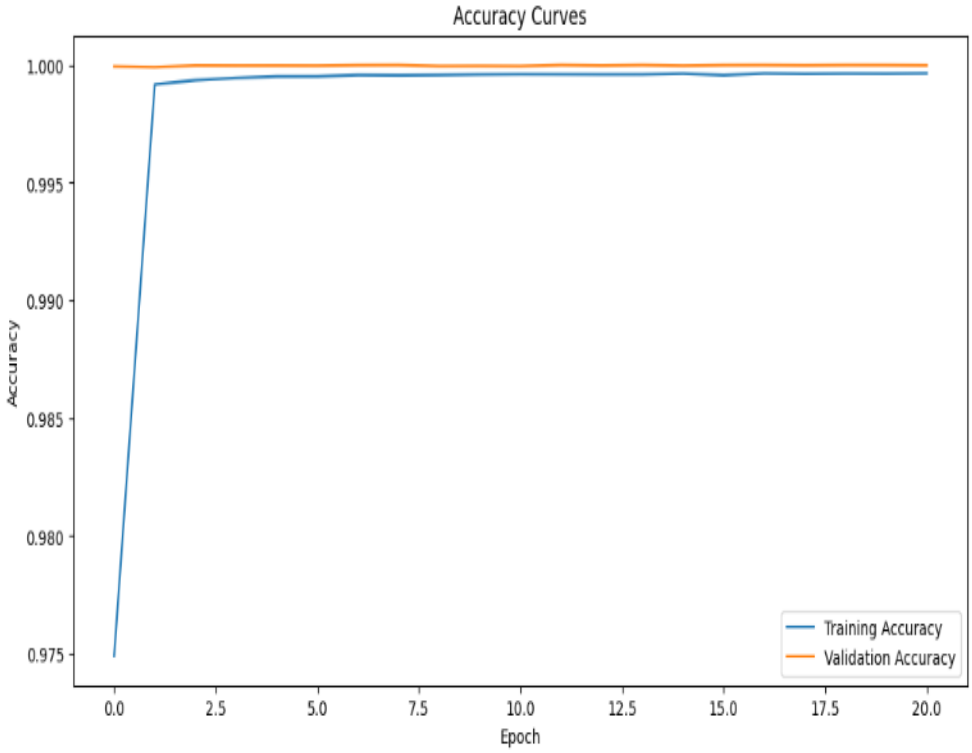
| Accuracy and loss curve |
| --- |
| **CNN**<br>  |

**CNNBiLST**


Accuracy Curves
Accuracy
Epoch
Training Accuracy
Validation Accuracy
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
0.984
0
5
10
15
20
25
30


Loss Curves
Loss
Epoch
Training Loss
Validation Loss
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
30

