DETECTING AND MITIGATING BOTNET ATTACKS USING DEEP LEARNING IN SOFTWARE-DEFINED NETWORKS

MUHAMMAD WAQAS NADEEM

DOCTOR OF PHILOSOPHY COMPUTER SCIENCE

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

DECEMBER 2023

APPROVAL SHEET

This thesis entitled "**DETECTING AND MITIGATING BOTNET ATTACKS USING DEEP LEARNING IN SOFTWARE-DEFINED NETWORKS"** was prepared by MUHAMMAD WAQAS NADEEM and submitted as partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at Universiti Tunku Abdul Rahman.

Approved by:

(Dr. Goh Hock Guan) Date: 24/11/2023
Supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

(Dr. Aun Yichiet) Date: 26/11/2023
Co-supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

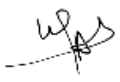
Date: 26/09/2023

SUBMISSION THESIS

It is hereby certified that ***Muhammad Waqas Nadeem*** (ID No:***20ACD06039***) has completed this thesis* entitled "*Detecting and Mitigating Botnet Attacks Using Deep Learning in Software-Defined Networks*" under the supervision of Goh Hock Guan (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology, and Aun Yichiet (Co-Supervisor) * from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my thesis in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



Muhammad Waqas Nadeem

DECLARATION

I Muhammad Waqas Nadeem hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

(Muhammad Waqas Nadeem)

Date: 26/09/2023

DEDICATION

Dedicated to my parents, wife, supervisors, and friends.

ACKNOWLEDGMENTS

I did not find a word to depict my deepest and heartiest gratitude, respectfulness, and thankfulness to my research advisor, Dr. Goh Hock Guan. During my Ph.D. journey, Dr. Goh acted as a life teacher by discussing my professional development and providing the space to improve my critical life-career skills, which helped me have a clear path to follow. Dr. Goh patiently listened to my challenges and pushed me to address them.

As a Ph.D. advisor, Dr. Goh provided significant advice, assistance, confidence, and the space that helped me to accomplish my Ph.D. successfully. This research work could not have been accomplished without the extraordinary guidance of Dr. Goh. Dr. Goh gave me the needed inspiration to be a successful researcher and teacher in the future. There is nothing I can give to repay Dr. Goh suitably. Thus, I will cherish your lessons and hopefully be an inspirational professor to others like you.

I'd like to express my deepest thankfulness and gratefulness to my co-advisor, Dr. Aun Yichiet and external co-supervisor, Dr. Vasaki Ponnusamy for their deep collaboration and strong support.

I'd also like to express my deepest thankfulness and gratefulness to Dr. Muhammad Adnan Khan and Dr. Muzammil Hussain, for their deep collaboration

and strong support. Dr. Muhammad Adnan Khan dedicated much effort and time to improving my research work by providing extraordinary advice and guidance. Besides, Dr. Muhammad Adnan Khan helped me improve my research and teaching skills by providing the needed inspiration and guidelines.

I also insistently and truly, wish to thank my parents, "Nadeem Ahmad and Yasmeen Bibi", and my sincere wife "Sufia Waris" for their unlimited support, encouragement, and assistance throughout my Ph.D. journey. They have told me that I can accomplish anything I have set out to achieve. My parents have also taught me patience and commitment to my goals. My parents and my wife are the main reasons for any personal success I have had and will have in the future. Many thanks to my brothers and friends Danyal Mahmood, Adnan, Salman Javed, and Ashar Wyne for standing with me through this process, for providing unlimited help, support, and encouragement, and that they have always trusted in my capabilities in combination with pushing me to achieve my goals.

A special thanks to the Department of Computer and Communication Technology for its support. It has been my honor to be a Ph.D. student in this department in the Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman.

DETECTING AND MITIGATING BOTNET ATTACKS USING DEEP LEARNING IN SOFTWARE-DEFINED NETWORKS

By

MUHAMMAD WAQAS NADEEM

A thesis submitted to the Department of Computer and Communication Technology,
Faculty of Information and Communication Technology,
Universiti Tunku Abdul Rahman,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science
December 2023

ABSTRACT

DETECTING AND MITIGATING BOTNET ATTACKS USING DEEP LEARNING IN SOFTWARE-DEFINED NETWORKS

MUHAMMAD WAQAS NADEEM

Software-Defined Networking (SDN) is a newly emerging network architecture separating control and data planes. It provides easy and flexible organization, management, and communication of complex or large-scale networks. Its programmable and centralized interfaces facilitate making complex and intelligent network decisions seamlessly and dynamically and can address the requirements of the data centers for managing the entire network. It also provides opportunities for individuals and businesses to build custom network applications based on their requirements and enhance their network services. Although SDN architecture offers high benefits, it introduced a new array of security and privacy challenges (i.e., single point of failure) that can preclude the wide adoption of SDNs. The SDN controller is a crucial element that attracts attackers to launch malicious attacks or

activities on the controller (s) through OpenFlow switches. Distributed Denial of Service (DDoS) and botnet attacks are considered dangerous threats for networks such as IoT, SDNs, cloud computing, etc. If the attacker accesses the SDN controller, it can reroute the network traffic, causing severe damage to the whole network. So, Network Intrusion Detection Systems (NIDSs) have become important tools to protect networks against malicious attacks. Deep learning (DL)-based network applications are trending and have shown promising results in detecting and mitigating potential threats with fast response. In this research, we analyze and show the classification performance in terms of detecting and real-time performance of various DL methods based on Recurrent Neural Networks (RNNs), Convolution Neural Networks (CNNs), Multilayer Perceptron (MLP), Deep Neural Networks (DNNs), and Long Short Term Memory (LSTM) for botnet-based DDoS attacks in an SDN environment. A new simulation-based dataset is developed and used to train deep learning methods. We also used feature weighting and threshold tuning methods to derive the significant features required for detection. The simulation outcomes and measurements are verified using a simulation-based dataset and a real-time testbed environment. The aim of comparative analysis among the DL methods is to find the lightweight DL method with baseline hyper-parameters, features and data that can be easily acquired to detect botnet-based DDoS attacks. The performance of the methods is evaluated using different metrics such as accuracy, detection rate, training and detection times, precision, F1 score, True Positive Rate (TPR), and False Positive Rate (FPR). The outcomes proved that the DL methods produced good results using optimal features. Finally, based

on the simulation results, we observed that the CNN method outperforms using the simulated dataset and in real testbed settings. The detection rate of CNN reaches 97% for attack flows and 99% for normal flows. We also adopted graph theory and dynamic flow deletion-based mitigation strategy to protect the SDN environment against botnet attacks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service Attacks
DDoS	Distributed Denial of Service Attacks
IDS	Intrusion Detection System
IoT	the Internet of Things
IT	Information Technology
LSTM	Long Short-Term Memory
NOS	Network Operating System
ML	Machine Learning
MLP	Multilayer Perceptron
SDN	Software Defined Networking

CHAPTER 1

INTRODUCTION

The rapid development of the Internet and the Internet of Things (IoT) [1] have explored the limitations of traditional networking [2]. The network community has used patching methods and techniques to address the emerging issues of the traditional network. These mechanisms make the networks more bloated, and the control over the network becomes weaker. Furthermore, traditional network architectures can no longer meet data centers' dynamic programming and storage needs. This makes it increasingly challenging to manage high heterogeneity networks in a distributed architecture, as production networks often have many network devices running various protocols and supporting different applications. So, the complexity ratio in traditional or heterogeneous network infrastructures is increasing, leading to resource management and optimization challenges.

Implementing intelligent solutions in future networks is necessary to address these challenges. A promising approach that has emerged in the last few years to introduce automation and intelligence to the internet is the Knowledge Plane (KP) approach [3], which utilizes cognitive and machine learning techniques [4]. However, the KP approach has yet to be implemented for several reasons. A significant obstacle is that conventional network systems are constructed with distributed characteristics, meaning that every node (like a router or switch) can only perceive and impact a limited network segment. As a result, controlling the local

network domain based on the limited information provided by the nodes can be extremely challenging [5]. Appreciatively, the recent development of Software Defined Networking (SDN) is expected to simplify the learning process.

Software Defined Networking (SDN) [6], [7], has resolved these issues by separating the control and data planes. Now a day, SDN has attracted the research and networking community due to its emerging and novel architecture, and it can meet the demands of fast-growing networking. It is also known as centralized control architecture, enabling the SDN controllers to access all the OpenFlow switches and control the whole network using southbound API interfaces [8], [9]. SDN controllers and the forwarding and distribution of OpenFlow switches is grasped to simplify the network's management and enhance the network capabilities over the traditional networks. The SDN controllers manages network resources. The controller has programmable qualities enabling it to program the entire network dynamically. It also has a global view of the network thanks to its real-time monitoring and information collection of packet and network data information. Furthermore, SDN has been applied in the networking world for different use cases, such as network traffic monitoring, traffic engineering, networking of data centers, and Quality of Services (QoS). Additionally, compared to traditional networks, SDN's flexibility enhanced security measures, including threat detection and prevention.

SDN architecture consists of three layers: data, control, and application. The application layer runs all the rules and policies defined by the network administrator. Then these rules and policies are dynamically forwarded to the SDN

controllers for execution. The controllers act and control the network according to the rules. Any modification in the application layer may change the entire network behavior. Hence, the application layer is a great innovation of the Open-Source (OS) platform, which does not force the network administrators to depend on vendors [10] completely. SDN allows administrators to build customized network applications over general-purpose hardware without license constraints. The control layer, which is referred to as the brain of the SDN architecture, is where the SDN controllers operate. The application layer passes administrator rules to the controllers; after that, these rules are decoded by the controller and then forwarded to the data layer. Once these rules are implemented in the data layers, the controllers collect the feedback from the data layer and pass it back to the application layer. The data layer is known as the non-intelligent layer, and it has different types of hardware devices, such as switches, routers, etc. The control layer passes the instructions to the data layer [11].

Furthermore, SDN facilities and simplifies the development, deployment, and maintenance of the networks compared to traditional networks. Updating or introducing new network applications may improve the network's features and services. SDN-based networks have almost no compatibility issues and can be run on simple hardware devices, making these networks cost-effective [12]. The entire network accessibility is allowed without revealing information about the underlying layers. Hence, SDN is an innovative invention that provides controllability and flexibility over a network. Due to its central control, it is known as a double-edged sword, meaning that a single controller can easily manage the

whole network. The security measures of traditional networks can be improved with the help of SDN.

Although SDN has great capabilities compared to traditional networks, it's emerging architecture and centralized control introduced a new array of security threats that can lead toward a single point of failure [13]. To support the vision of the next generation of networking [13], extensively trustworthy security measures are also needed for SDN. Different attacks, such as DDoS, botnets, saturation, and other types, can be easily launched toward SDN controllers due to their centralized nature.

Now a days, due to advancements in technology, botnet attacks have become more dangerous attacks, and they are considered a critical threat in the next generation of networking [14]. The botnet attacks are categorized into network-based attacks that can breach multiple computers into "bots" to make a "bot force." They could perform malicious activities such as information theft, spamming, DDoS, Domain Name System (DNS) spoofing, phishing, etc. In a botnet attack, a malicious actor, known as a "botmaster," attempts to get unauthorized access to a single device; after that, it tries to control the device using different botnet malware without disturbing normal users. The attacker also makes a Command and Control (C&C) center to establish a connection among the bots. The instructions related to malicious activities are passed from C&C to bots.

The Internet of Things (IoT) and SDN's most sophisticated DDoS attacks in recent years have often been launched using botnet technologies. [15]–[17]. The flexibility and power of the technology enable the botnets to generate several types of DDoS attacks. There are a few reasons behind the use of modern technology by the attackers, which are as follows:

- A big force of bots can fastly generate more powerful flooding attacks.
- The actual attacker cannot be found easily.
- The security mechanisms can be dodged using several network protocols.
- They can produce attack traffic that resembles regular traffic, making real-time detection challenging.

Negatively, the advancement in technology makes the attacker more competent, and they know they can control the whole SDN-based networks if they get access to the controllers. Hence, multiple computers can be easily breached into bots, and then a member of the botnet force to perform malicious actions on the SDM controllers. So, the botnets have resulted in the progression of huge and severe DDoS attacks against the SDN controllers and become a cause of a single point of failure.

DDoS attacks are typical threats that can be launched with bots for different purposes, such as sending many requests to exhaust the system’s or controller’s resources, synchronizing fake traffic, or blocking the network for legitimate users. In DDoS attacks, the victims’ sensitive information is not lost as in other network attacks. Instead, the users may experience out-of-service issues during the attack

period. Generally, a DDoS attack can be easily launched, but it is hard to trace, e.g., because of IP Spoofing [18]. In an SDN environment, attackers use spoofed sources' IP addresses to send many malicious packets to flood the network, and these packets cannot match the switch's flow table entries. Then, the switches send "Packet-In "messages to the controller to request how to handle these new packets. In response, the controller sends new flow rules to the corresponding switches. To process these spoofed packets, the controller uses excessive memory and computing resources, which make it unavailable to handle legitimate requests of normal users. Thus, the attackers successfully launched a DDoS attack against the controller. The DDoS attacks are categorized into application layer attacks, protocol attacks, and volumetric attacks. In application layer attacks, the attacker consumes controllers' computing resources and bandwidth to prevent them from providing legal services. In protocol attacks, the attackers attempt to take advantage of protocol rules by using the target server's current states, such as the "three-way handshake process of TCP." Attackers flooded the network with malicious traffic to perform volumetric attacks, depleting the controllers' resources, including CPU usage and bandwidth [19]. DDoS attacks against the data plane of SDN are quite similar to the traditional networks; they could impact the switch's processing capabilities of switches links. However, DDoS attacks against the control or application planes are considered more dangerous because these two planes are responsible for making and implementing network policies. In other words, these are high-profit targets for attackers to interrupt the network. This research focuses

on volumetric DDoS attacks because they are a long-term threat to SDN or other networks due to rapid growth in network-connected devices.

Deep learning-based techniques and methods are commonly applied for detecting and mitigating intrusion in different networking fields. For instance, In traditional networks, AI detection of Distributed Denial of Service (DDoS) attacks typically involves the use of anomaly detection algorithms and signature-based methods. Anomaly detection relies on establishing a baseline of normal network behavior and flagging deviations from this baseline as potential threats. Signature-based detection involves identifying known patterns or signatures of DDoS attacks by comparing network traffic to a predefined set of attack signatures. However, traditional networks may face limitations in adapting quickly to new and evolving DDoS attack techniques, and false positives/negatives are common challenges.

On the other hand, in Software-Defined Networking (SDN), AI detection of DDoS attacks can be more dynamic and responsive. SDN separates the control plane from the data plane, allowing for centralized network management and programmability. This enables the deployment of machine learning models that can adapt to changing network conditions in real-time. SDN's centralized control facilitates faster decision-making for traffic diversion or mitigation strategies, enhancing the network's ability to detect and respond to DDoS attacks more efficiently compared to traditional networks. The flexibility and programmability of SDN enable the integration of advanced AI techniques for improved threat detection and mitigation.

Furthermore, examining SDN behavior is a new research area in which deep learning techniques are applied to study the behavior, it could become the first attempt for early detection [20], [21] because machines can respond faster than humans. Deep learning techniques use training, trial, and error methods to empower the machines' decisions. These techniques realize the network behavior using historical data and can predict incoming traffic flows. Positively, DL techniques produced excellent results in classifying attacks or normal traffic flows. Instead of relying on packet payload, the DL-based methods take a particular set of features to make predictions. It means DL methods could help the SDN controller to monitor and recognize the network states in real-time. In different studies [4], [22], to predict DDoS attacks in SDN, the authors used machine learning(ML) and deep learning (DL) approaches. However, these methods and techniques have not achieved appropriate accuracies for detecting DDoS attacks in SDN. They also trained and evaluated their methods on old datasets generated in traditional network environments. Furthermore, in some studies [23]–[26], the authors used many features to train DL/ML methods. The extraction and collection of more features from real traffic flows is a tough and time-consuming task because of accessibility or authorization. Furthermore, the ML/DL methods show low efficiency in detection time (e.g., take a long detection time), which causes a delay in response, and the attack traffic cannot be detected in time. Their mitigation methods consume excessive computing and CPU resources [27], [28]. Moreover, we also need to focus on botnet-based DDoS attacks because these attacks can potentially affect the SDN controllers or the whole network due to centralized natures. Lastly, in existing

studies [7], [13], most of the methods are verified by simulations or experimental datasets only; the real-time testbed validation of these methods is intermittent.

1.1 Motivations and Problem Statement

In the current era of technology, cyber-attacks have become critical threats to computer networks. The security of communication and computer networks is becoming more important as more and more devices connect to the networks. Botnet attacks are categorized as the most dangerous cyber-attacks that become a major threat to the security of any IT infrastructure. As more devices come online everywhere, the potential for new and strong botnet malware has increased proportionally. Hackers have evolved and scaled their attacks to match modern security systems. In traditional networks, the Botnet owners can access several thousand computers at a time and command them to carry out malicious activities. Botnets can perform Distributed Denial-of-Service (DDoS) attacks, steal data, send spam, and allow the attacker to access the device and its connection. In traditional networks, the botnet attack easily breaks down a segment or whole network [29]. These networks are changed into SDN to prevent malicious attacks [30]. Botnet becomes smarter, and they know the architecture of the SDN as in an SDN network; the entire network goes down if the SDN controller is down. [31]. Hence, Botnet and DDoS attacks are widespread and can damage SDN-based networks [32]–[34]. Botnets can take control of the SDN network by attacking the control plane. In botnet-based DDoS attacks toward the control plane of SDN, the attacker can use spoofing IP addresses to generate many traffic flows that cannot match flow rules. The intelligent protection of the SDN from Botnet-based DDoS attacks is

significant because all traditional networks are converted into SDN-based networks [35], [36] to support the vision of the Next generation of networks. So, to maintain secure and sustainable cyberspaces for the SDN, advanced intrusion systems should be applied in the SDN controllers. These systems detect, prevent, and respond to the attacks. For example, an intrusion detection system uses those techniques that effectively detect external and internal intrusions or suspicious activities that can target a controller.

Traditional methods are based on statistical analysis [37], [38] and signature-based detection and prevention [39], [40], which are encountering scalability issues due to an increase in attack space. The security industry has focused on deep learning and machine learning-based approaches and techniques to detect and mitigate these attacks. Existing techniques, however, rely on statistical analysis, machine learning, or deep learning and thus exhibit low accuracy and reduce detection granularity. [37], [38]. Some other methods based on machine learning exhibit low efficiency in terms of detection time, which may cause a delay in the response, and attack traffic cannot be detected early. They are using many traffic features to detect botnet-based DDoS, increasing CPU usage and other computing resources. Furthermore, previous work focuses on old datasets (from the literature review part), and they have not ventured into the botnet, and previous work gives low prediction accuracy using old datasets.

As a result, machine learning and deep learning-based network security solutions for SDN are becoming increasingly common these days. This inspires me to research and analyze advanced deep learning methods to detect and prevent

botnet-based DDoS attacks in a setting supported by SDN. To protect the SDN controllers' computational resources, this research employed feature weighting and threshold tuning as the basis for the best feature selection. The selection of important traffic features can help to improve the detection accuracy and performance of the DL methods. A dynamic flow deletion and graph theory-based dropping strategy is adopted to defend the SDN. A botnet-based DDoS attack can control the controllers and flood the network with malicious traffic. So, a dedicated bot management system based on the DL method could become an optimal solution for the SDN controller to detect and protect from botnet-based DDoS traffic. Lastly, the progressive development to enhance the security measures of the SDN encouraged me to develop a DL method named "DepBot" to detect and mitigate botnet-based DDoS attacks. Table 1.1 shows the contribution of this research compared to the existing research.

Table 1.1: Comparison of this research with the existing research.

Study	Algorithm used	Feature Engineering	Flow-based	Packet-based	SDN Environment	Botnet	DDoS
[13]	IE-CNN	×	×	✓	✓	×	✓
[24]	Random Forest, Decision Tree, Bagging, k-Nearest Neighbor, and Support Vector Machine	✓	×	✓	×	✓	×
[26]	PSO, DNN, DT, and SVM	✓	✓	×	×	✓	✓

[41]	Deep auto encoders	×	✓	×	×	✓	✓
[42]	SVM	✓	✓	×	✓	×	✓
[43]	Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs)	×	✓	×	×	✓	×
[44]	LSTM-CNN	×	×	✓	×	✓	×
[45]	FT-EHO-DBN	×	×	✓	×	×	✓
[46]	SVM	×	✓	×	✓	×	✓
This research	RNN, CNN, MLP, LSTM, and DNN	✓	✓	×	✓	✓	✓

1.2 Research Questions

RQ1: Are the publically available intrusion detection datasets can use for SDN?

RQ2: Is Deep Learning methods can effectively detect botnet attacks in SDN?

RQ3: How can we protect the SDN from botnet attacks?

1.3 Research Objectives

RO1: To develop a specific dataset related to the botnet and simulation environment for training and testing deep learning methods. Because the existing datasets used in different studies [17] [20] are old or traditional datasets (i.e., NSL-

KDD, DARPA, Na-BaIoT, CIC-DoS-2017, etc.). These datasets are not specifically used the detection of botnet-based DDoS attacks. Furthermore, these datasets are unsuitable for SDN environments because they are developed in traditional network environments and suffer imbalanced problems.

RO2: To propose a deep learning-based method with baseline parameters for detecting botnets early in the SDN. Because the existing deep learning methods are trained using traditional or old datasets, and they use many traffic features that enhance the consumption of time by any DL methods during training and real-time detection. Furthermore, complex network architectures for the DL methods are also used to improve the accuracy or detection rate. Moreover, the existing DL methods are tested and verified only on experimental datasets; so, the real-time testbed evaluation is intermittent.

RO3: To develop a mitigation strategy to defend the SDN controller from DDoS attacks powered by botnets. Most of the existing mitigation strategy is based deletion of traffic flows after detecting the attacks, which enhances the chance of killing normal flows. The existing studies do not focus on finding the switches with attack flows. So, a mitigation strategy is needed to find the attack path and then adopt a more targeted dropping rate to avoid the by-mistake killing of normal flows.

1.4 Contributions of the Research

- The quality of the training dataset has a substantial impact on the classification performance of DL-based IDS systems. However, the unavailability of the benchmark datasets could interrupt the development of

advanced DL-based IDS for the SDN environments. The main issues behind the lack of benchmark datasets are the security and privacy of the networks. This is the first effort to construct a simulation dataset exclusively for detecting botnet-based DDoS attacks in an SDN-assisted environment. This dataset is collected in a pure SDN simulated network environment by generating both the normal and botnet-based DDoS attack traffics. The collected simulated dataset consists of both normal and attack traffic records and has a total around 89000 records. This dataset can be used for further research in the order to secure the SDN.

- The DL methods can explore and learn the intrusion patterns in the training data and achieve better results than traditional ML methods. So, in this research, a comprehensive analysis on the performance of the DL methods is performed for detecting the botnet-based DDoS attacks using the simulated dataset and real-time testbed. Furthermore, different subsets of the optimal features have been used to improve the detection accuracy of DL methods while detecting the attack flows and finding out the optimal subset of features. Based on the simulated results, we observed that the CNN methods produced superior results using a subset of 30 features and in the real-time testbed environment. To intelligently detect botnet-based DDoS attacks in the SDN, the CNN approach is implemented in the SDN controller.
- The controller(s) responsible for dynamically checking the network flows in specific time intervals and activating the defense shield after successfully

detecting the attacks in the SDN environments. We developed and deployed a graph theory and deletion-based defense strategy to defend the SDN controller(s) against botnet-based DDoS attacks. Our proposed defense methods calculate the variable dropping rates for different OpenFlow switches based on their location in the attack path to avoid the by-mistake killing of normal flows. We measured the effectiveness of the defense methods in terms of flow generation rates and CPU utilization and compared them with the most recent existing defense methods. Based on the reported results and comparison with existing methods, our defense strategy mitigates the botnet-based DDoS attack in a real-time SDN network with minimal resource consumption.

1.5 Organization of Thesis

The rest of the thesis is structured as follows: A review of the SDN architecture is provided in Chapter 2, along with a brief explanation of machine learning and deep learning techniques and their application to SDN security. The simulation model for identifying and thwarting botnet-based DDoS attacks in SDN is described in Chapter 3. The results of the simulation used to identify botnet attacks using deep learning techniques are described in Chapter 4. In addition to describing the suggested defense strategy's implementation and effectiveness, Chapter 5 also illustrates how it counters botnet attacks. The thesis is finally concluded in Chapter 6 with an outline of forthcoming research.

CHAPTER 2

LITERATURE REVIEW

The chapter covers the existing and recent research on securing the SDN from different types of malicious attacks (i.e., DDoS, Botnet, Saturation, and Ransomware, etc.) using machine learning and deep learning algorithms.

2.1 Architecture of SDN

Over the past few years, SDN has gained significant attention. The Open Network Foundation (ONF) [47] is a nonprofit organization with the development, standardization, and growth of SDN as one of its main objectives. As defined by the ONF, SDN design separated the control and data planes, centralized network state and intelligence, and derived the essential network infrastructure from applications. [48]. The control plane, data plane, and application plane are the three main planes that make up the high-level architecture of SDN, which is based on this concept. Figure 2.1 illustrates the components comprising the architecture of each plane, as well as how they interact with one another. In the subsequent sections, we overview three planes and how they interact.

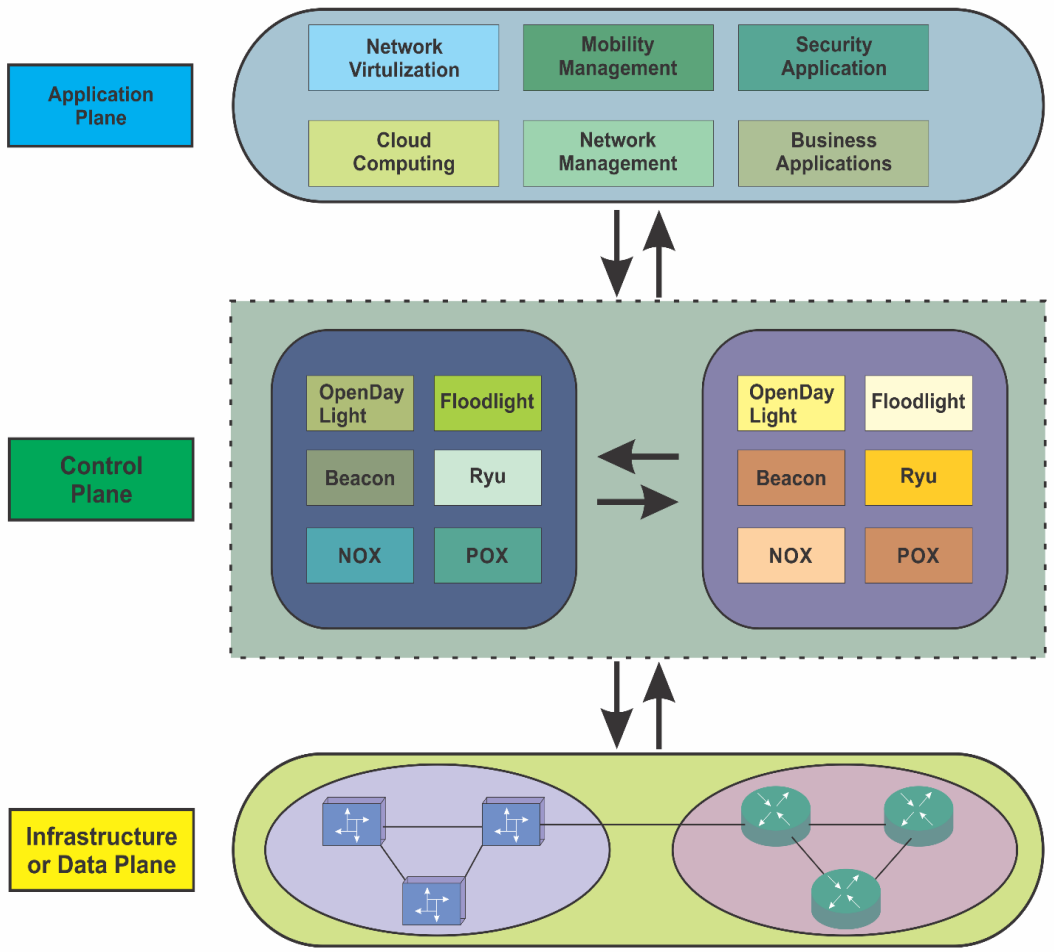


Figure 2.1: Block diagram of the SDN architecture. The physical and virtual network devices are operated at the data plane. The data and control planes are connected through the southbound interfaces. The westbound or eastbound is responsible for the communication among the controllers. The application and control planes are connected through northbound interfaces. The application plane is responsible for running different SDN-based business applications.

2.1.1 Data Plane

The data plane, often called the infrastructure plane, is the lowest level of the SDN architecture and contains forwarding components, including physical and virtual switches. Virtual switches run on standard operating systems like Linux [49]. On the other hand, physical switches are made of hardware, either merchant switches (which vendors install on networking gear) or open network hardware

switches (i.e., “NetFPGA. Pantou, Open vSwitch [50], and Indigo [51]”) are three examples of virtual switches [52], while examples of NetFPGA-based switches are (ie., “ServerSwitch [53] and SwitchBlade [54]”). Networking hardware vendors like Cisco, HP, Huawei, etc., have begun supporting SDN protocols in their commercial switches. Physical switches have a faster flow forwarding rate but are less full and flexible than virtual switches, which often sacrifice all SDN protocol features. The switches are in charge of flow-forwarding, modification, and packet dropping in the data plane following the control plane flow policies. These switches use Southbound Interfaces (SBIs) to communicate with the CP, which controls the processing and forwarding capabilities of the data planes.

2.1.2 Control Plane

SDN-based systems have a control plane, commonly called the "brain," responsible for dynamically updating forwarding rules, programming network resources, and improving network administration's flexibility and agility. The central component of the control plane is the logically centralized SDN controller that manages communication between applications and forwarding devices. The controller offers vital features that applications need, including shortest path routing, device configuration, network condition information notifications, and network topology storage. It converts application requirements into customized policies, which are then disseminated across the forwarding devices. Several controller architectures, such as (i.e., “Floodlight [55], POX [56], NOX [57], Beacon [58], Open Daylight [59], and Ryu [60]”), have been introduced. To interact with other planes, the controller utilizes three communication interfaces, including

northbound, southbound, and westbound/eastbound. On the one hand, the controller delivers a summary of network state data to the application plane.

Control Data Plane Interfaces (CDPIs), also known as Southbound Interfaces (SBIs), facilitate communication between the control and data planes. Through these interfaces, forwarding devices can communicate with the control plane about network state and control policies, offer forwarding features, promote device capabilities, produce statistical reports, and send event notifications. While the ONF-based OpenFlow [61] the primary and most popular open-standard SBI, other proposals are less popular, such as (i.e., "OpFlex [62], ForCES [63], OVSDB [64], Protocol Oblivious Forwarding (POF) [65], Open State [66], LISP [67], and NETCONF [68]").

- The Northbound Interfaces (NBIs) refer to interfaces defined between the application and control planes, allowing various applications to access and interpret the abstract network information the control plane provides. They facilitate SDN networks' management, innovation, and automation by enabling the expression of network requirements and behaviors. The Open Networking Foundation (ONF) has established a set of standard NBIs and a common information model [68].
- In multi-controller network architectures, westbound/eastbound interfaces are utilized. These interfaces are employed in large-scale networks, divided into several sub-domains, each with a separate controller. As the flow data in these networks exceeds the processing capacity of a single controller, multiple SDN controllers are deployed. To provide an overall view of the

network to the upper-layer applications, it is essential to establish communication between the various controllers for exchanging network information. The private westbound/eastbound interfaces of the Hyper Flow [69]and Onix [70] distributed control architectures do not allow them to communicate with each other. Several proposals have been put forward to facilitate communication between different types of controllers. These include the West-East Bridges [71], SDNi [72], and Communication Interface for Distributed Control Plane (CIDCP) [73], which serve as westbound/eastbound interfaces for exchanging network state information. However, the standardization of these interfaces is still necessary.

2.1.3 Application Plane

In the SDN design, the application layer is the uppermost layer and is in charge of numerous business applications that offer services, business administration, and optimization. The NBIs allow the distribution of network state information among these applications, which can use this information to modify network behaviors based on their specific business requirements.

The acceptance of SDN-based applications has grown significantly in academia and industries. For instance, [74] conducted a survey on SDN-based Traffic Engineering (TE) solutions, and SDN security has been reviewed in [27], [75]–[79], are focused on DDoS attacks in SDN and cloud computing. Additionally, fault management in SDN has been explored in [80], while [81] introduced a Four-Dimensional (4D) evaluation framework for SDN centralized Quality of Service mechanisms.

SDN has been implemented in numerous networks because to its inherent benefits, including logically centralized control, dynamically updated forwarding rules, and a global network perspective. Including optical networks [82], Internet of Things (IoT) [83], transport networks [84], wireless networks [85], Wide Area Network (WAN) [86], cloud computing [87], edge computing [88], and Network Function Virtualization (NFV) [89], [90]. Various resources are available in [91]–[97] for further reading on SDN.

2.1.4 SDN Workflow

The basic operations of an OpenFlow-based SDN network architecture involve the following steps:

- The OpenFlow switch uses flow entries to determine the proper data plane processing for incoming packets.
- After that, the switch collects the header fields from each packet it receives on the data plane and compares them to flow entries previously saved.
- The switch uses its local flow rules to act when any packet matches its current flow entries.
- The switch sends a Packet-In OpenFlow message to the SDN controller if the packet does not match the switch's current flow entries. (arrows 2 and 5).
- • Packet-In messages can contain the entire packet's contents or only the header information.

- After that, the controller develops fresh flow rules in accordance with the Packet-In message.
- The controller sends a Flow-Mod message to the switch to regulate and modify the flow rules (arrows 3 and 6).
- Finally, the switching process of the subsequent packets is based on new flow rules.

This workflow allows the SDN controller to manage the network by dynamically updating the switch flow entries. This flexibility enables the SDN architecture to adapt to changing network conditions and optimize performance. Figure 2.2 will help to understand the working of the SDN network [98].

2.2 Overview of Machine Learning and Deep Learning Algorithms

Machine learning is an advanced aspect of artificial intelligence (AI) that utilizes powerful algorithms and is widely employed in data mining. Its primary function is to enable models to recognize structural patterns from the training data. Training and decision-making are typically the two phases of a machine learning model, as shown in Figure 2.3. Machine learning models acquire the system model during training by utilizing the training datasets. Conversely, the trained model predicts the anticipated output for every new input in the decision-making phase.

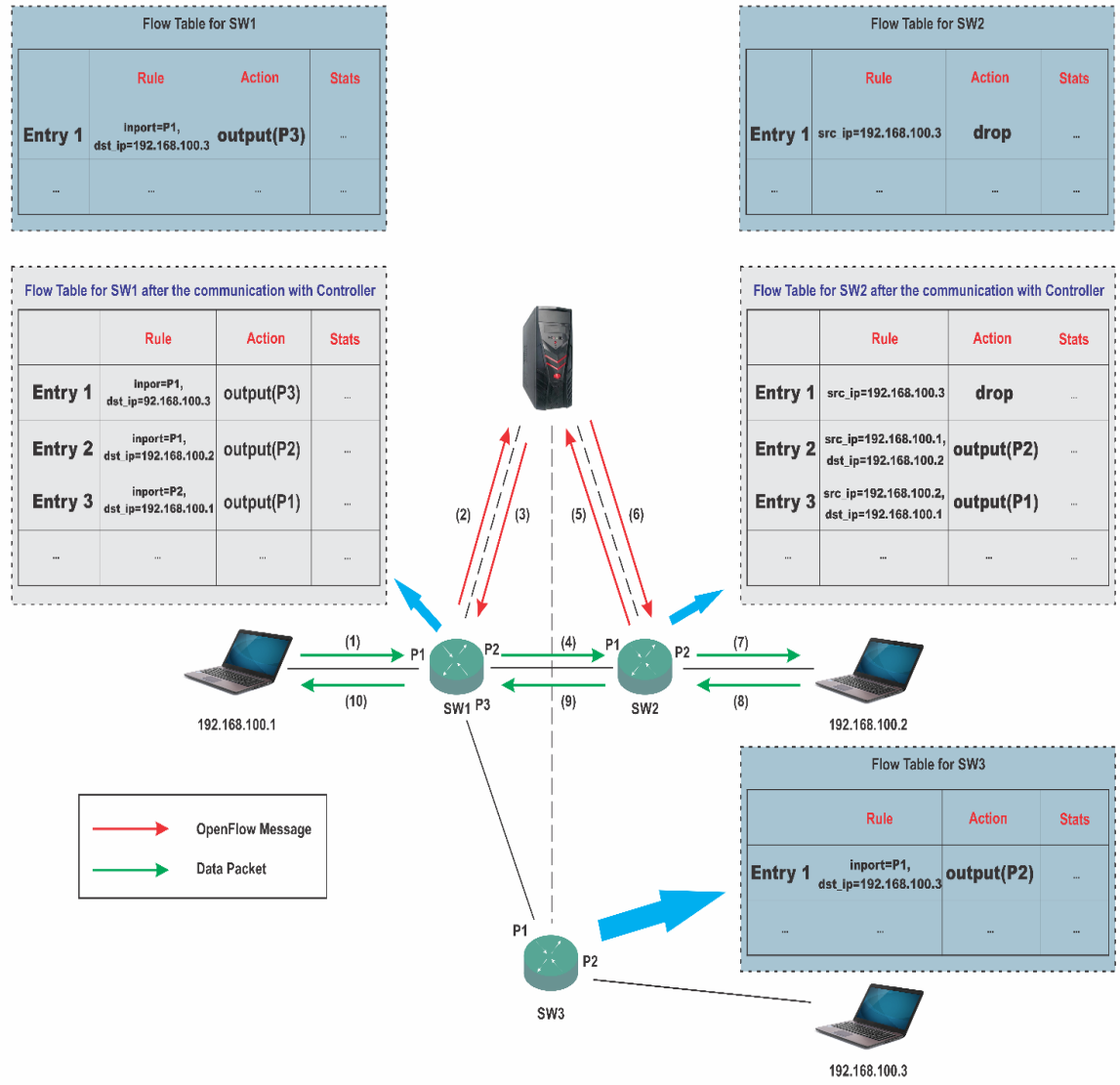


Figure 2.2: General working of an OpenFlow SDN network. Let, the "SW1" and "SW2" has two flow entries (i.e., Entry-2 and Entry-3), then the communications are allowed between two hots with IP "192.168.100.1" and "192.168.100.2". However, the traffic from "192.168.100.3" to "192.168.100.2" is not permitted due to security policies of "SW2".

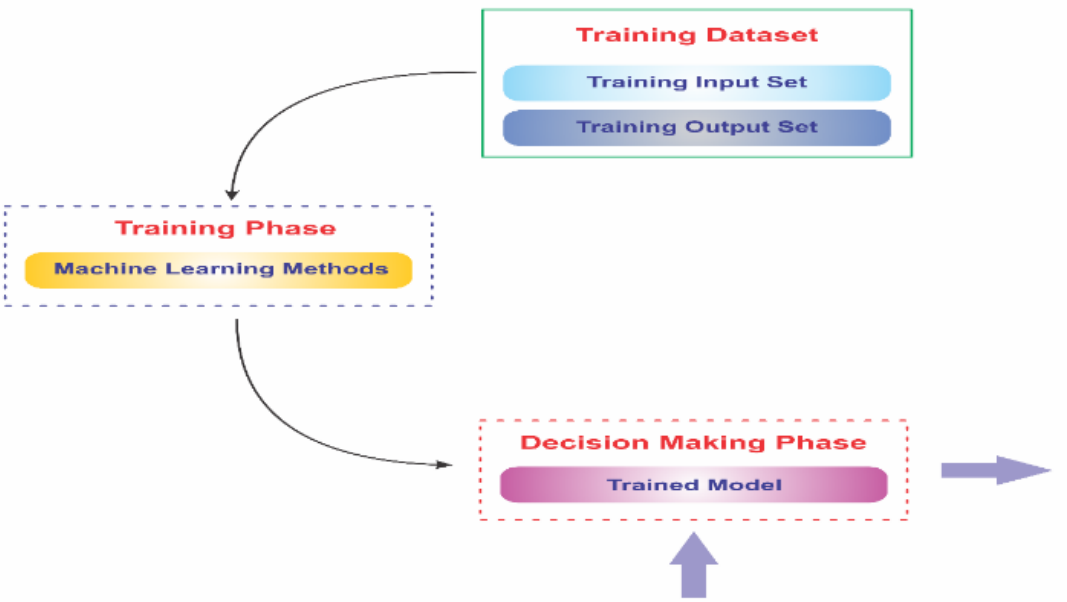


Figure 2.3: The working mechanism of a machine learning model.

Four different learning methods for machine learning algorithms have been defined: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning as shown in Figure 2.4. This section will provide a detailed description of the various machine learning algorithms introduced. To gain a deeper understanding of the theories and classical concepts related to these algorithms, we recommend referring to sources [99]–[102]. Machine learning models can also automatically learn from data [103] and uncover hidden patterns without explicit programming [104]. These techniques are recognized as highly efficient methods for reducing false alarm rates, communication, and computational costs [105] while improving the detection rate.

