Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Babylon
Collage of Information Technology
Department of Information Networks

DEVELOPED DEEP NEURAL NETWORK CROSS-LEVEL DOS/DDOS ATTACK DETECTION IN NETWORK SLICING

A Dissertation

Submitted to the Council of the College of Information Technology for Postgraduate Studies of University of Babylon in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Information Technology-Information Networks

Suadad Safaa Mahdi

Supervised by

Asst. Prof. Dr. Alharith A. Abdullah

2023A.D. **1445 A.H.**

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

﴿ وَيَسْأَلُونَكَ عَنِ الرُّوحِ قُلِ الرُّوحُ مِنْ أَمْرِ رَبِّي وَمَا أُوتِيتُمْ مِنَ الْعِلْمِ إِلَّا قَلِيلًا ﴾

سورة الاسراء ﴿٨٥﴾

صدقَ اللَّهُ العلي العظيم

Supervisor Certification

I certify that the dissertation entitled (**Developed Deep Neural Network Cross-Level DoS/DDoS Attack Detection in Network Slicing**) was prepared under my supervision at the department of Information Networks/ College of Information Technology/ University of Babylon as partial fulfillment of the requirements of the degree of Doctor of Philosophy in Information Technology-Information Networks.

Signature:
Supervisor Name: Asst. Prof. Dr. Alharith A. Abdullah
Date: / /2023

The Head of the Department Certification

In view of the available recommendations, I forward the dissertation entitled **(Developed Deep Neural Network Cross-Level DoS/DDoS Attack Detection in Network Slicing)** for debate by the examination committee.

Signature:
Asst. Prof. Dr. Alharith A. Abdullah
Head of Information Networks Department
Date: / /2023

Dedication

“Dedicated to My Father's Soul”

In memory of my beloved father, I dedicate this dissertation to his eternal soul.

Although his physical presence no longer graces this world, his spirit remains a constant source of inspiration in my life.

This dissertation stands as a testament to his profound influence on my life and my unwavering commitment to honoring his memory through my academic endeavors.

I humbly dedicate this dissertation to my wonderful father, whose influence on my heart and my life will remain indelible forever.

Suadad S. Mahdi

/ /2023

Acknowledgement

In the name of God, Most Gracious, Most Merciful.

Firstly, I would like to praise Allah for assisting me in facing the difficulties I encountered during my studies and for always helping me achieve my goals.

I would like to express my deepest gratitude and appreciation to all those who supported me throughout the completion of this dissertation.

First and foremost, I am immensely grateful to my supervisor, **(Dr. Alharith A. Abdullah)**, for his guidance, expertise, and unwavering support. I am truly fortunate to have had the opportunity to work under his supervision.

I would also like to extend my heartfelt appreciation to the faculty members of the College of Information Technology at the University of Babylon, whose dedication and commitment to academic excellence have provided me with a nurturing learning environment.

I am deeply grateful to my family for their unwavering love, encouragement, and belief in my abilities. Their constant support and understanding have been a source of strength and motivation throughout this challenging journey.

I would like to take this opportunity to express my heartfelt appreciation and love to my dear friend, **Shahad A. Hussain**. I am fortunate to have such a remarkable friend who stood by my side, offering guidance and reassurance when I needed it the most.

Lastly, I would like to acknowledge all the researchers and scholars whose work has laid the foundation for this study. Their contributions to the field have been fundamental in shaping my understanding and guiding my research endeavors.

Abstract

Network Slicing (NS) plays an essential role in the 5G network architecture as it involves dividing the physical network infrastructure into multiple virtual networks or slices, each designed to meet the specific requirements of different applications, industries, or user groups. Each network slice operates as an independent logical network with its own set of resources, configurations, and performance characteristics. While NS technology allows for efficient and flexible allocation of network resources based on the needs of different use cases, it is not without challenges, especially in the area of security.

One of the main security concerns related to NS is the risk of Denial-of-Service (DoS) attacks and Distributed Denial-of-Service (DDoS) attacks. These attacks involve flooding a network or service with a torrent of traffic from multiple sources, making it unavailable to legitimate users. In NS environment, DoS/DDoS attacks can disrupt the availability and performance of network slices, affecting not only the targeted slice but potentially other slices that share the same physical infrastructure.