**CNN-BiLSTM-Att**



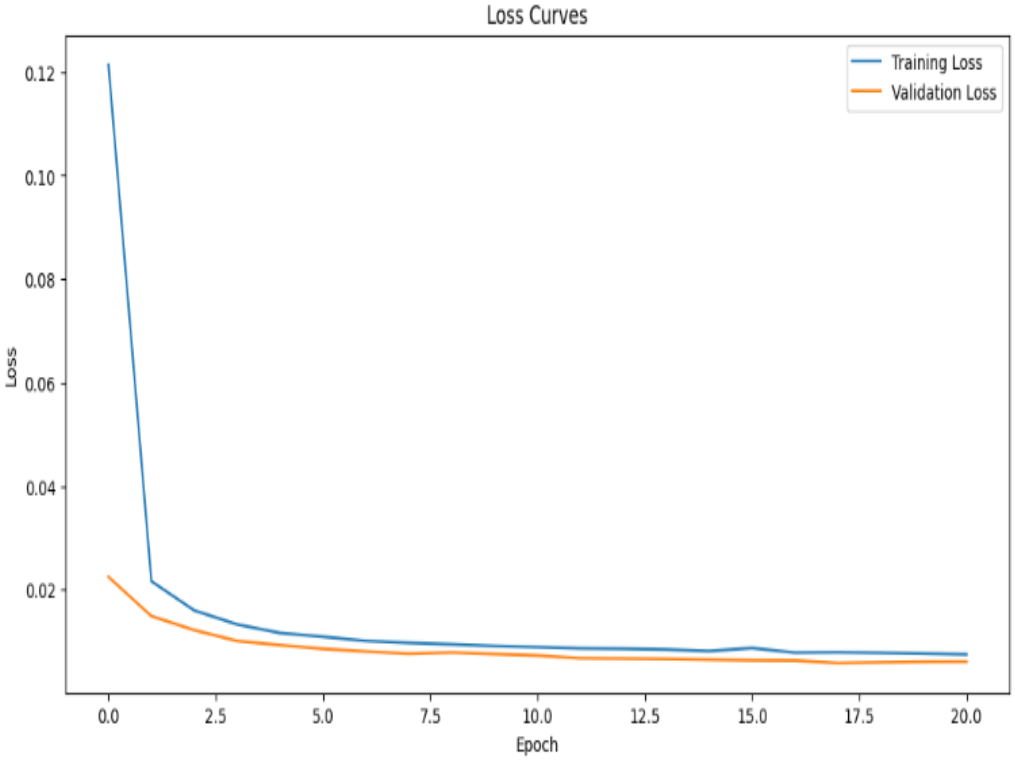
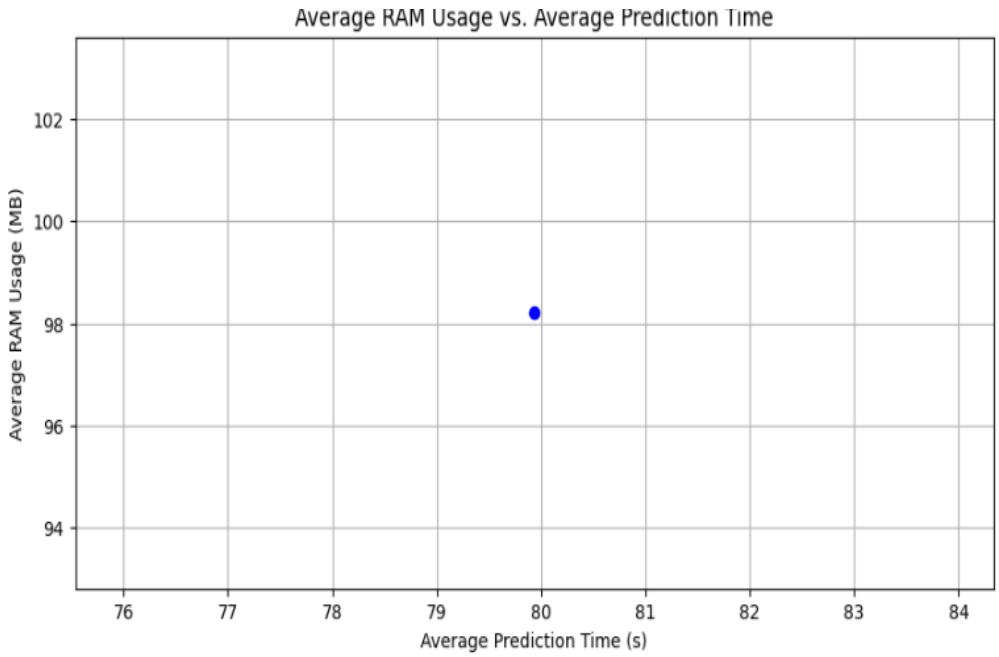
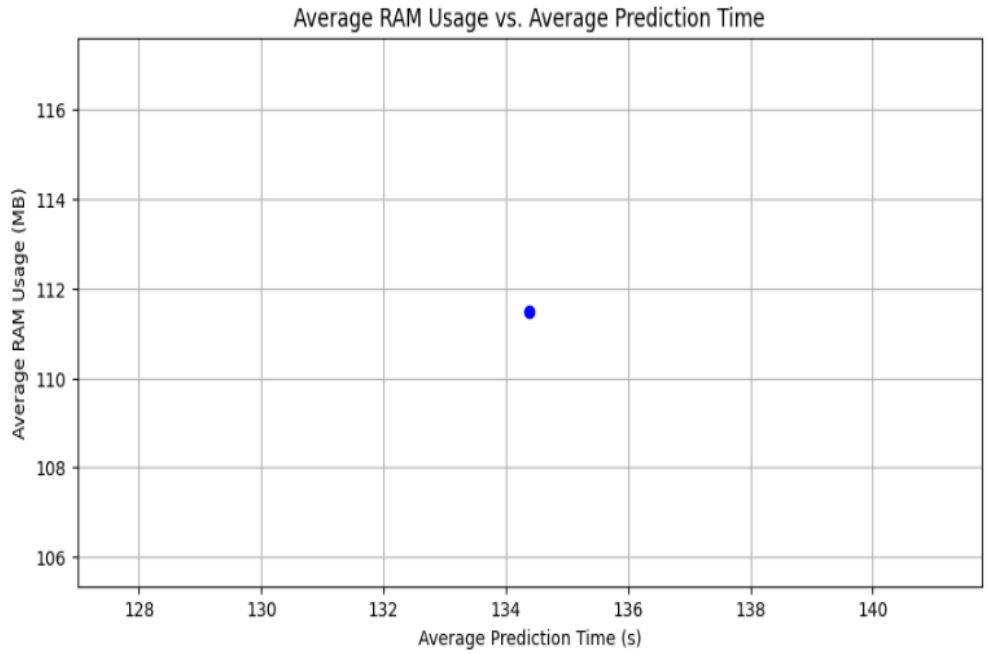
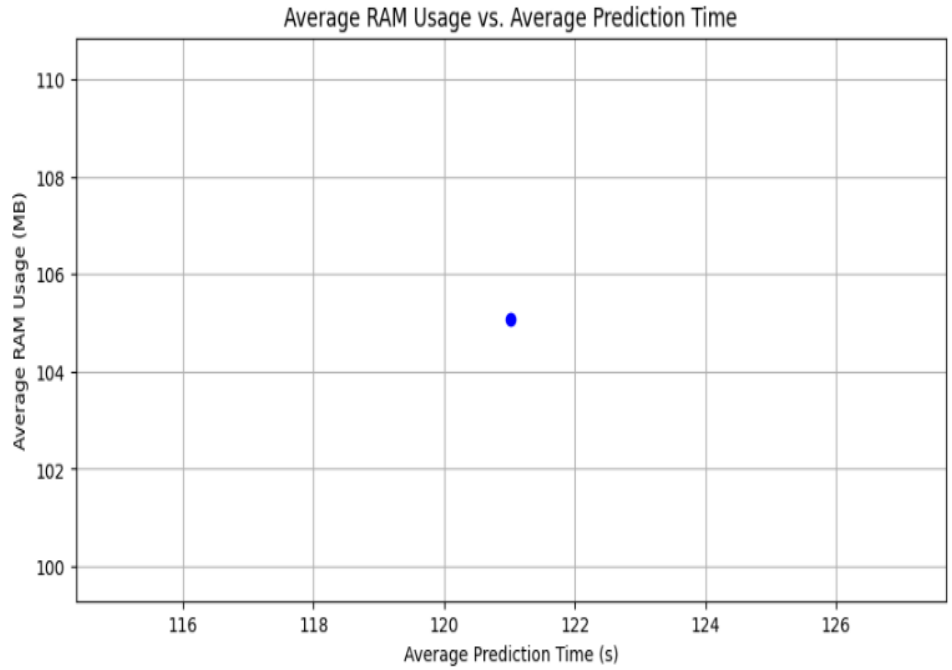



Table 10. RAM Utilization vs Time graph for  Stage 1 Models

| Ram vs Time graph |
| --- |
| **CNN**<br>  |
| **CNNBilSTM**<br>  |

**CNNBilSTM-Att**




### 6. 2. Evaluation of Stage 2 Unsupervised DDoS Detection Model

Similar to stage 1, stage 2 also employs the same performance measurement metrics. The Stage 2 unsupervised DDoS Detection Model is evaluated using a comprehensive set of metrics to thoroughly assess its performance. This evaluation framework ensures that every critical aspect of the model's performance is analyzed, from its predictive accuracy and reliability to its operational efficiency and resource utilization. By leveraging this suite of metrics, the model's capabilities are fully understood, confirming its readiness for real-world deployment.