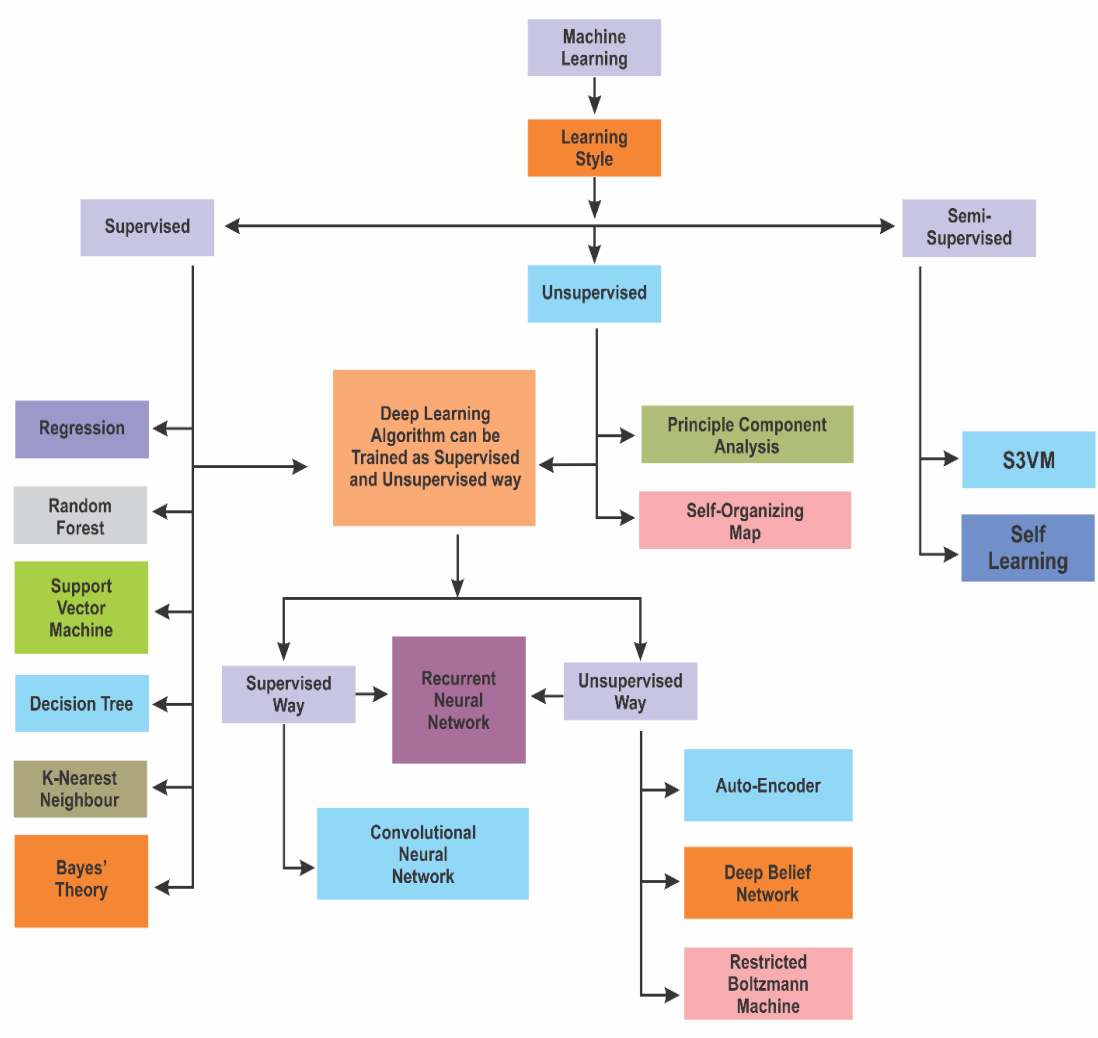


Figure 2.4: Taxonomy of standard machine learning and deep learning algorithms.

2.2.1 Supervised Learning

Supervised learning involves using labeled input data by ML algorithms to learn the representation and predict unknown cases [106], [107]. The most common algorithms used in supervised learning include SVM, Decision Tree (DT), RF, k-KNN, ANN, Bayes' Theory (BT), and Hidden Markov Models (HMMs) [108]. For

instance, SVM-based algorithms are utilized to solve classification problems, while RF-based algorithms are used for regression and classification problems [103]. SVM algorithms are prevalent in NIDS research due to their practicality in computation and classification prowess. These algorithms can yield successful outcomes for high-dimensional data; however, it is essential to choose an appropriate kernel function for optimal results. They also consume significant resources, requiring high computational power and memory [104]. Although RF algorithms [109] can handle variable data, they face the problem of over-fitting.

2.2.2 Unsupervised Learning

Unsupervised learning algorithms operate differently from supervised learning algorithms by being fed unlabeled input data. These algorithms aim to discover structures, patterns, or knowledge in the unlabeled data and group them into clusters based on their similarities. Unsupervised learning algorithms are mainly used for data aggregation, clustering, and feature reduction [99], [106]. For instance, Principle Component Analysis (PCA) is used for feature reduction, and Self-Organizing Map (SOM) is utilized for clustering. PCA also speeds up the unsupervised feature learning process [110]. In various studies, such as [111], the author used PCA as a feature selector before classification. Distance learning and clustering-based methods have been used for anomaly detection. Artificial Neural Network (ANN) based SOM is used to reduce the payload in NIDS [112]. However, clustering algorithms used in anomaly detection have some limitations that are

subjective to the Initial conditions of the clustering algorithms, such as centroid. They may produce a high False Positive Rate (FPR) [113].

2.2.3 Semi-Supervised Learning

Semi-supervised learning [114] combines labeled and unlabeled data to train algorithms, which is particularly useful when only a limited amount of labeled data is available. In situations such as image archives, where many images are unlabeled, and some are labeled, semi-supervised learning can be a more practical approach. This approach offers several advantages over supervised and unsupervised learning, including using a relatively cheap and easy source of unlabeled data to improve model performance. Different assumptions are considered in semi-supervised learning to effectively utilize labeled data, including clustering, manifold, smoothness, and low-density separation. Pseudo-labeling [115] is a popular semi-supervised learning technique that involves training a model using labeled data, predicting pseudo-labels for unlabeled data, and then combining this newly labeled data with the original labeled data to improve model accuracy. Several semi-supervised learning methods exist, including Co-training, transductive SVM, Expectation-Maximization (EM), and graph methods, each relying on different assumptions [116]. For example, transductive SVM is based on the low-density separation assumption, EM builds on the clustering assumption, and graph-based methods are built on various hypotheses. In network intrusion detection, semi-supervised learning techniques such as semi-supervised support vector machines, Gaussian Fields (GFs), Spectral Graph Transducers, and MPCK have been used to enhance the performance of detection systems [117], [118].

2.2.4 Reinforcement Learning

Three crucial elements make up the well-known learning method known as (i.e., “RL: an agent, a state space, and an action space”)[119]. The agent works with the environment to determine the optimum course of action and raise the cumulative reward—the total of all current and future rewards discounted over time. RL is used in the context of SDN architecture by treating the network as the environment and the SDN controller as the agent. The SDN controller learns to make the best decisions for managing traffic flow due to its ongoing monitoring of the network's status.

2.2.5 Deep Learning

DL algorithms are a recent update to Artificial Neural Networks (ANN) that take advantage of inexpensive and abundant computing power. DL allows algorithms to explore various levels of generalization in the different data representations. These algorithms have applications in network intrusion detection, object recognition, and other fields [120]. Both supervised and unsupervised techniques can be used to train DL algorithms. [103]. For instance, the Convolutional Neural Network (CNN), a standard DL algorithm, is typically trained in supervised [121], while Auto-encoders and Deep Belief Network (DBN) algorithms use unsupervised training [122]. Some algorithms, such as Recurrent Neural Networks (RNN), can be trained using supervised or unsupervised methods.

In conclusion, supervised learning approaches focus on classification and regression problems, whereas unsupervised learning and reinforcement learning strategies are frequently used for clustering and decision-making tasks.

2.3 Machine Learning and Deep Learning for Security in SDN

A centralized network view is provided by the SDN controller, which also makes control and maintenance easier. Leveraging deep learning algorithms and techniques can introduce intelligence into the controller, enabling it to analyze network data, optimize network performance, and automatically provide network services. The SDN controller can make the best decisions to adjust to changes in the network environment thanks to this learning capacity. This section thoroughly analyzes recent advances in deep learning and machine learning that address security issues in SDN. It also discusses how real-time these algorithms are employed in this context.

2.3.1 Benchmark Datasets

This section gives a brief overview of the commonly used network traffic-based datasets and attack types that are mentioned in these datasets. The datasets which were used in various studies are summarized in Table 2.1. The data amount for each class from the datasets is described in Table 2.2.

- NSL-KDD

This dataset is a modified version of KDD Cup 99. The repetitive and unnecessary records have been removed from this dataset, which now contains

sufficient records. After eliminating unnecessary and duplicate records, it has been reduced from 5 million records to around 150,000 records. This dataset is divided into predefined training and testing subsets for the IDS. It has the same classes and properties as KDD 99. The different attacks, such as Probing, U2R, R2L, and DoS, are also simulated for this dataset.

- CSE-CIC-IDS 2017

This dataset was created by Canadian Institute for Cybersecurity (CIC) and Communications Security Establishment (CSE) in 2017. A laboratory-based test environment with the victim and Offensive networks has been used to create this dataset. The network where the attacks were made, a computer with a Kali Linux operating system, a switch, and three computers with Windows 8. While in the target network, there is one router, one server with Ubuntu-12, one with Ubuntu-16, one Windows server-16, and one firewall. The Windows server-16 activated the directory features, and all the connected devices in the victim network were in the same domain. The Uplink port of the routers was mirrored to view the network traffic on the victim.

An agent based on the Java-B-profile system was used to generate the normal network traffic. With this agent, some protocols such as email, HTTPS, HTTP, SSH, and FTP have been reproduced using statistical and ML algorithms. The attack traffic was generated using tools such as Ares, Metasploit, Slowhttps, and slowloris (i.e., “DDoS, Dos, Heartbleed, Web attacks, Brute Force, and Infiltration attacks”). In this dataset, 14 types of attacks are labeled. These attacks

include (i.e., “DoS hulk, DoS Slowloris, Heartbleed, DoS Slow HTTP, DoS Golden Eye, SQL injection, Brute force, FP, DDoS, XSS, PortScann, Patator, SSH-Patator, infiltration, and Botnet”). In addition, the CIC Flow Meter was used to convert the captured network traffic into 80 features and create this dataset.

- CIC-DDoS2019

The CIC created this dataset in 2019 using a Wireshark-emulated environment. It comprises 50,063,112 records, containing 56,863 rows of normal traffic and 50,006,249 rows of attack traffic, with 80 different features. The training dataset consists of 12 DDoS attacks, (i.e., “DNS, LDAP, SNMP, MSSQL, UDP, NetBIOS, NTP, WebDDoS, TFTP, SSDP, SYN, and UDP-Lag”). On the other hand, the test dataset encompasses seven attacks, which include (i.e., “PortScan, MSSQL, UDP-Lag, SYN, NetBIOS, LDAP, and UDP”).

- ISCX-2012

This data collection, which includes both malicious and legitimate network activity, was created from network data collected over seven days. Attacks on malicious traffic include Brute Force SSH, DDoS, Infiltration, and HTTP-DoS.

- ISOT

5,424 network traffic flows from several variants of normal and botnet datasets were used to produce this dataset. The malicious traffic in this dataset was gathered from the Honeynet French Chapter, which uses the Waledac and Strom botnets. The normal traffic was obtained from Ericson Research Lab. The network

traffic was created using applications such as World of Warcraft, Azures, and HTTP Web Browsers.

- KDD Cup 99

This dataset is an advanced version based on DARPA 98 dataset program. The different attacks, such as Probing, User to Root (U2R), Remote to Local (R2L), and Dos, are simulated. The dataset has around 5 million lines collected during seven weeks of network traffic. This dataset is most commonly used for evaluating and assessing Intrusion Detection Systems (IDS).

- CTU-13

This dataset combines 13 attacks and is collected over a nonfictional network environment. It captures the real and mixed botnet network traffic. The dataset captures three network traffic types: botnet attacks, normal, and background. The infected hosts generated the Botnet traffic, and normal hosts were used for normal traffic. The 13 d scenarios were used to create the different botnet samples, and each was executed for a particular malware with various protocols. This dataset is a more labeled and largest dataset created by the Czech Technical University (CTU) in 2011. The capturing process of this dataset is carried out in a controlled network environment which is an advantage of this dataset.

- DARPA

This is a network-based dataset produced in 1998 at MT Lincoln Laboratory. The training data records network-based attacks for seven weeks. In contrast, the test dataset consists of two-week network-based attacks.

Table 2.1: Summary of publically available datasets.

Data Set	Year	Dataset literature	Feature Count	Attack Type (Classes)
NSL-KDD	2009	[109, 110,113,128,135]	43	Normal, U2R (User 2 Root), R2L (Root 2 Local), Probing DoS (Denial-of-Service)
CICIDS 2017	2017	[138,147]	80	Normal, Brute Force FTP, Brute Force SSH, DoS slowloris, DoS Slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed, Web Attack, Infiltration, Botnet, DDoS, SSH-Patator, SQL injection
CIC-DDoS2019	2019	[127,138,145]	80	DDoS-DNS, DDoS-LDAP, DDoS-MSSQL, DDoS, NetBIOS, DDoS-NTP, DDoS-SNMP, DDoS-SSDP, DDoS-SYN, DDoS-TFTP, DDoS-UDP, DDoS-UDP-Lag, DDoS-Web
ISCX-2012	2012	[115, 125, 153]	80	Normal, Attacker
ISOT	2010	[115,173,175,179,180]	27	Non-Malicious, Malicious
KDD Cup 1999	1999	[116,133,169]	43	Normal, Probe, R2L, U2R, DoS
CTU-13	2013	[115,173,175,178,179,180]	14	Botnet Flows, Normal Flows
DARPA 1999	1999	[124,157]	43	Probes, DoS, R2L, U2R

Table 2.2: Data count for each class of dataset.

Datasets	Classes	Data counts
NSL-KDD	Normal	6817
	U2R (User 2 Root)	3086
	R2L (Root 2 Local)	53
	Probe	988
	DoS	11617
CICIDS2017	Normal	1,589,806,681,514
	Brute Force FTP	55,742,361
	Brute Force SSH	41,241,773
	DoS slowloris	40,681,728
	DoS Slowhttptest	38,871,612
	DoS Hulk	161,067 69,057

	DoS GoldenEye	72,333,060
	Heartbleed	74
	Web Attack	462,190
	Infiltration	2,610
	Botnet	1,335,621
	DDoS	8,958,038,445
CIC-DDoS2019	Benign	56,863
	DDoS-DNS	5,071,011
	LDAP	2,179,930
	MSSQL	4,522,492
	NetBIOS	4,093,279
	NTP	1,202,642
	SNMP	5,159,870
	SSDP	2,610,611
	SYN	1,582,289
	TFTP	20,082,580
	UDP	3,134,645
	UDP-Lag	366,461
	Web	439
ISCX-2012	Normal	2517
	Attacker	2515
ISOT-Botnet	Non-Malicious	1,619,520
	Malicious	55,904
ISOT-Ransomware	Non-Malicious	103
	Malicious	669
KDD Cup 1999	Normal	1,033,374
	Probe	45,268
	R2L	15,676
	U2R	297
	DoS	4,114,845
CTU-13	Botnet Flows	432,755
	Normal Flows	369,806
DARPA 1999	Probes	37
	DoS	63
	R2L	53
	U2R	37

2.3.2 Detecting and Mitigating DDoS Attacks

A DDoS attack occurs when excessive traffic is deliberately directed toward a network, overwhelming its resources and rendering the server inaccessible to legitimate users. This results in network instability and reduced reliability. DDoS attacks can be classified into three categories: (i.e., “protocol-exploitation attacks, volumetric attacks, and application-layer attacks”) [123]. In an SDN-based network

architecture, the control plane manages the decision-making process, which can make intelligent decisions. However, it also presents a Single Point of Failure (SPF) vulnerability, where attackers who gain control of the controller can disrupt the entire network infrastructure [124]. SDN utilizes southbound protocols like OpenFlow to execute actions based on entries in the flow table. These entries encompass various fields serving specific purposes, such as timeouts, priorities, action fields, and counters [125]. An attacker can Initialize a DDoS attack on the controller by inserting malicious flow rules into the flow tables. Figure 2.5 provides an overview of the general procedure of a DDoS attack within an SDN environment. To counter such attacks, machine learning techniques are being developed to detect and mitigate them within the SDN controller. A detailed explanation of these techniques is presented in the subsequent section.

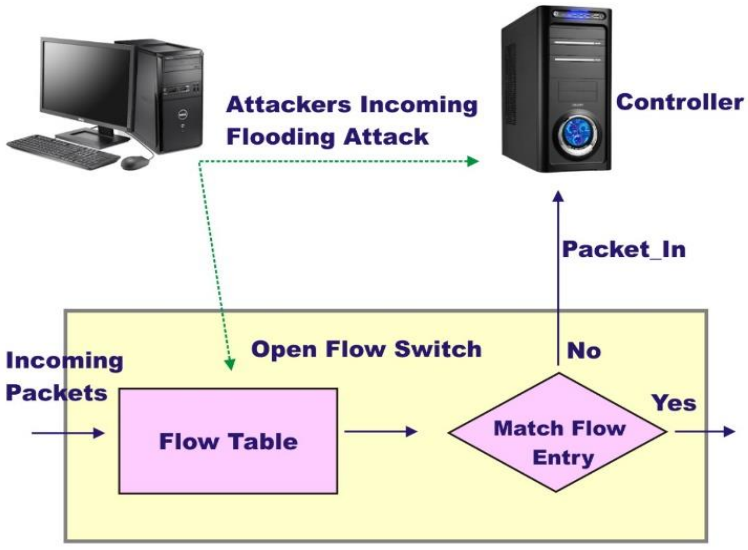


Figure 2.5: The general procedure of a DDoS attack on both control and forwarding planes.

In the context of SDN, many DDoS attack detection and mitigation methods are adaptations of techniques used in traditional networks. Among these, the most popular methods for detecting DDoS attacks are based on statistical information entropy algorithms. These methods offer the advantage of rapidly processing traffic data with minimal computational costs. However, their accuracy relies heavily on choosing an appropriate threshold, which can be limiting and result in one-sided detection.

Kalkan et al. [126] proposed a Joint Scoring System (JESS) based on entropy for detecting and mitigating DDoS attacks, which utilizes the joint entropy tool without increasing the workload of switches. Meanwhile, another study [127] utilized traffic entropy statistical analysis to effectively protect networks from DDoS attacks. They validate their model in the Mininet emulator. Additionally, Kumar et al. [128]developed a technique to identify and counteract SYN flooding attacks in SDN by using destination IP address entropy and specific TCP flags as random variables. To determine the attacker, an adaptive threshold is employed. In addition, machine learning algorithms effectively detect and mitigate anomalies in SDN [129]. These algorithms utilize training data, automatically make decisions, and classify traffic based on flow characteristics. Meanwhile, in [130], the authors proposed a lightweight algorithm for detecting DDoS attacks that utilizes traffic features. This algorithm processes switch data using the NOX controller and performs traffic analysis using the unsupervised learning algorithm Self-Organizing Map (SOM) and competitive learning of ANN. KNN is a straightforward and efficient algorithm that categorizes flows by determining the

arbitrary separation between traffic feature vectors. During the DDoS detection process, to increase the accuracy of anomalous flow detection and decrease the False Alarm Rate (FAR), Peng et al. [131] proposed the DPTCM-KNN method for detecting abnormal traffic. Furthermore, Zang et al. [132] proposed a set of finer-grained flow indices that extract nine single and 39 dual attributes from various dimensions such as category, time, space, and intensity.

This subset of fine-grained traffic features successively improves the detection accuracy for attacks. Xu et al. [133] introduced a modular detection system based on K-Means++ and Fast K-NN (K-FKNN), which enhances the controller's accuracy, efficiency, and stability against DDoS attacks.

The paper [6] proposes a new framework that employs a trigger mechanism to work with detection and defense methods in data and control planes to respond quickly to DDoS attacks and ease the workload on switches and controllers. Niyaz et al. [134] offer a multi-Vector DoS detection system that leverages a home wireless network to gather normal traffic data as a network application deployed on top of the controller. Hurley et al. [135] present a training-based Baum-Welch algorithm-based Hidden Markov Model-based detection system for the SDN environment. They utilize the Mininet emulator and OpenFlow Floodlight controller to carry out their experimental setup. Alshamrani et al. [136] propose a new technique to tackle NetFlow and Misbehavior attacks. Their system periodically gathers network information and applies ML algorithms to categorize network flow as normal or attack traffic. Hu et al. [137] developed an entropy and SVM-based flooding attack detection and mitigation system that collects traffic

information using an SDN controller and sFlow agents. They also add a mitigation agent to stop attack traffic while allowing authorized users to access network resources. Dehkordi et al. [138] offer a hybrid DDoS attack strategy that uses machine learning algorithms and statistical methods for feature extraction and categorization. Li et al. [139] describe a two-stage detection method that collects network traffic with a broad perspective and intelligently identifies network threats. Using the BDM and Bat algorithm with swarm division, they identify common features from network traffic flows and then input this data into Random Forest (RF) for classification. Guozi et al. [140] propose a KNN and φ-entropy-based hybrid system, where the φ-entropy is used for feature selection, and KNN is used for classification. Deepa et al. [141] introduce an ensemble method that detects anomalous network traffic behavior toward the SDN controller using different algorithms such as SVM, KNN, NB, and SOMs. Phan et al. [142] enhance the eHIFF scheme instead of HIPF to defend DDoS attacks in SDN and improve attack traffic's speed and detection rate. Myint et al. [143] suggested an advanced SVM-based technique that can distinguish between ICMP flood and UDP flood attacks to detect DDoS attacks with the least overhead. They validate their approach using the Open Daylight controller in the Mininet emulator.

In the paper [144], the authors proposed an artificial neural network (ANN) based method to detect known and unknown DDoS attack patterns. Cui et al. [145] provided a four-module SD-Anti DDoS approach includes attack detection, attack trigger for detection, attack tracing, and attack mitigation. Xu et al. [146] developed a method for detecting DDoS attacks using SOM classification and victim

detection. Cui et al. [147] suggested a back propagation neural network-based method to extract the temporal behavior of an attack, while Li et al. [148] demonstrated a deep learning-based DDoS detection solution for SDN. This approach involves preprocessing raw data samples before using the cleaned data set to train the deep learning algorithm. The Flow Table Generator module, which establishes the attack priorities and discards attack packets, receives network statistics from the Information Statistics Module (ISM). The authors used real and simulated datasets to train and test their techniques.

Nam et al. [149] proposed an approach to detect anomalous network behavior that integrates neural networks with statistical techniques. This method chooses the most important features from a set of features using an entropy metric and then utilizes a Self-Organizing Map (SOM) to categorize network behavior. The proposed approach is validated using the POX controller on the Mininet emulator.

In another study, Novaes et al. [150] proposed a hybrid system that contains three phases: characterization, attack detection, and mitigation, for detecting and preventing Port scan and DDoS attacks. LSTM was used to learn the characteristics of normal network traffic, and the scientists employed an entropy metric to quantify network features. Finally, they employed fuzzy logic to find network attacks. On the Mininet emulator, their strategy was implemented using a Floodlight controller.

Another approach is based on SVM assisted by Genetic Algorithm (GA) and Kernel Principal Component Analysis (KPCA), is proposed in [42] . The

authors used KPCA to reduce the dimension of feature vectors and then optimized the parameters of SVM through GA. They suggested an improved kernel function (N-RBF) to reduce the noise produced by feature differences. The proposed model was installed into the controller to define different security rules to detect the attack, and a DDoS mitigation module was designed separately inside the controller. The POX controller on the Mininet emulator was used to validate this approach.

Another study [9] introduced an SVM-based method for DDoS attack detection, which uses six-tuple characteristic values extracted from the switch's flow tables for classification. The authors validated this strategy using the Mininet emulator's Floodlight controller. In yet another research [151], To identify DDoS attacks in three categories—bandwidth attack, flow-table attack, and controller attack—the authors examined four different machine learning methods, including Random Forest (RF), SVM, Decision Tree (DT), and Multi-Layer Perceptron (MLP). The authors found that DT produces the best performance compared to the others. To validate their strategy, the Scapy tool produced each attack, a list of more than 20,000 IP addresses used by attackers, and a simulation using the POX controller on the Mininet emulator.

In their study, Meti et al. [152] proposed a method based on Neural Networks (NNs) and SVM for detecting DDoS attacks, which yielded promising results, achieving an accuracy of 80% and a precision of 100%. Likewise, Virupakshar et al. [153] analyzed four different algorithms NB, KNN, DT, and Deep Neural Network (DNN), to detect flooding attacks on an SDN-based OpenStack private cloud, concluding that the DNN algorithm outperformed the

other methods. Another framework for DDoS detection, proposed by [154], comprises three modules a traffic collection module, an attack detection module, and a flow table delivery module which work together to identify and mitigate attacks. SVM utilized multi-dimensional traffic features extracted from statistical flow table information to detect attack traffic. They used the KDD99 dataset to test their approach. [155] analyzed various machine learning algorithms, such as SVM, RF, KNN, and J48, and concluded that the J48 algorithm produced the best detection results. Once the J48 algorithm detected an attack, a REST message was sent to the controller to block the DDoS computer ports of the OF switches for 30 seconds. The authors used the RYU controller on the Mininet emulator to validate their approach. In the study, [156] proposed a DSM-based SVM algorithm for DDoS attack detection and mitigation, which involved pre-processing the input data, feature extraction using the MCA algorithm, and attack prediction with the DSM-SVM. After an attack was discovered, the mitigation server began to block attack traffic and take in the remaining normal traffic. The authors trained their machine learning algorithm using the KDD dataset and deployed it into the RYU controller (i.e., Mininet emulator) for real-time evaluation.

In the study [157], the authors analyzed the performance of various machine learning algorithms, including Linear Support Vector Machine (LSVM), Random Forest (RF), Naive Bayes (NB), Decision Tree (DT), and Deep Neural Network (DNN). The study found that DT provided better results than the other algorithms, and the models were validated using the CSE-CIC-IDS2018 dataset. Another study [158] examined six different machine learning algorithms, namely

KNN, Logistic Regression (LR), NB, RF, DT, and SVM, using extended native flow features for ML algorithm training. RF was the most effective for detecting attacks with a low probability of dropping normal traffic, and the approach was evaluated using the Floodlight controller on the Mininet emulator. Another study [159] suggested an architecture with four modules (i.e., "flow collector, preprocessing, attack detection, and flow manager") for detecting DoS/DDoS attacks on the SDN application and transport layers. The flow collector module generated and collected flows using the CIC Flow Meter application. PCA was employed in the preprocessing module to reduce the dimension of the flow features. The detection module used pre-trained machine learning models, including KNN, SVM, RF, MLP, CNN, Gated Recurrent Units (GRU), and LSTM, to classify the input flows as normal or suspicious. The Flow manager module then sent information regarding the suspicious flows to the controller for further action and created, and visualized flow logs for the classification received from the detection module. The approach was validated using the ONOS controller in the Mininet emulator.

An SVM-based Intrusion Detection System (IDS) for detecting DDoS attacks in SDN was proposed by [160], where traffic information is received by SVM and then classified as normal or attack traffic. [161] presented a model for mitigating different DDoS attacks by introducing adaptive polling, sFlow-based sampling, deep learning, and a Snort Intrusion Detection System (SIDS). In this model, adaptive polling and sFlow-based sampling are individually deployed in the data plane to reduce network overhead. In contrast, the Stacked Auto encoder

(SAE) and SIDS are deployed in the control plane to optimize detection accuracy. Meanwhile, [46] designed a Cognitive-Inspired Computing (CIC)-based DDoS attack detection and defense approach. This method's four components are collecting statistics, feature computing, attack detection, and attack defense and recovery. The attack detection module, which recognizes DDoS attacks, was created using SVM. In contrast, the attack defense and recovery module generates a fresh flow table and tosses any packets addressed to the targeted host address. [162] introduced an Extreme Gradient Boosting (XGBoost) based method that uses the flow packet dataset for detecting DDoS attacks. Finally, to overcome the challenge of detecting adversarial attacks caused by specific perturbations, a system for detecting DDoS attacks was proposed by [163], Using an adversarial training method and a generative adversarial network (GAN) makes the system less vulnerable to malicious attacks by using an adversarial training method and a generative adversarial network (GAN). Utilizing IP flow analysis, the system continuously analyses traffic to activate the detection system and respond in real time. The detection system is activated to take prompt action, and IP flow analysis is employed to continuously monitor traffic. The GAN module detects the attack, and the mitigation module is activated automatically, taking countermeasures to minimize the attack's effect. Their mitigation approach is based on Event-Condition Action (ECA), where the Event is associated with a set of specific rules for the anomaly, the Condition describes the rules where a particular event of anomaly occurs, and the Action comprises countermeasures taken against the anomaly event. Their method was evaluated using the CICDoS 2019 dataset.

The Generalized Entropy (GE) approach, Particle Swarm Optimization (PSO), and (BPNN) were combined in a method that was proposed.[164]. On the switch, the generalized entropy approach was originally used to detect attacks and separate normal from anomalous traffic flows. This causes the switch to sound an anomalous alarm, and the controller's PSO-BPNN, which is installed, collects characteristics from the abnormal traffic flow to predict potential DDoS attacks. The extracted features include average packets per flow, bytes per packet, percentage of pair flow, rate of flow entries, the entropy of source IP address, and average duration per flow. The effectiveness of this approach was validated using the Floodlight controller in the Mininet emulator. In a similar vein, Wang et al. [165] proposed a DDoS detection method that combines Information Entropy (IE) and Convolutional Neural Networks (CNNs). Their approach utilizes the IE to examine suspicious traffic flows, and then the CNN performs fine-grained packet-based detection to classify normal and abnormal traffic. They used the CICIDS2017 dataset to train the CNN, and the method’s performance was evaluated in real-time using the POX controller in the Mininet emulator.

In addition, a Decision Tree-based lightweight framework called DETPro was introduced by [166] for efficiently detecting DDoS attacks. DETPro is a modified version of the DT algorithm that uses the Pessimistic Error Pruning (PEP) strategy and Gini impurity. Traffic information is collected by sFlow agents and a POX controller embedded into OpenvSwitch. A white list mechanism is utilized in the mitigation module to block attack traffic and maintain the network's significant functionalities without delay. Meanwhile, [167] suggested an ensemble learning

and entropy-based DDoS attack detection system. A preliminary detection module on the edge switch uses entropy to continually monitor network condition information and notify the controller of unusual activity.

In addition, the authors used edge computing to transfer the responsibility of attack detection from the controller to the data plane, thereby decreasing the southbound communication overhead. Using cloud-edge collaboration, [168] designed a DDoS detection system based on Entropy-Measuring (EM), SOM, and K-Dimensional tree (EMSOM-KD). The authors selected Ideal SOM maps using EM and identified most traffic flows directly by the EMSOM. In [169], the authors proposed a hybrid solution to identify DDoS attacks, where an Information Entropy (IE)-based module first searches for anomalous traffic, and another detection module based on Stacked Sparse Auto encoder (SSA)-SVM subsequently verifies the suspected abnormal traffic. Their defense module speedily releases a new flow table to resume the network's regular communication after successfully identifying an attack.

In the study [170], a hybrid approach was suggested by the authors, which combined a Genetic Algorithm (GA) and Decision Tree (DT) called GA-DT. Their approach was compared to other machine learning algorithms like Logistic Regression (LR), SOM, Neural Network (NN), SVM, and KNN, and their approach was found to perform better than others. The ML algorithms were trained on the KDD dataset, and the Mininet emulator was utilized for real-time evaluation. Another study [171] used sFlow as a macro-detection to monitor the network, and SOM is used as a micro-detection to identify the attack traffic. An improved source-

based DDoS attack detection and defense approach for SDN was created in this research using SOM.

The ISCX-IDS2012 dataset was used for training, and real-time evaluation was carried out using the Floodlight controller in the Mininet emulator. A KNN and SVM-based detection method was proposed in another study [172] to reduce resource consumption during the DDoS attack detection process, which was validated using the Mininet emulator. In a different study, a detection technique based on Spatial-Temporal Graph Convolutional Networks (ST-GCN) that apply IN-band Network Telemetry (INT) to sense switch status with samples was developed [173]. An ensemble model based on optimized weighted voting for DDoS attack detection in SDN was introduced in another study [174], which used different hyper-parameter values of six base classifiers to build the ensemble model. A K-Means clustering-based method was proposed to deal with the unbalanced distribution of traffic data for DDoS attack detection [175]. A Random Forest (RF) based DDoS detection and mitigation system was proposed in another study [176] , which uses flow entries to classify them as normal or attack traffic. [177] introduced a system that consists of two modules named the trigger module and the detection module. The trigger module is based on Gini impurity, which analyses the source and destination IP data. The classification of the traffic flow as normal or attacked is done by the detection module, which is RF-based and based on discovered anomalies. Finally, a time series and RF-based detection method were suggested in another study [178], where the ARIMA model was used to predict the information of the current flow based on the Historical Information

Entropy (HIE), and the detailed traffic features were further extracted to detect the attack. The ML and DL-based solutions previously mentioned are compared in Table 2.3, along with their benefits and drawbacks.

Table 2.3: ML and DL-based solutions for DDoS detection and mitigation in SDN.

Study	Contributions	Controller Type	Layer	Protocols	Classifier Type	Environment Type	Data Set	Evaluation	Discussion
[130]	Proposed a simple DDoS detection method with very little overhead performance and classified traffic using flow-based data.	NOX	Control Plane	TCP, UDP and ICMP	Self-Organizing Map (SOM)	Physical	Custom developed	Detection Rate = 99.11%	The limitation of this work is that it cannot identify the ports of the OF switches from where the attack is launched.
[131]	To defend the controller's control plane from DDoS attacks, they created the Safe-Guard Scheme (SGS). The main objective of the SGS is to use a controller clustering technique to deploy many controllers in the control plane.	Ryu	Control plane and Data Plane	TCP, UDP and ICMP	Back Propagation Neural Network (BPNN)	Virtual	Custom developed	Not mentioned	The detection accuracy is not mentioned in the paper.

[133]	They deploy a modular detection system in the controller to improve the accuracy and stability against DDoS attacks.	Ryu	Application Plane	Not mentioned	K-means++ and K-FKNN	Virtual	NSL-KDD	Precision = 97.5%	This work may be extended to a distributed SDN environment to detect other flooding attacks.
[6]	The authors proposed a new trigger mechanism-based framework which cooperates the DDoS attack detection and defence methods in data and control planes.	ONOS	Control and Data Planes	Not mentioned	KNN and K-Means	Virtual	NSL-KDD	Accuracy = 98.85%	This work is not dealing with when the controller is under large-scale network traffic.
[134]	The authors introduced a multi-Vector DDOS detection system and installed it on top of the controller as a network application.	POX	Control and Data Planes	TCP, UDP and ICMP	Stacked Auto encoder (SAE)	Physical	Custom developed	Accuracy = 99.82%	To extend this study, deep learning algorithms can extract features from raw flows instead of feature reduction and are also used to detect other types of

									attacks in SDN.
[135]	It introduced and integrated a NIDS that can track and detect traffic attacks and protect the network based on information from the entire network. .	Floodlight	Data Plane	Not mentioned	Hidden Markov Models (HMMs)	Virtual	Custom developed	Accuracy = 96%	The use of feature vectors along with HMM improves the detection power of the proposed approach.
[136]	The authors developed a new method to solve two problems, Misbehavior and New-Flow attack in SDN.	POX	Control Plane	Not mentioned	Sequential Minimal Optimization (SMO)	Virtual	NSL-KDD	Accuracy = 99.40 %	This work is not efficient in detecting unknown attacks.
[138]	The authors proposed a hybrid method based on machine learning and statistical techniques.	Floodlight	Control Plane	Not mentioned	Bayes Net, J48, logistic regression, Random Tree and REP Tree	Virtual	ISOT, UNB-ISCX and CTU-13	Accuracy of REP Tree = 99.88%	They detect the DDoS attack on One controller.

[139]	Proposed a two-stage technique that records network flows from a global point of view and detects network attacks.	Not Mentioned	Application Plane	Not mentioned	Random Forest (RF)	Virtual	KDD Cup 1999	Accuracy = 96.03%	This approach is implemented in a virtual network environment. This work can be extended by implementing this approach in a real-time network environment.
[141]	Introduced an ensemble method that detects the anomalous behavior of the network traffic toward the SDN controller.	POX	Control Plane	TCP, UDP and ICMP	SVM, KNN, Naïve Bayes (NB), and Self-Organizing Maps (SOMs)	Virtual	CAIDA 2016	Accuracy of SVM-SOM = 98.12%	This study used an old dataset version and may be extended by using new datasets.
[147]	Proposed a method to extract the temporal behavior of an attack in SDN.	Not Mentioned	Control Plane	Not mentioned	Back Propagation Neural Network(BPNN)	Physical	DARPA 1999	Not mentioned	The achieved accuracy of the method is not mentioned. This method is effective in performing a

									port recovery after an attack.
[148]	Suggested a method that can learn the traffic patterns from the sequences of flows and then historically trace the attack activities.	Not Mentioned	Control Plane	TCP, UDP, HTTP	Bidirectional Recurrent Neural Network (BRNN)	Virtual	ISCX2012	Accuracy = 99%	The significant advantage of this study is that it can help to reduce the degree of dependence on the software and hardware environments. It simplifies the updating of detection systems in real-time.
[149]	The authors proposed a system that combined the neural network with entropy metric to identify the abnormal network behaviour.	POX	Control Plane	Not mentioned	SOM	Virtual	CAIDA2015	Detection Rate = 97.28%	In this study, the authors used manual methods for the selection of features and these methods can be replaced with

									automatic feature selection methods.
[150]	Introduced a hybrid approach for Port scan and DDoS attack detection based on Long Short Term Memory (LSTM).	Floodlight	Control Plane	Not mentioned	Long Short Term Memory (LSTM) and Fuzzy Logic	Virtual	CIC DDoS 2019	Area Under Curve (AUC) = 99.62%	This study is an excellent effort toward securing the SDN from DDoS attacks and their mitigation approach appears
[42]	Proposed a method that reduces and selects the relevant features and then performs DDoS detection using machine learning classifiers.	POX	Control Plane	TCP and UDP	SVM, Genetic Algorithm (GA) and Kernel Principal Component Analysis (KPCA)	Virtual	NSL-KDD	Accuracy = 98.90%	This approach is effective in a single controller environment and may fail in a multi-controller environment.
[9]	Proposed a technique that does classification and gets the 6-tuple characteristic values	Floodlight	Control Plane	TCP, UDP and ICMP	SVM	Virtual	Custom developed	Average Detection Rate = 95.24%	This approach has low detection accuracy for the ICMP attack flows.

	from the switch's flow table.								
[156]	Suggested a DSM based SVM algorithm for DDoS attack detection and mitigation.	RYU	Control Plane	Not mentioned	SVM	Virtual	NSL-KDD	Accuracy = 99.7%	This method achieved the best detection accuracy, but in the mitigation phase, there is a chance to drop the normal packets along with attack packets.
[159]	Proposed an architecture to detect DoS/DDoS attacks on application and transport layers of the SDN.	Open Network Operating System (ONOS)	Application Plane	TCP and UDP	KNN, SVM, RF, MLP, CN), GRU, LSTM	Virtual	CICDoS2017 and CICDDoS2019	Accuracy = 99%	In this study, the authors observed that the DL algorithms produced the best results compared to ML algorithms. Their mitigation strategy is simple and

									works for the edge switches. So, there is a need to optimize the mitigation strategy to save the computation resources of the controller.
[161]	They have implemented a deep learning model that utilizes adaptive polling and sFlow sampling to mitigate various DDoS attacks.	RYU	Control Plane	TCP, UDP and ICMP	Stacked Auto encoder (SAE)	Virtual	Custom developed	Accuracy for sFlow = 91% and Accuracy for Adaptive Pooling = 89%	Implementing this framework with real-time traffic streams can help to lower the SDN controller's crucial overhead. Further, training deep learning models with signature-based and rule-based network data could help to

									improve the detection accuracy of DDoS attacks as a whole.
[46]	The authors suggest a Cognitive-Inspired Computing (CIC) mechanism that integrates dual address entropy for detecting and protecting against DDoS attacks.	Floodlight	Control Plane	UDP	SVM	Virtual	Custom developed	Detection Rate = 97.65%	In this study, the authors try to restore the communication function to a single victim host, which could be extended to multiple victim machines.
[163]	The authors suggested a system for identifying adversarial attacks that employ Generative Adversarial Networks (GANs).	Floodlight	Application Plane	UDP	Generative Adversarial Network (GAN)	Virtual	CICDoS 2019	Detection Rate = 99.87%	The GAN framework has the potential to detect adversarial attacks in SDN due to its adversarial training nature, and it makes the controller less sensitive to DDoS-based

									adversarial attacks. But the proposed framework is implemented in a simple test scenario.
[165]	A DDoS detection technique based on information entropy (IE) and convolutional neural networks (CNN) was presented.	POX	Control Plane	HTTP, TCP, UDP, ICMP	CNN	Virtual	CICIDS2017	Accuracy = 98.98%	In the proposed method, two-level detection is performed, enhancing the controller's workload and taking a long time to identify the attack.
[167]	The authors proposed an entropy and ensemble learning-based cooperative DDoS attack detection scheme.	Ryu	Data Plane	ICMP	Fine-Grained and RF	Virtual	Custom developed	Not mentioned	The authors do not mention the accuracy of the proposed approach.

[169]	The Stacked Sparse Auto encoder (SSA)-SVM and Information Entropy (IE) based hybrid approach for dual identification of attack.	Floodlight	Control Plane	TCP, UDP and ICMP	Stacked Sparse Auto encoder (SSA)-SVM	Virtual	DARPA and two custom developed	Detection Rate = 96.86% and Accuracy = 98.63%	The proposed approach performs dual detection for the attack, the Initial detection through information entropy and then by SSAE-SVM. They observed that the time cost and CPU utilization for the combination of both modules are less than the SSAE-SVM alone, indicating that the combined approach is more effective.

[171]	Source-based DDoS detection using improved SOM and sFlow.	Floodlight	Control Plane	TCP	SOM	Virtual	ISCX-IDS2012	Detection Rate = 95.41%	This study accurately identifies the attacker host and focuses on source-based DDoS detection in SDN. Although it imposes tight limits on more falsely alerted normal flows that damage regular users, the suggested method loses effectiveness when the normal flows are substantially similar to the attack flows.

[173]	To map the network into a graph for attack detection, the authors presented a method based on Spatial-Temporal Graph Convolutional Networks (ST-GCN).	ONOS	Data Plane	TCP, UDP, HTTP	Spatial-Temporal Graph Convolutional Network (ST-GCN	Physical	CAIDA	Maximum Accuracy = 91.11%	The proposed method detects the DDoS attack in the data plane by extracting the data characteristics from both spatial and temporal perspectives. The main focus of this study is to identify the switches containing the attack flows extending toward the controller end.

2.3.3 Detecting and Mitigating Low-Rate DDoS Attacks

LR-DDoS is a form of attack that is distinct from the well-known HR-DDoS. Although identifying HR-DDoS is common, detecting LR-DDoS can be equally complex. The primary aim of these attacks is to exhaust computing resources. Unlike HR-DDoS, LR-DDoS does not involve a network flood with high traffic volumes. Instead, it strategically manipulates specific protocols, such as Congestion Control mechanisms [179] , TCP's timeout retransmission [180], [181] and HTTP's keep-alive mechanism [182], to deplete the target's computing resources.

Wu et al. [183] proposed a detection mechanism for low-rate DDoS attacks based on Factorization Machine (FM) [184] using features derived from SDN data layer flow rules. The technique extracts the packet number, duration time, relative packet interval dispersion, and relative match byte dispersion. The FM algorithm uses these features to categorize flow as normal or attack flow, integrating the features to create a correlation between all feature samples. This correlation is then used to update other parameters, improving real-time detection rate against LR-DDoS and providing reliable conditions to resist attacks. Dynamic flow rule deletion is adopted for defense against DDoS attacks. Another adaptable architecture is suggested that consists of the Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) modules to detect and prevent LR-DDoS attacks [11]. Six machine learning algorithms are used to train the IDS, which is embedded into the controller to perform various tasks, such as identifying and classifying APIs and flows. The IPS detects HTTP flows and generates new flow rules for malicious

flow mitigation, blocking potential attackers. The CIC DoS 2017 dataset trains the machine learning models [185]. On the Mininet emulator in an SDN environment, the Open Network Operation System (ONOS) controller is used to validate the methodology.

Zhang et al. [186] introduced an LR-DDoS detection model that utilizes SVM and Power Spectral Density (PSD), claiming that PSD-entropy enhances the system's efficiency and detection power while minimizing computation costs. The SVM component learns traffic patterns and selects the most appropriate features for the detection algorithm. The algorithm determines two classification thresholds by averaging attack traffic and regular traffic. The communication is deemed an attack if the calculated PSD entropy is below the threshold. To address the inability of traffic volume analysis to detect LR-DoS attacks, Liu et al. [187] proposed a Deep Convolutional Neural Network (DCNN)-based system that automatically extracts available features from traffic flows. The extracted features are then fed into a Q-Network, a reinforcement learning algorithm, which detects edge LR-DoS attacks. This system produced satisfactory results in a simulated environment for protecting SDN from LR-DoS attacks.

[188] proposed a flow-based LR-DDoS attack detection and mitigation framework that employs ML models such as DT, SVM, and NB. In the mitigation phase, the controller acquires information on attack flows and uses mitigation rules to prevent LR-DDoS attacks from the same source. Compared to DT and NB, SVM exhibited the highest detection accuracy. The above-discussed solutions are summarized in Table 2.4.

Table 2.4: ML and DL-based solutions for Low-Rate DDoS detection and mitigation in SDN.

Study	Contributions	Controller Type	Layer	Protocols	Classifier Type	Environment Type	Data Set	Evaluation	Discussion
[183]	Introduced a Factorization Machine (FM)-based multi-feature low-rate DDoS attack detection technique.	Ryu	Data Plane	TCP, UDP and ICMP	Factorization Machine (FM)	Virtual	CAIDA	Detection Accuracy = 95.80%	The suggested method can be expanded to a multi-controller environment and is acceptable for single-controller network architecture. Implementing the recommended strategy in a real network environment can demonstrate its usefulness further.

[11]	The authors proposed a flexible architecture consisting of two modules: the Intrusion Detection System (IDS) and the Intrusion Prevention System (IPS), for identifying and mitigating LR-DDoS attacks.	ONOS	Control Plane	TCP and UDP	J48, Random Tree (RT), REP Tree, RF, MLP, and SVM	Virtual	CIC DoS Dataset (2017)	Maximum Accuracy Achieved by MLP = 95.01%	According to this study, when the false positive rate rises, it may be possible to restrict real users. This method is tested in a simple network setting. Therefore, this method is compatible with a complicated network design.
[186]	Introduced a Power Spectral Density (PSD) based method to improve detection accuracy with minimum computation cost.	Not mentioned	Not mentioned	TCP	SVM	Physical	KDD99	Detection Rate = 99.19%	They did not mention the controller type and layer from which they will detect the attack.

[187]	Proposed a Deep Convolutional Neural Network (DCNN) based method which automatically extracts the available features from traffic flows to identify the attack.	Ryu	Control Plane	TCP, UDP and ICMP	Deep Convolutional Neural Network (DCNN) and Q-Network	Virtual	Custom Developed	Detection Rate = 97.80%

2.3.4 Detecting and Mitigating Botnet Attacks

Researchers have recently utilized machine learning techniques to develop precise and scalable frameworks for detecting and preventing botnet attacks in SDN. They have employed centralized learning with distributed detection to achieve scalability in detection. In the past few years, the use of machine learning techniques for identifying botnets has grown significantly. This section discusses the latest developments in machine learning for this type of attack.

A study [189] investigated different types of botnets (P2P, IRC, and HTTP botnets) in SDN controllers. The study found that Decision Tree (DT) effectively detects Peer-to-Peer botnets, while Naïve Bayes and SVM detect IRC and HTTP botnets more successfully. Another study [190] used centralized network flow statistics collected by OpenFlow counters for detection, applying decision trees and C4.5 to the collected counters. The proposed method achieved an 80% detection rate for botnets, using a publicly available real-world botnet dataset for the experimental analysis. In another research paper [191], the authors analyzed potentially vulnerable hosts and malicious codes using four different classifiers: NB, DT, Bayesian Networks (BNs), and C4.5. They used historical data for prediction and deployed security rules in the SDN controller to protect potentially compromised hosts and block the entire subnet to restrict the attackers' access. Bayesian Networks achieved a higher precision rate compared to the other classifiers.

Several research studies have used machine learning (ML) algorithms to suggest various techniques for detecting and overcoming botnet attacks in software-

defined networking (SDN). One way suggested by [192] is using a flow-based approach instead of packet payload inspection to detect botnets in SDN. Their system combines real-time flows with historical context to extract an enriched feature set for classification, which achieved 90% detection accuracy for unknown botnets and 97% for known botnets. Another framework introduced by [193] integrates an ML algorithm into the SDN controller to detect and categorize peer-to-peer (P2P) network traffic in real time. They achieved high accuracy in detecting different P2P network traffic using a Strom and Zeus botnet dataset for attack traffic and Skype, eMule, and uTorrent network data for normal traffic. The study proposed by [194] suggested an ML-based framework that uses traffic flow classes to reduce detection complexity and determine high-level policies for the derived flow classes. The K-mean algorithm was used for unsupervised learning to classify NetFlow features, and the DT was used for supervised learning to classify traffic as normal or attack. In the study, [195] proposed a framework that integrates ML with SDN/NFV to detect and mitigate botnet attacks. They suggested a network function which use network protocols to detect known attacks, and collect real-time network traffic as a data set for detecting additional distributed attacks.

To detect botnets, a study [196] proposed a method using Multi-Layer Perceptron (MLP) that analyzes malware traffic data collected from an existing network. This technique creates network isolation and adds a connection block to the external network to avoid internal infection. The approach, which had a 99.2% accuracy rate, was tested using the CTU-13 and ISOT data sets. Another approach [197], involved building a system that uses SDN's northbound and southbound API

to detect botnets At predetermined intervals during the time window, the switch notifies the controller with an OpenFlow message that includes statistical data. The deep learning classifier, which has five hidden layers and is based on ReLU, receives instructions from the controller to block communication and isolate the infected host. The detection accuracy of this technique was 99%. The discussed solutions for botnet attack are summarized in Table 2.5.

Table 2.5: ML and DL-based solutions for botnet attacks detection and mitigation in SDN.

Study	Contributions	Controller Type	Layer	Type of Botnet	Classifier Type	Environment Type	Data Set	Evaluation	Discussion
[189]	They used IPFIX generic template for the detection of botnets.	POX	Control Plane	IRC, HTTP, P2P	Bayesian Network, Neural Network, SVM, DT	Physical	Custom developed	Not Mentioned	The performance results are not discussed.
[190]	Propose a method on centralized network flow statistics, which are collected by OpenFlow counters	Open Daylight	Application Plane	IRC, HTTP, P2P	Decision Tree and C4.5	Virtual	CTU-13, ISOT	Accuracy = 80%	They analyzed that OpenFlow counters have the potential to identify botnet behavioral patterns and are a suitable candidate for flow-based botnet detection techniques.
[191]	They used historical data to predict the potentially vulnerable hosts and malicious codes and deploy the security rules	Not Mentioned	Control Plane	Not Mentioned	NB, DT, Bayesian Networks (BNs), and C4.5	Not Mentioned	LongTail	Accuracy = 91.68%	The BN achieved a high precision rate compared to the other classifiers.

	in the SDN controller.								
[192]	A flow-based approach is used instead of reading packet payload for the detection of the botnet in SDN	Open Daylight	Application Plane	IRC, HTTP	C4.5	Virtual	ISOT and CTU-13	Accuracy for unknown botnets = 90% and Accuracy for known botnets =97%	This method needs extensive computations.
[193]	An ML framework is integrated with the controller, which detects and categorizes the P2P botnet attack.	Ryu	Control Plane	Not Mentioned	SVM, KNN and RF	Virtual	Custom developed	Accuracy =99.7%	SVM produces good detection results as compared to other classifiers.

[194]	An ML-based framework uses traffic flow classes to reduce the detection complexity and determines the SDN high-level policies for the derived flow classes.	Not Mentioned	Control Plane	HTTP	K-mean and SVM	Virtual	Custom developed		This method did not be implemented for the new types of network traffic.
[195]	They integrated the ML method with SDN/NFV for the detection and mitigation of botnet attack.	Floodlight	Control Plane	IRC, HTTP, P2P	RF	Virtual	CTU-13	Accuracy = 100%	This work is limited to a few protocols that need to add more protocols for botnet detection in SDN.
[196]	They use malware traffic data collected over the existing network to	Ryu	Application Plane	IRC, HTTP, P2P	Multi-Layer Perceptron (MLP)	Physical	CTU-13 and ISOT	Accuracy = 99.2%	The study just focused on the controller and did not experiment with the terminals that were

	detect the botnet.								infected by the botnets.
[197]	The authors proposed a system that detects the botnets using northbound and southbound APIs of SDN.	Not Mentioned	Application Plane	Not Mentioned	Artificial Neural Network	Physical	CTU-13 and ISOT	Not-Mentioned	Their method is suitable to block the attack traffic at the source. Further, this method cannot be used to perform real-time detection due to the constrained time window duration.

2.3.5 Detecting and Mitigating Saturation Attacks

A saturation attack is a form of adversarial attack that can impact the entire SDN network due to its prolonged duration. When the control plane of the controller is overloaded, the SDN may become unavailable. The attack involves a malicious host generating a large volume of table-miss packets, which can deplete the resources of the control plane. In SDN-based networks, a saturation attack operates by manipulating the OpenFlow switch, which receives network packets.

- If a new incoming packet does not match the local flow rules, a miss-table error occurs.
- If the switch's buffer is partially full, a Packet-In message that contains the header of the table-miss packet will be produced.
- The table-miss packet is encapsulated in the Packet-In message and forwarded to the controller if the switch's buffer is full.
- The controller receives the Packet-In message, and processes the table-miss packet.
- Additionally, the controller sends Packet-Mod and Packet-Out messages to the switch's flow table to add new flow rules.
- This reactive packet processing mechanism exposes the OpenFlow network to the attacker. The attacker gets an opportunity to consume the different computation resources such as CPU, memory, etc. of the switches and controller and saturate the channels of OpenFlow connections which are

responsible for delivering the forwarding messages between OpenFlow switches and controller.

- Once the attacker gets access to the controller, it can launch different saturation attacks such as TCP-SYN, ICMP, UDP, TCP-SARFU flooding, and IP Spoofing and their combinations (i.e., hybrid saturation attacks) at data-to-control planes. The attacker overtakes the SDN network's numerous hosts (also known as "zombie machines") and transmits forged packets, making it hard for the controller to match the new packets with the switch's flow rules. As a result, the controller begins to receive numerous Packet-In notifications. As a result, the attack from the data-to-control plane depletes the controller's computing capabilities.
- The controller sends Packet-Out and Packet-Mod messages in response to a data-to-control plane flooding attack, which results in flooding attacks from the control-to-data plane. Therefore, the flow tables of the targeted switches are filled with fake flow rules.
- So, the whole switch buffer is consumed and becomes unavailable for the legitimate new packets.

Finally, the OpenFlow channel's bandwidth is used up, which prevents OpenFlow messages from being delivered between the switches and the controller. The above-discussed scenario is shown in Figure 2.6.

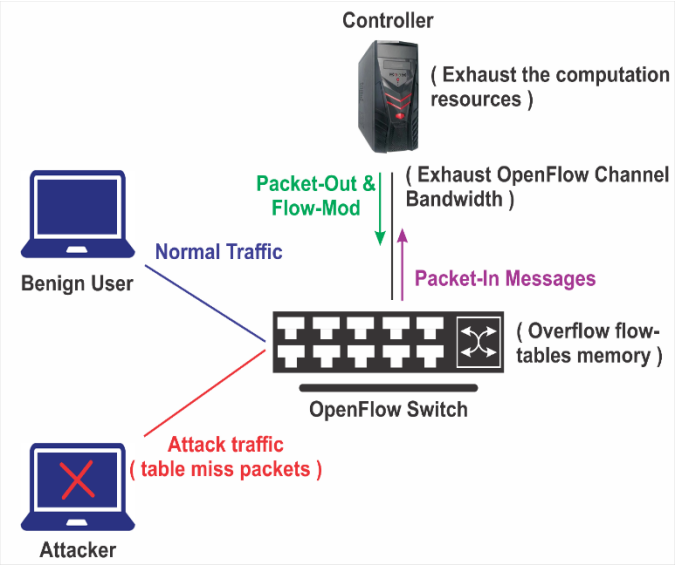


Figure 2.6. General procedure for a saturation attack in SDN.

A study [198] examined how machine learning-based systems detect saturation attacks within an SDN environment. The study found that adversaries could bypass machine learning classifiers by creating adversarial attacks that evade detection. The authors proposed an adversarial testing tool that generated four types of saturation attacks by manipulating different traffic features to address this. They also suggested using various machine learning classifiers to improve detection, but their tests showed that the saturation attacks reduced the detection power of the system.

Another study [199] suggested a time window-based machine-learning method for identifying saturation attacks in SDN. The authors found that if the window size were too large, the response time of the detection method would be too slow, giving the attacker time to saturate the network. Conversely, if the window size were too small, it would cause frequent false alarms and high-

performance overheads for the controller. They investigated the impact of time windows on three different classifiers using OpenFlow traffic data.

Abusnaina et al. [200] introduced a method called FlowMerge that used a Convolutional Neural Network to detect different types of saturation attacks, but the approach had some limitations. The authors generated the attack samples based on the machine learning classifier. It was unclear whether the attacks could evade other machine learning classifiers installed in the SDN controller. Mossavi-Dezfooli et al. [201] presented a Deep Neural Network-based Deep-fool algorithm to generate and detect saturation attacks in SDN. They used image inputs to perform classification. Papernot et al. [202] proposed a library named Clearhans v0.1 to create attacks in SDN and to help improve the robustness of machine learning classifiers.

Finally, a framework called Fast Recovery Saturation Attack Detection and Mitigation (FSDM) was proposed [203]. The FSDM framework employed various strategies to stop the attack flows after using Control Channel Occupation Rate (CCOR) distribution to identify the ports from which the attacker originates. Additionally, the framework contained a brand-new function module called Force Checking, which enables the SDN controller to recover and clean up any leftover attack quickly flows. The discussed solutions for saturation attacks are summarized in Table 2.6.

Table 2.6: ML-based solutions for saturation attacks detection in SDN.

Study	Contributions	Controller Type	Layer	Type of Saturation Attack	Classifier Type	Environment Type	Data Set	Evaluation	Discussion
[198]	A machine learning-based solution was suggested by the author to identify saturation	Floodlight	Application Plane	TCP-SYN, UDP, ICMP, and TCP-SARFU	K-NN, NB, SVM, ANN and Isolation-Forest	Physical	Custom developed	The maximum accuracy achieved by KNN = 96%	This study focus on four types of saturation attacks, so more types of attacks can be added.
[199]	The time window concept has been used to analyze and detect saturation attacks in SDN.	Floodlight	Application Plane	TCP, UDP, ICMP, SCTP (Stream Control Transmission Protocol)	NB, KNN, SVM	Use both Physical and Virtual	Custom developed	KNN achieved the best results Precision = 97% Recall = 99% F-1 score = 98%.	This study is suitable for a single-controller SDN environment and maybe extend to multiple controllers. It was the first attempt to detect unknown saturation attacks.

[200]	The author proposed a method named Flow Merge to detect different saturation attacks.	POX	Application Plane	TCP, ICMP, and UDP	Convolutional Neural Network (CNN)	Physical	Custom developed	Accuracy = 99.83%	The authors created Flow-Merge after demonstrating how generic adversarial examples (AEs) techniques result in unrealistic flows. Weight merging techniques are used rather than ratio-based features to develop the adversarial inputs. The adversarial methods provide a strong defense against general attacks, but it is necessary to investigate the defenses against Flow-Merge.

2.3.6 Detecting and Mitigating Ransomware Attacks

Ransomware is malware that encrypts and locks a user's files and demands a ransom to release them. To spread the attack, the perpetrator seeks control of the SDN controller and employs HTTPS to deliver the malware, making extracting and identifying features through deep packet inspection difficult. However, detection and mitigation techniques are available to safeguard the SDN controller from ransomware attacks. One such technique is machine learning, and this section focuses on its development for ransomware detection and prevention in SDN.

According to a study [204], the authors propose a K-Nearest Neighbor (KNN)-based prediction system that identifies ransomware traffic packets and integrates a dynamic isolation method in SDN. The system achieves 97.7% prediction precision for ransomware. In another study [205], a two-phase approach is introduced consisting of stream processing and classification. In the stream processing phase, the system reads a flow, manages a custom flow table, and extracts flow features. In the classification phase, the Random Forest (RF) classifier trains on the extracted features to distinguish normal traffic from ransomware traffic. In yet another proposal [206], a federated learning-based anti-ransomware learning mechanism is suggested for detecting and mitigating four types of ransomware attacks: (i.e., “Petya, PowerGhost, BadRabbit, and WannaCry”). During the defense phase, the trained federated learning classifier is installed in the SDN controller, which detects ransomware attacks and blocks traffic from the victim device.

2.4 Summary

In this chapter, first, we describe the architecture and workflow of the SDN in detail. Second, a comprehensive overview with a taxonomy of the machine learning and deep learning algorithms is given. Finally, a related work on ML and DL development for securing SDN is explained in detail. The section related to security in SDN covers the following: a) a description related to the available benchmark dataset, b) machine learning and deep learning in detecting and mitigating attacks such as DDoS, low-rate DDoS, botnet, saturation and ransomware.

CHAPTER 3

SYSTEM MODEL

3.1 Overview of the Proposed Method

To successfully protect the SDN controllers from botnet-based DDoS attacks, it is important to detect and block the attack traffic flows. A flow consists of large number of packets with same information's (e.g., source IP address, destination IP address, source port number, destination port number, protocols, etc.). During an attack, the source IP addresses are belonging to the attackers. For example, we have "y" classes with "N" number of flow samples. Let the flow samples are represented as $\boldsymbol{X = \{F_1, F_2, F_3, \ldots\ldots\ldots, F_N\}\ \varepsilon\ \mathbb{R}^{d \times N}}$, where $\boldsymbol{F_i}$ is the ith flow, $\boldsymbol{d}$ represents the number of original features of the flow, and $\boldsymbol{N}$ is total number of flows. The actual tables for a $\boldsymbol{F_i}$ flow can be defined as $\boldsymbol{y_i = \{0, 1\}}$. The aim of this research is to develop an end-to-end method which can predict a label as an actual label ($y_{pred(i)} = y_i$). This chapter discusses the system model which is proposed in this research for the flow classification. A comparative analysis on deep learning methods with features selection is performed. The DL methods deployed in the SDN controllers would help to classify the attack and normal flows. A systematic diagram which shows the operational phases of the proposed research is described in Figure 3.1. First, a pretty table is created to place the incoming packets, Second, the flows features are computed and extracted from each traffic flow. Third, features weight and threshold tuning method is used to select the

optimal features, Then, the selected optimal features are combined and then converted into five different subsets, finally, these subsets are passed as input to the DL methods (e.g., MLP, DNN, CNN, RNN, and LSTM) for the training and validation. Lastly, these methods are deployed in the controller for real time flow classification.

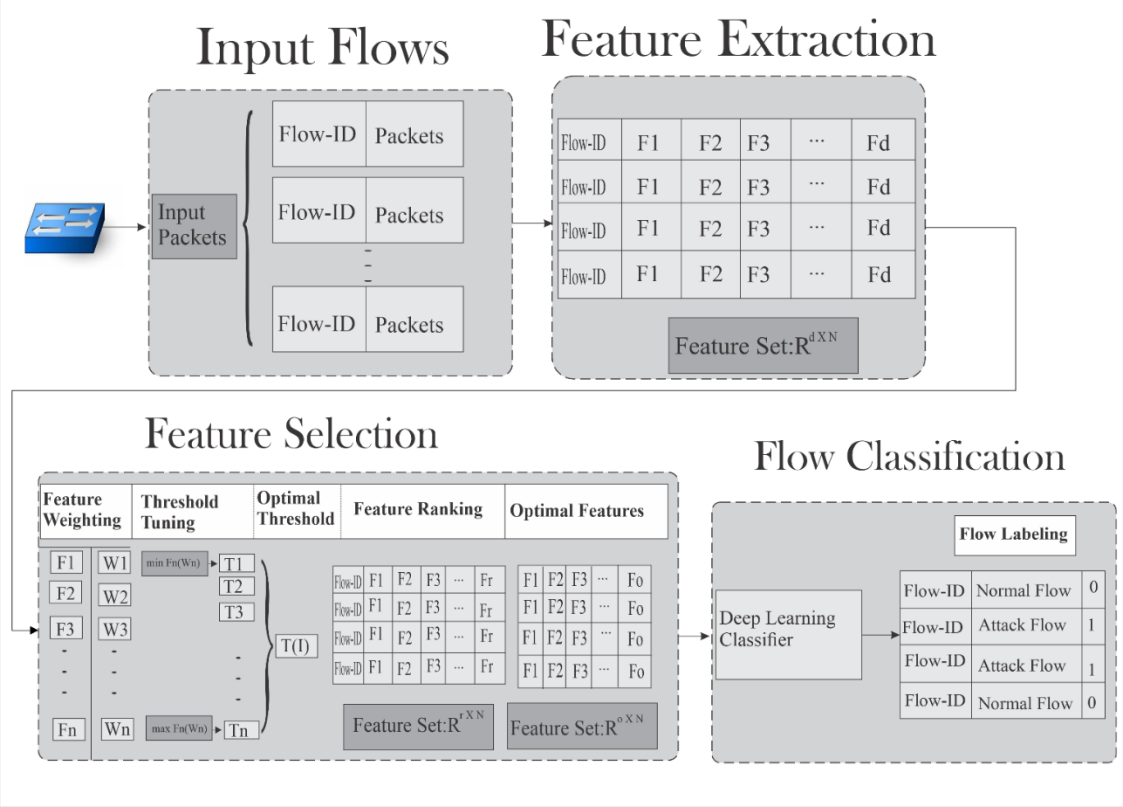


Figure 3.1: Systematic diagram of the proposed research.

3.2 Simulated Dataset

The quality of the training datasets is highly important for the better performance of the IDS methods and techniques. However, one of the main issues is the availability of benchmark datasets for the detection of intrusion in various fields of networking. This problem interrupts the development of efficient intrusion

detection systems. Some other fields such as biomedical engineering, language translation, etc., have a massive amount of benchmark datasets to evaluate the performance of ML/DL methods. However, security and privacy are the main reasons behind the lack of benchmark datasets for the detection of network intrusion. These datasets may have sensitive information, so, the customer information may be revealed to the public by the availability of these datasets publically. Besides, most of the available datasets have different issues such as being laboriously anonymized, don't contains modern attacks found in the current networks, and being outdated. To the best of our knowledge, there are currently no publicly accessible benchmark datasets for the training, testing, and assessment of deep learning-based IDS in an SDN environment besides the issues above. These datasets also don't contain botnet-based DDoS attack records. In several recent studies, typical datasets generated by conventional networks have been employed extensively in SDNs to detect intrusion. However, in SDN networks, the characteristics of the network traffic are quite different from those of traditional network environments. SDN architecture is more vulnerable to threats that do not exist in conventional networks because of its centralized nature. For instance, the decoupling of network devices from the SDN controllers increases the chances for the attackers to launch different types of malicious attacks or activities on SDN controllers themselves or data communication systems. So, attacks on the SDN controllers become hard to detect because the attacker uses an authorized way to connect with the victim server.

To tackle the above-mentioned problems, in this research, a custom dataset is developed in a pure SDN-supported environment to evaluate the performance of the different DL methods (e.g., MLP, DNN, CNN, RNN, LSTM). This dataset contains the botnet-based DDoS attack and normal network traffic. We developed this dataset in two different formats (i.e., "Pcap" and "CSV" file formats), where CIC Flow Meter is used to extract more than 83 statistical flow features. To capture both attack and normal network traffic and mimic real-world attack scenarios, a custom network topology consisting of different hosts, OpenFlow switches, and an SDN controller is used.

3.3 Virtual Experimental Setup and Data Collection

A Mininet virtual environment with a POX controller [207] is used to conduct all the experiments in this research. The Mininet V 2.3.2 ("version 2.3.2) which supports Open Virtual Switches (OVS) [208] is used in testing environments. In recent years, Mininet is widely preferred by the network and research community to perform SDN-based network emulations. The OVS is an open-source virtual machine that supports various commonly used OpenFlow protocols. Because the POX is an interface-rich SDN controller and permits the development of network applications in Python, the network and research communities promote it.

All the experiments are performed on an Intel Core i7 with Windows 10 operating system and 8GB RAM. The deep learning methods are coded using Python language with the Keras framework. To adopt and maintain a pure SDN

environment in the research, we developed a customized centralized network topology in Mininet.

The used network topology is developed in a complex network three structure manner to adopt real-network structure. Figure 3.2 shows the experimental virtual network topology. Figure 3.2 depicts the three planes that make up the SDN architecture: the application, control, and data planes. The application plane has four different planes Flow Statistics Collector (FSC), Feature Extractor (FE), DL classifier, and Mitigator. The Flow Statistics Collector module collects the flow information in Δ time and then stores these flows in a pretty table for further analysis. Feature Extractor module extracts the flow features. The DL classifier module accepts the extracted flows and classifies normal and attack flows. The Mitigator module activates the defense shield to protect the SDN controller from attacks. The data plane has a POX SDN controller which centrally controls the OpenFlow switches which exist in the data layer. The data plane consists of several switches and hosts. The adopted network topology has one POX controller and 7 OpenFlow switches, and 18 hosts. Each OpenFlow switch is connected to three hosts except OpenFlow S1. The S1 is connected to the six hosts.

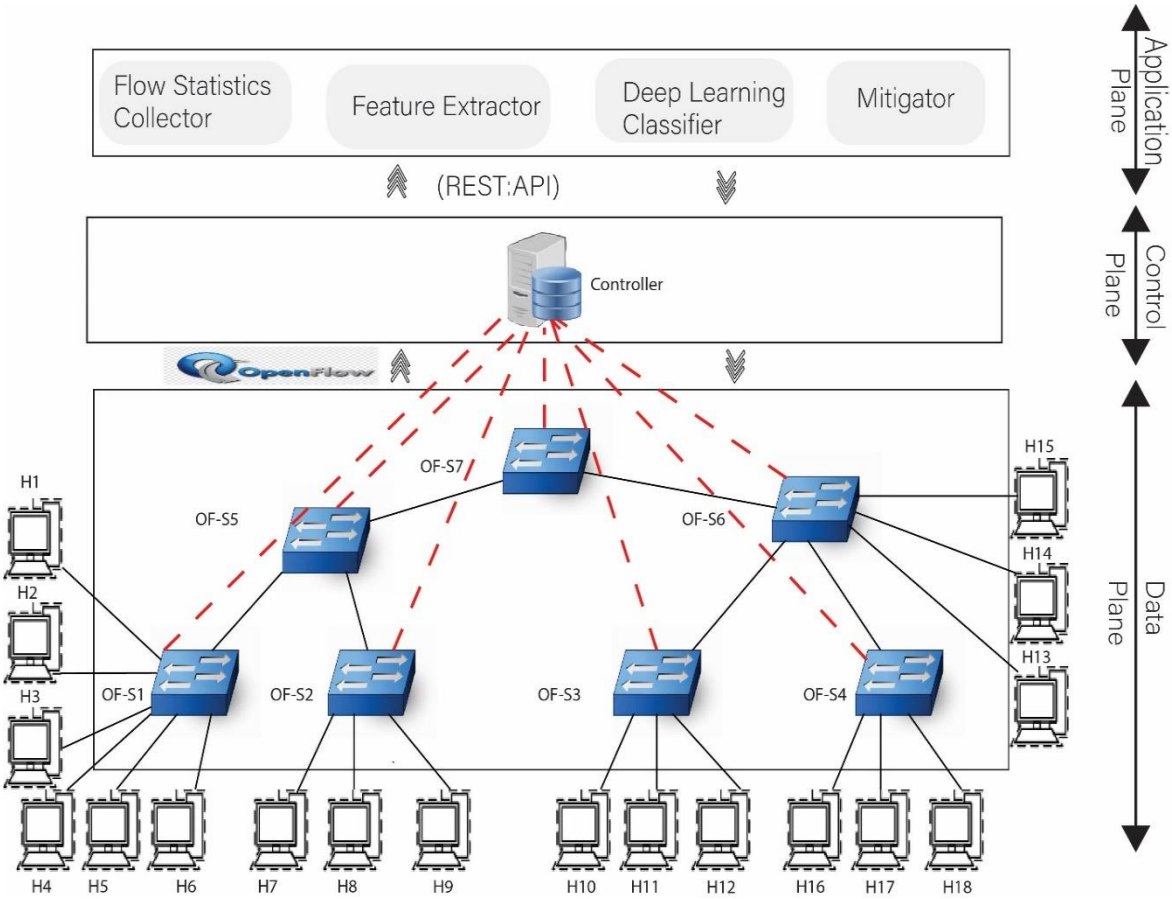


Figure 3.2: SDN-based experimental network topology.

After successfully developing the network topology in the Mininet, we run the "ping" command on all the connected hosts to verify the access to the controller, switches, and other hosts. The host H2 is selected as a botmaster, and H3, H4, H5, and H6 are corresponding bots, while H13 is selected as the target server and the other remaining hosts operated as normal users. The hosts H1, H7, H8, H9, H10, H11, H12, H14, H15, H16, H17, and H18 are used to generate background or normal network traffic while collecting the dataset and real-time evaluation. The OpenFlow switches are used to forward the network traffic, while the POX controller controls the entire network and helps to detect the attack. The port and

IP address settings of the network topology are described in Table 3.1. for further understanding.

Table 3.1: The setting of IP addresses and device ports for experimental network topology.

Device	Networks Port	Address	State	Device	Networks Port	Address	State
Controller	eth0	127.0.0.1	-	H10	eth0	10.0.0.10	Normal
H1	eth0	10.0.0.1	Normal	H11	eth0	10.0.0.11	Normal
H2	eth0	10.0.0.2	Bot-master	H12	eth0	10.0.0.12	Normal
H3	eth0	10.0.0.3	Bot	H13	eth0	10.0.0.13	Target Server
H4	eth0	10.0.0.4	Bot	H14	eth0	10.0.0.14	Normal
H5	eth0	10.0.0.5	Bot	H15	eth0	10.0.0.15	Normal
H6	eth0	10.0.0.6	Bot	H16	eth0	10.0.0.16	Normal
H7	eth0	10.0.0.7	Normal	H17	eth0	10.0.0.17	Normal
H8	eth0	10.0.0.8	Normal	H18	eth0	10.0.0.18	Normal
H9	eth0	10.0.0.9	Normal				

3.3.1 Design of Attack Traffic

Botnet-based DDoS attack in the custom-developed network topology is generated using Python scripts in all our experiments. These files have the code written in Python language to design and launch the attack. Initially, the “ping” command is executed on all the hosts to verify their reachability in the network topology. Once the complete topology is set up, then we run a file named “target.py” on the host H13 to set it as a target server. After that, we execute the

"bot.py" file on hosts H3, H4, H5, and H6 to set these hosts as bots. Finally, the "botmaster.py" Python file is executed on host H2 to set it as bot master. Here, the concept of socket programming is used to design and create specific ports for the corresponding bot hosts to connect with the bot master and listen to the instructions. For example, the bot master instructs the bots to remain ready and sends the date and time for the attack. When the date and time of all the bots are matched with instructed time and date of the bot master, then the bots quickly start to send the attack traffic to the specified target server. To collect the attack traffic for the data set, we launch the attack in the network topology for approximately "14.26" minutes.

3.3.2 Design of Normal Traffic

We used Distributed Internet Traffic Generator (D-ITG) [209] to generate background or normal network traffic to collect the normal traffic for the data set as well as for real-time verification. In our experiments, we used the D-ITG-2.8.1-r1023 version. The ITGSend and ITGRecv commands are executed on the normal hosts such as H1, H7, H8, H9, H10, H11, H12, H14, H15, H16, H17, and H18 to send and accept the normal network traffic. We approximately injected more than 200 flows as background flows into the network. The aim of this is to make the background traffic like the real network traffic. Different transmission rates such as constant, exponential, gamma distribution, uniform, and Poisson are followed with TCP protocol to inject each flow in the network. During each flow, the packet size also variated using different distributions such as constant, exponential, gamma distribution, uniform, and Poisson. Here, our focus was not on the same size of

packets so, we used variations in the packet size to make the virtual network like a real network. Figure 3.3 shows an example of flow rules written in D-ITG to generate the background traffic.

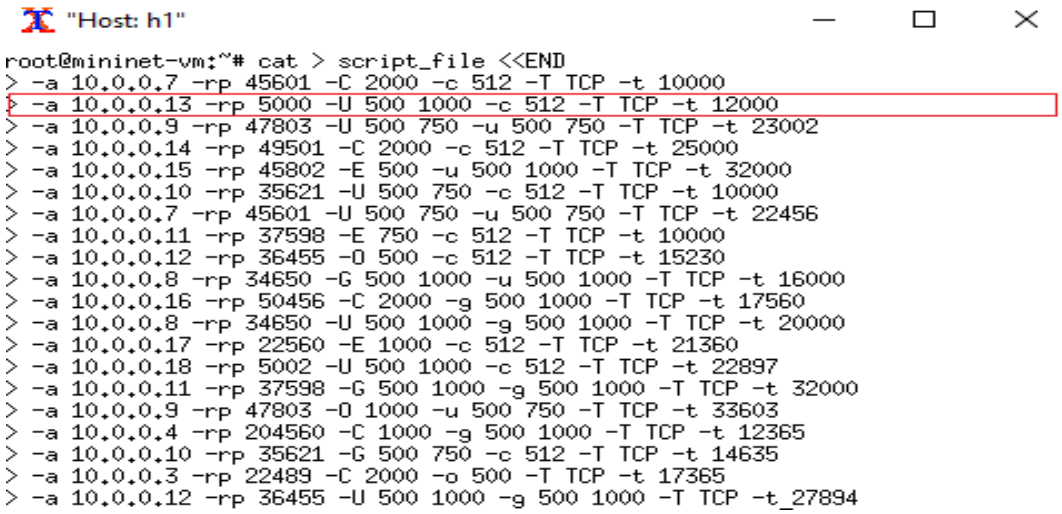


Figure 3.3: An example of flow rules to generate normal traffic.

Each row as shown in Figure 3.3 has a flow rule to generate the background traffic. So, to generate background traffic these flow rules are executed on different hosts which are specified as normal users. By taking the red box in Figure 3.3 as an example, we can observe that the 3rd flow rules are follows the 2nd flow rules and so on. Where the aim is to send the normal traffic to the destination host H13 ("IP address 10.0.0.13 with port 5000"). The uniform distribution from 500 to 1000 follows to send the number of packets/per flow, and the size of the transmitted packets is fixed at 512 bytes. The duration of the traffic for the flow is set to 12000 milliseconds, where the TCP protocol-based traffic is generated for this flow rule. Figure 3.4 shows a portion of the screenshot of the OpenFlow switches' flow table. The red box in Figure 3.4 highlights the background traffic is successfully injected

into the network. The red box also shows a flow entry of host H1 (“IP address 10.0.0.1) to host H13 (“IP address 10.0.0.13”) with destination port 5000 in the flow table of the OpenFlow switch. The above-discussed procedure can be used to successfully inject the background traffic using G-ITG into the SDN-based network.

Figure 3.4: Partial screenshot of the flow table of an OpenFlow switch during normal traffic.

3.4 Feature Extraction and Labeling

A software-defined network environment is adopted to capture the attack and normal traffic. The captured traffic is initially stored in the “Pcap” format in Wireshark. After that, CIC Flow Meter V4 (version 4) [210], [211] is used to convert the “Pcap” format to “CSV” and make a dataset to train and validate deep learning methods. The following subsection gives an overview of the CIC Flow Meter.

3.4.1 CIC Flow Meter

In the field of network intrusion detection, the CIC Flow Meter is used to generate bidirectional flows and to convert “Pcap” files to “CSV” files. Each flow consists of 83 features. A network flow is the traveling of packets in a unidirectional sequence from source to destination with a particular protocol in a specific period. CIC Flow Meter terminates any flow in two ways: time out is reached and the connection is closed. First, we can set a specific time duration to terminate the flows, and when the set time out is reached then the flow automatically terminates. Second, any flow is terminated by detecting any packet having a “FIN flag” either from destination to source or source to destination side. The definition of TCP specification [212] is ignored in the connection closed method in this way: the flow is terminated when both side source and destination send packets that are containing “FIN flags”. This creates two severe consequences. First, the construction of that flow which are having “ACK” and “FIN” flags because that part of the current flow and the other flows. The second one is ignorance of flows that do not have “FIN” flags packets. For example, CIC Flow Meter originally ignored the packets that are having “RST” flags and did not consider them to close connection and terminate flows.

The second method has more consequences than the first method to terminate the flows. So, in this research, we adopted the first method (e.g., flow time out is reached) and set the “600” seconds to terminate the flows. The CIC Flow Meter converted the “Pcap” file into a dataset that has a total of 89, 632 flow records with 83 features. Furthermore, the dataset has 41,242 attack flow records and

48,390 normal flow records with the same 83 features. Table 3.2. shows the feature names, feature codes, and descriptions of all the extracted features.

Table 3.2: List of extracted features from traffic flows.