Table 11. Prediction Loss Curve of Autoencoder and Variational Autoencoder

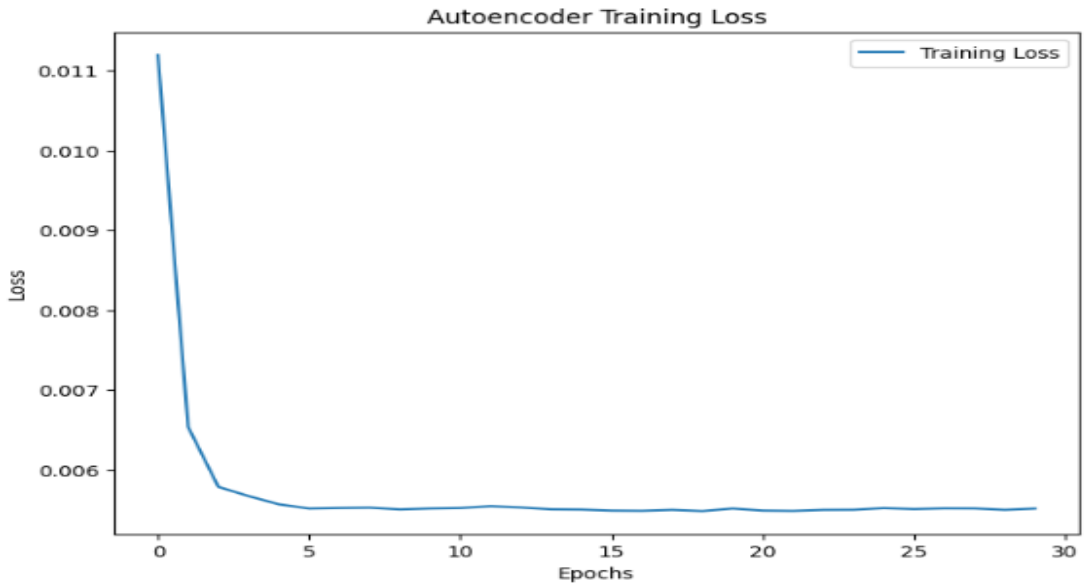
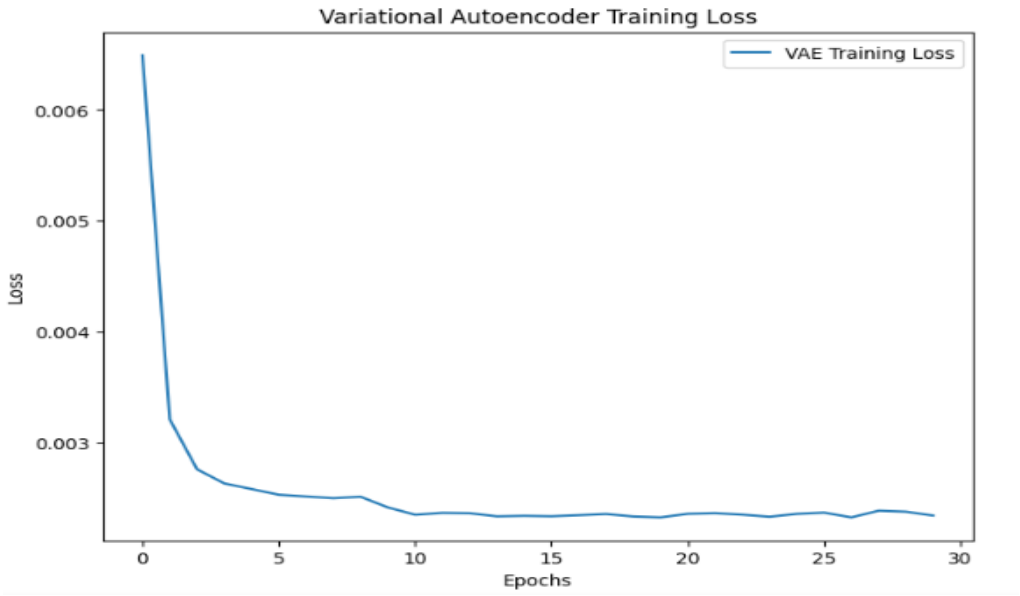
| Training loss curve |
| --- |
| **Autoencoder(AE)**<br>  |
| **Variational Autoencoder(VAE)**<br>  |

Table 12. Stage 2 Models Comparison

| Performance Metrics | Autoencoder | Variational Autoencoder |
|---|---|---|
| Loss | 0.0055 | 0.0023 |
| Accuracy | 0.9986 | 0.9983 |
| Recall Score | 0.9986 | 0.9983 |
| Precision Score | 0.9986 | 0.9983 |
| AUC ROC Score | 0.9966 | 0.9960 |
| F1 Measure | 0.9986 | 0.9983 |
| Time(S) | 2.09 | 2.17 |
| RAM(MB) | 6.6 | 10.5 |
| CPU% | 1.6 | 3.80 |

Based on the performance metrics presented in the table, it is clear that the Autoencoder model outperforms the Variational Autoencoder. It demonstrates superior results across various key indicators, such as accuracy, recall, precision, AUC ROC score, and F1 measure, which collectively suggest that the Autoencoder is the best model for this specific application. These metrics are crucial for evaluating the effectiveness of models in detecting and classifying DDoS attacks in SDN environments, and the Autoencoder's higher scores indicate its enhanced capability in handling such tasks efficiently and accurately. the Autoencoder model not only showcases better performance in terms of the above metrics but also demonstrates more efficient resource utilization when compared to the Variational Autoencoder. This makes the Autoencoder model particularly suitable for environments where computational resources are a concern.

### 6.3. Evaluation Stage 2 Autoencoder on Different Thresholding Techniques

Autoencoders, when paired with CUSUM (Cumulative Sum) thresholding techniques, have demonstrated exceptional performance in anomaly detection tasks. This combination has achieved a remarkable 100% precision score, distinguishing it as the best among various tested techniques.

Table 13. Result with Different Thresholding Techniques

| 1 | Thresholding Techniques | Accuracy | Average Precision Score |
|---|---|---|---|
| 1 | Autoencoder with Percentile | 0.9999 | 0.9997 |
| 2 | Autoencoder with Z-Score | 0.8304 | 0.3483 |
| 3 | Autoencoder with CUSUM | 100 | 100 |
| 4 | Autoencoder with IQR | 100 | 0.9999 |
| 5 | Autoencoder with Peak-to-Peak | 0.999 | 0.9998 |
| 6 | Autoencoder with Control Chart | 0.8540 | 0.3128 |

### 6.4. Evaluation Models Performance on Benchmark Public Dataset

Cross-dataset validation is an evaluation method that measures the adaptability of a machine learning model's architecture. It involves training and evaluating the model on various datasets to determine how the architecture handles different data characteristics. This method is crucial for assessing the robustness of the model and its specialization to the training dataset.

By applying the model to datasets with diverse features, such as size, complexity, and noise levels, one can assess the model's ability to capture and learn from a range of patterns and relationships. A model that performs consistently well across different datasets is considered adaptable and reliable for real-world applications. On the other hand, if a model's performance falters on new datasets, it may indicate that the architecture requires adjustments to better manage data variability. Cross-dataset validation is therefore a vital strategy for understanding a model's learning process and ensuring its accuracy in various scenarios.

This study conducted cross-dataset validation on a selected model during stage one and stage two using the CICDDoS2019 and InSDN datasets. The model was trained and tested separately on each dataset

Table 14.Validate Model with Other Benchmark Datasets

| Datasets | Performance Metrics | CNN | AE |
|---|---|---|---|
| CICDDOS2019 | Loss | 0.0286 | 0.0459 |
| | Accuracy | 0.9949 | 0.980 |
| | Recall Score | 0.9948 | 0.980 |
| | Precision Score | 0.9948 | 0.980 |
| | ROC Score | 0.9964 | 0.980 |
| | F1 Measure | 0.9946 | 0.980 |
| InSDN | Loss | 0.0568 | 1.00 |
| | Accuracy | 0.9957 | 1.00 |
| | Recall Score | 9960 | 1.00 |
| | Precision Score | 0.9956 | 1.00 |
| | ROC Score | 0.9876 | 1.00 |
| | F1 Measure | 9953 | 1.00 |

### 6. 5. Stage 1 Result Summary

This study evaluates the performance of three Deep learning models: Convolutional Neural Network (CNN), Convolutional Neural Network with Bidirectional Long Short-Term Memory (CNNBiLSTM), and Convolutional Neural Network with Bidirectional Long Short-Term Memory and Attention Mechanism (CNNBiLSTM-Attention). The primary focus lies in analyzing the trade-off between computational efficiency (resource usage and processing speed) and model accuracy (effectiveness in performing the designated task).

#### 6.5.1. Speed and Resource Consumption Analysis

The Convolutional Neural Network (CNN) model stands out due to its computational strength. It takes only 80 seconds to complete a prediction task, reflecting its quick processing ability. This efficient performance is further highlighted by its low memory requirement, using just 98 MB of RAM. The model also boasts a minimal CPU usage of only 4% during operation. In comparison, other models, while delivering similar accuracy, require more time and greater CPU usage, emphasizing the CNN model's superior efficiency.

6.5.2. Emphasizing Computational Efficiency

The data clearly demonstrates the CNN model's outstanding balance of high accuracy with minimal resource use. Its capability to process data rapidly and efficiently positions it as the optimal choice for real-world applications where computational efficiency is valued. The model's skill in managing large datasets quickly, even under resource limitations, solidifies its role as the preferred solution in such contexts.

6.5.3 Considering the Full Spectrum of Model Capabilities

Although the CNN model is the epitome of efficiency, the merits of other models like the CNNBiLSTM and CNNBiLSTM-Attention should not be overlooked. These models feature more complex structures, allowing them to navigate through more complicated data patterns and identify longer-term dependencies within the data. They may demand more computational power, but they offer essential advantages in situations where the complexity and sequential relationships in the data are the main focus.

**6.6. Stage 2 Result Summary**

The Autoencoder (AE) outperforms the Variational Autoencoder (VAE) on most metrics, including loss, accuracy, and AUC ROC, while also being significantly faster and more resource-efficient, as indicated by lower RAM usage. However, the choice of model may vary depending on the specific application and its priorities. For instances where processing speed and resource efficiency are paramount, the AE is the superior option. Conversely, if the task benefits from a slight improvement in metrics like loss or AUC ROC, and computational resources are not a limiting factor, the VAE might be a suitable alternative.

The objective of this study was to prioritize the evaluation of machine learning models that balance accuracy and efficiency. After selecting the efficient AE model, various thresholding techniques were applied. Among these, the AE model combined with the Cumulative Sum (CUSUM) thresholding technique achieved the best performance, outshining other methods.

## 6.7. Discussion and Interpretation

The approach and metrics presented demonstrate that this research is both realistic and applicable. Furthermore, the proposed method for DDoS detection in SDN environments outperforms other benchmarks in terms of accuracy, efficiency, and innovation. This method employs a two-stage process that can operates independently and collaboratively, introducing enhancements across various aspects. Let explore research question in detail as follows

1. **Suitable Deep Learning Architectures for Known DDoS Attacks in SDN Environments:**

To address the first research question, this study carefully evaluated three deep learning architectures. Among these, the Convolutional Neural Network (CNN) emerged as the most suitable choice. Not only does it exhibit remarkable performance in terms of detection accuracy, but it also operates efficiently in resource-constrained SDN environments. The CNN successfully detects known DDoS attacks using both custom-collected datasets and publicly available datasets. Its ability to generalize across different attack scenarios makes it a robust choice for SDN-based security.

2. **How can deep learning models be trained and adapted to detect unknown or novel DDoS attacks in SDN environments?**

The second research question focuses on detecting unknown or novel DDoS attacks from stage 1 . For this purpose, an Autoencoder-based model was leveraged. This model not only achieves impressive accuracy but also operates efficiently in terms of computational resources. By learning to represent normal network traffic patterns, the Autoencoder effectively identifies deviations from the norm. Whether it's a current attack variant or a future, previously unseen attack, the model adapts dynamically. Its ability to capture subtle anomalies sets it apart, achieving extraordinary performance in detecting previously unknown threats.