Forward	Description	Feature Code and Names	Description
F1. Flow ID	The assigned ID of a Flow	F43. Forward Packets/second	The number of packets transmitted in the forward direction per second.
F2. Source IP	Source IP address	F44. Backward Packets/second	The number of packets transmitted in the backward direction per second.
F3. Source Port	Source port number	F45. Packet Len Min	The minimum length of a packets in a flow.
F4. Destination IP	Destination IP address	F46. Packet Len Max	The maximum length of a packets in a flow.
F5. Destination Port	Destination port number	F47. Packet Len Mean	Mean of a flow length
F6. Protocol	Type of protocol	F48. Packet Len Std	Standard deviation of a flow length
F7. Timestamp	Capture time	F49. Packet Len Var	Minimum nter-arrival time of a packet
F8. Flow Duration	The duration of a flow	F50. FIN Flag Cnt	FIN-flagged packet count for a flow
F9. Total Forward Packets	Total number of forward packets	F51. SYN Flag Cnt	SYN-flagged packet count for a flow
F10. Total Backward Packets	Total number of backward packets	F52. RST Flag Cnt	RST-flagged packet count for a flow
F11. Total Length Forward Packets	The total length of forward packets	F53. PSH Flag Cnt	PSH-flagged packet count for a flow
F12. Total Length Backward Packets	Total length of backward packets	F54. ACK Flag Cnt	ACK-flagged packet count for a flow
F13. Forward Packets Length Maximum	The maximum length of forward packets	F55. URG Flag Cnt	URG-flagged packet count for a flow

F14. Forward Packets Length Minimum	The minimum length of forward packets	F56. CWE Flag Count	CWE-flagged packet count for Flows
F15. Forward Packets Length Mean	The length Mean of the forward packets	F57. ECE Flag Count	ECE-flagged packets count for a flow
F16. Forward Packets Length Strand Deviation	The length variance of the forward packets	F58. Up/Down Ratio	Download and Upload ratio
F17. Backward Packets Length Maximum	The maximum length of backward packets	F59. Packet Size Average	Size of a package on average
F18. Backward Packets Length Minimum	The minimum length of the backward packets	F60. Forward Segment Size Average	Observed average packet size moving in forward direction
F19. Backward Packet Length Mean	The length Mean of the backward packets	F61. Backward Segment Size Average	Observed average packet size moving in backward direction
F20. Backward Packet Length Strand Deviation	The length variance of the backward packets	F62.Forward Bytes/b Average	Observed average counts of bytes per bulk moving in forward direction
F21.Flow Bytes/second	Number of bytes of the flow per second	F63. Forward Packets/b Average	Observed average counts of packets per bulk moving in forward direction
F22. Flow Packets/second	Number of packets of the flow per second	F64. Forward Bulk Rate Average	The average rate of bulk data transmission observed in the forward direction.
F23. Flow IAT Mean	Mean of Packets flow inter arrival time	F65. Backward Bytes/b Average	Observed average counts of bytes per bulk moving in backward direction
F24. Flow IAT Strand Deviation	Standard deviation of Packets flow inter arrival time	F66. Backward Packets/b Average	Observed average counts of packets per bulk moving in backward direction
F25. Flow IAT Maximum	Maximum of Packets flow inter arrival time	F67. Backward Bulk Rate Average	The average rate of bulk data transmission

			observed in the backward direction.
F26. Flow IAT Minimum	Minimum of Packets flow inter arrival time	F68. Subflow Forward Packets	Observed subflow packets counts in forward direction
F27. Forward IAT Total	Total time interval of the forward packets	F69. Subflow Forward Bytes	Observed subflow bytes counts in forward direction
F28. Forward IAT Mean	Mean time interval of the forward packets	F70. Subflow Backward Packets	Observed subflow packets counts in backward direction
F29. Forward IAT Strand Deviation	Standard deviation time interval of the forward packets	F71. Subflow Backward Bytes	Observed subflow bytes counts in backward direction
F30. Forward IAT Maximum	Maximum time interval of the forward packets	F72. Initially Forward Window Bytes	Initially window byte counts moved in forward
F31. Forward IAT Minimum	Minimum time interval of the forward packets	F73. Initially Backward Win Bytes	Initially window byte counts moved in backward
F32. Backward IAT Total	Total time interval of the backward packets	F74. Forward Act Data Packets	Number of packets in the forward direction with at least one byte of TCP data payload
F33. Backward IAT Mean	Mean time interval of the backward packets	F75. Forward Segment Size Minimum	Observed Minimum size of a segment moving in the forward direction
F34. Backward IAT S Strand Deviation	Standard deviation time interval of the backward packets	F76. Active Mean	Mean of time when a flow was active before becoming idle
F35. Backward IAT Maximum	Maximum time interval of the backward packets	F77. Active Standard Deviation	Standard deviation of time when a flow was active before becoming idle
F36. Backward IAT Minimum	Minimum time interval of the backward packets	F78. Active Maximum	Maximum of time when a flow was active before becoming idle

F37. Forward PSH Flags	PSH-flagged packet counts moving in the forward direction	F79. Active Minimum	Minimum of time when a flow was active before becoming idle
F38. Backward PSH Flags	PSH-flagged packet counts moving in the backward direction	F80. Idle Mean	Mean of time when a flow was idle before becoming active
F39. Forward URG Flags	URG-flagged packet counts moving in the forward direction	F81. Idle Standard Deviation	Standard deviation of time when a flow was idle before becoming active
F40. Backward URG Flags	URG-flagged packet counts moving in the backward direction	F82. Idle Maximum	Maximum of time when a flow was idle before becoming active
F41. Forward Header Length	Bytes count for headers moving in the forward direction	F83. Idle Minimum	Minimum of time when a flow was idle before becoming active
F42. Backward Header Length	Bytes count for headers moving in the backward direction		

3.5 Data Pre-Processing

In this research, three steps including quantization, removing irrelevant features, and normalization are performed to pre-process the dataset. The developed dataset has different categorical attributes (e.g., protocol, flags, services, classes, etc.). So, the quantization process is performed to convert the categorical values

into numerical values by assigning a unique number to each attribute category. Quantization is an important task in deep learning because DL methods cannot process the nominal features directly. In the second step, the irrelevant features such as Flow ID, source IP address, destination IP address, source port number, destination port number, timestamp, and protocol has been removed to generalize the dataset. Removing the irrelevant above-mentioned features prevents the DL methods from attributing the specific ports, IPs, protocols, etc. as attack nodes and maintains the generalized nature of DL methods. Lastly, the normalization tasks are performed to evaluate the dataset for missing and infinities values. The normalization of data is particularly helpful for the DL methods represented at several levels. It also helps the learning process of the DL methods may not be affected by the value ranges of different features of the dataset. In this research, we used Min-Max normalization to normalize the values of the dataset and it helps the neural networks to generalize themselves in a more consistent way. This method can effectively and accurately conduct all data connections. The increasing function follows the min-max range for the true values which was added during the classification process. Nevertheless, the values of the features can lie within the existing range [213] . This method is mathematically formulated as follows:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

After performing the above-mentioned steps for data pre-processing, the dataset is encoded (the normal flows labeled as 0 and attack flows labeled as 1). As mentioned earlier, the dataset has 41,242 attacks and 48,390 normal flow records.

The imbalance is retained in the dataset to replicate real-world scenarios for the DL methods ("where the number of normal flows is always greater than attack flows"). The following formula is used to maintain the imbalanced structure.

$$Imbalance\ Ratio\ (R) = \frac{the\ majority\ class\ (\max)}{the\ minority\ class\ (\min)} \tag{3.2}$$

For all experiments, we divided the dataset into 75% and 25% ratios. 75% of the dataset is used for training and 25% for the validation of all DL methods for all subsets of features. The distribution of the dataset for the training and validation for all subsets is given in Table 3.3.

Table 3.3: Distribution of the dataset for training and testing.

Classes	Total Records	75% for Training	25% for Testing
Normal Flows	48,390	36,293	12,097
Attack Flows	41,242	30,932	10,310
	89,632	67,225	22,407

Data Reshaping. A CNN can accept input in three-dimensional image format (height, width, and channel). CNNs can deal with network traffic data that is typically one-dimensional. We must add one more step to modify the input traffic data to meet CNN's resolution parameters. For example, for a subset with 43 features, we convert the 73-dimensional vector into an image with a "7 ×6" shape, and for the 30-dimensional input vector an image with "6 ×5" is created, etc. In this research, for all experiments and DL methods, only grayscale images were generated with a single channel, and we set "1" for the channel number.

3.6 Feature Selection

Feature engineering or feature selection is an important pre-processing task before the training and evaluation of DL methods. Because the presence of redundant and irrelevant features in a dataset can reduce the performance of the predictive models, due to the curse of dimensionality and over-fitting problems. Moreover, even if any model has effective performance with noise and redundancy, the presence of those features poses some other disadvantages such as increasing computational and storage costs, increasing time costs and reducing the model interpretability. So, the feature selection methods can help to mitigate these problems through the identification and selection of important or relevant features, and discarding the of redundant and irrelevant features.

Especially, the growing interest in explainable AI encourages the researcher to focus on improving the interpretability of models. The interpretability of deep or machine learning models becomes essential because it enables the models to adopt social requirements such as privacy, fairness, reliability, unbiasedness, and trust.

Since modern and advanced DL methods and applications have an ever-growing number of features, so, a virtual understanding of prediction outcomes without dimension reduction becomes quite difficult. On the other side of model interpretation, for example, data visualization helps to reduce the number of features. The other advantage of feature selection is for those applications where the acquisition of data is costly. In such cases, the identification of redundant and irrelevant features is important to reduce the acquisition cost of data.

In current years, the usage of feature selection methods has increased in deep learning-based network intrusion detection applications. Because the attackers are using modern technology to launch malicious attacks and they are hard to detect. So, the detection of these attacks becomes easier and more effective with optimal features. As deep learning-based intrusion detection systems take a short time to analyze a large number of traffic time. Thus, the subset of optimal features can become an optimal solution to improve the performance and operations of the IDS in terms of accuracy, speed, response time, and simplicity.

In the current research, CIC Flow Meter generates 83 statistical features for a flow which are described in Table 3.2. Although some of them are important for the detection of attack flows, some may have no or little effect on the prediction accuracy of the DL methods, they just upsurge the time and computational costs. So, we need to select the subset of optimal features that can help the DL methods to discriminate the attack and normal flows, also accurate, and boost the classification performance. In all our experiments, the optimal feature selection is performed using two different methods: (i) Feature Weighting and (ii) Threshold Tuning. This method selects the optimal features without changing the original features and then converts them into five different subsets. Let's consider a given set of features can be represented as $\{F_1, F_2, F_3, \dots \dots \dots F_d\}$ are d features of X, where d indicates the number of high-dimension features. First, we convert the high-dimension features d to low-dimension features $r(r < d)$ using the above-mentioned methods and then make a subset of optimal features to recognize the

attack and normal flows. The complete procedure of feature selection is described in Algorithm 2.

Algorithm 2: Procedure of feature selection

Input: Feature Set F = {F1, F2, F3, F4, ……. Fd}, the threshold of weight α, and the number of selected features r;

Output: Selected feature set F'

// Calculate the weights of the features

1: F' ⟵ Ø

2: for (i = 0; i < d; i++) do

3: r(i) ⟵ compute weights (F_i)

4: if (|r(i)| < α) then

5: remove feature F_i;

6: else

7: FA [] = F_i

8: end if

9: end for

10: F' ⟵ store (FA [])

11: return F'

3.6.1 Feature's Weighting

In this research, we used iterative wrapper-based feature selection with Support Vector Machine (SVM). The whole dataset is converted into five different subsets with optimal features. SVM is a powerful classifier that assigns weights to all the features while predicting the output. The SVM allocates a weight that can be

used to rank the significance of the feature. It divides the various classes using a hyperplane and employs a kernel function to translate the Initially input feature space to a high-dimensional feature space. The following procedure is used to compute the weights for a binary soft margin SVM and rank the weights.

Let's consider $\{X, Y\}$ is a training dataset with $\{1, -1\}$ are labels, where X represents the input features and Y represents labels. The training dataset has k samples. $\emptyset$ Feature mapping is applied to the inputs, then the decision function can be formulated as follows:

$$f(X) = \big(\omega, \emptyset(X)\big) + b \tag{3.3}$$

In the above Equation 3.3 b and ω defined the properties of the SVM hyperplane. The training goal of SVM is to achieve the optimal values for b and ω that maximize the distance between the hyperplane and the mapped training samples $\emptyset(X)$. The loss function (L) can be minimized by applying the quadratic penalties to the misclassified examples.

$$min_{\omega,\delta}\, L = \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{k}\delta_i^2 \tag{3.4}$$

$$\forall i, y_i f(x_i) \geq 1 - \delta_i \tag{3.5}$$

Where C represents the penalty factor and δ_i are slack variables representing the distances by which the soft margins are despoiled by the misclassified examples. x_i are input variables and y_i represents the labels of a single training sample. So, the weights are calculated as follows:

$$W = \sum_{i=1}^{k} a_i^* y_i \emptyset\,(x_i) \tag{3.6}$$

Where a_i^* is the solution of:

$$min_{a_i}\, W(a_i) = \sum_{i=1}^{k} a_i + \frac{1}{2}\sum_{i,j=1}^{k} a_i a_j\, y_i y_j (K(x_i, x_j) + \frac{1}{C}\vartheta_{i,j}) \tag{3.7}$$

$$s.t. \qquad \sum_{i=1}^{k} a_i y_i = 0 \tag{3.8}$$

$$\forall i, a_i \geq 0$$

Where $\vartheta_{i,j}$is the Kronecker symbol and $K(x_i, x_j) = \{\ \emptyset\,(x_i), \emptyset\,(x_j)\}$ represents the Gram matrix of the training samples. So, the SVM weights ω can be used to score the feature's importance. The feature with a higher weight is considered more important compared to the feature with a lower weight. It means that the features with higher weights are more important for detecting attack flows. Table 3.4. Described the SVM assigned absolute weights values for the distinct features. After obtaining the weight values for all the features, a threshold tuning method is used to derive an optimal threshold value. During each iteration, this optimal threshold value is used to evaluate the feature weight and select the optimal features. The features with weights equal to or greater than the threshold $(F_n(w_n) \geq \alpha)$ were selected and placed into the corresponding subset. Then, the optimal feature subsets are given as input to the DL methods for classification.

Table 3.4: List of features with assigned weights by SVM.

Feature Code and Names	Assigned Weights	Feature Code and Names	Assigned Weights
F1. Flow ID	-	F43. Forward Packets/s	6.17

F2. Source IP	-	F44. Backward Packets/s	5.25
F3. Source Port	-	F45. Packet Len Min	1.17
F4. Destination IP	-	F46. Packet Len Max	2.53
F5. Destination Port	-	F47. Packet Len Mean	1.22
F6. Protocol	-	F48. Packet Len Standard deviation	5.12
F7. Timestamp	-	F49. Packet Len Var	1.05
F8. Flow Duration	4.51	F50. FIN Flag Cnt	2.75
F9. Tot Forward Packets	2.05	F51. SYN Flag Cnt	6.23
F10. Tot Backward Packets	4.30	F52. RST Flag Cnt	1.94
F11. Tot Len Forward Packets	4.70	F53. PSH Flag Cnt	1.56
F12. Tot Len Backward Packets	9.48	F54. ACK Flag Cnt	2.07
F13. Forward Packet Len Max	1.58	F55. URG Flag Cnt	0.00
F14. Forward Packet Len Min	2.92	F56. CWE Flag Count	0.00
F15. Forward Packet Len Mean	2.13	F57. ECE Flag Cnt	0.00
F16. Forward Packet Len Standard deviation	5.70	F58. Down/Up Ratio	4.82
F17. Backward Packet Len Max	1.16	F59. Packet Size Average	2.10
F18. Backward Packet Len Min	3.78	F60. Forward Segment Size Average	2.13
F19. Backward Packet Len Mean	2.81	F61. Backward Segment Size Average	2.81
F20. Backward Packet Len Standard deviation	3.38	F62. Forward Bytes/b Average	0.00
F21. Flow Bytes/s	1.01	F63. Forward Packets/b Average	0.00

F22. Flow Packets/s	1.55	F64. Forward Blk Rate Average	0.00
F23. Flow IAT Mean	1.81	F65. Backward Bytes/b Average	0.00
F24. Flow IAT Standard deviation	3.39	F66. Backward Packets/b Average	0.00
F25. Flow IAT Max	1.16	F67. Backward Blk Rate Average	0.00
F26. Flow IAT Min	1.82	F68. Subflow Forward Packets	2.05
F27. Forward IAT Tot	4.18	F69. Subflow Forward Bytes	4.70
F28. Forward IAT Mean	1.09	F70. Subflow Backward Packets	4.30
F29. Forward IAT Standard deviation	1.30	F71. Subflow Backward Bytes	9.48
F30. Forward IAT Max	6.00	F72. Initial Forward Win Bytes	0.00
F31. Forward IAT Min	1.86	F73. Initial Backward Win Bytes	1.16
F32. Backward IAT Tot	5.72	F74. Forward Act Data Packets	1.05
F33. Backward IAT Mean	9.59	F75. Forward Segment Size Minimum	0.00
F34. Backward IAT Standard deviation	8.10	F76. Active Mean	1.48
F35. Backward IAT Maximum	9.67	F77. Active Standard deviation	3.02
F36. Backward IAT Minimum	1.27	F78. Active Max	1.17
F37. Forward PSH Flags	0.00	F79. Active Min	7.51
F38. Backward PSH Flags	1.56	F80. Idle Mean	6.81
F39. Forward URG Flags	0.00	F81. Idle Standard deviation	1.61

F40. Backward URG Flags	0.00	F82. Idle Maximum	1.07
F41. Forward Header Len	2.52	F83. Idle Minimum	5.27
F42. Backward Header Len	2.09		

3.6.2 Threshold Tuning

An optimal threshold value is calculated from feature weights using a simple threshold tuning method. This method takes absolute feature weight values from minimum to maximum and then returns an optimal value between the given range. The best threshold value is computed as the value which reduce the features dimensions. So, based on the optimal values we make the five different subsets of the features to evaluate the performance of the DL methods.

3.7 Deep Learning Methods and Hyper-Parameters Settings

3.7.1 Recurrent Neural Networks (RNNs)

Generally, RNNs are used to deal the problems related to time series, because these methods have the capabilities to learn the features from the time-series data compared to CNNs and other DL methods. For example, one sentence in the natural language is considered a type of time-series data. It means that each word in a sentence has a correlation with the other words so, the previous and current words can be used as input to predict the next word. Further, the feed-forward methods cannot store or remember the previous input information, that's why these methods are not suitable for tasks related to time-series data. RNNs can

learn from data sequentially. In a neural network, the information related to previous inputs can be stored in an internal state and the RNN methods can learn from time-series data. In intrusion detection, the malicious traffic may be hidden within the normal traffic, so, the RNNs can perform better than others for attack detection [214], [215]. Figure 3.5 shows a directed cell, which can be used to construct a connection among various neurons.

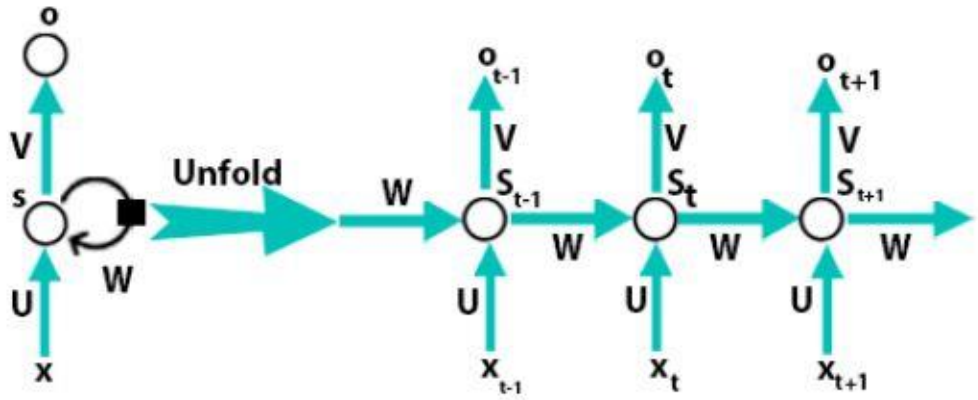


Figure 3.5: General diagram of RNN architecture.

An RNN consists of input, hidden, and output units. Let's consider the input units $\{x_0, x_1, x_2, \ldots\ldots\ldots x_t \ldots\ldots\ldots x_{t+1}\}$, hidden units $\{h_0, h_1, h_2, \ldots\ldots\ldots h_t \ldots\ldots\ldots h_{t+1}\}$, and output units $\{y_0, y_1, y_2, \ldots\ldots\ldots y_t \ldots\ldots\ldots y_{t+1}\}$ for an RNN. So, in Figure 3.5, it is shown that at time step t, RNN takes as input the current sample x_t and the previously hidden representation h_{t-1} to obtain the currently hidden representation h_t, and it is performed using the following mathematical formulation:

$$h_t = f(x_t, h_{t-1}) \tag{3.9}$$

Where f represents an encoder function for an RNN. For time step t, the most commonly useable vanilla for the RNN can be formulated as follows:

$$h_t = f\left(W_{hx,x_t} + W_{hy,h_{t-1}} + b_h\right) \tag{3.10}$$

$$y_t = g\left(W_{hy,h_{t-1}} + b_y\right) \tag{3.11}$$

In the above Equations 3.10, 3.11 f and g represent the encoder and decoder, respectively and $\theta = \{W_{hx}, W_{hh}, W_{hy}, b_h, b_y\}$ is a set of parameters. During a forward pass, RNN can capture the dependencies between the current sample x_t and the previous sample x_{t-1} by integrating the previously hidden representation h_{t-1}. RNN also can capture arbitrary-length dependencies in the data. However, RNNs do not produce effective results while dealing with long-term dependencies in the data due to gradient vanishing problems. Other methods such as Long Short-Term Memory (LSTM) etc., have solved the gradient vanishing and gradient exploding problems of the RNNs. In recent years, the usage of RNN and its variants have increased in different applications such as machine translation, intrusion detection, etc. The parameter settings for the RNN which is used in this research are given in Table 3.5.

3.7.2 Convolutional Neural Networks (CNNs)

A convolutional neural network is a type of multi-layer ANN that is mostly used for intrusion detection. CNNs run a simple mathematical operation which is known as convolution. Convolution is a specialized type of linear operation. These networks have at least one convolutional layer rather than general matrix multiplication [216]. Generally, CNNs composed of a convolutional, pooling, and fully connected layer. CNNs can automatically learn and extract complex attributes. The convolutional layer provides an advanced representation of the attributes [217]. The CNN architecture with one layer is shown in Figure 3.6.

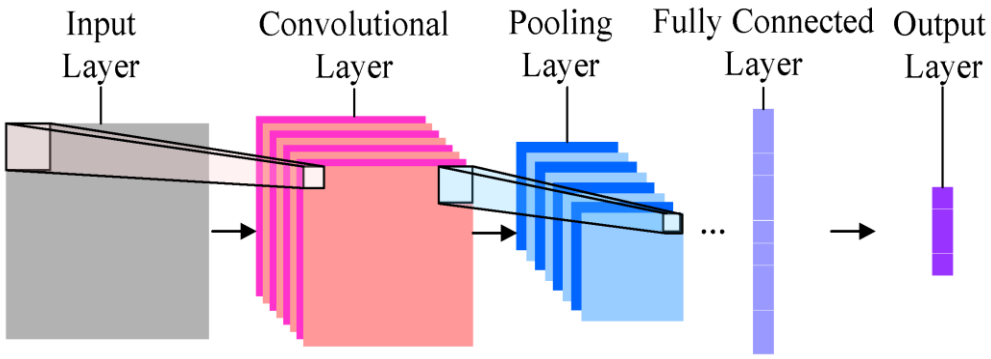


Figure 3.6: Structural diagram of CNN architecture.

The convolutional operation for any CNN can be formulated as follows:

$$X_i^a = \emptyset\left[\sum_{i \epsilon k_i} X_j^{a-1} \times W_{ij}^a + b_j^a\right] \quad (3.12)$$

Where X_i^a represents the attribute map i of the convolutional layer a. The $\emptyset$ demonstrate an activation function. k_i represent the set of input features for the

layer $(a-1)$. W_{ij}^{a} is the connection weight between the feature i of the convolutional layer a and feature j of the convolutional layer $(a-1)$. The b_j^a is used to show the deviation among the related layers.

The pooling layer is next to the convolutional layer. This layer aims to reduce the size of the feature map. The operations of the pooling layer are responsible for the proper identification of important features, reducing the complexities in the data, and improving the network tolerance against environmental changes. The following mathematical equation demonstrates the operation in the pooling layer.

$$X_i^a = \emptyset\left[\beta_i^a c(X_j^{a-1} + b_j^a)\right] \tag{3.13}$$

In Equation 3.13, β represents the weighting matrix while c is the sub-sampling function. In CNNs, the classification process is performed from convolutional layers to pooling layers through fully connected layers. The output function for the fully connected layers can be formulated as follows:

$$Y^m = \emptyset[W^m X^{m-1} + b^m] \tag{3.14}$$

In the above Equation 3.14, the layer index is represented by the m, Y^m indicates the output of the fully connected layer, X^{m-1} is the fully connected layer input, W^m is the weighting coefficient, and b^m term is deviation [56]. The hyper-parameter setting of the CNN which is used in this research is given in Table 3.5.

3.7.3 Multilayer Perceptron (MLP)

It maps the input data to output in a feed-forward way [218]. The general architecture of the MLP is shown in Figure 3.2. Its architecture has multiple layers of interconnected neurons and each layer is fully connected with its previous and next layers [219]. Our research used one hidden layer for the MLP method to reduce its complexity and computing resources. We used the ReLu activation function in hidden layers and the sigmoid activation function at the output layer. Table 3.5 shows the parameter settings of the used MLP method. The architecture of MLP is shown in Figure 3.7.The hit-and-trial method is used for all the DL classifiers to find the best parameters.

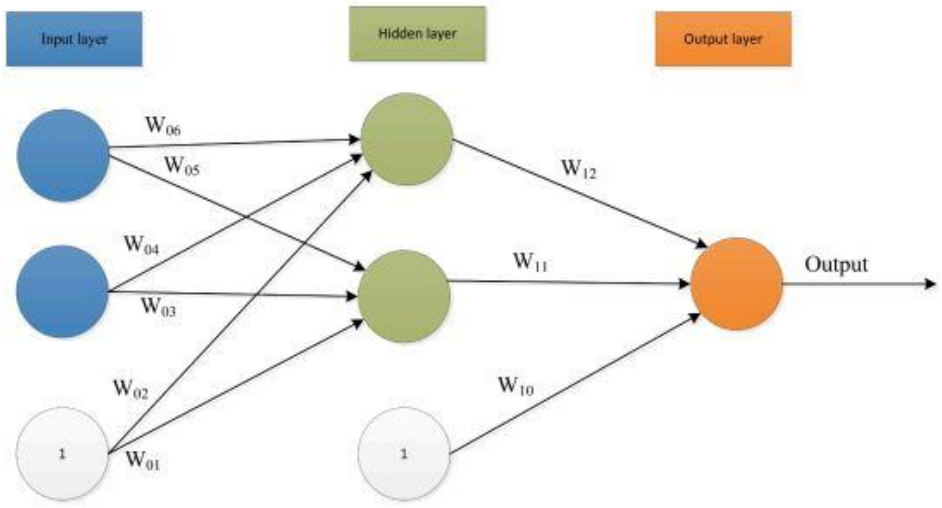


Figure 3.7: The architecture of MLP.

During the training of MLP, the connection weights are adapted to minimize the difference between the obtained and actual output. To achieve this, we used the backpropagation method. The output of each neuron is referred to as a weight unit, followed by an activation function to discriminate the linearly or nonlinearly

separable data [220]. The following mathematical equation can be used to calculate the output activation α^{l+1} at layer $l+1$.

$$\alpha^{l+1} = \emptyset(W^l \alpha^l + b^l) \tag{3.15}$$

Where l indicates the layer, $\emptyset$ is the activation function (i.e., rectified linear unit, hyperbolic tangent, sigmoid), W^l represents the weights, and b^l is biased at the particular layer. For example, if we have MLP with m layers then the first and last layers can be formulated as follows:

$$\alpha^l = x$$

$$h_{W,b}(x) = \alpha^m \tag{3.16}$$

The back-propagation method is used to decide the learning weights and bias and to get an approximation of unknown input and output relation. The objective function described below reduces the divergence between expected and actual outcomes.

$$J(W, b; x, y) = \frac{1}{2}||h_{W,b}(x) - y||^2 \tag{3.17}$$

3.7.4 Deep Neural Networks (DNNs)

Generally, a vector format is used in the neural networks for the inputs. These inputs then pass through numerous hidden layers. The network's output layer shows the output outcomes of processing from hidden layers finally. Each of the network's hidden layers contains several neurons linked to the previous neurons. Each layer's neurons function independently and have no connections to other

neurons. The final completely connected layer's output layer is in charge of determining a grade for each class. CNNs frequently aren't the best choice for regular data. So, DNNs [221], [222] take advantage of the inputs containing accurate examples and appropriately restrict the network's architecture. An ANN is a term that describes a subfield of machine learning known as deep learning. The term "deep learning" is exceedingly popular in artificial intelligence. The neural network's hidden layer count increases. On a large scale, ANNs learn from observational data. The original format of any neural network is shown in Figure 3.8.

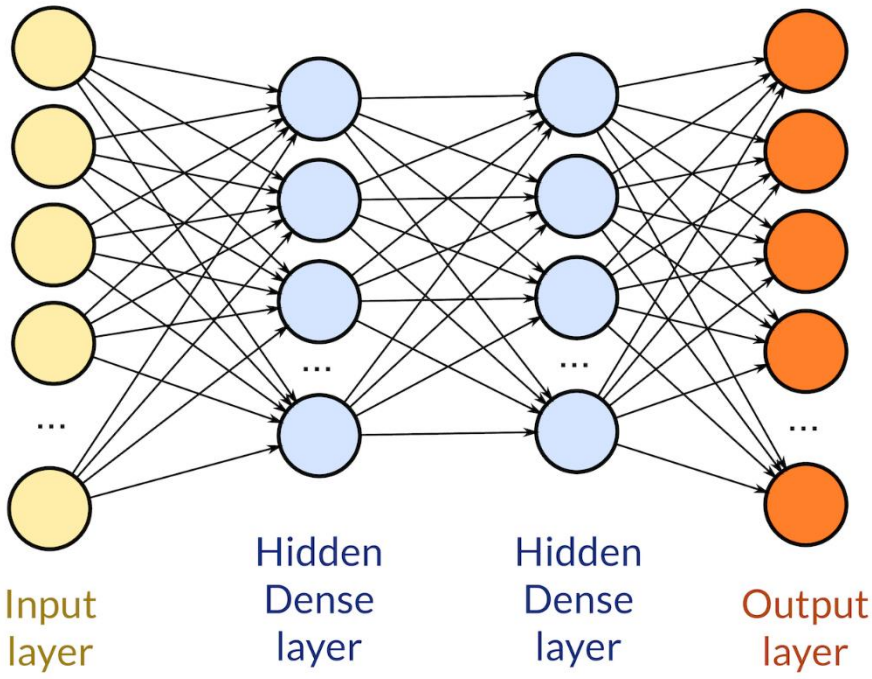


Figure 3.8: Diagram of DNN architecture.

An ANN is typically created by embedding two sets of neurons in the input and output layers. The input layer receives the features X_i, and the output layer

responds with a view of the features. The output layer creates a bias and a set of weights for the input features. The results are then produced by applying non-linear or linear transfer functions, also called non-linear or linear transfer functions, also referred to as "activation functions." A building block of actual neural networks is the activation law for neurons. The following equation is used to calculate the total weight for the input feature X_i and allocated weight to it W_i from an N-dimensional features vector.

$$Z = \sum_{i=1}^{n} (X_i \,.\, W_i) + b \tag{3.18}$$

Where W_i is the weight for the input vector and b represents the bias term. In neural networks, the bias term is a significant factor for arranging activator flexibility when calculating neurons' output. The ANN architecture performance is specified using the activation function. The activation function terms the output of a neuron corresponding to given input features. Researchers have proposed several activation functions, and some specific parameters, and new functions have also been investigated in recent years. There are some commonly used activation functions such as Rectified Linear units (ReLu), Sigmoid (logistic), Hyperbolic Tangent (Tanh), etc. The range of a sigmoid function is between 0 and 1 and it can be easily understood and implemented. But, it has a few drawbacks: a) slow convergence, and b) vanishing gradient problems. The sigmoid activation function is mathematically formulated as follows:

$$Sigmoid(x) = \frac{1}{1+\exp(-x)} \tag{3.19}$$

The output of the Tanh activation function is zero-centered because it has a range between -1 to 1(e.g., -1 < output <1). Although this method simplifies optimization more than the sigmoid function, vanishing gradient issues still exist. The mathematical formula of the Tanh function is given in Equation 3.20.

$$Tanh(x) = \frac{1-exp(-2x)}{1+exp(-2x)} \tag{3.20}$$

ReLu is known as a general activation function, easy to understand, efficient, and fast, and most widely used in different cases. ReLu is simple because it does not require any exponential computation and normalization and also has simple mathematical operations compared to Tanh or sigmoid activation functions. Furthermore, this function rectifies and avoids the vanishing gradient problems and also has improved in convergence. This function is mathematically formulated in Equation 3.21.

$$ReLu\ (x) = \max(0, x) = \begin{cases} 0 & ,x < 0 \\ x & ,x \geq 0 \end{cases} \tag{3.21}$$

Furthermore, The Tanh or sigmoid activation functions are unsuitable for the hidden layers because if x is too small or too large that makes the gradient very small and slows down the gradient descent. For hidden layers, the ReLu activation function is the right choice, its derivative is 0 when x is negative and 1 when x is positive.

For binary classification problems, given that the output value is either 1 or 0, the sigmoid function, compared to Relu or Tanh, is appropriate for the output layer. In this research, a DNN which consists of three hidden layers shown in Figure

3.5 is used. The development of hidden layers with activation functions makes its architecture deeper to solve complicated problems. we used ReLu activation functions in hidden layers and sigmoid activation functions in the output layer. The hyper-parameters setting of DNN is given in Table 3.5.

3.7.5 Long Short-Term Memory (LSTM)

Another commonly useable Deep learning algorithm is Long Short-Term Memory (LSTM), which can learn the data's short-term dependencies. For some DL algorithms such as DNNs, the inputs can be considered as independent of each other. These algorithms produce fixed-sized outputs by accepting fixed-sized inputs. They cannot handle the inputs with varying sequence lengths or output data. Recurrent Neural Networks (RNNs) overcome these drawbacks by processing the sequences of variable or fixed lengths. Compared to DNNs, RNNs compute the next output by using all the information of the previous inputs (i.e., previous timestamp information effect on the prediction of current timestamp). So, RNNs can learn or capture short-term dependencies between input data and outputs. The architecture of LSTM with one hidden layer which is used in this research is given in Figure 3.9.

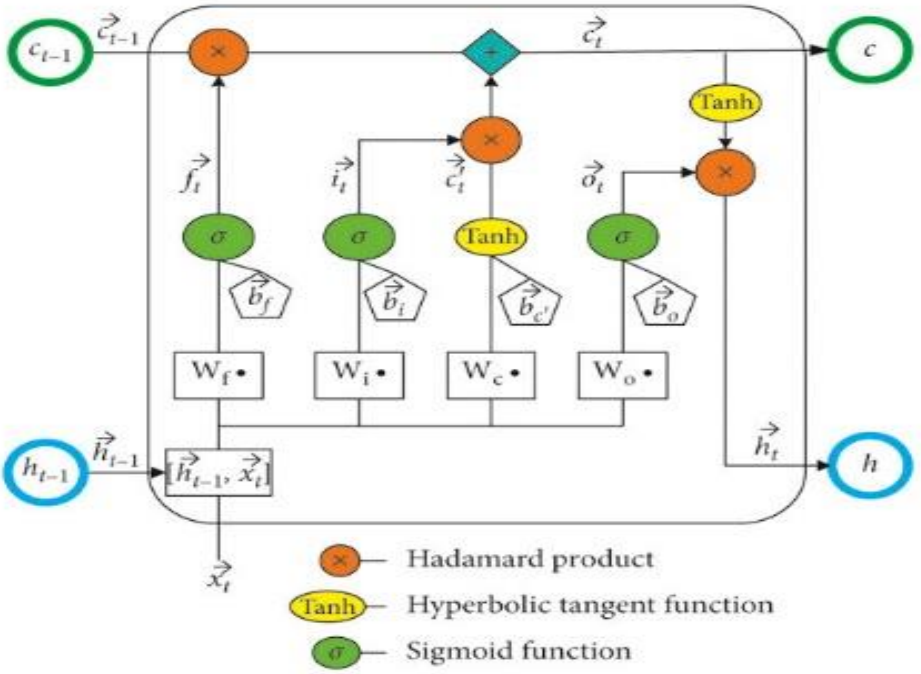


Figure 3.9. The architecture of LSTM.

RNNs can perform better when they used past or recent information to predict the next output. However, with the large gaps between the relevant information, RNNs cannot produce good results. Because they have vanishing gradient problems, which arise with exponential decay in back-propagated error. LSTM is an advancement in the RNNs, that utilizes the inputs at the current timestamp and the previous timestamps information to produce outputs. In their architecture, they have layers of neurons that can capture data's long-term dependencies and can remember selective information for long periods. They have the power to decide what previous information they can retain and what information they can discard from memory. LSTMs use three gates such as input, forget, and output while adding and discarding information, and producing outputs.

Input/Update Gate: It helps the LSTM to decide which information is going to store in the cell state. First, the input gate uses a sigmoid activation function to decide the updates in the information and then a new vector is added to the cell state through the Tanh activation function. After that, LSTM updates the cell state with new vector values and forgets the information that information who was decided to forget. It can be calculated using the following mathematical formulas:

$$Input\ Gate = \ i_t = \emptyset(W_i \times [H_{t-1}, X_t] + b_i) \tag{3.22}$$

$$Intermidate\ Cell\ State = \ C_t' = Tanh(W_C \times [H_{t-1}, X_t] + b_C) \tag{3.23}$$

Forget Gate: It looks at the input and previously received hidden layer data to decide which information needs to delete from the cell state using the sigmoid activation function (i.e., 0 means delete it, and 1 means keeps it). This gate is mathematically formulated as:

$$Forget\ Gate = \ f_t = \emptyset\big(W_f \times [H_{t-1}, X_t] + b_f\big) \tag{3.24}$$

$$Cell\ State = \ C_t = \ f_t C_{t-1} + \ i_t C_t' \tag{3.25}$$

Output Gate: It executes the sigmoid activation function to decide which part of LSTM cells is going to output. After that, the output results are calculated using Tanh (i.e., a value between 1 and -1) and also decide which output information is passed to the next neuron. The mathematical formulation of this gate is as follows:

$$Output\ Gate = \ o_t = \emptyset(W_o \times [H_{t-1}, X_t] + b_o) \quad (3.26)$$

$$Hidden\ State = \ H_t = Tanh(C_t \times \ o_t) \quad (3.27)$$

In the above Equations 3.22-3.27 C represents the state of the cell, $\emptyset$ represent the sigmoid activation function, $Tanh$ is the hyperbolic Tangent activation function, X_t indicates the inputs at the time t, H_{t-1} are output, the weights are represented by W_i, W_C, W_f, $W_{o,}$ and biases are represented using b_i, b_C, b_f, b_o. The architecture of the LSTM can help to address the vanishing gradient problems in the RNNs. The hyper-parameter settings for LSTM architecture are given in Table 3.5.

Table 3.5: Hyper-parameters settings of DL methods.

Hyper-parameters settings	MLP	DNN	CNN	RNN	LSTM
Number of hidden layers	1	3	1	1	1
Hidden layer neurons	128	128	128	128	128
Activation function in hidden layers	ReLu	ReLu	ReLu	Tanh	Tanh
Activation function at the output layer	Sigmoid	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Learning rate (p)	0.1	0.1	0.1	0.1	0.1
Optimizer	adam	adam	adam	adam	adam
Epochs	20	20	20	20	20
Batch size	32	32	32	32	32

Algorithm 1: Training of deep learning methods for attack detection

Input: dataset: data-subsets, learning rate: learning rate (p), optimizer: adam, training rounds: epochs

Output: accuracy: acc

1: Set the parameters of the DL classifiers according to TABLE 2.

2: Initialize a parameter matrix using random values

3: for each training round, do

4: select a subset from the dataset to form a batch for the training

5: if the number of training rounds%100=0 then

6: return acc

7: end if

8: input batch and calculate predicted value y

9: Calculate the loss value between the actual value of the label y and the predicted value y

10: Calculate loss value for gradient descent direction with optimizer adam

11: Update the parameter matrix with the gradient descent direction and learning rate (p)

12: if training rounds reach the epochs, then

13: stop training of DL classifier

14: end if

15: end for

3.8 Summary

In this chapter, first, we give an overview of the proposed system for detecting botnet-based DDoS attacks in an SDN network. Second, explain the simulated dataset, virtual simulation setup and data collection, design of attack and normal traffic, data pre-processing, selection of optimal features. Finally we discuss the architectures of DL methods and their hyper-parameter settings.