3. **For unsupervised deep learning-based DDoS detection in SDN environments, which threshold calculation method achieves the best balance between attack detection rate and false alarm rate?**

The third research question investigates the impact of thresholding techniques on the performance of an unsupervised model, specifically the Autoencoder (AE), in detecting

DDoS attacks within SDN environments. The study's exploration of various thresholding methods revealed significant differences in the balance between attack detection rates and false alarm rates. The Cumulative Sum (CUSUM) method stood out, achieving a perfect precision of 100%, indicating its effectiveness in minimizing false alarms while maintaining high detection rates. These findings underscore the importance of selecting the appropriate thresholding technique to optimize the AE model's performance, as it directly influences the reliability and accuracy of DDoS attack detection in SDN environments. The superior results of CUSUM suggest that it could be the most suitable method for environments where the cost of false alarms is high and precision is paramount.

CHAPTER SEVEN

# 7. Conclusion and Future Works

This research proposes a two-stage deep learning framework for efficient and accurate detection of both known and unknown DDoS attacks in Software-Defined Networking (SDN) environments. Incoming traffic is first evaluated by a highly accurate Convolutional Neural Network (CNN) in the initial stage, effectively identifying known attack patterns with a 99.99% success rate. Traffic classified as benign in the first stage is then forwarded to the second stage for further refinement. Here, an Autoencoder (AE) leverages unsupervised learning to detect novel DDoS attacks, achieving an efficiency rate of 99.86%. The study also explored various thresholding techniques for the AE, with Cumulative Sum (CUSUM) demonstrating the best performance at 100% precision. These findings highlight the potential of deep learning to significantly enhance the ability of SDN systems to defend against the evolving landscape of DDoS attacks by combining high accuracy with efficient multi-stage detection.

Future work can explore the applicability of this approach in diverse SDN architectures, considering potential challenges like scalability and compatibility. Additionally, investigating real-time detection and mitigation techniques within deployment environments is crucial for practical implementation. Furthermore, incorporating online learning algorithms to dynamically update the deep learning model holds significant promise. This would enable the model to adapt to evolving attack patterns and maintain high detection accuracy over time. Addressing concept drift, a potential issue with online learning, would be an important area of focus.

# REFERENCES


[1] I. A. Valdovinos, J. A. Pérez-Díaz, K. K. R. Choo, and J. F. Botero, "Emerging DDoS attack detection and mitigation strategies in software-defined networks: Taxonomy, challenges and future directions," Journal of Network and Computer Applications, vol. 187. Academic Press, Aug. 01, 2021. doi: 10.1016/j.jnca.2021.103093.

[2] S. Kaur, K. Kumar, N. Aggarwal, and G. Singh, "A comprehensive survey of DDoS defense solutions in SDN: Taxonomy, research challenges, and future directions," Computers and Security, vol. 110. Elsevier Ltd, Nov. 01, 2021. doi: 10.1016/j.cose.2021.102423.

[3] A. El Kamel, H. Eltaief, and H. Youssef, "On-the-fly (D)DoS attack mitigation in SDN using Deep Neural Network-based rate limiting," Comput Commun, vol. 182, pp. 153–169, Jan. 2022, doi: 10.1016/j.comcom.2021.11.003.

[4] B. Alhijawi, S. Almajali, H. Elgala, H. Bany Salameh, and M. Ayyash, "A survey on DoS/DDoS mitigation techniques in SDNs: Classification, comparison, solutions, testing tools and datasets," Computers and Electrical Engineering, vol. 99, Apr. 2022, doi: 10.1016/j.compeleceng.2022.107706.

[5] L. F. Eliyan and R. Di Pietro, "DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges," Future Generation Computer Systems, vol. 122, pp. 149–171, Sep. 2021, doi: 10.1016/j.future.2021.03.011.

[6] "DDoS attacks in Q3 grow by 24%, become more sophisticated | Kaspersky." Accessed: Jan. 15, 2022. [Online]. Available: https://www.kaspersky.com/about/press-releases/2021_ddos-attacks-in-q3-grow-by-24-become-more-sophisticated

[7] N. Gupta, M. S. Maashi, S. Tanwar, S. Badotra, M. Aljebreen, and S. Bharany, "A Comparative Study of Software Defined Networking Controllers Using Mininet," Electronics (Switzerland), vol. 11, no. 17. MDPI, Sep. 01, 2022. doi: 10.3390/electronics11172715.

[8] Y. Cui et al., “Towards DDoS detection mechanisms in Software-Defined Networking,” Journal of Network and Computer Applications, vol. 190. Academic Press, Sep. 15, 2021. doi: 10.1016/j.jnca.2021.103156.

[9] N. Ahmed et al., “Network Threat Detection Using Machine/Deep Learning in SDN-Based Platforms: A Comprehensive Analysis of State-of-the-Art Solutions, Discussion, Challenges, and Future Research Direction,” Sensors (Basel, Switzerland), vol. 22, no. 20. NLM (Medline), Oct. 17, 2022. doi: 10.3390/s22207896.

[10] M. S. Abdullah, S. Alshra’, N. Jochen Seitz, I. Giovanni, D. Galdo, and I. J. Schussmann, “Intrusion Detection System against Denial of Service Attack in Software-Defined Networking”, doi: 10.22032/dbt.51449.

[11] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[12] J. Singh and S. Behal, “Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions,” Computer Science Review, vol. 37. Elsevier Ireland Ltd, Aug. 01, 2020. doi: 10.1016/j.cosrev.2020.100279.

[13] N. Ahmed, I. Hussain, and Z. Yousaf, “Analysis and Detection of DDoS Attacks Targetting Virtualized Servers.”

[14] V. Srihari and R. Anitha, “CCIS 467 - DDoS Detection System Using Wavelet Features and Semi-supervised Learning,” 2014.

[15] I. PES Institute of Technology (Bangalore, IEEE Communications Society, IEEE Photonics Society. Bangalore Chapter, IEEE Robotics and Automation Society. Bangalore Chapter, and Institute of Electrical and Electronics Engineers, 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI) : 19-22 Sept. 2018.