CHAPTER 4

SIMULATION RESULTS FOR ATTACK DETECTION

This chapter provides a detailed analysis of the results achieved using DL methods. The performance of the methods is evaluated on the generated dataset and in real time by conducting a series of experiments.

4.1 Evaluation Metrics

Different metrics can be used to assess the performance and efficacy of machine learning- or deep learning-based intrusion detection techniques. These evaluation metrics are accuracy, Detection Rate (DR), precision, F1 score, and False Positive Rate (FPR). Furthermore, these evaluation metrics are computed using the confusion matrix of network anomaly classification. The confusion matrix has four different parameters as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). These parameters for any deep learning-based IDS can be explained as follows:

- True Positive (TP): indicates the number of attack records correctly identified as botnet-based DDoS attacks.
- True Negative (TN): indicates the number of normal records accurately detected as normal.
- False Positive (FP): indicates the inaccurately detected number of normal records as an attack traffic.

- False Negative (FN): indicates the number of attack records incorrectly identified as normal traffic.

The following mathematical equations can be used to compute the evaluation mentioned above metrics:

Accuracy: The accuracy evaluation metric is the proportion of the accurately identified number of attacks and normal records to the all-over records. This can be formulated as follows:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (4.1)$$

Detection Rate (DR): DR is the proportion of correctly identified accurate attack records. It is also called True Positive Rate (TPR) and recall or sensitivity. This metric is mathematically computed using the following formula:

$$Detection\ Rate\ (DR)\ or\ TPR\ or\ recall\ or\ sensitivity = \frac{TP}{TP+FN} \quad (4.2)$$

Precision: A proportion of correctly identified attack records are accurate attack records.

$$Precision = \frac{TP}{TP+FP} \quad (4.3)$$

F1 Score: This metric is computed through the harmonic means of recall and precision. This metric is considered more reliable than the accuracy when the DL methods are trained using imbalanced datasets. The following mathematical equation can be used to compute this metric:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision \times Recall} \quad (4.4)$$

False Positive Rate (FPR): The ratio of the incorrectly detected number of normal records as attack records. The following mathematical equation can be used to compute this metric:

$$FPR = \frac{FP}{FP+TN} \quad (4.5)$$

So, if the values of accuracy, detection rate, precision, and F1 score are high for any implemented DL/ML method the same way, the value of FPR is low. That method is considered best for botnet-based DDoS attack detection.

4.2 Feature Selection Results

The importance of each feature present in the dataset is calculated using features weighting and threshold tuning mechanism and then split the whole dataset into five different subsets. As discussed earlier, in this research, first, we used SVM to assign the weight to each feature, and then the weighted values were used by the tuning methods to determine an optimal threshold value for feature selection. The tuning method automatically compares the weight value of each feature with the threshold value. The features with equal and higher weight values than the threshold value are picked and placed in a subset of features. First, all features are placed in a set called Subset-1. Second, the tuning method returns an optimal value of "1.8", resulting in 43 features being selected with weights $\{\alpha \geq 1.80\}$ and placed in subset-2. For subset-3, the tuning method returns a "2.70" optimal value, so the features with weights $\{\alpha \geq 2.70\}$ are selected for this subset. Similarly, the features

with weights $\{\alpha \geq 3.15\}$ are picked and placed in subset-4. Lastly, a threshold value of $\{\alpha \geq 4.90\}$ is used to select features for subset-5. Table 4.1 shows the description of selected features for each subset. Figure 4.1 shows an example of selected features for subset-3.

Table 4.1: Number of selected features for each subset based on optimal threshold value.

Feature Sets	Threshold value	Number of Selected Feature
Subset-1	-	76
Subset-2	1.80	43
Subset-3	2.70	30
Subset-4	3.15	23
Subset-5	4.90	15

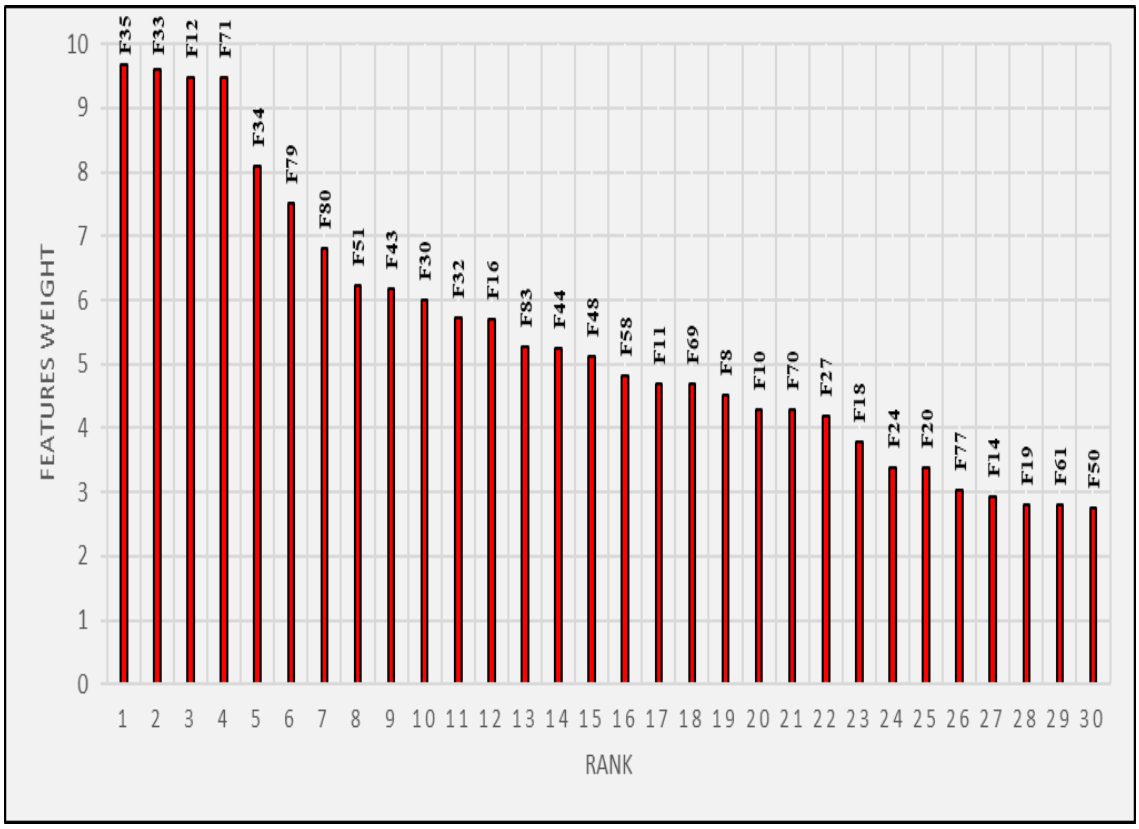


Figure 4.1: List of optimal features in subset-3.

4.3 Effectiveness of the DL Methods for Attack Detection

The effectiveness of the DL methods for detecting botnet-based DDoS attacks in an SDN environment is also verified in this research. We used identical structures with minimum values of the hyper-parameters for the DL methods to simulate and overserve their classification performances. As discussed in the above section, the confusion matrix is used to measure the performance evaluation metrics for each DL method for all subsets of features. The general structure of the confusion matrix is shown in Figure 4.2.

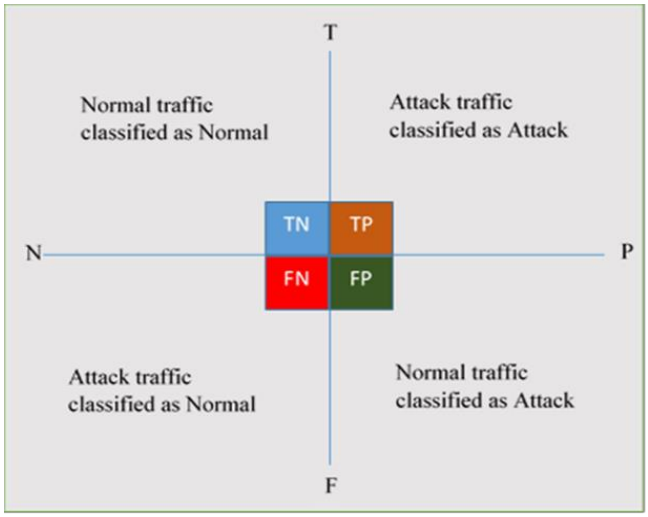


Figure 4.2. The general structure of a confusion matrix for anomaly detection.

4.4 Structural Performance of the DL Methods

Five DL approaches (MLP, DNN, CNN, RNN, and LSTM) are tested for classification performance to detect botnet-based DDoS attacks. The five feature subsets are divided into training and testing sets individually. The DL method uses the same neural network architecture, including learning rates, optimizers, batch

sizes, hidden layer counts, hidden layer neuron counts, hidden and output layer activation functions, and hidden and output layer number of hidden layers. It is observed that on the same subset of features with the same network structure, the methods produced variable results. The main aim is to adopt the same structure for all the methods to find the best method without increasing the method's complexities.

4.5 Results of Subset-1 Features

This section describes the results of the DL methods using a set of 76 features collected in an SDN environment.

4.5.1 Accuracy and Loss Trends of Methods

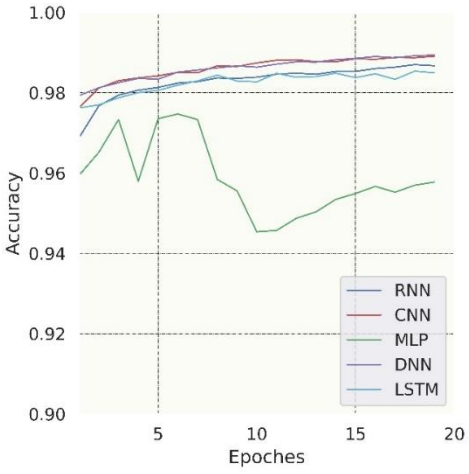
Figure 4.3 shows the change in training and validation accuracy and loss across the total number of epochs for several DL algorithms using 76 features. After 20 iterations, it is shown that the accuracy and loss trends for the training and validation sets suited to one another converge. The maximum training accuracies of DL methods, RNN, CNN, MLP, DNN, and LSTM using 76 features are 98.64%, 98.91%, 97.47%, 98.94%, and 98.54%, respectively. The maximum achieved validation accuracies of methods are 98.68%, 98.85%, 98.01%, 99.00%, and 98.94%, respectively.

It is also observed that the accuracy curves of CNN and DNN are more stable than the other methods, and the accuracy curve of MLP continuously fluctuates during each epoch. Since more, the MLP method has more noise in the

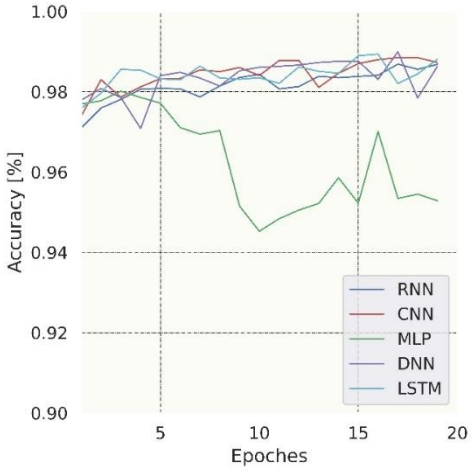
loss curves. Because dropout layers are used when the techniques are being trained, it is also seen that the validation loss methods have more minor relative losses than the training loss methods. The noise that was not injected during the validation period is made more apparent by the dropout layers. As a result, the error is decreased during the validation phase due to greater generalization to address overfitting issues.

4.5.2 Performance Evaluation Using Confusion Matrix

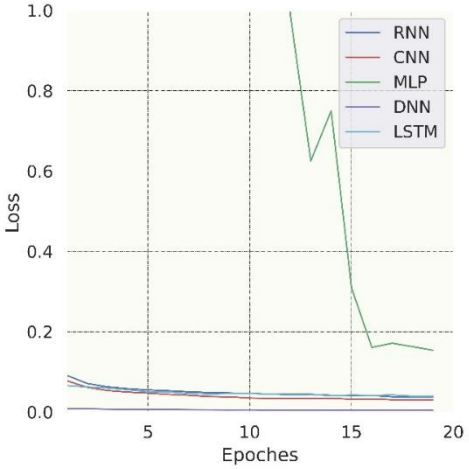
The performance of the DL methods using a set of 76 features is also evaluated using the confusion matrix. It summarizes the false and correct predictions. The confusion matrix for all five methods is given in Figure 4.4. It is observed that the normal data recognition ability of RNN is 98.17%, CNN is 98.18%, MLP is 92.17%, DNN is 97.71%, and LSTM is 98.56%. Similarly, RNN, CNN, MLP, DNN, and LSTM recognition abilities for attack data are 99.28%, 99.37%, 99.94%, 99.75%, and 99.37%, respectively. It is also noticed that the identifying attacks underreporting rate (i.e., False Negative Rate (FNR) is 2.16%, 2.16%, 11.11%, 2.77%, and 1.69%, respectively. MLP has a greater FNR of 11.11% compared to the other methods. High sensitivity is needed for the DL-based detection methods because the attack traffic can harm the SDN controllers. Among all the methods, MLP could perform better using the full feature set.



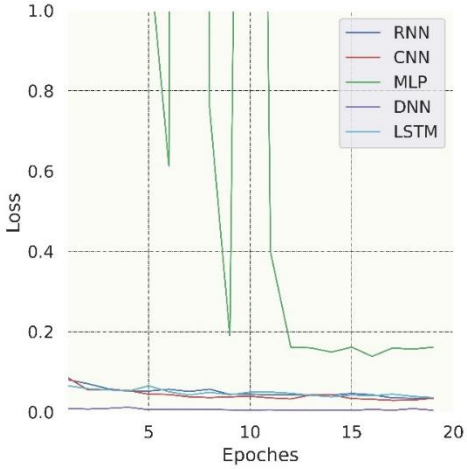
(a) Training Accuracy Curves



(b) Validation Accuracy Curves



(c) Training Loss Curves



(d) Validation Loss Curves

Figure 4.3: Accuracy and Loss Curves of DL methods using 76 features.

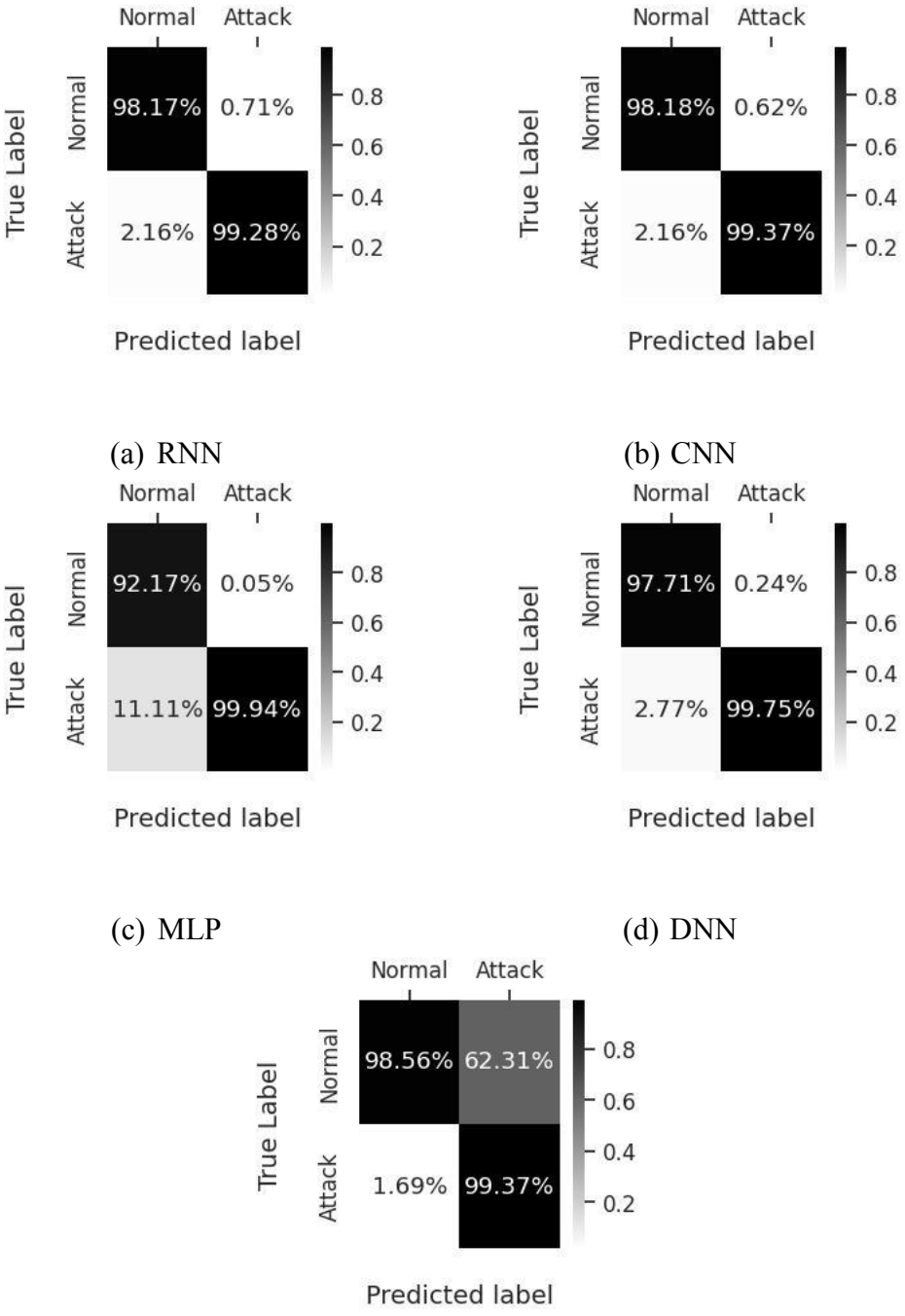


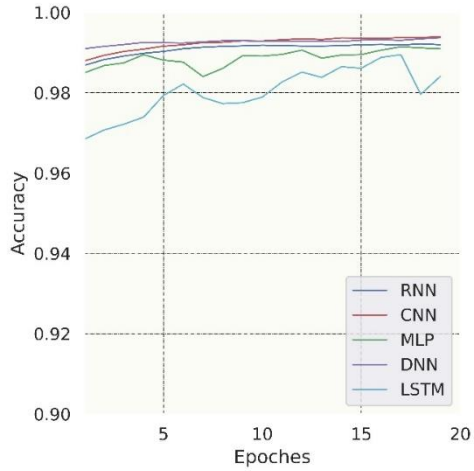
Figure 4.4: Confusion Matrix of RNN, CNN, MLP, DNN, and LSTM using 76 features.

4.6 Results of Subset-2 Features

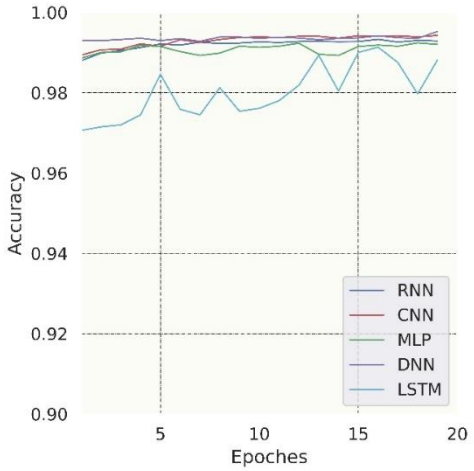
This section discusses the results of the DL methods using a set of 43 optimal features collected through the feature selection method.

4.6.1 Accuracy and Loss Trends of Methods

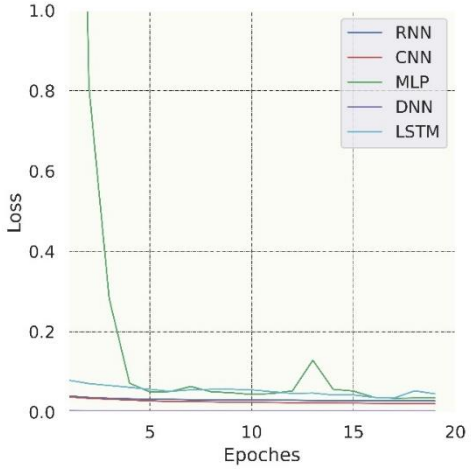
The classification performance of the DL methods using a set of 43 optimal features is evaluated through training accuracy, training loss, validation accuracy, and validation loss curves shown in Figure 4.5. Similarly, it is observed that after the 20 epochs, the training and validation curves start to converge, and we stopped the training and validation processes on 20 epochs. For the 43 features set, the maximum achieved accuracy by the RNN is 99.22%, CNN is 99.40%, MLP is 99.14%, DNN is 99.37%, and LSTM is 98.95%. Similarly, the validation accuracies of the methods are 99.33%, 99.43%, 99.24%, 99.53%, and 99%, respectively. For subset-2, the DNN method achieved maximum validation accuracy compared to others. The accuracy curves of the MLP are stable with 43 features compared to the 76 features set. But the LSTM method does not produce good results using the 43 feature set and has more fluctuation in the curve than other methods. Compared to other algorithms, the loss ratio of MLP is more excellent. Although MLP achieved better results using the 43 features set compared to the 76 features set due to its more excellent loss ratio, it is unsuitable for attack detection.



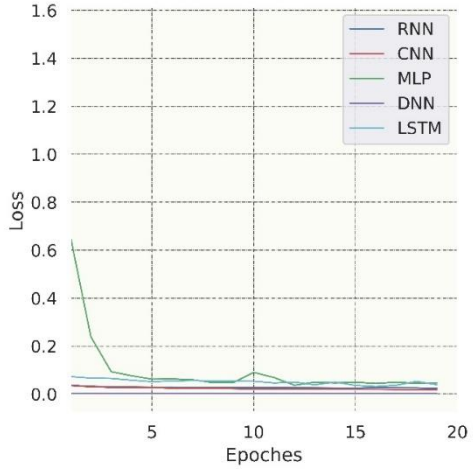
(a) Training Accuracy Curves



(b) Validation Accuracy Curves



(c) Training Loss Curves



(d) Validation Loss Curves

Figure 4.5: Accuracy and Loss Curves of DL methods using 43 features

4.6.2 Performance Evaluation Using Confusion Matrix

Similarly, as in the above section, the confusion matrix is used to evaluate the DL methods using 43 features. The confusion matrix of all methods using 43 features is shown in Figure 4.6. We can observe using Figure 4.6 that the recognition abilities for the normal data of methods RNN, CNN, MLP, DNN, and

LSTM are 99.32%, 99.58%, 99.52%, 99.57%, and 99.39%, respectively. The attack data recognition ability of RNN is 99.24%, CNN is 99.23%, MLP is 99.23%, DNN is 99.47%, and LSTM is 98.85%. The False Negative Rates (FNRs) of all the methods are 0.80%, 0.48%, 0.55%, 0.50%, and 0.69%, respectively. So, we can conclude that all the methods have over 99% recognition ability for normal and attack data using a set of 43 optimal features except LSTM. The FNR of CNN is lower than the other methods. Considering the sensitivity metric, we can conclude that CNN can detect the attack using 43 optimal features.

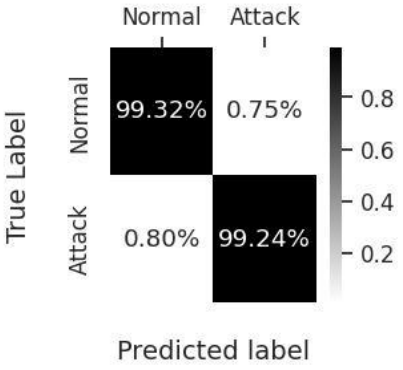
4.7 Results of Subset-3 Features

A set of 30 optimal features are selected and placed in subset-3. The classification performance of the DL techniques is assessed using 30 features in this section. The accuracy and loss curves and confusion matrix is used for evaluation.

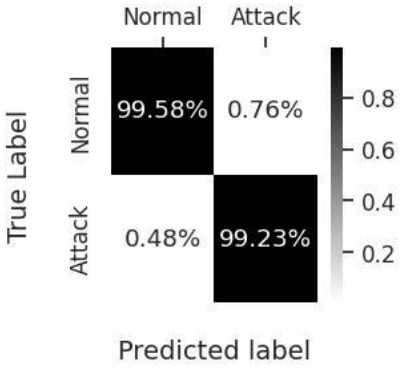
4.7.1 Accuracy and Loss Trends of Methods

The training accuracy, validation accuracy, training loss, and validation loss of all the methods using a set of 30 features are shown in Figure 4.7. During the training and validation phase, it is also observed that these curves start to converge on 20 epochs. So, we stop the further training and validation of the DL methods using 30 features at epoch number 20. The maximum achieved training accuracies by the DL methods (i.e., RNN, CNN, MLP, DNN, LSTM) with 30 features are 99.21%, 99.31%, 99.12%, 99.27%, 99%, and their validation accuracies are

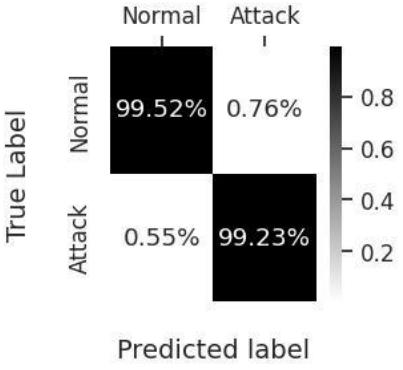
99.25%, 99.37%, 99.33%, 99.30%, and 99.16%, respectively. For the subset-3 features, it is observed that all the DL methods achieved over 99% training and



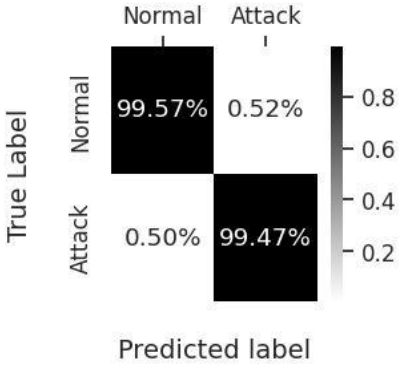
(a) RNN



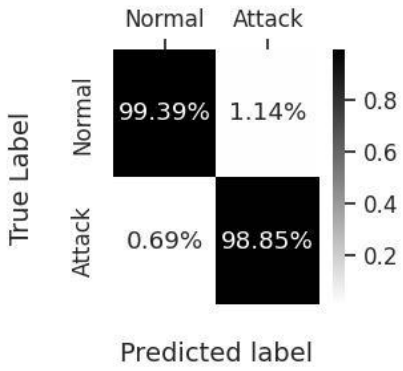
(b) CNN



(c) MLP



(d) DNN



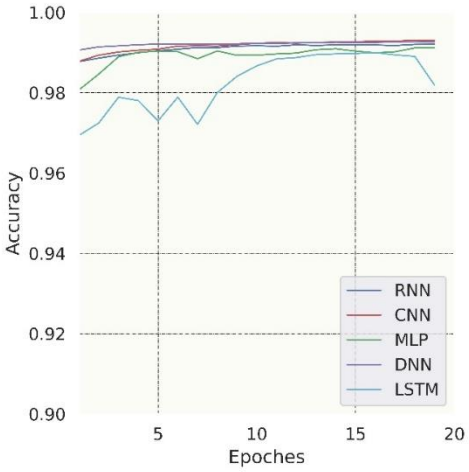
(e) LSTM

Figure 4.6: Confusion Matrix of RNN, CNN, MLP, DNN, and LSTM using 43 features.

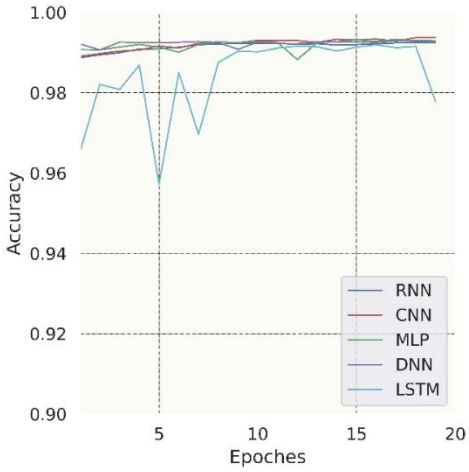
validation accuracies. CNN has become a top performer in training and validation accuracies using 30 features. The loss curve of the CNN is also more stable than the other methods. Although all the methods achieved maximum accuracies over 90%, MLP again has more fluctuation in its accuracy curves, indicating MLP is not suitable for attack detection in the adopted scenario. Furthermore, the DL methods achieved good accuracies, so we can conclude that the set with 30 features becomes an optimal set of features.

4.7.2 Performance Evaluation Using Confusion Matrix

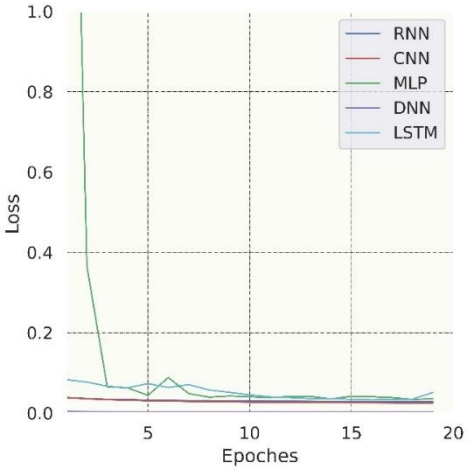
Similarly, as in the above two sections, the performance of the DL methods using a set of 30 features is analyzed through a confusion matrix. Figure 4.8 shows the confusion matrixes of all the methods using 30 features. For the subset-3 features, the recognition ability for normal data of RNN is 99.33%, CNN is 99.65%, MLP is 99.49%, DNN is 99.24%, and LSTM is 99.25%. It is also observed that the attack recognition ability of RNN is 99.14%, CNN is 99.04%, MLP is 99.05%, DNN is 99.29%, and LSTM is 99.05%. So, the recognition abilities of all the DL methods for the normal and attack data is over 99% using 30 features. Since more, the FNR of RNN is 0.78%, CNN is 0.40%, MLP is 0.59%, DNN is 0.89%, and LSTM is 0.87%. Here, we can conclude that the CNN method has more than 99% recognition abilities for normal and attack data and has lower FNR than other methods using 30 optimal features. Compared to previous methods employing 30 features, the CNN method can successfully identify the botnet-based DDoS attack.



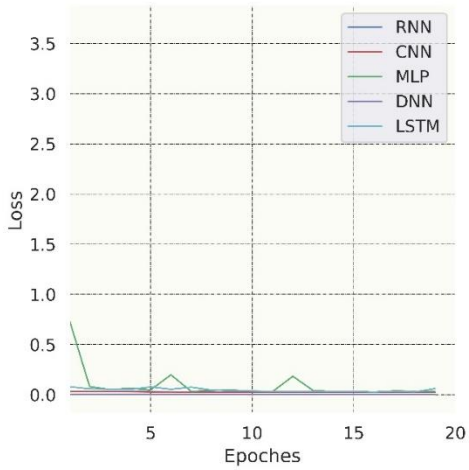
(a) Training Accuracy Curves



(b) Validation Accuracy Curves

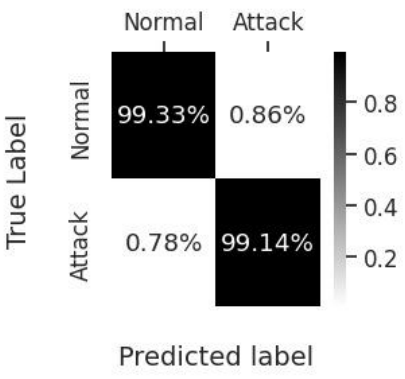


(c) Training Loss Curves

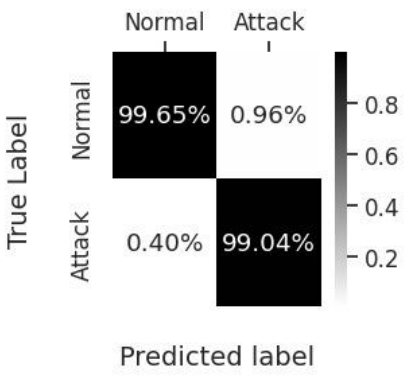


(d) Validation Loss Curves

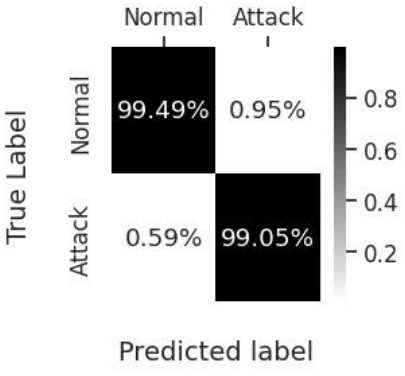
Figure 4.7: Accuracy and Loss Curves of DL methods using 30 features



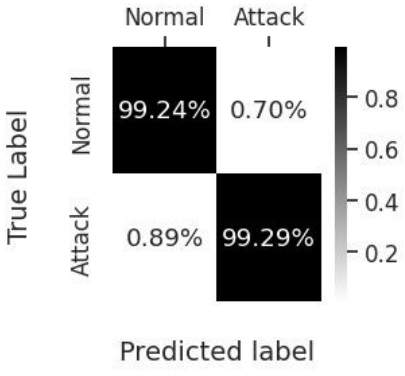
(a) RNN



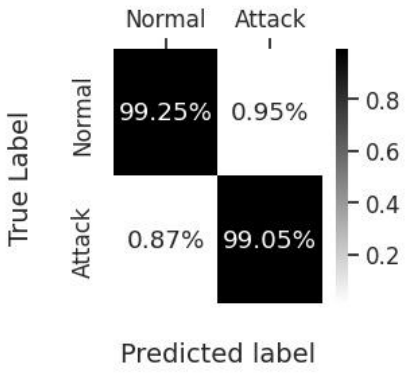
(b) CNN



(c) MLP



(d) DNN



(e) LSTM

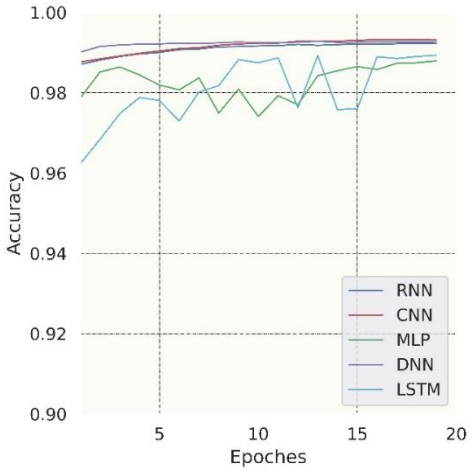
Figure 4.8: Confusion Matrix of RNN, CNN, MLP, DNN, and LSTM using 30 features.

4.8 Results of Subset-4 Features

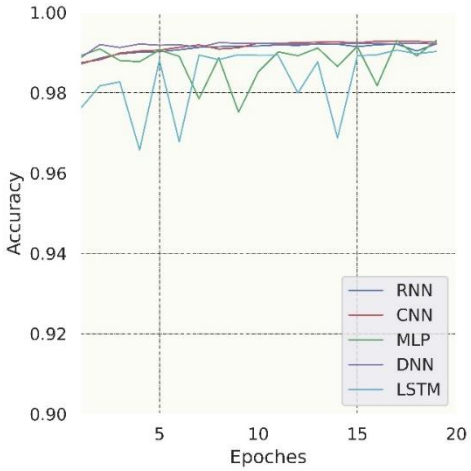
A set of 23 features is used in this section to assess the classification performance of the DL approaches. Their performance is analyzed using accuracy, loss curves, and confusion matrix.

4.8.1 Accuracy and Loss Trends of Methods

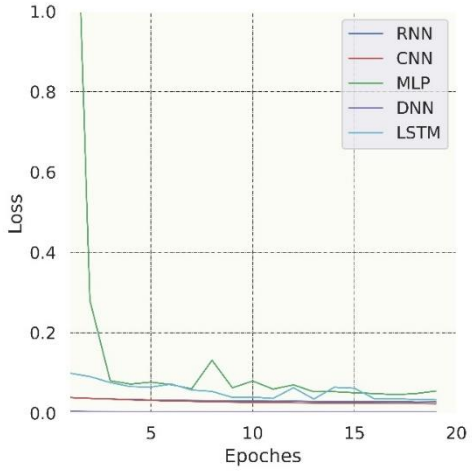
Here, the DL methods are trained and validated using a set of 23 optimal features, and their training and validation accuracy curves and training and validation loss curves are depicted in Figure 4.9. The training and validation processes of all the methods are stopped on epoch 20 because, at 20 epochs, all the methods start to converge. It is observed in Figure 4.9 that the maximum achieved training accuracy by the RNN is 99.21%, CNN is 99.34%, MLP is 99.05%, DNN is 99.28%, and LSTM is 98.50%. Similarly, the maximum achieved validation accuracy by the RNN is 99.22%, CNN is 99.29%, MLP is 99.31%, DNN is 99.25%, and LSTM is 99.06%. The maximum training and validation accuracies achieved by all the methods using 23 features is over 99%, except for LSTM. So, among all the DL methods, the CNN method achieved the highest training accuracy using 23 features, and the validation accuracy is also reasonable compared to RNN, DNN, and LSTM. Here, we also observed that the validation accuracies of the DL methods become lower than training accuracies, indicating that some important features may be dropped in subset-4. Furthermore, the loss curves of the CNN and DNN are more stable than other methods.



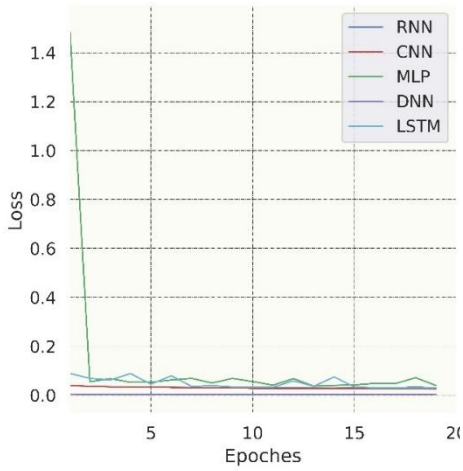
(a) Training Accuracy Curves



(b) Validation Accuracy Curves



(c) Training Loss Curves



(d) Validation Loss Curves

Figure 4.9: Accuracy and Loss Curves of DL methods using 23 features.

In the case of MLP, it again fluctuates its accuracy curves and has the highest loss ratio compared to other methods. So, the MLP is not a suitable classifier for attack detection for the adopted scenarios of this research.

4.8.2 Performance Evaluation Using Confusion Matrix

The DL methods' performance through the confusion matrix is also observed using 23 features. The confusion matrix of all the methods using a set of 23 features is depicted in Figure 4.10. It is observed that the normal data recognition ability using 23 features of RNN is 99.18%, CNN is 99.43%, MLP is 99.33%, DNN is 99.18%, and LSTM is 99.16%. Similarly, the attack data recognition ability of RNN is 99.26%, CNN is 99.04%, MLP is 99.20%, DNN is 99.24%, and LSTM is 98.94%. All the methods' normal and attack data recognition abilities are more than 99% instead of LSTM, which shows 98.94% recognition ability for attack data. Through the confusion matrix, we also observed that the FNR of RNN is 0.94%, CNN is 0.64%, MLP is 0.78%, DNN is 0.94%, and LSTM is 0.97%. By analyzing the performance of all the DL methods using a set of 23 features, we can conclude that CNN is the best method for the detection of botnet-based DDoS attacks in the adopted scenario because it achieved over 99% recognition abilities for both normal and attack data and also has lower False Negative Rate (FNR) compared to other methods.

4.9 Results of Subset-5 Features

In this section, the classification performance of the DL methods is evaluated using a set of 15 features. Their performance is analyzed using accuracy, loss curves, and confusion matrix.