[16] K. J. Singh and T. De, “An Approach of DDOS Attack Detection Using Classifiers,” in Emerging Research in Computing, Information, Communication and

Applications, Springer India, 2015, pp. 429–437. doi: 10.1007/978-81-322-2550-8_41.

[17] H. S. Abdulkarem and A. Dawod, “DDoS Attack Detection and Mitigation at SDN Data Plane Layer,” in Proceedings - 2020 IEEE 2nd Global Power, Energy and Communication Conference, GPECOM 2020, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 322–326. doi: 10.1109/GPECOM49333.2020.9247850.

[18] H. Elubeyd and D. Yiltas-Kaplan, “Hybrid Deep Learning Approach for Automatic DoS/DDoS Attacks Detection in Software-Defined Networks,” Applied Sciences (Switzerland), vol. 13, no. 6, Mar. 2023, doi: 10.3390/app13063828.

[19] W. G. Gadallah, H. M. Ibrahim, and N. M. Omar, “A deep learning technique to detect distributed denial of service attacks in software-defined networks,” Comput Secur, vol. 137, p. 103588, Feb. 2024, doi: 10.1016/J.COSE.2023.103588.

[20] Y. Al-Dunainawi, B. R. Al-Kaseem, and H. S. Al-Raweshidy, “Optimized Artificial Intelligence Model for DDoS Detection in SDN Environment,” IEEE Access, vol. 11, pp. 106733–106748, 2023, doi: 10.1109/ACCESS.2023.3319214.

[21] S. Wang et al., “Detecting flooding DDoS attacks in software defined networks using supervised learning techniques,” Engineering Science and Technology, an International Journal, vol. 35, p. 101176, Nov. 2022, doi: 10.1016/J.JESTCH.2022.101176.

[22] A. Mansoor, M. Anbar, A. A. Bahashwan, B. A. Alabsi, and S. D. A. Rihan, “Deep Learning-Based Approach for Detecting DDoS Attack on Software-Defined Networking Controller,” Systems, vol. 11, no. 6, Jun. 2023, doi: 10.3390/systems11060296.

[23] M. A. Setitra, M. Fan, B. L. Y. Agbley, and Z. E. A. Bensalem, “Optimized MLP-CNN Model to Enhance Detecting DDoS Attacks in SDN Environment,” Network, vol. 3, no. 4, pp. 538–562, Dec. 2023, doi: 10.3390/network3040024.

[24] L. Chen, Z. Wang, R. Huo, and T. Huang, “An Adversarial DBN-LSTM Method for Detecting and Defending against DDoS Attacks in SDN Environments,” Algorithms, vol. 16, no. 4, Apr. 2023, doi: 10.3390/a16040197.

[25] G. Nawaz et al., “Detecting and Mitigating DDOS Attacks in SDNs Using Deep Neural Network,” Computers, Materials and Continua, vol. 77, pp. 2157–2178, 2023, doi: 10.32604/cmc.2023.026952.

[26] S. Dong, K. Abbas, and R. Jain, “A Survey on Distributed Denial of Service (DDoS) Attacks in SDN and Cloud Computing Environments,” IEEE Access, vol. 7, pp. 80813–80828, 2019, doi: 10.1109/ACCESS.2019.2922196.

[27] 2019 IEEE Conference on Network Softwarization (NetSoft). IEEE, 2019.

[28] Y. H. Tung, H. C. Wei, Y. W. Ti, Y. T. Tsou, N. Saxena, and C. M. Yu, “Counteracting UDP flooding attacks in SDN,” Electronics (Switzerland), vol. 9, no. 8, pp. 1–28, Aug. 2020, doi: 10.3390/electronics9081239.

[29] R. Swami, M. Dave, and V. Ranga, “Detection and Analysis of TCP-SYN DDoS Attack in Software-Defined Networking,” Wirel Pers Commun, vol. 118, no. 4, pp. 2295–2317, Jun. 2021, doi: 10.1007/s11277-021-08127-6.

[30] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, “Machine-Learning-Enabled DDoS Attacks Detection in P4 Programmable Networks,” Journal of Network and Systems Management, vol. 30, no. 1, Jan. 2022, doi: 10.1007/s10922-021-09633-5.

[31] A. Sangodoyin, B. Modu, I. Awan, and J. Pagna Disso, “An Approach to Detecting Distributed Denial of Service Attacks in Software Defined Networks,” in Proceedings - 2018 IEEE 6th International Conference on Future Internet of Things and Cloud, FiCloud 2018, Institute of Electrical and Electronics Engineers Inc., Sep. 2018, pp. 436–443. doi: 10.1109/FiCloud.2018.00069.

[32] Institute of Electrical and Electronics Engineers. Turkey Section. and Institute of Electrical and Electronics Engineers, 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies : proceedings : 11-13 October 2019, Ankara, Turkey.

[33] A. A. Aizuddin, M. Atan, M. Norulazmi, M. M. Noor, S. Akimi, and Z. Abidin, “DNS amplification attack detection and mitigation via sFlow with security-centric SDN,” in Proceedings of the 11th International Conference on Ubiquitous

Information Management and Communication, IMCOM 2017, Association for Computing Machinery, Inc, Jan. 2017. doi: 10.1145/3022227.3022230.

[34] Institute of Electrical and Electronics Engineers, 2019 IEEE 4th International Conference on Computer and Communication Systems : ICCCS 2019 : February 23-25, 2019, Singapore.

[35] Z. Liu, M. Xu, J. Cao, and Q. Li, "TSA: A two-phase scheme against amplification DDoS attack in SDN," in Communications in Computer and Information Science, Springer Verlag, 2018, pp. 483–496. doi: 10.1007/978-981-10-8890-2_37.

[36] Anna University and IEEE Aerospace and Electronic Systems Society, 2019 International Carnahan Conference on Security Technology (ICCST) : ICCST 2019 : IEEE 53rd International Carnahan Conference on Security Technology : October 01-03, 2019, Anna University, Chennai, India.

[37] M. S. Elsayed, N. A. Le-Khac, and A. D. Jurcut, "InSDN: A novel SDN intrusion dataset," IEEE Access, vol. 8, pp. 165263–165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[38] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed, "A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU." 2023.

[39] B. Gupta, P. Prakasam, and T. Velmurugan, "Integrated BERT embeddings, BiLSTM-BiGRU and 1-D CNN model for binary sentiment classification analysis of movie reviews," Multimed Tools Appl, vol. 81, no. 23, pp. 33067–33086, Sep. 2022, doi: 10.1007/s11042-022-13155-w.

[40] A. Khacha, R. Saadouni, Y. Harbi, C. Gherbi, S. Harous, and Z. Aliouat, "Robust Intrusion Detection for IoT Networks: an Integrated CNN-LSTM-GRU Approach," in 2023 International Conference on Networking and Advanced Systems (ICNAS), 2023, pp. 1–6. doi: 10.1109/ICNAS59892.2023.10330519.

[41] K. Wu et al., "An attention-based CNN-LSTM-BiLSTM model for short-term electric load forecasting in integrated energy system," International Transactions on Electrical Energy Systems, vol. 31, no. 1, Jan. 2021, doi: 10.1002/2050-7038.12637.

[42] L. Shan, Y. Liu, M. Tang, M. Yang, and X. Bai, “CNN-BiLSTM hybrid neural networks with attention mechanism for well log prediction,” J Pet Sci Eng, vol. 205, Oct. 2021, doi: 10.1016/j.petrol.2021.108838.

[43] S. Zavrak and M. Iskefiyeli, “Anomaly-Based Intrusion Detection from Network Flow Features Using Variational Autoencoder,” IEEE Access, vol. 8, pp. 108346–108358, 2020, doi: 10.1109/ACCESS.2020.3001350.

[44] I. H. Sarker, “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions,” SN Computer Science, vol. 2, no. 6. Springer, Nov. 01, 2021. doi: 10.1007/s42979-021-00815-1.

[45] T. Yu and H. Zhu, “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.05689

[46] B. Bischl et al., “Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges,” Jul. 2021, [Online]. Available: http://arxiv.org/abs/2107.05847

APPENDICES

# Appendix A :Stage 1 CNN Model Training Code

```
# Initialize the Sequential model
cnn = keras.Sequential()
# Convolutional layers
    layers.Conv1D(
        filters=best_hyperparameters["conv1d_1_filters"],
        kernel_size=6,
        input_shape=(_features, 1),
        kernel_regularizer=regularizers.l2(best_hyperparameters["l2_1"]),
    )
)
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.MaxPooling1D(pool_size=2))
cnn.add(
    layers.Conv1D(
        filters=best_hyperparameters["conv1d_2_filters"],
        kernel_size=6,
        padding="same",
        kernel_regularizer=regularizers.l2(best_hyperparameters["l2_2"]),
    )
)
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.MaxPooling1D(pool_size=2))
cnn.add(
    layers.Conv1D(
        filters=best_hyperparameters["conv1d_3_filters"],
        kernel_size=6,
        padding="same",
        kernel_regularizer=regularizers.l2(best_hyperparameters["l2_3"]),
    )
)
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.MaxPooling1D(pool_size=2))
```

```
# Flatten layer
cnn.add(layers.Flatten())
# Dense layers
cnn.add(
    layers.Dense(
        units=best_hyperparameters["dense_1_units"],

kernel_regularizer=regularizers.l2(best_hyperparameters["l2_dense_1"]),
    )
)
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.Dropout(best_hyperparameters["dropout_1"]))
cnn.add(
    layers.Dense(
        units=best_hyperparameters["dense_2_units"],

kernel_regularizer=regularizers.l2(best_hyperparameters["l2_dense_2"]),
    )
)
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.Dropout(best_hyperparameters["dropout_2"]))
# Output layer
cnn.add(layers.Dense(n_classes, activation="softmax"))
# Optimizer
opt =
keras.optimizers.Adam(learning_rate=best_hyperparameters["learning_rate"])
# Compile the model
cnn.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])
```

## Appendix B: Stage 2 Autoencoder Model Code

```
# Extract features (X) and labels (Y)
AX = dp[:, 1:22]
Ay = dp[:, -1]
# Filter benign data
X_benign_Autoencoder = AX[Ay == 0]
```

```
# Split the benign data for training (90%) and testing (10%)
x_train_Autoencoder, x_test_benign_Autoencoder, _, _ = train_test_split(
    X_benign_Autoencoder,
    np.zeros(X_benign_Autoencoder.shape[0]),  # Labels are all zeros for
benign data
    test_size=0.1,
    stratify=np.zeros(X_benign_Autoencoder.shape[0]),
    random_state=42  # Set random state for reproducibility
)
# Randomly select 10,000 anomalous instances with labels from 1 to 7 for
testing
X_anomalous_Autoencoder_test = AX[(Ay >= 1)]
selected_anomalous_indices =
np.random.choice(X_anomalous_Autoencoder_test.shape[0], size=10000,
replace=False)
X_anomalous_Autoencoder_test =
X_anomalous_Autoencoder_test[selected_anomalous_indices, :]


# Labels: 1 for anomaly, 0 for normal
y_anomalous_Autoencoder_test = np.ones(X_anomalous_Autoencoder_test.shape[0])


# Combine the benign test data and anomalous test data
x_test_Autoencoder = np.vstack([x_test_benign_Autoencoder,
X_anomalous_Autoencoder_test])
y_test_Autoencoder =
np.concatenate([np.zeros(x_test_benign_Autoencoder.shape[0]),
y_anomalous_Autoencoder_test])


# Scale the data using MinMaxScaler
min_max_scaler = MinMaxScaler(feature_range=(0, 1))
x_train_Autoencoder_scaled =
min_max_scaler.fit_transform(x_train_Autoencoder.copy())
x_test_Autoencoder_scaled =
min_max_scaler.transform(x_test_Autoencoder.copy())


# Build the model with the best hyperparameters
AE = Sequential()
AE.add(Dense(units=128, activation='relu',
input_dim=x_train_Autoencoder_scaled.shape[1]))
```

```
AE.add(Dropout(rate=0.30000000000000004))

AE.add(Dense(units=16, activation='relu'))
AE.add(Dropout(rate=0.2))
AE.add(Dense(x_train_Autoencoder_scaled.shape[1], activation='sigmoid'))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
AE.compile(optimizer=optimizer, loss='mean_squared_logarithmic_error')
```