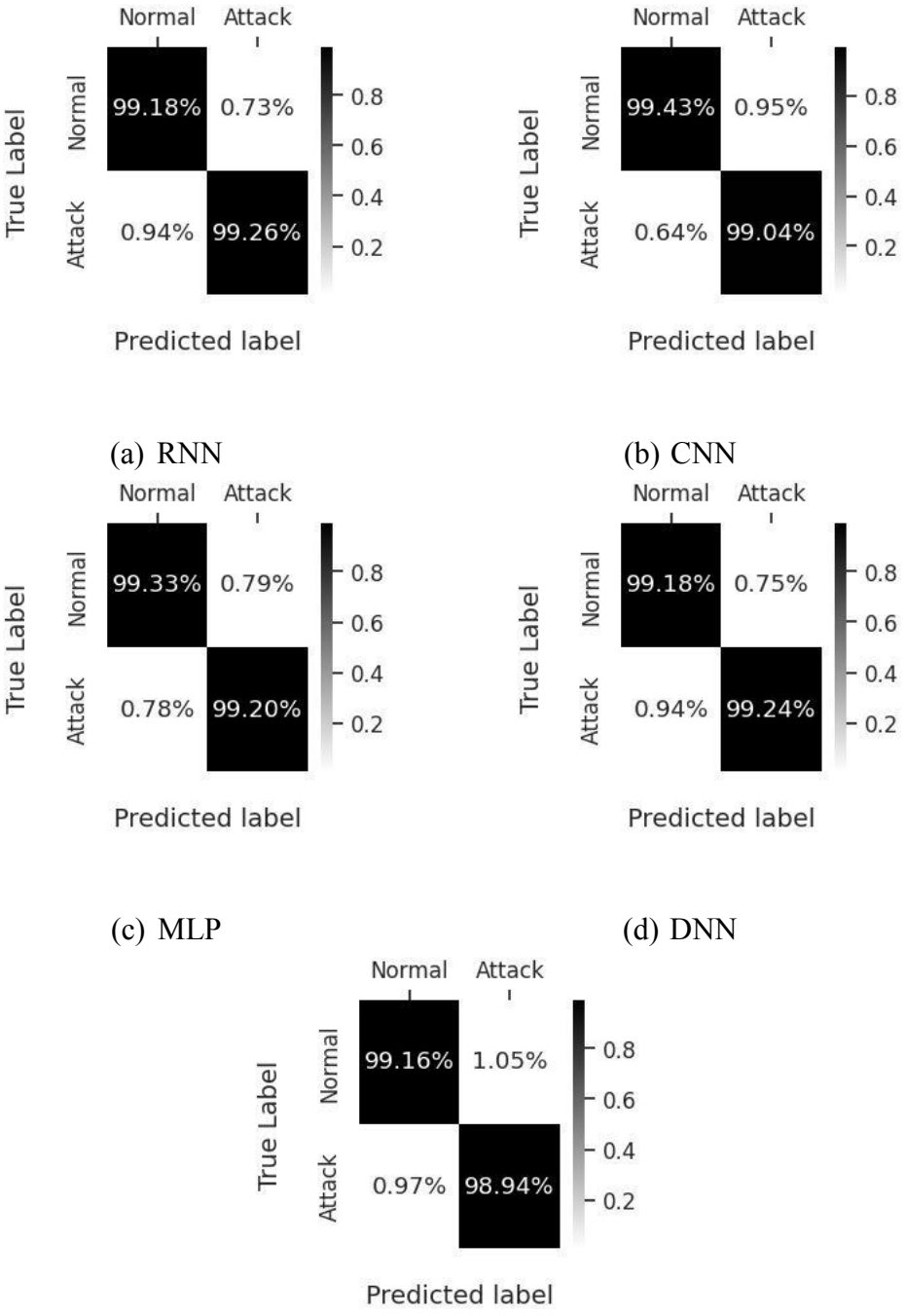


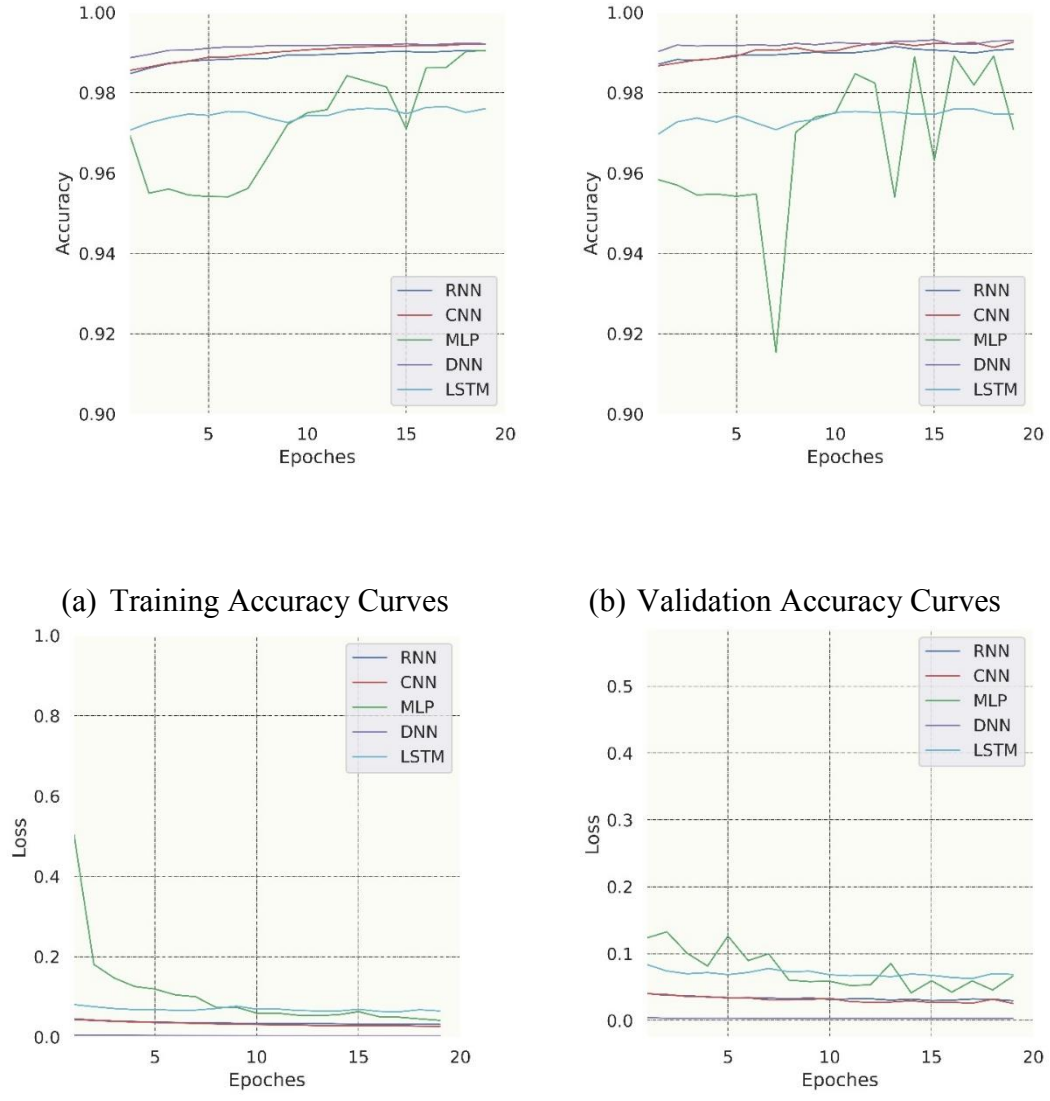
Figure 4.10: Confusion Matrix of RNN, CNN, MLP, DNN, and LSTM using 23 features.

4.9.1 Accuracy and Loss Trends of Methods

This section discusses the performance of the DL methods in terms of accuracy and loss using 15 optimal features. The curves of all the methods related to training accuracy, validation accuracy, training loss, and validation loss are given in Figure 4.11.

The training and validation curves of all the methods using 15 features started to converge in 20 epochs, so we stooped further training and validation of the methods. The maximum achieved accuracies by the DL methods (i.e., RNN, CNN, MLP, DNN, and LSTM) are 99.06%, 99.21%, 99.06%, 99.24%, and 99.63%, respectively. The maximum achieved validation accuracy by the RNN is 99.15%, CNN is 99.26%, MLP is 98.91, DNN is 99.32%, and LSTM is 97.60%. Here, we observed that the training accuracies of all the methods are over 99%. But, during the validation phase, some methods drop their accuracies. For example, the drop in the validation accuracy of CNN is 0.05%, MLP is 0.15%, DNN is 0.08%, and LSTM is 2.03%. So, LSTM has more drop in the validation accuracies than the other methods. The decrease in the validation accuracies of the methods resulted in some important features being discarded from the subset-5. The training and testing accuracy of the CNN is over 99% using 15 features. In contrast, the training and validation curves of the MLP are not stable, and these curves are continuously variating with the change in epochs. The loss ratio of MLP is also high than the other methods. Similarly, due to continuous variations in accuracy curves, the LSTM method is also unsuitable for detecting botnet-based DDoS attacks with 15

features. RNN, CNN, and DNN are influential classifies in detecting botnet-based DDoS attacks using 15 features in the adopted scenario.



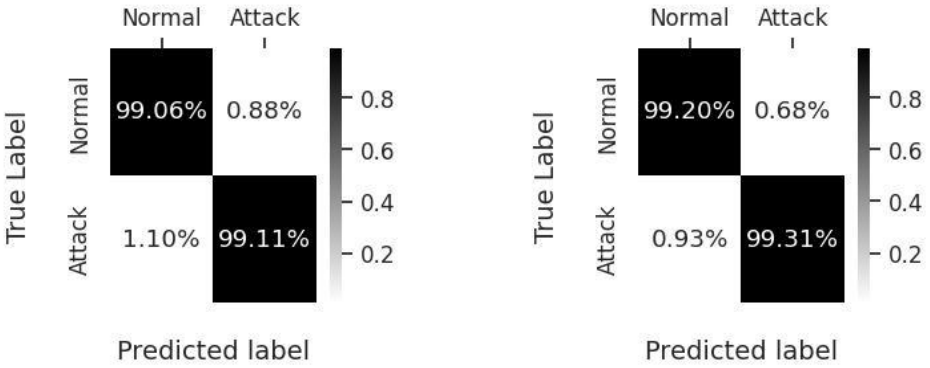
(a) Training Accuracy Curves (b) Validation Accuracy Curves

(c) Training Loss Curves (d) Validation Loss Curves

Figure 4.11: Accuracy and Loss Curves of DL methods using 15 features.

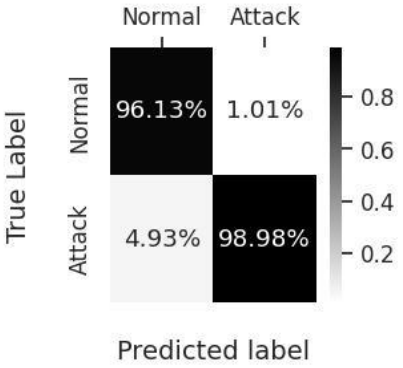
4.9.2 Performance Evaluation Using Confusion Matrix

The confusion matrix is also used to analyze the performance of the DL methods using 15 features. Figure 4.12 shows the confusion matrix of all the methods using 15 features. The recognition ability for the normal data of RNN is 99.06%, CNN is 99.20%, MLP is 96.13%, DNN is 99.41%, and LSTM is 97.81%. The attack data recognition ability of RNN is 99.11%, CNN is 99.31%, MLP is 98.98%, DNN is 99.16%, and LSTM is 97.36%. Furthermore, the FNR for RNN is 1.10%, CNN is 0.93%, MLP is 4.93%, DNN is 0.68%, and LSTM is 2.55%. It is observed that the normal and attack data recognition abilities of RNN, CNN, and DNN are over 99%, but MLP and LSTM methods have lower than 99%. The FNR of CNN and DNN is lower than 1%, indicating that CNN and DNN are effective methods for attack detection. In contrast, MLP has the highest FNR, which is 4.93%. The FNR of MLP and LSTM using 15 features is also high compared to other subsets of features, indicating that this subset of features is unsuitable for detecting botnet-based DDoS attacks in the adopted scenarios. Furthermore, all the methods produce poor results using this subset of features than others. It means we can reduce the number of features at a certain level.

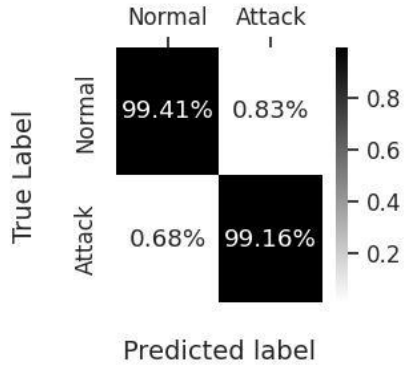


(a) RNN

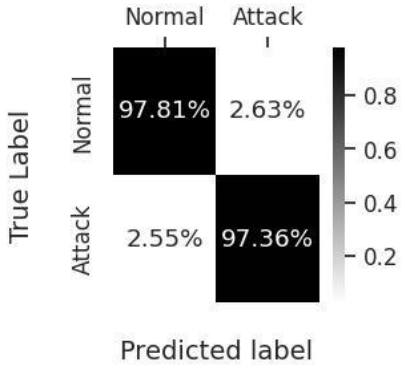
(b) CNN



(c) MLP



(d) DNN



(e) LSTM

Figure 4.12: Confusion Matrix of RNN, CNN, MLP, DNN, and LSTM using 15 features.

4.10 Overall Analysis of the Performance of Methods

This section discusses the overall performance in terms of training time, accuracy, detection rate, precision, F1 score, True Positive Rate (TPR), and False Positive Rate (FPR) of the methods for detecting botnet-based DDoS attacks in SDN. The performance comparison results of all the methods are shown in Table 4.2. The overall classification performance of all the methods is improved by reducing the number of features at a certain level. Table 4.2 shows that CNN and DNN have a trend to achieve maximum accuracy of 99.43% and 99.53% using subset-2 features, respectively.

The CNN method achieved the highest detection rate of 99.60% using subset-3 features. The CNN also performed the second highest detection rate of 99.51% using subset-2 features. The training time of CNN is 185.92 seconds using subset-2 features and 181.88 seconds using subset-3 features. The training time of the CNN is reduced by 4.04 seconds while using subset-3 features. The decreased accuracy and detection rate and increased training time while using subset-4 and subset-5 features by the CNN indicate that some important features are removed by making these two subsets. In contrast, LSTM has achieved minimum accuracy of 97.60% using subset-5 features. MLP has a minimum detection rate of 89.99% using subset-1 features, for example, by considering the results of all the methods using subset-3 features. The accuracy of the CNN is 0.12%, 0.04%, 0.07%, and 0.21% is higher than the other four classifiers (i.e., RNN MLP, DNN, and LSTM), respectively. Also, the detection rate of CNN is 0.39% higher than that of RNN, 0.19% higher than that of MLP, 0.49% higher than that of DNN, and 0.47% higher

than that of LSTM. In addition, the training time of the CNN is 39.51 seconds more heightened than that of MLP and 27.75 seconds more elevated than that of DNN, while 81.29 seconds lower than that of RNN and 29.85 seconds lower than LSTM. Furthermore, CNN has increased in accuracy by nearly 0.04% to 0.21%, and its detection rate is improved from 0.19% to 0.49%. Since all the methods are trained in an offline way and are not frequently updated, so, while ensuring the best accuracy and detection rate, a slightly higher training time could be accepted for any ML/DL method. Here, we can summarize that the ML/DL-based detection methods are trying to achieve maximum accuracy or detection rate with reasonable training time. It is observed that the CNN method achieved a maximum detection rate of 99.60% while using subset-3 features compared to other methods and other subsets. The accuracy of CNN is 99.29% using subset-4 features and 99.26% using subset-5 features. Similarly, the detection rate of CNN is 99.35% using subset-4 features and 99.07% using subset-5 features. Although these subsets consist of minimum features compared to subset-3, the training time of the CNN using subset-3 is 3.98 seconds and 20.53 seconds lower than that of subset-4 and subset-5 features. The detection rate of CNN is 0.25% and 0.53% higher while using the other two subsets.

Table 4.2: Comparison of performance results with all five sets of features.

Feature Sets	DL Models	Computational Time (Seconds)	Maximum Accuracy (%)	Detection Rate (%)
All feature Set-1	RNN	264.43	98.68	97.86
	CNN	202.39	98.85	97.87
	MLP	142.94	98.01	89.99
	DNN	202.39	99.00	97.29
	LSTM	323.46	98.94	98.32
Subset-2	RNN	256.78	99.33	99.20
	CNN	185.92	99.43	99.51
	MLP	117.76	99.24	99.45
	DNN	185.92	99.53	99.49
	LSTM	223.06	99.14	99.31
Subset-3	RNN	263.17	99.25	99.21
	CNN	181.88	99.37	99.60
	MLP	142.37	99.33	99.41
	DNN	154.13	99.30	99.11
	LSTM	211.73	99.16	99.13
Subset-4	RNN	278.09	99.22	99.06
	CNN	185.86	99.29	99.35
	MLP	119.37	99.31	99.21
	DNN	155.66	99.25	99.06
	LSTM	231.18	99.06	99.02
Subset-5	RNN	259.45	99.15	98.90
	CNN	202.41	99.26	99.07
	MLP	117.40	98.91	95.24
	DNN	202.42	99.32	99.31

	LSTM	203.40	97.60	97.44

Furthermore, the performance of all the methods using all five subsets of the features in terms of precision is shown in Figure 4.13. For subset-1, the precision of RNN is 99.29%, CNN is 99.37%, MLP is 99.94%, DNN is 99.75%, and LSTM is 99.37%. The precision of all the methods (i.e., RNN, CNN, MLP, DNN, and LSTM) using subset-2 is 99.24%, 99.23%, 99.23%, 99.47%, and 98.85%, respectively. For subset-3, the precision of RNN is 99.13%, CNN is 99.03%, MLP is 99.05%, DNN is 99.29%, and LSTM is 99.05%. Similarly, the precision for subset-4 features is 99.26%, 99.04%, 99.20%, 99.24%, and 98.94%, respectively, for all methods. Lastly, the precision for subset-5 features is 99.11%, 99.31%, 98.98%, 99.16%, and 97.36%, respectively. Based on Figure 4.13, we can analyze the highest precision percentage of MLP using subset-1 features compared to other methods and subsets. In contrast, the lowest precision rate is 97.36 of LSTM using features subset-5. With the reduction in the number of features, the precision percentage of the MLP decreases from 99.94 to 98.98. Similarly, for LSTM, it also reduced from 99.37% to 97.36%. The precision percentage ratio of RNN, CNN, and DNN methods remains over 99% for all subsets, which indicates that these methods' precision performance is more stable than MLP and LSTM. Although MLP has the highest precision percentage compared to other ways, it may not be recommended for attack detection because of the continuous decrease. The DNN is more stable in terms of precision compared to RNN and CNN.

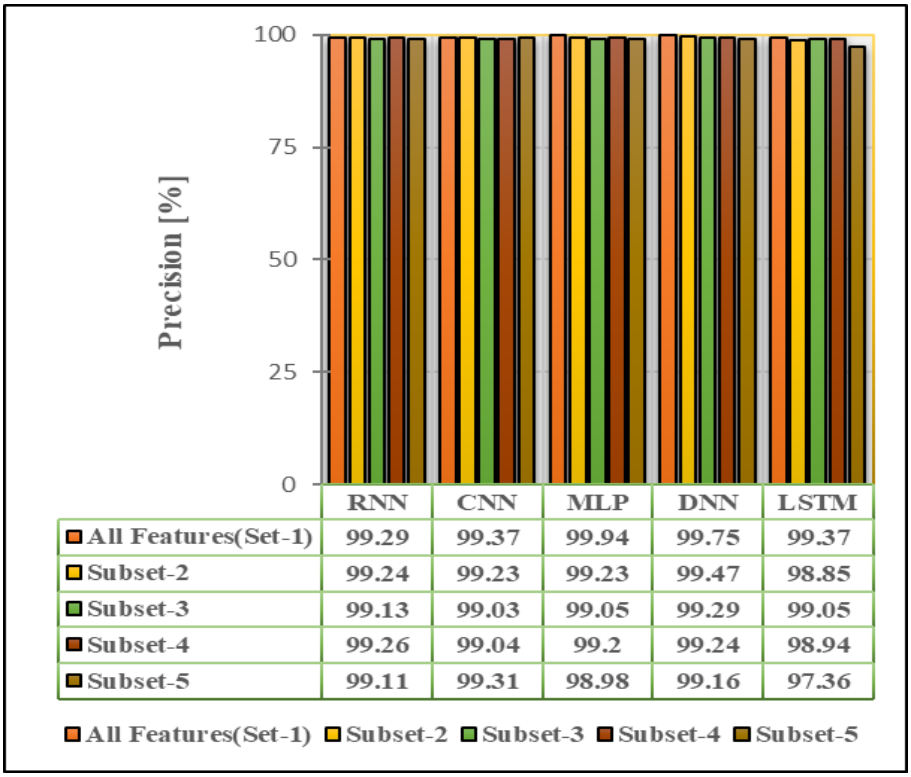
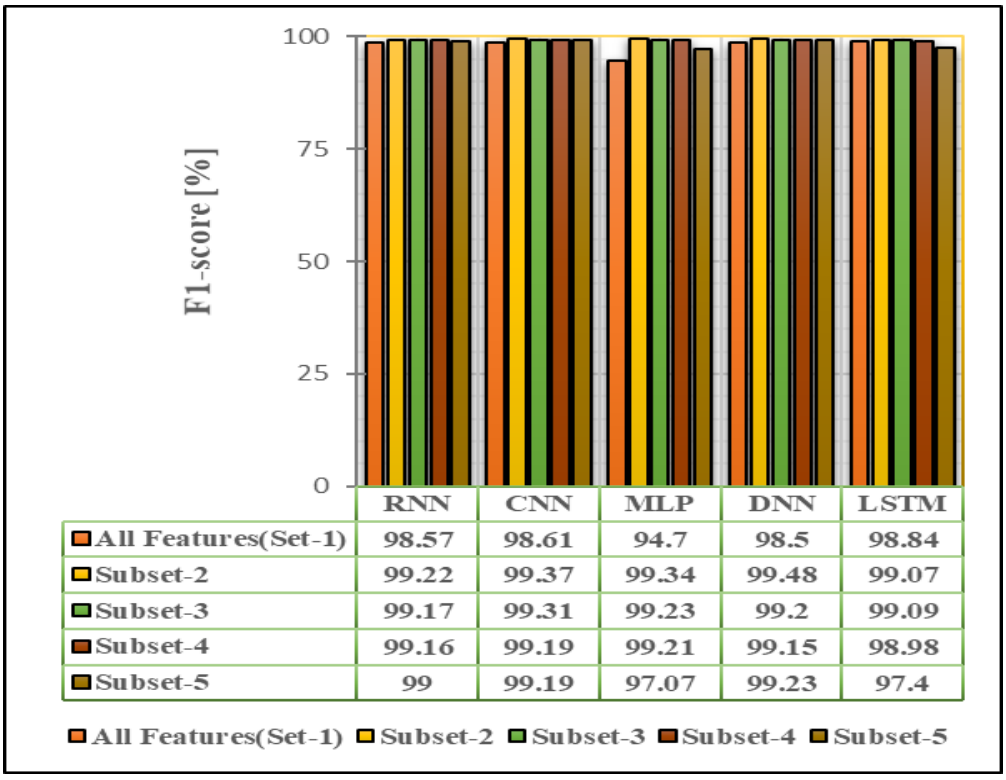


Figure 4.13: Comparison of performance results between all methods in terms of precision.

The classification performance of all the methods is evaluated using the F1 score parameter using all subsets. Figure 4.14 shows the performance of all the methods regarding the F1 score. The F1 score of RNN is 98.57%, CNN is 98.61%, MLP is 94.70%, DNN is 98.50%, and LSTM is 98.84% using subset-1 features. For subset-2, the F1 score of all the methods (i.e., RNN, CNN, MLP, DNN, and LSTM) is 99.22%, 99.37%, 99.34%, 99.48%, and 99.07%, respectively. The F1 score for subset-3 features of RNN is 99.17%, CNN is 99.31%, MLP is 99.23%, DNN is 99.20%, and LSTM is 99.09%. Similarly, for subset-4 features, RNN is 99.16%, CNN is 99.19%, MLP is 99.21%, DNN is 99.15%, and LSTM is 98.98%. Lastly, the F1 score using subset-5 features of RNN is 99%, CNN is 99.19%, MLP is 97.07%, DNN is 99.23%, and LSTM is 97.4%. The DNN method achieved the highest F1 score of 99.48% using subset-2 features, while MLP has the lowest of

94.70% using subset-1. The F1 score of RNN lies in the range of 98.57%-99.21%, CNN of 98.61%-99.37%, MLP of 94.70%- 99.34%, DNN of 98.50%-99.48%, and LSTM of 97.40% to 99.07% using different subsets. Here, the F1 score of MLP and LSTM is continuously variable, which indicates these methods are unsuitable for attack detection in the adopted scenarios. RNN, CNN, and DNN methods achieved over 99% F1 score using all subsets of features except subset-1, which indicates that these methods are effective in attack detection.



	RNN	CNN	MLP	DNN	LSTM
All Features(Set-1)	98.57	98.61	94.7	98.5	98.84
Subset-2	99.22	99.37	99.34	99.48	99.07
Subset-3	99.17	99.31	99.23	99.2	99.09
Subset-4	99.16	99.19	99.21	99.15	98.98
Subset-5	99	99.19	97.07	99.23	97.4

Figure 4.14: Comparison of performance results between all methods in terms of F1 score.

The True Positive Rate (TPR) evaluation metric is also used for the performance evaluation of all the methods using five different subsets of features. The TPR of the methods using all subsets of features is shown in Figure 4.15. It is observed in Figure 4.15, that the TPR of RNN is 97.86% CNN is 97.87%, MLP is

89.99%, DNN is 97.29%, and LSTM is 98.32% using subset-1 features. For subset-2, the TPR of RNN is 99.20%, CNN is 99.51%, MLP is 99.45%, DNN is 99.49%, and LSTM is 99.31%. Similarly, the TPR of all methods (i.e., RNN, CNN, MLP, DNN, and LSTM) using subset-3 features is 99.21%, 99.60%, 99.41%, 99.11%, and 99.13%, respectively. For subset-4, the TPR of RNN is 99.06%, CNN is 99.35%, MLP is 99.21%, DNN is 99.06%, and LSTM is 99.02%. Lastly, the TPR of RNN is 98.90%, CNN is 99.07%, MLP is 95.24%, DNN is 99.31%, and LSTM is 97.44% using subset-5 features. The TPR of RNN lies in the range of 97.86%-99.21%, and it has the highest TPR using subset-3 features and the lowest using subset-1. The range of TPR for CNN is 97.87%-99.60, with the highest using subset-3 and the lowest using subset-1. The range of TPR for MLP is 89.99%-99.45%, with the highest using subset-2 and the lowest using subset-1. Similarly, the range of TPR for DNN is 97.59%-99.49%, with the highest using subset-2 and the lowest using subset-1. The range of TPR for LSTM is 97.44%-99.31%, with the highest using subset-2 and the lowest using subset-5. CNN and DNN have over 99% TPR for all subsets except subset-1, while RNN, MLP, and LSTM have over 99% for 2,3 and 4 subsets. The ratio of TPR for CNN and DNN is more stable than other methods. Among all the methods, CNN achieved the highest TPR of 99.60%. We can conclude that CNN becomes an effective method for detecting attacks than other methods by evaluating the performance of the methods using this evaluation parameter.

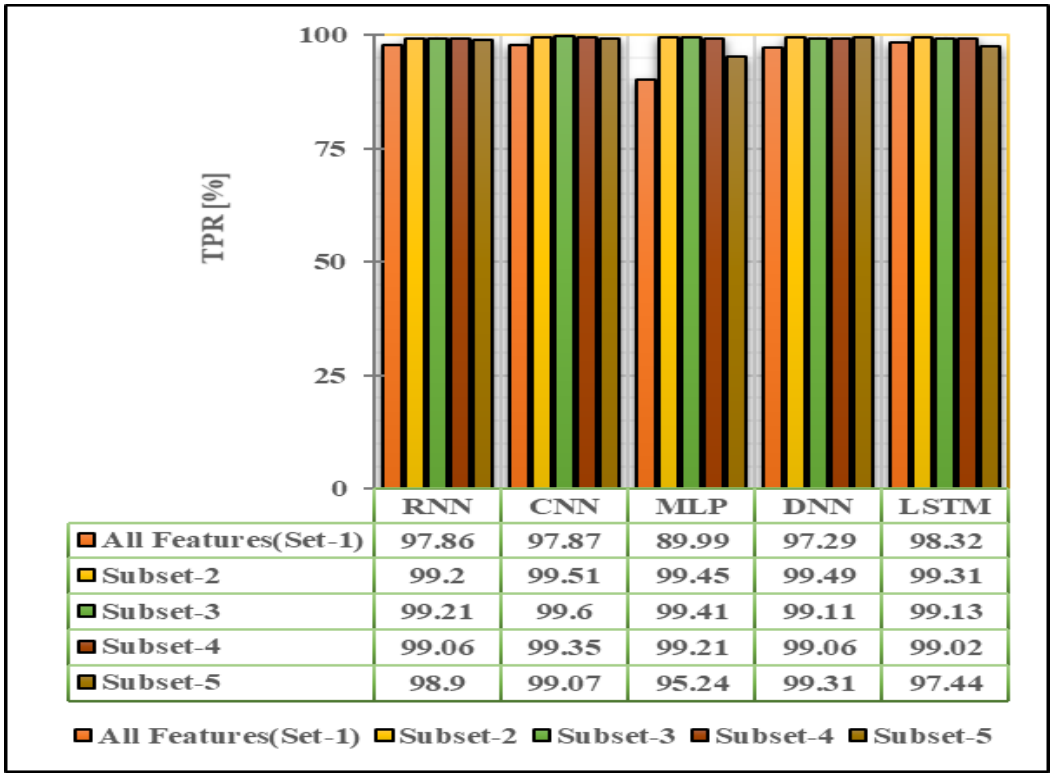
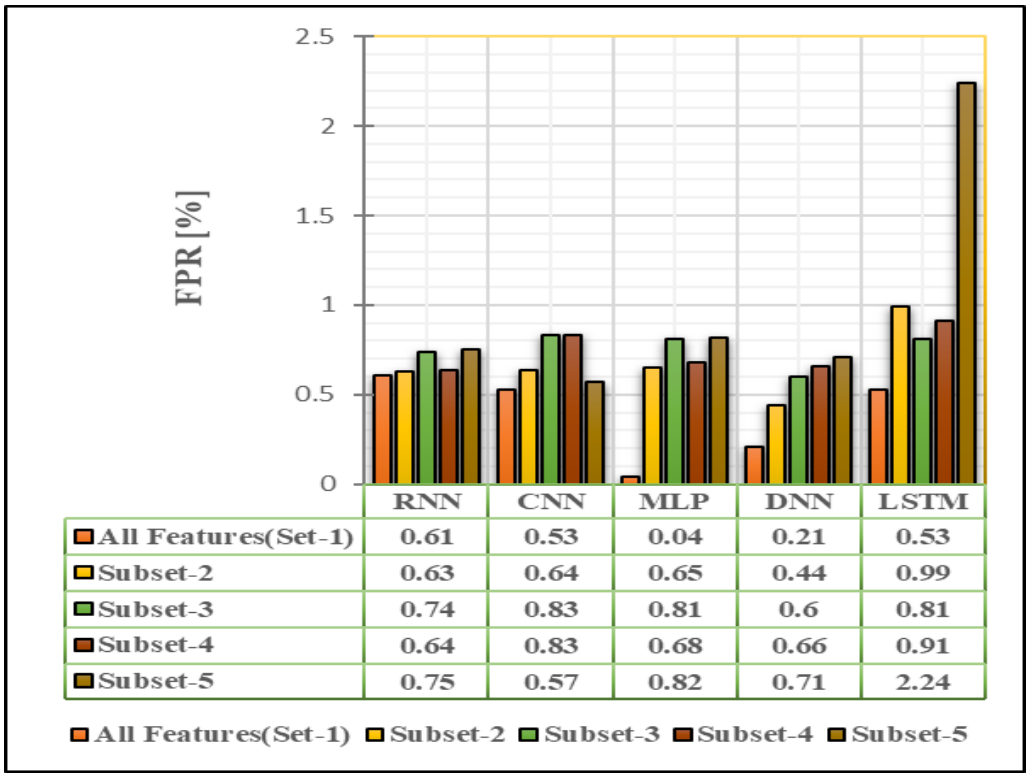


Figure 4.15: Comparison of performance results between all methods in terms of True Positive Rate (TPR).

Lastly, the False Positive Rate (FPR) is also used to analyze the performance of all the DL methods using five different subsets of features. The FPR of all the methods using different subsets of features is shown in Figure 4.16. Here, the ML/ DL methods with the lowest FPR are considered effective for attack detection. For subset-1, the FPR of RNN is 0.61%, CNN is 0.53%, MLP is 0.04%, DNN is 0.21%, and LSTM is 0.53%. Similarly, for subset-2, the FPR of RNN is 0.63%, CNN is 0.64%, MLP is 0.65%, DNN is 0.44%, and LSTM is 0.99%. the FPR of RNN is 0.74%, CNN is 0.83%, MLP is 0.81%, DNN is 0.6%, and LSTM is 0.81% using subset-3 features. For subset-4, the FPR of RNN is 0.64%, CNN is 0.83%, MLP is 0.68%, DNN is 0.66%, and LSTM is 0.91. Lastly, the FPR of RNN is 0.75%, CNN is 0.57%, MLP is 0.82%, DNN is 0.71%, and LSTM is 2.24%. The

range of the FPR for RNN is 0.61%-0.75%, CNN is 0.53%-0.83, MLP is 0.04%-0.82%, DNN is 0.21%-0.71%, and LSTM is 0.53%-2.24%. RNN has the lowest FPR using subset-1 and the highest using subset-5, CNN has the lowest using subset-1 and the highest using 3 and 4 subsets features, MLP has the lowest using subset-1 and the highest using subset-5, DNN has the lowest using subset-1 and the highest using subset-5, and LSTM has the lowest using subset-1 and the highest using subset-5. By analyzing the overall performance of all the methods in terms of FPR, we can conclude that the MLP has the lowest of 0.04% using subset-1 features, and LSTM has the highest of 2.24% using subset-5 features. RNN, CNN, and DNN seem more stable for all the subsets than MLP and LSTM.



	RNN	CNN	MLP	DNN	LSTM
All Features(Set-1)	0.61	0.53	0.04	0.21	0.53
Subset-2	0.63	0.64	0.65	0.44	0.99
Subset-3	0.74	0.83	0.81	0.6	0.81
Subset-4	0.64	0.83	0.68	0.66	0.91
Subset-5	0.75	0.57	0.82	0.71	2.24

Figure 4.16: Comparison of performance results between all algorithms in terms of False Positive Rate (FPR).

In summary, based on the above-discussed and evaluating the classification performance results in terms of accuracy and loss curves, confusion matrix, training time, accuracy, detection rate, precision, F1 score, TPR, and FPR of all the methods using five different subsets of features, we can conclude that the all the methods produced good results using subset-2 and subset-3 features. Subset-1 has many features, and some of the features need to be participating in improving the classification performance. At the same time, subset-4 and subset-5 may be lost some important features, which causes a decrease in the performance of the classifiers. Although the performance of all the methods using subset-2 and subset-3 is good, the number of features in subset-2 is more than in subset-3. After the overall analysis, we select and recommend the subset-3 features for detecting botnet-based DDoS attacks in the SDN environment. We also observed that the CNN method became a top performer with more effective and stable performance than other classifiers in the adopted network scenarios of this research. Thus, there are some advantages of this outcome: (i) the training sets are collected easily without knowing the details of traffic flows; (ii) with optimal features, the training phase becomes simple; (iii) the resource consumption and complexity of the methods is reduced due to training set with optimal features.

4.11 Implementation and Evaluation in Real Testbed

4.11.1 Performance of the DL Methods

This section discusses the performance of the DL methods on the real testbed. As we overserved and concluded in the above sections, the DL methods

achieved the best results using a subset of 30 features. So, to evaluate and validate the performance of the methods on the real testbed, we have selected the methods trained using subset-3 features of the self-generated dataset. The same network topology shown in Figure 3.2 is used for the real testbed analysis. To generate and collect the attack and normal flow statistics, the same process explained in section 3.8 has been followed. All DL methods trained using a subset of 30 features are individually implemented in the controller. The implemented methods inside the SDN controller classify the incoming flows with "0" or "1" labels (i.e., as we used a binary classes data set in our experiments, so, the DL methods have only two options, and each method labels the attack flows with "1" and normal flows with label "0"). Furthermore, we used "50" consecutive decisions made by any method under two network states (attack flow or normal flow) to validate the overall performance of the methods during real-time network traffic. Figure 4.17 shows the rate of correct detections of each method during real-time traffic. We observed that the ratio of output prediction for normal flows is superior to the attack flows by all the methods. The prediction rate of RNN for normal flows is 93%, CNN is 99%, MLP is 87%, DNN is 95%, and LSTM is 92%. Similarly, the prediction rate of RNN for attack flows is 87%, CNN is 97%, MLP is 85%, DNN is 93%, and LSTM is 85%. Here we observed that all the methods have more than a 90% detection rate for normal flows instead of MLP, while the CNN method achieves a 99% detection rate. The performance of the CNN method in detecting the attack flows is superior to the other methods. So, we can conclude that the performance of the CNN for both normal and attack flows is better than the other methods. A graphical

comparison of the training time (i.e., seconds) for all methods using subset-3 features is shown in Figure 4.18. Here we have observed that the RNN method took longer to train than other methods. The training time of the CNN using subset-3 features is reasonable. The detection time (in microseconds (µs)) per flow is shown in Figure 4.19. The detection time per flow of RNN is 6.7µs, CNN is 1.4µs, MLP is 4.5µs, DNN is 1.7µs, and LSTM is 8.4µs. The detection time of CNN is slightly lower than the other methods. In comparison, LSTM is a significantly higher detection time, indicating that during attack traffic, LSTM can handle a few flows per second. We may conclude that CNN becomes a practical approach for detecting botnet-based attacks by considering the various characteristics, such as detection rate, training time, and detection time, for evaluating the DL methods during the real-time testbed.

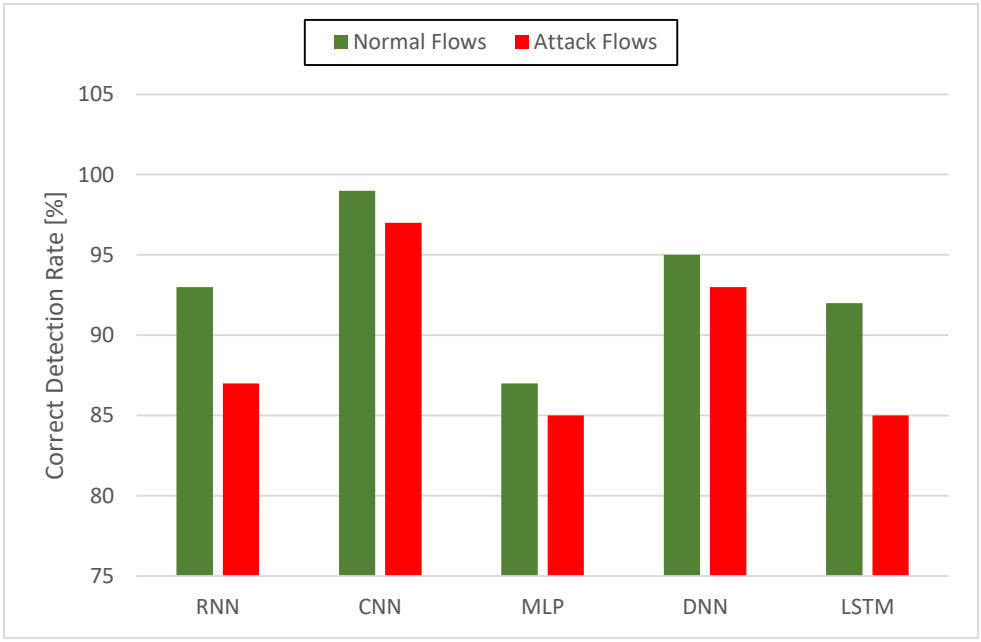


Figure 4.17: The correct detection rate of each algorithm during real-time traffic.

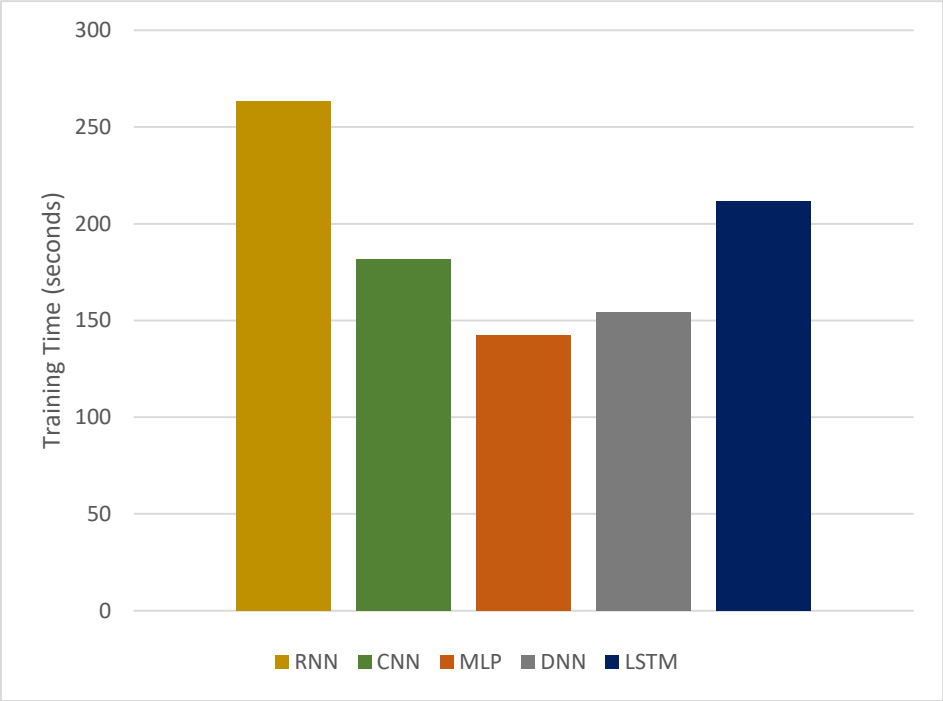


Figure 4.18: Comparison of training time of DL methods using subset-3 features.

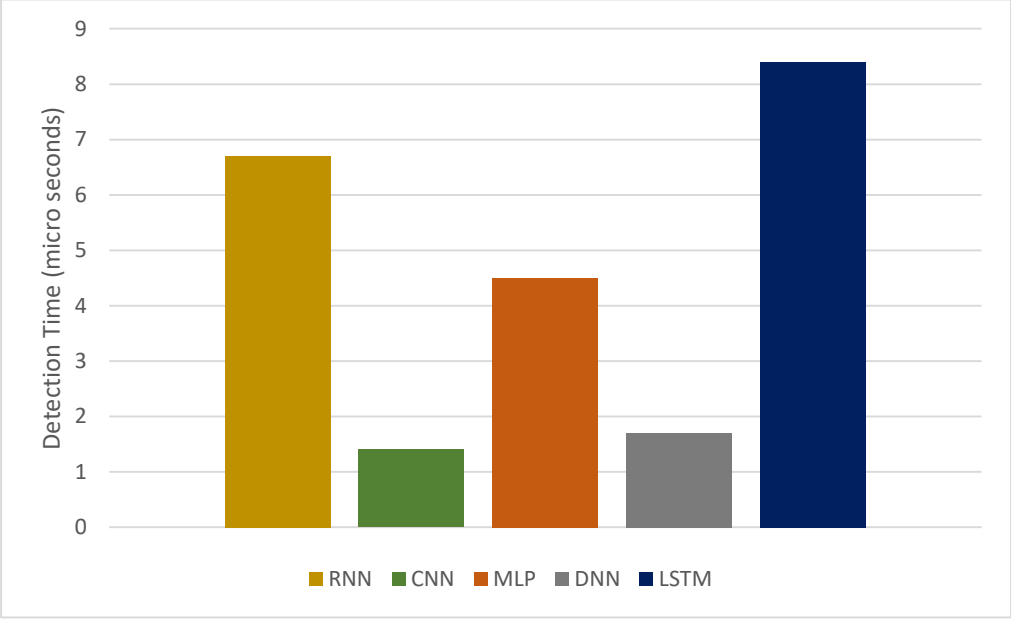


Figure 4.19: Comparison of detection time during real traffic.

4.12 Summary

This chapter addresses the security issues in SDN networks by analyzing the performance of deep learning methods. Especially the focus of this chapter is to detect the botnet-based DDoS attacks in SDN using different deep learning methods. This chapter analyzes and describes the simulation results on detecting botnet-based DDoS attacks in SDN. First, we discuss the results of the selection of optimal features. Second, the structural performance of the DL methods is analyzed. Then the classification performance of the DL methods is described in detail.

The performance of the DL methods is evaluated using different evaluation metrics (i.e., accuracy, detection rate, precision, F1 score, True Positive Rate (TPR), False Positive Rate (FPR), training time, and detection time). The efficiency of the DL or ML methods can be enhanced using a set of optimal features. So, we only replay a portion of the dataset with a few features. The whole dataset is converted into five different subsets of features based on their importance (i.e., subset-1 consists of 76 features, subset-2 consists of 43 features, subset-3 consists of 30 features, subset-4 consists of 23 features, and subset-5 consists of 15 features) using features weighting and threshold tuning methods. Then, these subsets of features are individually used to measure the impact of optimal features on the method's performance while detecting botnet-based DDoS attacks. Simulation results show that DL methods with the same hyper-parameter settings produced different results using different feature subsets. We also observed that each DL method's performance differs on the same feature subset. It means that the DL

method trained with optimal features could improve the detection rate and detect the attack flows more effectively and quickly. Based on the above discussion and experimental performance results of the DL methods, we found that the CNN produced effective performance results compared to other methods (i.e., RNN, MLP, DNN, and LSTM) in the adopted network experimental scenarios. It achieved a maximum detection rate of 99.60% and an accuracy of 99.37% using a subset of 30 features during offline training. We also considered timing metrics (i.e., training and real-time detection time) for evaluating DL methods. We observed that the CNN method took reasonable time during training and real-time detection of flows. As a result, the CNN method shows a respectable accuracy or detection rate for detecting botnet-based DDoS attacks in an SDN environment during real-time testbed evaluation.

CHAPTER 5

MITIGATION STRATEGY AND PERFORMANCE ANALYSIS

This chapter discusses the proposed defense method and then analyze the effectiveness of the defense method on the real testbed. We observed and measured our mitigation strategy's effectiveness during normal and attack traffic.

5.1 Mitigation Strategy Implementation

The SDN controllers are responsible for regularly analyzing the network traffic flows to protect them from several types of malicious attacks. For example, if the deep learning classifier inside the controller detects any type of malicious activity or attack, then it needs to speedily activate the defense shield to diminish the conceivable impact of the attack and ensure that normal network operations remain continue. In the existing studies [6], [11] the concepts of block or modify attack flows are commonly used by the authors in their mitigation methods. There is a chance the OpenFlow switches may contain many malicious flow entries after blocking the attack flows, which influence the network’s normal forwarding process and quickly consume switches and controller resources. To effectively protect the SDN controller from botnet-based DDoS attacks and overcome the above-mentioned problems, a graph theory and dynamic flow deletion-based

mitigation strategy are adopted in this research. The same network topology which is discussed in chapter 3 used for the implementation of mitigation strategy.

In our mitigation strategy, Initially, the controller sends in every ΔT (e.g., 5 sec) a request to the connected corresponding switches for the flow statistics. After collecting the flow statistics, the features extractor module inside the controller extracts the flow features according to the subset-3, and then the controller passes these features' information to the already trained DL classifier (e.g., CNN) to detect the flows whether the attack or normal. Once, any flow is detected as an attack flow by the DL classifier, then it sharply starts to alert the controller.

Once the classifier inside the controller detects any attack flow, it informs the controller. After that, a gray list S_g is created in the database by the controller to put the incoming flows of the switches to the controller for further analysis. It helps to reduce the mistakenly killing of normal packets and continue the normal operation of the residual network. After successfully placing the attack flows in the S_g list, the controller redirects the flows of the S_g list to the DL classifier (CNN) to classify the normal flows with the label "0" and attack flows with the label "1". Here we create a counter variable C that starts to calculate those flows which are labeled as attack flows by the classifier and set a limit for attack flows (i.e., $C \geq 10$). At the same time, the controller creates other two new lists; the S_d delete list and the S_b block list in the database. The S_d list stored the attack flows that need to be deleted from the OpenFlow switches and the S_b list stored the information of hosts which are involved in the attack to block these hosts and for further future use. The controller has a host tracker feature that can extract the attacking hosts' information

(e.g., IP or MAC address, UDP or TCP port number, entry port, etc.) When the counter C research to its set limit (i.., C≥10), then we used the concept of graph theory derived to trace out the attacking path in the network.

5.1.1 Attack Path Identification

Here finding the path from where the attack flows to pass and locating the switches through where the attack flows enter the network is more important. The attacking path between more than two switches can be formulated as:

$$E_{i,j} = \sum(s_i, r_i) \rightarrow (s_j, r_j), where\ s_i, s_j \varepsilon\ S_{attack} \quad (5.1)$$

Where $E_{i,j}$ is the edge (i.e., the path for attack), and S_{attack} represents the set of switches from where the botnet-based DDoS attack is passed. When the attack flows pass through both s_i and s_j OpenFlow switches and the forwarding rules are meet. Then we can predict that the edge between the (si, sj) switches is a close hop in the attacking path. So, our mitigation strategy is based on two principles: 1) use a dynamic flow deletion mechanism to delete or shield as many as possible attack flows, and 2) avoid the by-mistake killing of the normal flows. Hence, by finding the attacking edge or path $E(i,j)$, We can implement a more targeted flow deletion strategy to mitigate botnet-based DDoS attacks in an SDN environment.

To effectively implement the graph theory concept to find out the attacking path, we need to consider the characteristics of the network traffic. Let's assume (e.g., "the closer the hop is to the attack sources on an attacking path, the proportion

of the botnet-based DDoS attack is greater in the link traffic"). It can be mathematically formulated as follows:

$$U\left(h_{(s_i,s_{i+1})}\right) < U\left(h_{(s_j,s_{j+1})}\right), \quad where\ i < j \tag{5.2}$$

By considering our network topology, let us consider two attack paths $(s_1, s_5, s_7)$and (s_2, s_5, s_7) are detected in the network, and they are overlapped to each other on the link (s_5, s_7) as shown in Figure 5.1. For the links (s_1, s_5) and (s_2, s_5), the proportions of the attacks are $U\left(h_{(s_1,s_5)}\right)$ and $U\left(h_{(s_2,s_5)}\right)$, are respectively. Then the proportion of the botnet-based DDoS attack on the link (s_5, s_7) can be represented as $U\left(h_{(s_5,s_7)}\right) = \left(B^d_{(s_1,s_5)} + B^d_{(s_2,s_5)} + B^d_{others}\right) \Big/ \left(B^t_{(s_1,s_5)} + B^t_{(s_2,s_5)} + B^t_{others}\right)$, so it can be pointed that the link (s_5, s_7) may contain the traffic from the other links such as (s_3, s_5) beside then (s_1, s_5) and (s_2, s_5) links. Since these links do not have the attack traffic, $B^d_{others} = 0$ and $B^t_{others} \geq 0$. If the links (s_5, s_7) do not contain the traffic from the other links except $\left(s_1, s_{5,}\right)$ and (s_2, s_5), then $B^t_{others} = 0$. Therefore, $U\left(h_{(s_5,s_7)}\right)/U\left(h_{(s_1,s_5)}\right) \leq 1$and $U\left(h_{(s_5,s_7)}\right)/U\left(h_{(s_2,s_5)}\right) \leq 1$. An attacking path $P_{s,d} = (s_s, s_{s+1}, \dots\dots\dots, s_d), \quad whereN_{s_i,d} = s_{i+1}, U\left(h_{(s_{i-1},s_i)}\right) \geq U\left(h_{(s_{i+1},s_{i+2})}\right)$ holds presume that they hold for each link in the path. So, according to Figure 5.1, the links $\left(s_1, s_{5,}\right)$ and (s_2, s_5), contain much larger attack traffic instead of normal business traffic due to botnet-based DDoS attack (i.e., $U\left(h_{(s_1,s_5)}\right) >> 1 - U\left(h_{(s_1,s_5)}\right)$ and $U\left(h_{(s_2,s_5)}\right) >> 1 - U\left(h_{(s_2,s_5)}\right)$. The flow dropping rate should be increased to minimize the attack at switches and controllers since the switches from

which the attack traffic enters the network typically contain more attack flows than other switches. For convenience, the switches which are closer to the attack hop can be called edge switches S_e while the other switches in the path $P_{i,j}$ are called intermediate switches S_m letter on. So, the dropping rate on the intermediate switches should be smaller compared to the edge switches to shrink the impact on the normal requests. Hence the significant issues are: to find the proper attacking path $P_{i,j}$ and trace out the edge switches S_e form where the attack enters, and set the appropriate dropping rate on the switches which are present in the attacking path. And, there is no need to set the dropping rate on that switches only contain normal traffic.

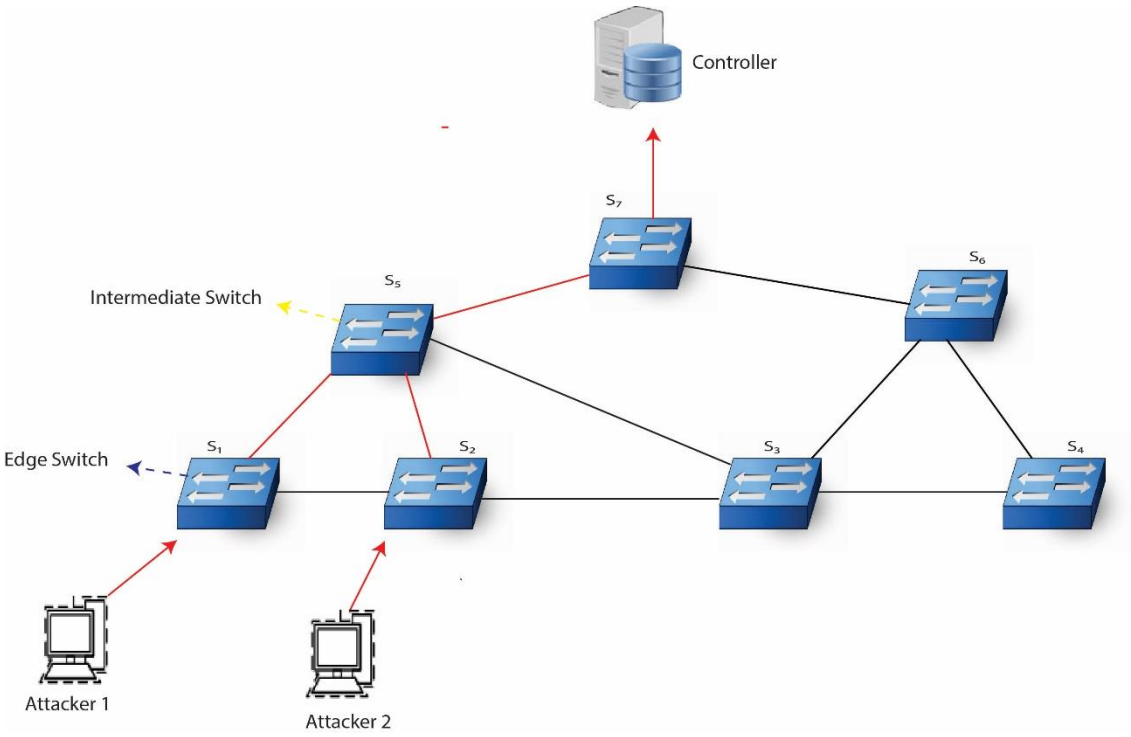


Figure 5.1: An example of finding the attack path in adopted network topology.

5.1.2 Dropping Strategy

After successfully finding the attack path, there is a need to implement an effective dropping strategy against the attack flows. To avoid the chance of by-mistake dropping the normal flows, the dropping rate at the intermediate switches S_m should be smaller than the edge switches S_e. So, by considering the characteristics of botnet-based DDoS attacks, we can use the entropy H of source IP address packets and change the number of passing packets N through an edge switch s_e per unit of time. So, the dropping rate for the edge switch can be calculated using the following formula.

$$Dr_{edge} = k(\Delta H, \Delta N) \tag{5.3}$$

Where in the above Equation 3.30, ΔH represents the difference in entropy of source IP addresses of packets per unit of time, and ΔN is the difference in the number of packets passing through a switch per unit of time, and $Dr_{edge} \epsilon (0,1)$. The values of the H and N start to rapidly increase in some switches when there is a botnet-based DDoS attack, so, to mitigate the attack traffic there is an increase expected in the dropping rate of that switches. However, a continuous increase in the dropping rate can affect normal network traffic. Additionally, we must indicate the upper bound for dropping rates much less than 1. Therefore, the dropping rate Dr_{edge} can be calculated as follows:

$$\varphi(w) = \frac{1}{(1+e^{w})} - (1 - l) \tag{5.4}$$

In the above Equation 5.4, w represents the weighted sum of ΔH and ΔN values and $w > 0$. l represents the upper limit for the dropping rate during the attack, thus, $Dr_{edge} \epsilon\ [l - 0.5, l]$

The intermediate switches S_m may be connected with the edge or other switches. There is a chance the intermediate switches have fewer attack flows or only normal flows. It means that the dropping rate Dr_{min} at the intermediate switches keeps smaller than the edge switches. So, the edge switches have a dropping rate relatively smaller than the upper limit l, which indicates that is attack is not severe and can be easily mitigated at the edge switches. Otherwise, there is also need to set an appropriate dropping rate Dr_{min} at the intermediate switches. The dropping rate for the intermediate switches may be computed by including edge switches to prevent the attack as $Dr_{min} = k \times\ Dr_{edge}$, where $k\ \epsilon\ (0,1)$ The shortest hops in the attack path from the edge to the intermediate switches, and k is a coefficient whose value depends on the distance between the edge and intermediate switches. If the edge and intermediate switches are far away, then the intermediate switches perform dropping at a smaller dropping rate. Hence we can get:

$$Dr_{min} = \begin{cases} 0, & l - Dr_{edge} > \epsilon \\ k \times\ Dr_{edge}, & l - Dr_{edge} \leq \epsilon \end{cases} \tag{5.5}$$

Where ϵ is the smallest constant that determines the difference between l and Dr_{edge}.The following formula can be used to obtain the value of k.

$$k = \frac{N_{hops} - n_{hops}}{N_{hops}} \quad (5.6)$$

Where N_{hops} is the distance in the attack path $P_{i,j}$ from the edge switch to the destination switch, and n_{hops} is the distance from the edge to intermediate switch.

Finally, after successfully finding the attack path, and the switch near the attacking source, and calculating the dropping rates for edge and intermediate switches, the controller sends the "OFPFC_ADD" messages to the corresponding switches to insert new flow entries in the flow tables of the switches. After receiving the intrusion by the controller, the corresponding edge and intermediate switches start to dynamically delete or drop the incoming attack flows according to the delete list S_d using the calculated dropping rates. Here, if the dropping rate for any host reaches 100%, then, first, the corresponding host is blocked, and the information of that host is placed in the block list S_b for future use. In short, this mitigation strategy successfully protects the SDN controller from botnet-based DDoS attacks and reduces the by-mistake killing of normal flows, and also helps the controller to continue its normal services.

5.2 Performance Analysis and Discussion

The performance of the mitigation strategy is evaluated through flow table utilization and the computational resources utilization. We also evaluated how effectively our suggested defence method performed against those already used. Southbound links are frequently utilized to transmit the packet headers to the

control plane. The packet headers are examined to classify the normal or attack traffic on the controller plane. Generally, the transfer of instructions and communication between the control and data planes typically take place across the southbound interfaces. The increased traffic load on the southbound interfaces will hamper the normal connection between the control and data planes.

5.2.1 Flow Table Utilization

The flow table utilization is measured in terms of generation of flow rules in both network states (i.e., normal traffic and attack traffic). The number of flow rules generated by the SDN controller under two different network states (i.e., normal and attack) are shown in Figure 5.2. Here the storage limit for the flow rules on the various switches is set to a maximum of “1500”. Furthermore, the number of flow rules generated by the controller under the attack network traffic and normal network traffic are calculated separately, as shown in Figure 5.2. The experimental network is kept in a relatively network-stable condition with only background traffic that is typical for networks, and the number of flow rules is always maintained at 180 per second. On the other hand, the botnet-based DDoS attack started in the 20th second, and we noticed that the number of flow rules dramatically rose. The amount of flow rules exceeded the OpenFlow switches' maximum storage capacity in less than 5 seconds. Therefore, installing new flow rules for normal network users won't be possible if the switches' flow rules storage is full.

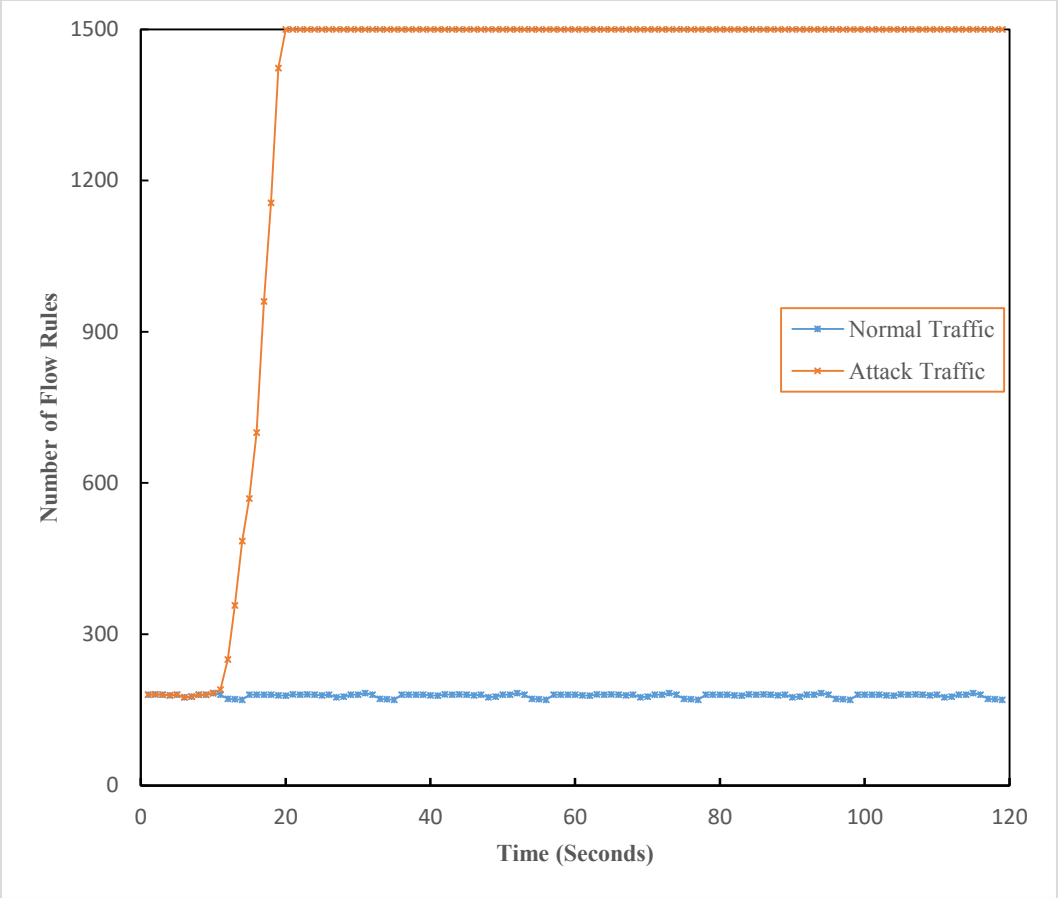


Figure 5.2: The Number of flow rules generated in different network states.

We start the defense method after the 40th seconds when both normal and attack traffic is running in the network. Here we can observe that after activating the defense method, the number of flow rules starts to decrease gradually and maintain the flow rules limit for the attack flows around 250 within 15 seconds as shown in Figure 5.3. So, our defense methods kept generating flow rules below 250 per second.

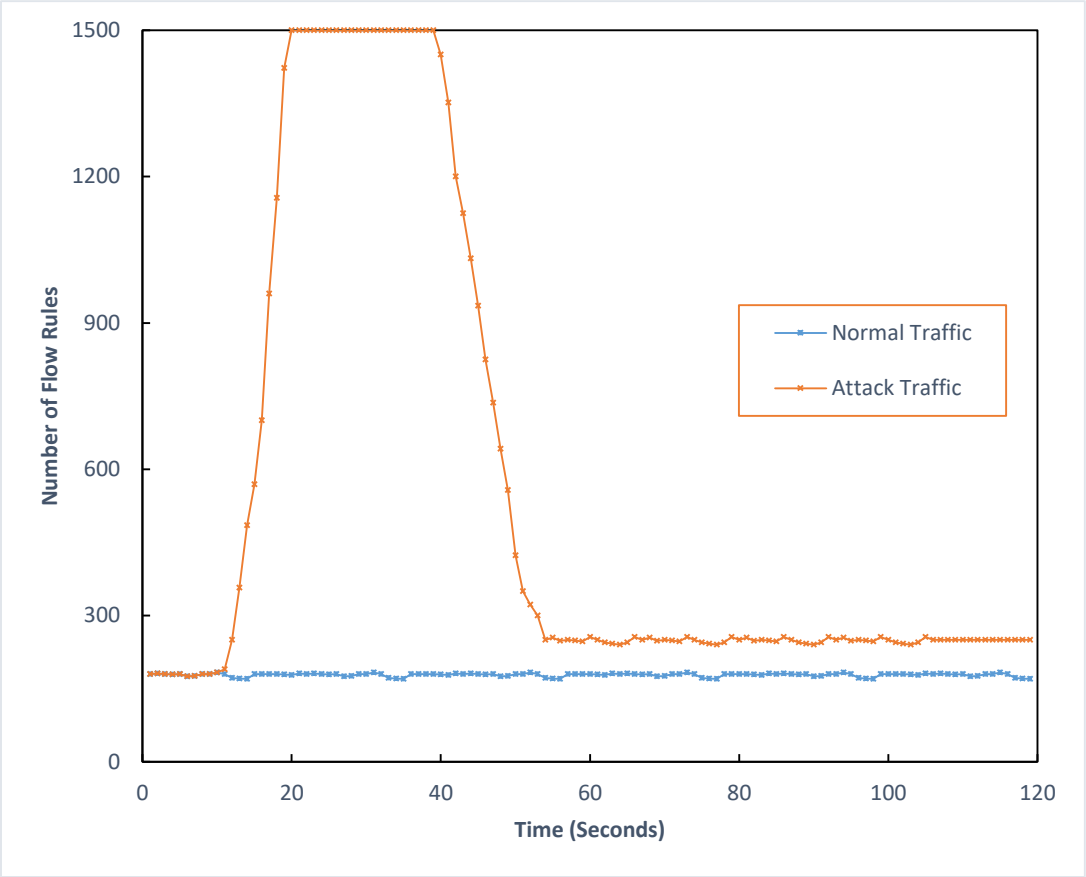


Figure 5.3: Shows the change in the generated flow rules after activating the proposed defense method.

A comparison of the proposed defense methods with some existing methods is shown in Figure 5.4. We individually implemented different defense methods in the SDN controller to evaluate their performance in our adopted network scenarios. Figure 5.3 shows that the number of flow rules remains stable in the first 10 seconds for all methods without attack traffic. Gradually more flow rules are added to the network due to the botnet-based DDoS attack. When the attack is launched after 10 seconds, the number of flow rules steadily increases, and we activate the various defense strategies to assess how well they perform in terms of flow rules. Our proposed method effectively controls the growth of flow rules and always keeps

them around 250 per second. The growth of flow rules is unaffected by the SIFT approach, and they quickly reach the maximum rate of 1500 per second. The defense strategy based on dynamic flow deletion slows the rate of flow rule growth to about 350 per second. Similarly, the SD-Anti DDoS defense approach consistently limits the expansion of the flow rules at around 400 per second. Last but not least, when the Load-Aware technique is activated, the flow rules expand at a rate of 1000 per second, and after 60 seconds, they continue to grow at a rate of about 800 per second.

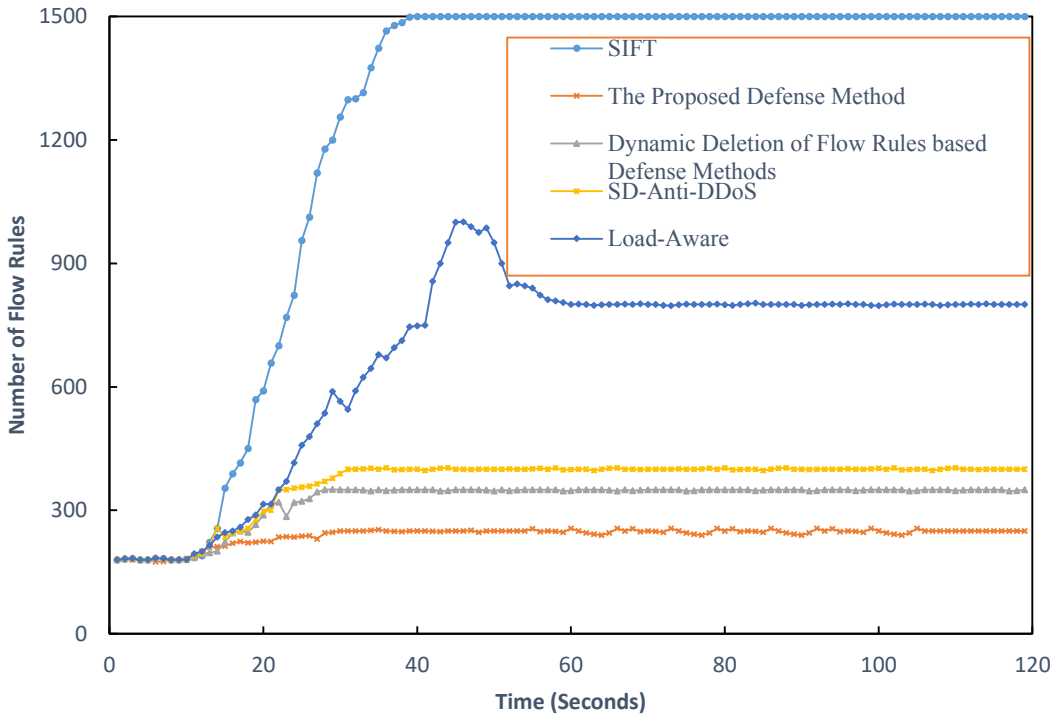


Figure 5.4: Comparison of the proposed defense method with existing methods in terms of flow rules.

Compared with the other defense methods, our defense method keeps the number of rules at the lowest and helps the controller process the flow rules request of the normal users even when the network is under attack.

5.2.2 Computational Resources Utilization

The effectiveness of the proposed defense method is also validated through the evaluation of the consumption of CPU usage by the controller during normal and attack network states. Because the controller is responsible for managing and configuring every OpenFlow switch in the network, assessing the controller's CPU resource use is crucial. Therefore, we need to verify that the controller is not excessively consuming the CPU resources during the attack and after activating the defense method. Figure 5.5 compares the CPU utilization under two network states (i.e., normal or attack). We observed that the CPU utilization remains controlled when only the background traffic runs in the network (i.e., at around 20%). After 10 seconds, when we inject the attack traffic into the network, the CPU utilization gradually increases and reaches about 70%-90%. After 15 seconds of the attack, we activate the defense methods, which gradually decrease the CPU utilization by dynamically deleting the attack flows. After 40 seconds, it controls the utilization of the CPU and maintains it at around 40%. A graphical representation of CPU utilization after activating the defense methods is shown in Figure 5.6.

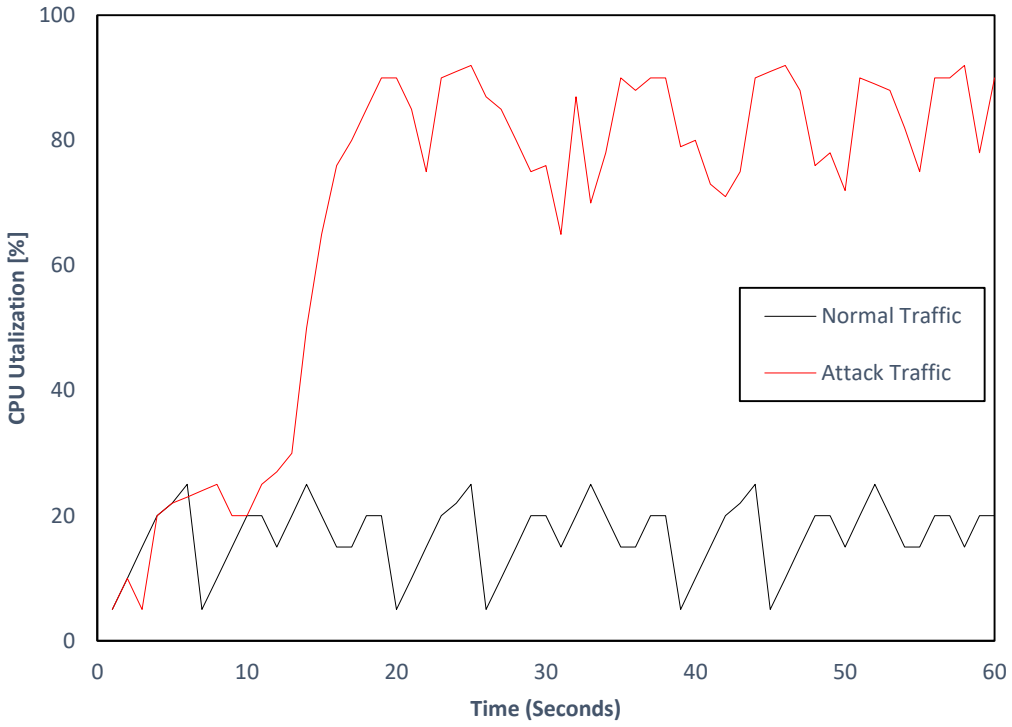


Figure 5.5: Comparison of controller CPU resource consumption in different network states.

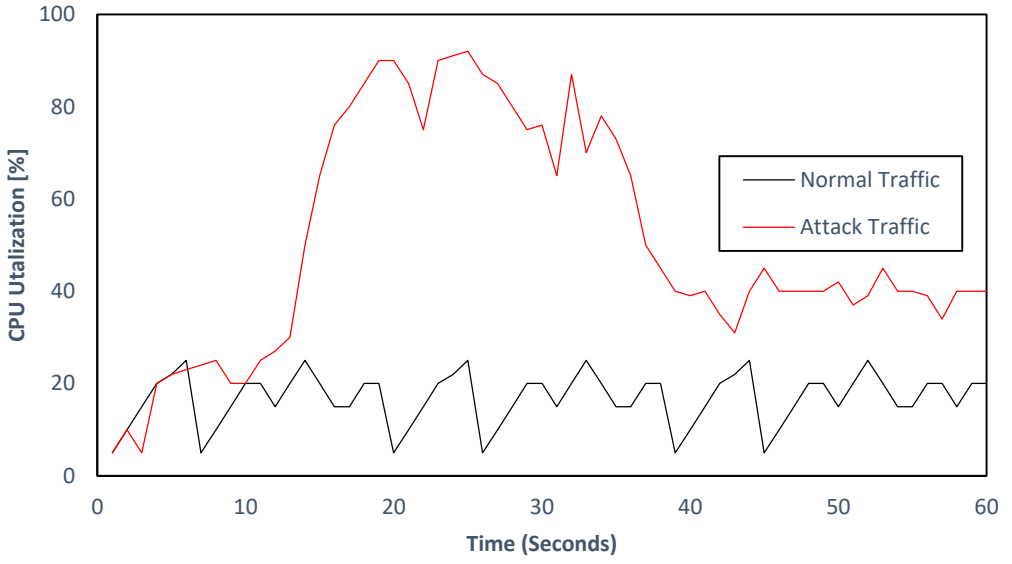


Figure 5.6: Comparison of controller CPU resource consumption in different network states after activating the defense method.

In addition, a CPU utilization comparison between the proposed method and those that are currently in use is shown in Figure 5.7. As discussed above, we implemented different defense methods (i.e., SIFT, dynamic flow deletion method, SD-Anti DDoS, and Load-Aware) in the controller to observe their performance while controlling the CPU utilization. Figure 5.7 shows that for the first 10 seconds, when only the background traffic runs in the network, the CPU utilization remains stable at around 20%-25%. When the botnet-based DDoS attack traffic is injected into the network, the CPU utilization increases in a gradient way. After 10 seconds of the attack, when the CPU utilization increases gradually, we activate the defense methods to observe their performance in controlling the CPU utilization. Our proposed method does not increase the workload of the controller and also controls the CPU utilization and always keeps it around 40%. After activating the SIFT method, we observed that it does not help to control the CPU utilization and reaches 90%-95%. The dynamic flow deletion method controls the CPU utilization within the range of 45%-50% in our adopted network scenarios. Similarly, the SD-Anti DDoS method keeps CPU utilization at around 55%. Lastly, the Load-Aware method maintains CPU utilization at around 70% during the attack.

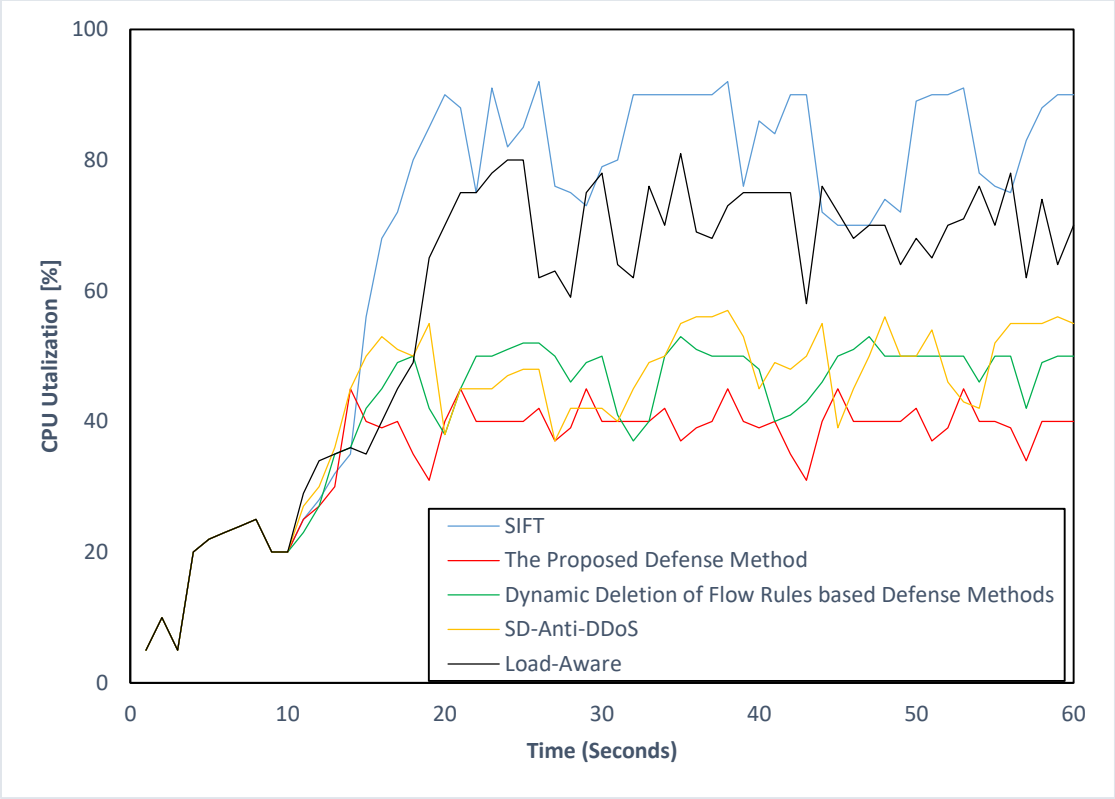


Figure 5.7: Comparison of the proposed defense method with existing methods in terms of CPU utilization.

In short, our proposed defense method proved effective in controlling the generation of flow rules and CPU utilization in normal and attack network states compared to other defense methods. Our proposed method differs from other defense methods as data and control planes are programmable. However, in our method, when the CNN methods detect the attack traffic, we use the graph theory concept to find switches in the attack path and dynamically delete attack packets with different dropping rates on different switches. Most existing defense methods try to delete the attack traffic of the last switch on the attack path. This makes our defense method more effective in protecting the SDN controller from botnet-based DDoS attacks.

5.3 Summary

This chapter discusses the proposed defense method's effectiveness in protecting SDN networks from botnet-based DDoS attacks. The proposed defense method has two main principles (a) find the attacking path in the network, (b) adopt a more systematic dropping strategy to prevent accidentally killing normal flows. First, graph theory is used to find the switches in the attack path. Second, dropping rates for the different OpenFlow switches are calculated based on the switch position in the attack path, and then dynamically, attack flows are deleted using estimated dropping rates. Furthermore, we studied different defense approaches to provide more scalable and effective countermeasures against these attacks without changing the SDN network design or adding new network devices. We validate the effectiveness of our defense methods using two different metrics (i.e., Rate of generated flow rules and CPU utilization). Our proposed method maintains the Flow rate at around 250 per second and CPU Utilization at approximately 40% during the attack. We also compared our defense method with the existing methods. Based on the reported results, we can conclude that the proposed defense method in this research can protect the SDN controllers from botnet-based DDoS attacks without increasing the overhead of the SDN controller.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

SDN redefines the network's organization and management, which leads to network reforms. However, SDN resolves emerging problems, such as flexibility, security, scalability, etc., in traditional networks. Simultaneously, the emerging architecture of the SDN introduced a new array of privacy and security challenges that needs great attention from the research and network community. Because malicious attacks against computer networks and SDN are potentially growing, intelligent intrusion detection systems are required to detect and mitigate these attacks. The botnet attacks are considered more dangerous in the modern era of networking technologies. Attacks from botnets can target controllers in the control plane or network devices in the data plane (such as routers, OpenFlow switches, etc.) in an SDN environment. The sophistication of network flow features makes the detection of botnets and DDoS attacks in SDNs more challenging than in traditional networks. The decoupled architecture of the SDN allows to develop and deploy the intrusion detection models in the architecture to detect and mitigate botnet-based DDoS attacks. Nevertheless, secure SDN networks shall be adopted by providing a reliable definition of the behavior of botnet-based DDoS attacks. Recently, deep learning-based networking applications have been in trading and become a

viable option for network intrusion detection systems. DL methods use large historical training datasets for learning intrusion patterns. They can predict the real-time network state (i.e., normal traffic flows or attack traffic flows) and update the controller(s). This research proposed an IDS based on deep learning methods named "DepBot" to detect and mitigate botnet-based DDoS in an SDN environment. First, we generate a simulation-based dataset in a pure SDN-supported environment. Then the attack detection is performed using various DL methods such as RNN, CNN, MLP, DNN, and LSTM. We implemented and tested the DL methods using a simulated dataset to detect the attack flows. The simulated dataset consists of two classes (i.e., normal and attack). We also used pre-processing techniques such as feature elimination and selection to select a set of best features and improve the performance of the DL methods while detecting attacks. The dataset with 83 features is converted into five different subsets of features based on their importance, where 30 optimal features were selected out of 83. We also performed min-max normalization at the preprocessing stage to give equal weightage to all features and observe their effect on the method's performance. The whole system model, from the dataset collection to attack detection and mitigation, is implemented over a single controller and gratitude to the centralized control of the SDNs.