## Appendix C : DDoS Detection and Mitigation Code

```
def predict_network_flow(self):
    try:
        # Load the dataset and clean the specified columns
        flow_data = pd.read_csv('PredictFlowStatsfile.csv')
        columns_to_clean = [2, 3, 5]
        for column in columns_to_clean:
            flow_data.iloc[:, column] = flow_data.iloc[:,
column].str.replace('.', '')

        # Prepare the feature set for prediction
        features = flow_data.iloc[:, :21].dropna().values
        # Standardize the feature set
        standardized_features = StandardScaler().fit_transform(features)

        # Predict using the pre-trained model
        predictions = self.flow_model.predict(standardized_features)
        # Determine the most likely class for each prediction
        class_predictions = np.argmax(predictions, axis=1)

        # Count the occurrences of each class
        traffic_counts = np.bincount(class_predictions, minlength=2)
        total_predictions = class_predictions.size

        # Calculate the percentage of each traffic type
```

```
        percent_legitimate = (traffic_counts[0] / total_predictions) * 100
        percent_ddos = (traffic_counts[1] / total_predictions) * 100

        # Log the results
        self.logger.info(f"Percentage of Legitimate Traffic:
{percent_legitimate:.2f}%")
        self.logger.info(f"Percentage of DDoS Traffic: {percent_ddos:.2f}%")

        # Determine the status of the network traffic
        if percent_legitimate > 80:
            self.logger.info("Most of the traffic is legitimate.")
        else:
            self.logger.info("ALERT: Potential DDoS attack detected!")
            self.logger.info("Initiating mitigation process.")
            self.mitigation = True
```

## Appendix D: Find Average RAM usage During Prediction Code

```
# Function to measure RAM usage and prediction time
def measure_performance(model, xtest, iterations=2):
    ram_usages = []
    prediction_times = []
    for _ in range(iterations):
        # Measure RAM usage before prediction
        ram_before = psutil.virtual_memory().used
        # Start the timer
        start_time = time.time()
        # Run the model prediction
        model.predict(xtest)
        # Stop the timer
        end_time = time.time()
        # Measure RAM usage after prediction
        ram_after = psutil.virtual_memory().used
        # Calculate the difference in RAM usage and prediction time
```

```
        ram_usage = (ram_after - ram_before) / (1024 ** 2)  # Convert to MB
        prediction_time = end_time - start_time
        # Append to lists
        ram_usages.append(ram_usage)
        prediction_times.append(prediction_time)
    # Calculate averages
    avg_ram_usage = sum(ram_usages) / len(ram_usages)
    avg_prediction_time = sum(prediction_times) / len(prediction_times)

    return avg_ram_usage, avg_prediction_time

# Use this function to measure the performance for your model
avg_ram_usage, avg_prediction_time = measure_performance(model, X_test)
```

## Appendix E: Command to generate DDoS traffic

1. *Performing DNS amplification DDoS attack*

```
  src.cmd("timeout 20s hping3 -1 -V -d 120 -w 64 -p 53 --flood {dst}")
```

2. *Performing Performing NTP amplification DDoS attack*

```
  src.cmd("timeout 20s hping3 -1 -V -d 120 -w 64 -p 123 --flood {dst}")
```

3. *Performing Performing ICMP (Ping) Flood DDoS attack*

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

4. *print("Performing UDP Flood DDoS attack ")*

```
  src.cmd("timeout 20s hping3 -2 -V -d 120 -w 64 --rand-source --flood
{}".format(dst))
```

5. *print("Performing TCP-SYN Flood DDoS attack ")*

```
  src.cmd('timeout 20s hping3 -S -V -d 120 -w 64 -p 80 --rand-source --flood
10.0.0.1')
```

6. *Performing LAND DDoS attack*

```
  src.cmd("timeout 20s hping3 -1 -V -d 120 -w 64 --flood -a {}
{}".format(dst,dst))
```

7. *Performing Slowloris DDoS attack*

```
    src.cmd("timeout 20s hping3 -S -V -d 0 -w 0 -p 80 --flood {}".format(dst))
```

## Appendix F. Dataset Features

| No | Column | Description |
|---|---|---|
| 1 | time_stamp | The exact time and date when the flow was recorded. |
| 2 | data_path_id | Unique identifier for the physical or virtual network path (Datapath) where the flow was observed |
| 3 | flow_identifier | Unique identifier for the specific flow of packets, irrespective of the Datapath |
| 4 | source_ip | Source IP address of the flow |
| 5 | source_trans_port: | Source port number of the flow. |
| 6 | destination_ip: | Destination IP address of the flow |
| 7 | destination_trans_port: | Destination port number of the flow. |
| 8 | ip_protocol: | IP protocol number (e.g., 6 for TCP, 17 for UDP, 1 for ICMP) |
| 9 | icmp_type_value: | ICMP code (only present for ICMP flows). |
| 10 | icmp_type: | ICMP type (only present for ICMP flows). |
| 11 | duration_seconds: | Duration of the flow in seconds. |
| 12 | duration_nanoseconds: | Duration of the flow in nanoseconds (more precise timing) |
| 13 | idle_time_out: | Idle timeout for the flow in seconds. |
| 14 | hard_time_out: | Hard timeout for the flow in seconds. This is more strict than the idle timeout and will forcefully terminate the flow even if it becomes active again after exceeding the idle timeout. |
| 15 | flag_values | Set of flags associated with the flow, each represented by a single letter (e.g., S for SYN flag, F for FIN flag). |
| 16 | total_packets | Total number of packets in the flow |
| 17 | total_bytes: | Total number of bytes transmitted and received in the flow. |
| 18 | packets_per_second: | Average number of packets per second within the flow duration. |
| 19 | nanosecond_packets: | Average number of packets per nanosecond within the flow duration (even finer-grained rate). |
| 20 | bytes_per_second: | Average number of bytes per second within the flow duration. |
| 21 | nanosecond_bytes: | Average number of bytes per nanosecond within the flow duration (even finer-grained rate for byte volume). |
| 22 | Traffic_Class | Multi-label network traffic flow classification: benign, UDP flood, TCP SYN, ICMP ping flood, Slowloris, Land attack, DNS amplification, NTP amplification. |

# Appendix G: Running of Mininet and Ryu