RQ1: Are the publically available intrusion detection datasets can use for SDN?

Aa core contributing factor of this research is to develop a dataset in a pure SDN environment to train DL methods instead of relying on traditional or old datasets. Because, in most of the existing studies [17] [20], the authors are using old or traditional datasets (i.e., NSL-KDD, DARPA, Na-BaIoT, CIC-DoS-2017, etc.) for the training of machine learning or deep learning methods. These datasets are unsuitable for SDN environments because they are developed in traditional network environments and suffer imbalanced problems. The power and efficiency of the DL methods are evaluated by adopting baseline hyper-parameters in their architectures.

RQ2: Is Deep Learning methods can effectively detect botnet attacks in SDN?

According to the results on the simulation dataset, the CNN method achieved superior accuracy and detection rate results. CNN achieved a maximum accuracy of 99.37% and a detection rate of 99.60% using 30 optimal features. In addition, the performance of the DL methods training using a set of 30 optimal features was also evaluated in real-time testbed settings where the CNN method achieved a higher real-time detection rate compared to RNN, MLP, DNN, and LSTM. The real-time detection rate of CNN reaches 97% for attack flows and 99% for normal flows. The performance of the CNN method proved that the DL methods can effectively detect the botnet-based DDoS attacks in SDN.

RQ3: How can we protect the SDN from botnet attacks?

After successfully detecting botnet-based DDoS attacks, we deployed that graph theory and dynamic flow deletion-based mitigation strategy to protect the SDN controller against these attacks. The effectiveness of our defense methods is measured using two parameters: rate of flow rules generation by controller and CPU utilization. The proposed method maintains a flow generation rate of around 250 per second and CPU utilization at about 40% during the attack, which is better than the existing work.

The limitations of this research are: This research focuses on detecting botnet-based flooding attacks in SDN environments. This produced dataset cannot be used for detecting non-volumetric attacks (i.e., slow-rate DDoS attacks). This research also did not help to detect other types of malicious attacks (i.e., saturation, ransomware, other types of DDoS attacks, etc.) In this research, we used around 89000 records to train and test DL methods, so the number of records for the dataset can be enhanced for better training of machine learning or deep learning methods. Lastly, we just used a single SDN controller in our experimental network, it may not be effective in a distributed or multi-controller network architecture.

6.2 Future Work

Our current research focuses on detecting and mitigating botnet-based DDoS attacks in a single-controller SDN architecture. For future work, we intend to extend this research to investigate other security issues of SDN. For example,

detection and mitigation of other malicious attacks such as slow-rate DDoS, DDoS-web, other types of DDoS attacks, saturation attacks, ransomware attacks, etc.

This research can be extended to detecting other types of malicious attacks, such as spoofed, low-rate, botnet-based low-rate DDoS, etc., in the SDN environment.

This research can extend the development of new optimization solutions to reduce the false positive rate and improve the true positive rate for detecting botnet-based DDoS attacks in SDN-based networks.

In the future, a score reporting scale can be introduced by using this research to measure the performance of the machine learning or deep learning methods for the SDN environments (i.e., selection of datasets, features selections with advanced methods, overfitting problems).

A fair comparison scale can be implemented to assess the effectiveness of deep learning methods in identifying various attacks in SDN systems.

Furthermore, in the future, this research will help researchers to apply more advanced deep learning and machine learning algorithms with optimized feature selection methods for intrusion detection in SDN as other domains such as IoT, smart grid, cloud computing, etc.

In the future, hybrid or ensemble deep learning approaches can be introduced for the SDN networks for intrusion detection.

Furthermore, deep learning or machine learning methods can be trained in real-time to keep the DL/ML-based systems updated.

This research can be extended to detect malicious attacks in other SDN-based environments such as cloud computing, IoT, and ISP networks.

Recently, researchers are trying to adapt SDN multi-controller networks to solve the security problems of a single-controller architecture (i.e., “single point of failure”). In the future, this research may be extended to the investigation of security issues of the multi-controller SDN architecture. A multi-controller architecture has a positive aspect because it can be divided into a hierarchal or flat architecture. The flat SDN architecture can consist of multiple domains in different locations, and each domain can be controlled by a single controller, so the controllers can use east-bound interfaces to communicate with each other. The hierarchal SDN architecture maybe consists of multiple layers, the top layer is referred to as the master layer, where the master controller can work and control the whole network, and the rest of the layer is referred to as the slave layers that have multiple controllers, and these controllers can control multiple domains.

Although multi-controller SDN architecture has several advantages over single-controller architecture, it may face a set of major challenges. For example, high availability and reliability cannot be assured for multi-controller architecture because the attackers can target the connection links between the multiple controllers or send many malicious packets to overwhelm the controllers' processing. Thus, the connected switches or targeted controllers will be isolated

from the other part of the network. Therefore, it is crucial to design defense methods for the multi-controller that can detect incoming flows, monitor the multiple controllers in different domains, and countermeasure them without affecting the other network.

The proposed defense method is tested in a single-controller network architecture that can protect the SDN controller against botnet-based DDoS attacks. In future research, we will focus to develop a defense method that can protect the controller(s) in a multi-controller architecture.

REFERENCES

[1] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for internet of things (IoT) security," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 3, pp. 1646–1685, 2020.

[2] S. H. Haji *et al.*, "Comparison of software defined networking with traditional networking," *Asian J. Res. Comput. Sci.*, vol. 9, no. 2, pp. 1–18, 2021.

[3] A. Mestres *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, 2017.

[4] M. W. Nadeem, H. G. Goh, V. Ponnusamy, and Y. Aun, "Ddos detection in sdn usingmachine learning techniques," *Comput. Mater. Contin.*, vol. 71, no. 1, 2022, doi: 10.32604/cmc.2022.021669.

[5] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surv. tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[6] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, and Y. Deng, "A new framework for DDoS attack detection and defense in SDN environment," *IEEE Access*, vol. 8, pp. 161908–161919, 2020.

[7] S. Wang *et al.*, "Detecting flooding DDoS attacks in software defined networks using supervised learning techniques," *Eng. Sci. Technol. an Int. J.*, vol. 35, p. 101176, 2022.

[8] Y. Cui *et al.*, “Towards DDoS detection mechanisms in software-defined networking,” *J. Netw. Comput. Appl.*, vol. 190, p. 103156, 2021.

[9] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, “A DDoS attack detection method based on SVM in software defined network,” *Secur. Commun. Networks*, vol. 2018, 2018.

[10] A. A. Diro and N. Chilamkurti, “Distributed attack detection scheme using deep learning approach for Internet of Things,” *Futur. Gener. Comput. Syst.*, vol. 82, pp. 761–768, 2018.

[11] J. A. Perez-Diaz, I. A. Valdovinos, K.-K. R. Choo, and D. Zhu, “A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning,” *IEEE Access*, vol. 8, pp. 155859–155872, 2020.

[12] R. K. Chouhan, M. Atulkar, and N. K. Nagwani, “A framework to detect DDoS attack in Ryu controller based software defined networks using feature extraction and classification,” *Appl. Intell.*, pp. 1–21, 2022.

[13] Y. Liu, T. Zhi, M. Shen, L. Wang, Y. Li, and M. Wan, “Software-defined DDoS detection with information entropy analysis and optimized deep learning,” *Futur. Gener. Comput. Syst.*, vol. 129, pp. 99–114, 2022.

[14] O. Habibi, M. Chemmakha, and M. Lazaar, “Imbalanced tabular data modelization using CTGAN and machine learning to improve IoT Botnet attacks detection,” *Eng. Appl. Artif. Intell.*, vol. 118, p. 105669, 2023.

[15] M. Wazzan, D. Algazzawi, A. Albeshri, S. Hasan, O. Rabie, and M. Z. Asghar, “Cross Deep Learning Method for Effectively Detecting the Propagation of IoT Botnet,” *Sensors*, vol. 22, no. 10, p. 3895, 2022.

[16] M. M. Alani, “BotStop: Packet-based efficient and explainable IoT botnet detection using machine learning,” *Comput. Commun.*, vol. 193, pp. 53–62, 2022.

[17] G. L. Nguyen, B. Dumba, Q.-D. Ngo, H.-V. Le, and T. N. Nguyen, “A collaborative approach to early detection of IoT Botnet,” *Comput. Electr. Eng.*, vol. 97, p. 107525, 2022.

[18] D. Yin, L. Zhang, and K. Yang, “A DDoS attack detection and mitigation with software-defined Internet of Things framework,” *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[19] S. T. Zargar, J. Joshi, and D. Tipper, “A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks,” *IEEE Commun. Surv. tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[20] H. S. Ilango, M. Ma, and R. Su, "A FeedForward–Convolutional Neural Network to Detect Low-Rate DoS in IoT," *Eng. Appl. Artif. Intell.*, vol. 114, p. 105059, 2022.

[21] M. W. Nadeem, H. G. Goh, Y. Aun, and V. Ponnusamy, "A Recurrent Neural Network based Method for Low-Rate DDoS Attack Detection in SDN," in *2022 3rd International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, 2022, pp. 13–18.

[22] K. N. Rao, K. V. Rao, and P. R. PVGD, "A hybrid intrusion detection system based on sparse autoencoder and deep neural network," *Comput. Commun.*, vol. 180, pp. 77–88, 2021.

[23] P. L. S. Jayalaxmi, G. Kumar, R. Saha, M. Conti, T. Kim, and R. Thomas, "DeBot: A deep learning-based model for bot detection in industrial internet-of-things," *Comput. Electr. Eng.*, vol. 102, p. 108214, 2022.

[24] H.-T. Nguyen, Q.-D. Ngo, D.-H. Nguyen, and V.-H. Le, "PSI-rooted subgraph: A novel feature for IoT botnet detection using classifier algorithms," *ICT Express*, vol. 6, no. 2, pp. 128–138, 2020.

[25] A. Al Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and Grey Wolf optimization for IoT botnet detection," *J. Ambient Intell. Humaniz. Comput.*, vol. 11, pp. 2809–2825, 2020.

[26] M. Asadi, M. A. J. Jamali, S. Parsa, and V. Majidnezhad, "Detecting botnet by using particle swarm optimization algorithm based on voting system," *Futur. Gener. Comput. Syst.*, vol. 107, pp. 95–111, 2020.

[27] J. Xie *et al.*, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[28] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Commun. Surv. tutorials*, vol. 21, no. 1, pp. 686–728, 2018.

[29] S. Almutairi, S. Mahfoudh, S. Almutairi, and J. S. Alowibdi, "Hybrid botnet detection based on host and network analysis," *J. Comput. Networks Commun.*, vol. 2020, pp. 1–16, 2020.

[30] X. Pei, S. Tian, L. Yu, H. Wang, and Y. Peng, "A two-stream network based on capsule networks and sliced recurrent neural networks for DGA botnet detection," *J. Netw. Syst. Manag.*, vol. 28, pp. 1694–1721, 2020.

[31] F. Ja'fari, S. Mostafavi, K. Mizanian, and E. Jafari, "An intelligent botnet blocking approach in software defined networks using honeypots," *J.*

Ambient Intell. Humaniz. Comput., vol. 12, pp. 2993–3016, 2021.

[32] T. E. Ali, Y.-W. Chong, and S. Manickam, "Comparison of ML/DL Approaches for Detecting DDoS Attacks in SDN," *Appl. Sci.*, vol. 13, no. 5, p. 3033, 2023.

[33] G. O. Anyanwu, C. I. Nwakanma, J.-M. Lee, and D.-S. Kim, "RBF-SVM kernel-based model for detecting DDoS attacks in SDN integrated vehicular network," *Ad Hoc Networks*, vol. 140, p. 103026, 2023.

[34] W. G. Negera, F. Schwenker, T. G. Debelee, H. M. Melaku, and Y. M. Ayano, "Review of Botnet Attack Detection in SDN-Enabled IoT Using Machine Learning," *Sensors*, vol. 22, no. 24, p. 9837, 2022.

[35] P. W. Eslinger *et al.*, "Projected network performance for next generation aerosol monitoring systems," *J. Environ. Radioact.*, vol. 257, p. 107088, 2023.

[36] R. Ahmad, M. Hämäläinen, R. Wazirali, and T. Abu-Ain, "Digital-care in next generation networks: Requirements and future directions," *Comput. Networks*, vol. 224, p. 109599, 2023.

[37] D. Kwon, H. Kim, D. An, and H. Ju, "DDoS attack volume forecasting using a statistical approach," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1083–1086.

[38] M. E. Ahmed, S. Ullah, and H. Kim, "Statistical application fingerprinting for DDoS attack mitigation," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 6, pp. 1471–1484, 2018.

[39] Y. Otoum and A. Nayak, "As-ids: Anomaly and signature based ids for the internet of things," *J. Netw. Syst. Manag.*, vol. 29, pp. 1–26, 2021.

[40] M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based intrusion detection systems," *Appl. Soft Comput.*, vol. 92, p. 106301, 2020.

[41] Y. Meidan *et al.*, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, 2018.

[42] K. S. Sahoo *et al.*, "An evolutionary SVM model for DDOS attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 132502–132513, 2020.

[43] A. A. Ahmed, W. A. Jabbar, A. S. Sadiq, and H. Patel, "Deep learning-based classification model for botnet attack detection," *J. Ambient Intell. Humaniz. Comput.*, pp. 1–10, 2020.

[44] H. Alkahtani and T. H. H. Aldhyani, "Botnet attack detection by using CNN-LSTM model for Internet of Things applications," *Secur. Commun. Networks*, vol. 2021, pp. 1–23, 2021.

[45] S. Velliangiri and H. M. Pandey, "Fuzzy-Taylor-elephant herd optimization inspired Deep Belief Network for DDoS attack detection and comparison with state-of-the-arts algorithms," *Futur. Gener. Comput. Syst.*, vol. 110, pp. 80–90, 2020.

[46] J. Cui, M. Wang, Y. Luo, and H. Zhong, "DDoS detection and defense mechanism based on cognitive-inspired computing in SDN," *Futur. Gener. Comput. Syst.*, vol. 97, pp. 275–283, 2019.

[47] S. Schaller and D. Hood, "Software defined networking architecture standardization," *Comput. Stand. interfaces*, vol. 54, pp. 197–202, 2017.

[48] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, 2013.

[49] "Pantou: OpenFlow 1.3 for OpenWRT." online: https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-for-OpenWRT

[50] "Open vSwitch."

[51] "Indigo: Open Source OpenFlow Switches."

[52] J. W. Lockwood *et al.*, "NetFPGA--an open platform for gigabit-rate network switching and routing," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007, pp. 160–161.

[53] G. Lu *et al.*, "{ServerSwitch}: A Programmable and High Performance Platform for Data Center Networks," 2011.

[54] M. B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster, "Switchblade: A platform for rapid deployment of network protocols on programmable hardware," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 183–194.

[55] "Floodlight, "Project Floodlight open source software for building softwaredefined networks."

[56] M. McCauley, "About Pox," 2013.

[57] N. Gude *et al.*, "NOX: towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[58] D. Erickson, “The beacon openflow controller,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13–18.

[59] J. Medved, R. Varga, A. Tkacik, and K. Gray, “Opendaylight: Towards a model-driven sdn controller architecture,” in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, pp. 1–6.

[60] Ryu, “Ryu SDN Framework,” 2013.

[61] N. McKeown *et al.*, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[62] M. E. A. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, “OpFlex control protocol,” *IETF, Apr*, 2014.

[63] A. Doria *et al.*, “Forwarding and control element separation (ForCES) protocol specification,” 2010.

[64] B. Pfaff and B. Davie, “The open vswitch database management protocol,” 2013.

[65] H. Song, “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 127–132.

[66] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “Openstate: Programming platform-independent stateful openflow applications inside the switch,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.

[67] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, “The locator/ID separation protocol (LISP),” 2013.

[68] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network configuration protocol (NETCONF),” 2011.

[69] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, vol. 3, pp. 10–5555.

[70] T. Koponen *et al.*, “Onix: A distributed control platform for large-scale production networks,” 2010.

[71] P. Lin, J. Bi, and Y. Wang, “East-west bridge for SDN network peering,” in *Frontiers in internet technologies*, Springer, 2013, pp. 170–181.

[72] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, “Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains,” *IETF Draft. Work Prog.*, 2012.

[73] F. Benamrane and R. Benaini, “An East-West interface for distributed SDN control plane: Implementation and evaluation,” *Comput. Electr. Eng.*, vol. 57, pp. 162–175, 2017.

[74] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, “A survey on the contributions of software-defined networking to traffic engineering,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 2, pp. 918–953, 2016.

[75] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, “A survey on the security of stateful SDN data planes,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[76] Q. Yan, F. R. Yu, Q. Gong, and J. Li, “Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges,” *IEEE Commun. Surv. tutorials*, vol. 18, no. 1, pp. 602–622, 2015.

[77] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, “A survey of securing networks using software defined networking,” *IEEE Trans. Reliab.*, vol. 64, no. 3, pp. 1086–1097, 2015.

[78] S. Scott-Hayward, S. Natarajan, and S. Sezer, “A survey of security in software defined networks,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 623–654, 2015.

[79] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, “Security in software defined networks: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[80] P. C. Fonseca and E. S. Mota, “A survey on fault management in software-defined networks,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2284–2321, 2017.

[81] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, “Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 388–415, 2017.

[82] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, “Software defined optical networks (SDONs): A comprehensive survey,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 4, pp. 2738–2786, 2016.

[83] S. Bera, S. Misra, and A. V Vasilakos, “Software-defined networking for

internet of things: A survey," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1994–2008, 2017.

[84] R. Alvizu *et al.*, "Comprehensive survey on T-SDN: Software-defined networking for transport networks," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2232–2283, 2017.

[85] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.

[86] O. Michel and E. Keller, "SDN in wide-area networks: A survey," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, 2017, pp. 37–42.

[87] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, 2013.

[88] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[89] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 358–380, 2014.

[90] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[91] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 655–685, 2015.

[92] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and Y. Liu, "A survey on large-scale software defined networking (SDN) testbeds: Approaches and challenges," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 2, pp. 891–917, 2016.

[93] C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 4, pp. 2687–2712, 2016.

[94] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, 2015.

[95] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-

defined networking," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[96] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Commun. Surv. tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[97] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surv. tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[98] OpenFlow Switch Consortium, "OpenFlow Switch Specification Version 1.0 of OpenFlow."

[99] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[100] M. Kubat and Kubat, *An introduction to machine learning*, vol. 2. Springer, 2017.

[101] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[102] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine learning: algorithms and applications*. Crc Press, 2016.

[103] J. Brownlee, "Supervised and unsupervised machine learning algorithms," *Mach. Learn. Mastery*, vol. 16, no. 03, 2016.

[104] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," *arXiv Prepr. arXiv1701.02145*, 2017.

[105] M. Zamani and M. Movahedi, "Machine learning techniques for intrusion detection," *arXiv Prepr. arXiv1312.2177*, 2013.

[106] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning. Springer series in statistics," *New York, NY, USA*, 2001.

[107] M. W. Nadeem, H. G. Goh, M. A. Khan, M. Hussain, M. F. Mushtaq, and V. A. P. Ponnusamy, "Fusion-Based Machine Learning Architecture for Heart Disease Prediction," *Comput. Mater. Contin.*, vol. 67, no. 2, 2021, doi: 10.32604/cmc.2021.014649.

[108] M. Anam *et al.*, "Osteoporosis prediction for trabecular bone using machine learning: a review," *Comput. Mater. Contin.*, vol. 67, no. 1, 2021, doi: 10.32604/cmc.2021.013159.

[109] S. Thaseen and C. A. Kumar, “An analysis of supervised tree based classifiers for intrusion detection system,” in *2013 international conference on pattern recognition, informatics and Mobile engineering*, 2013, pp. 294–299.

[110] F. E. Heba, A. Darwish, A. E. Hassanien, and A. Abraham, “Principle components analysis and support vector machine based intrusion detection system,” in *2010 10th international conference on intelligent systems design and applications*, 2010, pp. 363–367.

[111] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in *2016 international conference on wireless networks and mobile communications (WINCOM)*, 2016, pp. 258–263.

[112] S. Zanero and S. M. Savaresi, “Unsupervised learning techniques for an intrusion detection system,” in *Proceedings of the 2004 ACM symposium on Applied computing*, 2004, pp. 412–419.

[113] I. Syarif, A. Prugel-Bennett, and G. Wills, “Unsupervised clustering approach for network anomaly detection,” in *International conference on networked digital technologies*, 2012, pp. 135–145.

[114] X. Zhou and M. Belkin, “Semi-supervised learning,” in *Academic Press Library in Signal Processing*, vol. 1, Elsevier, 2014, pp. 1239–1269.

[115] H. Wu and S. Prasad, “Semi-supervised deep learning using pseudo labels for hyperspectral image classification,” *IEEE Trans. Image Process.*, vol. 27, no. 3, pp. 1259–1270, 2017.

[116] O. Chapelle, B. Scholkopf, and A. Zien, “Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews],” *IEEE Trans. Neural Networks*, vol. 20, no. 3, p. 542, 2009.

[117] S. Ding, Z. Zhu, and X. Zhang, “An overview on semi-supervised support vector machine,” *Neural Comput. Appl.*, vol. 28, no. 5, pp. 969–978, 2017.

[118] C. Chen, Y. Gong, and Y. Tian, “Semi-supervised learning methods for network intrusion detection,” in *2008 IEEE international conference on systems, man and cybernetics*, 2008, pp. 2603–2608.

[119] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[120] M. W. Nadeem *et al.*, “Brain tumor analysis empowered with deep learning: A review, taxonomy, and future challenges,” *Brain Sci.*, vol. 10, no. 2, 2020, doi: 10.3390/brainsci10020118.

[121] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[122] M. Z. Alom, V. Bontupalli, and T. M. Taha, “Intrusion detection using deep belief networks,” in *2015 National Aerospace and Electronics Conference (NAECON)*, 2015, pp. 339–344.

[123] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, “DDoS attacks in cloud computing: Issues, taxonomy, and future directions,” *Comput. Commun.*, vol. 107, pp. 30–48, 2017.

[124] K. S. Sahoo, S. K. Panda, S. Sahoo, B. Sahoo, and R. Dash, “Toward secure software-defined networks against distributed denial of service attack,” *J. Supercomput.*, vol. 75, no. 8, pp. 4829–4874, 2019.

[125] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, “Survey on SDN based network intrusion detection system using machine learning approaches,” *Peer-to-Peer Netw. Appl.*, vol. 12, no. 2, pp. 493–501, 2019.

[126] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, “JESS: Joint entropy-based DDoS defense scheme in SDN,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2358–2372, 2018.

[127] N. A. S. Lima and M. P. Fernandez, “Towards an efficient DDoS detection scheme for software-defined networks,” *IEEE Lat. Am. Trans.*, vol. 16, no. 8, pp. 2296–2301, 2018.

[128] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, “SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN,” *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 4, pp. 1545–1559, 2018.

[129] B. V Karan, D. G. Narayan, and P. S. Hiremath, “Detection of DDoS attacks in software defined networks,” in *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, 2018, pp. 265–270.

[130] R. Braga, E. Mota, and A. Passito, “Lightweight DDoS flooding attack detection using NOX/OpenFlow,” in *IEEE Local Computer Network Conference*, 2010, pp. 408–415.

[131] Y. Wang, T. Hu, G. Tang, J. Xie, and J. Lu, “SGS: Safe-guard scheme for protecting control plane against DDoS attacks in software-defined networking,” *IEEE Access*, vol. 7, pp. 34699–34710, 2019.

[132] X.-D. Zang, J. Gong, and X.-Y. Hu, “An adaptive profile-based approach for detecting anomalous traffic in backbone,” *IEEE Access*, vol. 7, pp. 56920–56934, 2019.

[133] Y. Xu, H. Sun, F. Xiang, and Z. Sun, “Efficient DDoS detection based on K-FKNN in software defined networks,” *IEEE access*, vol. 7, pp. 160536–160545, 2019.

[134] Q. Niyaz, W. Sun, and A. Y. Javaid, “A deep learning based DDoS detection system in software-defined networking (SDN),” *arXiv Prepr. arXiv1611.07400*, 2016.

[135] T. Hurley, J. E. Perdomo, and A. Perez-Pons, “HMM-based intrusion detection system for software defined networking,” in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 617–621.

[136] A. Alshamrani, A. Chowdhary, S. Pisharody, D. Lu, and D. Huang, “A defense system for defeating DDoS attacks in SDN based networks,” in *Proceedings of the 15th ACM international symposium on mobility management and wireless access*, 2017, pp. 83–92.

[137] D. Hu, P. Hong, and Y. Chen, “FADM: DDoS flooding attack detection and mitigation system in software-defined networking,” in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, 2017, pp. 1–7.

[138] A. Banitalebi Dehkordi, M. Soltanaghaei, and F. Z. Boroujeni, “The DDoS attacks detection through machine learning and statistical methods in SDN,” *J. Supercomput.*, vol. 77, no. 3, pp. 2383–2415, 2021.

[139] J. Li, Z. Zhao, R. Li, and H. Zhang, “Ai-based two-stage intrusion detection for software defined iot networks,” *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2093–2102, 2018.

[140] S. U. N. Guozi, W. Jiang, G. U. Yu, R. E. N. Danni, and L. I. Huakang, “DDoS attacks and flash event detection based on flow characteristics in SDN,” in *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*, 2018, pp. 1–6.

[141] V. Deepa, K. M. Sudar, and P. Deepalakshmi, “Design of ensemble learning methods for DDoS detection in SDN environment,” in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, pp. 1–6.

[142] T. V Phan and M. Park, “Efficient distributed denial-of-service attack defense in SDN-based cloud,” *IEEE Access*, vol. 7, pp. 18701–18714, 2019.

[143] M. Myint Oo, S. Kamolphiwong, T. Kamolphiwong, and S. Vasupongayya, “Advanced support vector machine-(ASVM-) based detection for distributed denial of service (DDoS) attack on software defined networking (SDN),” *J. Comput. Networks Commun.*, vol. 2019, 2019.

[144] J. Li, Y. Liu, and L. Gu, “DDoS attack detection based on neural network,” in *2010 2nd international symposium on aware computing*, 2010, pp. 196–199.

[145] Y. Cui *et al.*, “SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks,” *J. Netw. Comput. Appl.*, vol. 68, pp. 65–79, 2016.

[146] Y. Xu and Y. Liu, “DDoS attack detection under SDN context,” in *IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications*, 2016, pp. 1–9.

[147] J. Cui, J. He, Y. Xu, and H. Zhong, “TDDAD: Time-based detection and defense scheme against DDoS attack on SDN controller,” in *Australasian Conference on Information Security and Privacy*, 2018, pp. 649–665.

[148] C. Li *et al.*, “Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN,” *Int. J. Commun. Syst.*, vol. 31, no. 5, p. e3497, 2018.

[149] T. M. Nam *et al.*, “Self-organizing map-based approaches in DDoS flooding detection using SDN,” in *2018 International Conference on Information Networking (ICOIN)*, 2018, pp. 249–254.

[150] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proença, “Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment,” *IEEE Access*, vol. 8, pp. 83765–83781, 2020.

[151] R. Santos, D. Souza, W. Santo, A. Ribeiro, and E. Moreno, “Machine learning algorithms to detect DDoS attacks in SDN,” *Concurr. Comput. Pract. Exp.*, vol. 32, no. 16, p. e5402, 2020.

[152] L. Barki, A. Shidling, N. Meti, D. G. Narayan, and M. M. Mulla, “Detection of distributed denial of service attacks in software defined networks,” in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 2576–2581.

[153] K. B. Virupakshar, M. Asundi, K. Channal, P. Shettar, S. Patil, and D. G. Narayan, “Distributed denial of service (DDoS) attacks detection system for OpenStack-based private cloud,” *Procedia Comput. Sci.*, vol. 167, pp. 2297–2307, 2020.

[154] L. Yang and H. Zhao, “DDoS attack identification and defense using SDN based on machine learning method,” in *2018 15th international symposium on pervasive systems, algorithms and networks (I-SPAN)*, 2018, pp. 174–178.

[155] O. Rahman, M. A. G. Quraishi, and C.-H. Lung, “DDoS attacks detection

and mitigation in SDN using machine learning," in *2019 IEEE world congress on services (SERVICES)*, 2019, vol. 2642, pp. 184–189.

[156] M. Revathi, V. V Ramalingam, and B. Amutha, "A machine learning based detection and mitigation of the DDOS attack by using SDN controller framework," *Wirel. Pers. Commun.*, pp. 1–25, 2021.

[157] T.-K. Luong, T.-D. Tran, and G.-T. Le, "Ddos attack detection and defense in sdn based on machine learning," in *2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*, 2020, pp. 31–35.

[158] F. Khashab, J. Moubarak, A. Feghali, and C. Bassil, "DDoS attack detection and mitigation in SDN using machine learning," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 395–401.

[159] N. M. Yungaicela-Naula, C. Vargas-Rosales, and J. A. Perez-Diaz, "SDN-based architecture for transport and application layer DDoS attack detection by using machine and deep learning," *IEEE Access*, vol. 9, pp. 108495–108512, 2021.

[160] R. T. Kokila, S. T. Selvi, and K. Govindarajan, "DDoS detection and analysis in SDN-based environment using support vector machine classifier," in *2014 sixth international conference on advanced computing (ICoAC)*, 2014, pp. 205–210.

[161] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," *Futur. Gener. Comput. Syst.*, vol. 111, pp. 763–779, 2020.

[162] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, and J. Peng, "XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud," in *2018 IEEE international conference on big data and smart computing (bigcomp)*, 2018, pp. 251–256.

[163] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proença Jr, "Adversarial Deep Learning approach detection and defense against DDoS attacks in SDN environments," *Futur. Gener. Comput. Syst.*, vol. 125, pp. 156–167, 2021.

[164] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Commun.*, vol. 16, no. 7, pp. 144–155, 2019.

[165] L. Wang and Y. Liu, "A DDoS attack detection method based on information entropy and deep learning in SDN," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2020, vol. 1, pp. 1084–1088.

[166] Y. Chen, J. Pei, and D. Li, "DETPro: a high-efficiency and low-latency system against DDoS attacks in SDN based on decision tree," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[167] S. Yu, J. Zhang, J. Liu, X. Zhang, Y. Li, and T. Xu, "A cooperative DDoS attack detection scheme based on entropy and ensemble learning in SDN," *EURASIP J. Wirel. Commun. Netw.*, vol. 2021, no. 1, pp. 1–21, 2021.

[168] Y. Xu, Y. Yu, H. Hong, and Z. Sun, "DDoS detection using a cloud-edge collaboration method based on entropy-measuring SOM and KD-tree in SDN," *Secur. Commun. Networks*, vol. 2021, 2021.

[169] Z. Long and W. Jinsong, "A hybrid method of entropy and SSAE-SVM based DDoS detection and mitigation mechanism in SDN," *Comput. Secur.*, vol. 115, p. 102604, 2022.

[170] P. Preamthaisong, A. Auyporntrakool, P. Aimtongkham, T. Sriwuttisap, and C. So-In, "Enhanced DDoS detection using hybrid genetic algorithm and decision tree for SDN," in *2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2019, pp. 152–157.

[171] M. Wang, Y. Lu, and J. Qin, "Source-Based Defense Against DDoS Attacks in SDN Based on sFlow and SOM," *IEEE Access*, vol. 10, pp. 2097–2116, 2021.

[172] Z. Ma and B. Li, "A DDoS attack detection method based on SVM and K-nearest neighbour in SDN environment," *Int. J. Comput. Sci. Eng.*, vol. 23, no. 3, pp. 224–234, 2020.

[173] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and mitigating ddos attacks in SDN using spatial-temporal graph convolutional network," *IEEE Trans. Dependable Secur. Comput.*, 2021.

[174] A. Maheshwari, B. Mehraj, M. S. Khan, and M. S. Idrisi, "An optimized weighted voting based ensemble model for DDoS attack detection and mitigation in SDN environment," *Microprocess. Microsyst.*, vol. 89, p. 104412, 2022.

[175] J. Cui, J. Zhang, J. He, H. Zhong, and Y. Lu, "DDoS detection and defense mechanism for SDN controllers with K-Means," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 394–401.

[176] H. Nurwarsito and M. F. Nadhif, "DDoS Attack Early Detection and Mitigation System on SDN using Random Forest Algorithm and Ryu Framework," in *2021 8th International Conference on Computer and Communication Engineering (ICCCE)*, 2021, pp. 178–183.

[177] J. Tan, S. Jing, L. Guo, and B. Xiao, “DDoS detection method based on Gini impurity and random forest in SDN environment,” in *2021 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, 2021, pp. 601–606.

[178] H. Meigen and C. Yunqiang, “A DDoS attack detection method based on time series and random forest in SDN,” in *2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, 2021, pp. 323–327.

[179] X. Luo and R. K. C. Chang, “On a new class of pulsing denial-of-service attacks and the defense.,” 2005.

[180] A. Kuzmanovic and E. W. Knightly, “Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 75–86.

[181] A. Shevtekar, K. Anantharam, and N. Ansari, “Low rate TCP denial-of-service attack detection at edge routers,” *IEEE Commun. Lett.*, vol. 9, no. 4, pp. 363–365, 2005.

[182] E. Adi, Z. Baig, C. P. Lam, and P. Hingston, “Low-rate denial-of-service attacks against HTTP/2 services,” in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, 2015, pp. 1–5.

[183] W. Zhijun, X. Qing, W. Jingjie, Y. Meng, and L. Liang, “Low-rate DDoS attack detection based on factorization machine in software defined network,” *IEEE Access*, vol. 8, pp. 17404–17418, 2020.

[184] S. Rendle, “Factorization machines,” in *2010 IEEE International conference on data mining*, 2010, pp. 995–1000.

[185] “CIC Dos Dataset,” 2017.

[186] N. Zhang, F. Jaafar, and Y. Malik, “Low-rate DoS attack detection using PSD based entropy and machine learning,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2019, pp. 59–62.

[187] Z. Liu, X. Yin, and Y. Hu, “CPSS LR-DDoS detection and defense in edge computing utilizing DCNN Q-learning,” *IEEE Access*, vol. 8, pp. 42120–42130, 2020.

[188] K. M. Sudar and P. Deepalakshmi, “Flow-Based Detection and Mitigation of Low-Rate DDOS Attack in SDN Environment Using Machine Learning Techniques,” in *IoT and Analytics for Sensor Networks*, Springer, 2022, pp.

193–205.

[189] U. Wijesinghe, U. Tupakula, and V. Varadharajan, “Botnet detection using software defined networking,” in *2015 22nd International Conference on Telecommunications (ICT)*, 2015, pp. 219–224.

[190] F. Tariq and S. Baig, “Botnet classification using centralized collection of network flow counters in software defined networks,” *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 8, p. 1075, 2016.

[191] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, “Predicting network attack patterns in SDN using machine learning approach,” in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 167–172.

[192] F. Tariq and S. Baig, “Machine learning based botnet detection in software defined networks,” *Int. J. Secur. Appl*, vol. 11, no. 11, pp. 1–12, 2017.

[193] S.-C. Su, Y.-R. Chen, S.-C. Tsai, and Y.-B. Lin, “Detecting p2p botnet in software defined networks,” *Secur. Commun. Networks*, vol. 2018, 2018.

[194] D. Comaneci and C. Dobre, “Securing networks using SDN and machine learning,” in *2018 IEEE International Conference on Computational Science and Engineering (CSE)*, 2018, pp. 194–200.

[195] Y. Park, N. V. Kengalahalli, and S.-Y. Chang, “Distributed security network functions against botnet attacks in software-defined networks,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018, pp. 1–7.

[196] S. Maeda, A. Kanai, S. Tanimoto, T. Hatashima, and K. Ohkubo, “A botnet detection method on SDN using deep learning,” in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–6.

[197] I. Letteri, M. Del Rosso, P. Caianiello, and D. Cassioli, “Performance of Botnet Detection by Neural Networks in Software-Defined Networks.,” 2018.

[198] S. Y. Khamaiseh, I. Alsmadi, and A. Al-Alaj, “Deceiving machine learning-based saturation attack detection systems in sdn,” in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, pp. 44–50.

[199] S. Khamaiseh, E. Serra, Z. Li, and D. Xu, “Detecting saturation attacks in sdn via machine learning,” in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–8.

[200] A. Abusnaina, A. Khormali, D. Nyang, M. Yuksel, and A. Mohaisen,

"Examining the robustness of learning-based ddos detection in software defined networks," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, 2019, pp. 1–8.

[201] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[202] N. Papernot *et al.*, "Technical report on the cleverhans v2. 1.0 adversarial examples library," *arXiv Prepr. arXiv1610.00768*, 2016.

[203] X. Huang, K. Xue, Y. Xing, D. Hu, R. Li, and Q. Sun, "FSDM: Fast recovery saturation attack detection and mitigation framework in SDN," in *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2020, pp. 329–337.

[204] H.-Y. Chang, T.-L. Lin, T.-F. Hsu, Y.-S. Shen, and G.-R. Li, "Implementation of ransomware prediction system based on weighted-KNN and real-time isolation architecture on SDN Networks," in *2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, 2019, pp. 1–2.

[205] G. Cusack, O. Michel, and E. Keller, "Machine learning-based detection of ransomware using SDN," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2018, pp. 1–6.

[206] C. Thapa, K. K. Karmakar, A. H. Celdran, S. Camtepe, V. Varadharajan, and S. Nepal, "FedDICE: A ransomware spread detection in a distributed integrated clinical environment using federated learning and SDN based mitigation," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2021, pp. 3–24.

[207] P. V Shalini, V. Radha, and S. G. Sanjeevi, "Early detection and mitigation of TCP SYN flood attacks in SDN using chi-square test," *J. Supercomput.*, pp. 1–33, 2023.

[208] K. Subratie, S. Aditya, and R. J. Figueiredo, "EdgeVPN: Self-organizing layer-2 virtual edge networks," *Futur. Gener. Comput. Syst.*, vol. 140, pp. 104–116, 2023.

[209] S. A. Wagan, J. Koo, I. F. Siddiqui, N. M. F. Qureshi, M. Attique, and D. R. Shin, "A fuzzy-based duo-secure multi-modal framework for IoMT anomaly detection," *J. King Saud Univ. Inf. Sci.*, vol. 35, no. 1, pp. 131–144, 2023.

[210] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw.*

Comput. Appl., vol. 169, p. 102767, 2020.

[211] T. H. H. Aldhyani and H. Alkahtani, "Cyber Security for Detecting Distributed Denial of Service Attacks in Agriculture 4.0: Deep Learning Model," *Mathematics*, vol. 11, no. 1, p. 233, 2023.

[212] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del-Rincon, and D. Siracusa, "LUCID: A practical, lightweight deep learning solution for DDoS attack detection," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 2, pp. 876–889, 2020.

[213] D. K. Thara, B. G. PremaSudha, and F. Xiong, "Auto-detection of epileptic seizure events using deep neural network with different feature scaling techniques," *Pattern Recognit. Lett.*, vol. 128, pp. 544–550, 2019.

[214] M. Sheikhan, Z. Jadidi, and A. Farrokhi, "Intrusion detection using reduced-size RNN based on feature grouping," *Neural Comput. Appl.*, vol. 21, pp. 1185–1190, 2012.

[215] C. Yue, L. Wang, D. Wang, R. Duo, and X. Nie, "An ensemble intrusion detection method for train ethernet consist network based on CNN and RNN," *IEEE Access*, vol. 9, pp. 59527–59539, 2021.

[216] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[217] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv Prepr. arXiv2106.11342*, 2021.

[218] A. I. Georgevici and M. Terblanche, "Neural networks and deep learning: a brief introduction," *Intensive Care Med.*, vol. 45, no. 5, pp. 712–714, 2019.

[219] H. Zheng, G. Wang, and X. Li, "Swin-MLP: A strawberry appearance quality identification method by Swin Transformer and multi-layer perceptron," *J. Food Meas. Charact.*, vol. 16, no. 4, pp. 2789–2800, 2022.

[220] F. Sharifzadeh, G. Akbarizadeh, and Y. Seifi Kavian, "Ship classification in SAR images using a new hybrid CNN–MLP classifier," *J. Indian Soc. Remote Sens.*, vol. 47, pp. 551–562, 2019.

[221] P. Devan and N. Khare, "An efficient XGBoost–DNN-based classification model for network intrusion detection system," *Neural Comput. Appl.*, vol. 32, pp. 12499–12514, 2020.

[222] S. P. RM *et al.*, "An effective feature engineering for DNN using hybrid PCA-GWO for intrusion detection in IoMT architecture," *Comput. Commun.*, vol. 160, pp. 139–149, 2020.