This dissertation primarily focuses on the detection of attacks within NS environment, and the work is divided into three main parts. The first part involves the creation of NS environment using software-defined networks and network virtualization, which includes two network slices with dedicated controllers for each slice. The second part focuses on the creation of a specialized dataset called the Network Slicing Dataset (NSDS), specifically tailored for NS. This dataset effectively replicates normal network traffic scenarios as well as DoS/DDoS attack traffic within NS environment. Finally, the core of the dissertation revolves around an innovative methodology that utilizes the NSDS to detect attacks through statistical techniques and deep neural networks, enabling accurate detection of attacks within the NS environment.

The architecture of the innovative methodology consists of two levels of attack detection: slice-level and cross-level. At the slice-level, statistical analysis techniques based on joint entropy and dynamic thresholds are used for the early detection of attacks within each network slice. Moving to cross-level detection, the architecture extends detection across multiple slices of the network. At this level, attacks are detected using a Developed Deep Neural Network (DDNN) model that was trained and tested using two distinct datasets: the well-known CICIDS2017 dataset and NSDS. The DDNN model incorporates a new activation function called SETAF, which combines elements from the sigmoid and exponential functions along with a dynamic threshold.

In the implementation of the emulator, the network slicing testbed was established using the Mininet network emulator for infrastructure emulation. The FlowVisor served as the virtualization layer on the infrastructure components, enabling the creation of two network slices. Each network slice was controlled by a dedicated POX controller. At the slice level, statistical analysis was performed by individual controllers for each network slice. Furthermore, the execution of the DDNN model took place on a third-party server, which was linked to each network slice. The emulator results obtained from the network slicing testbed provided solid confirmation of the methodology's effectiveness in detecting attacks, achieving a remarkable detection accuracy rate of 0.99 within 49 milliseconds. Consequently, the results of this dissertation clearly prove that the methodology is effective in detecting attacks and maintaining network slicing security in a 5G network.

Declaration Associated with this Dissertation

(First Paper)

- **Name of Journal:** Infocommunications Journal
- **Paper Title:** Enhanced Security of Software-defined Network and Network Slice Through Hybrid Quantum Key Distribution Protocol
- **Publication:** Scientific Association for Infocommunications, Hungary (HTE), a Sister Society of IEEE and IEEE Communications Society
- **Authors:**
  - Suadad S. Mahdi
  - Alharith A. Abdullah

(Second Paper)

- **Name of Conference:** Information Systems and Intelligent Applications (ICISIA2022)
- **Paper Title:** Survey on Enabling Network Slicing Based on SDN/NFV
- **Publication:** Springer Book Series (Lecture Notes in Networks and Systems)
- **Authors:**
  - Suadad S. Mahdi
  - Alharith A. Abdullah

(Third Paper)

- **Name of Conference:** International Conference on New Trends in Information and Communications Technology Applications (NTICT 2022)
- **Paper Title:** Implementation of Network Slicing for Multi-controller Environment Based on FlowVisor
- **Publication:** Springer Book Series (Communications in Computer and Information Science)
- **Authors:**
  - Suadad S. Mahdi
  - Alharith A. Abdullah

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

Abbreviation	Description
3GPP	Third Generation Partnership Project
4G LTE	Fourth Generation Long-Term Evolution
4K	4K Resolution
5G	Fifth Generation
ACL	Access Control List
Adagrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
ADMM	Alternating Direction Method of Multipliers
ADR	Attack Detection Rate
AMMF	Access and Mobility Management Function
API	Application Programming Interface
AWT	Average Waiting Time
BF	Bloom Filters
CAVs	Connected Autonomous Vehicles
CCA	Canonical Correlation Analysis
CN	Core Network
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DDNN	Developed Deep Neural Network
DDoS	Distributed Denial of Service
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service

Dst IP	Destination IP Address
Dst MAC	Destination MAC Address
Dst Port	Source Port
Dst Port	Destination Port
FAR	False Alarm Rate
FDIA	False Data Injection Attacks
FL	Federated Learning
FN	False Negative
FP	False Positive
FPR	False Positive Rate
Fwd IAT	Forward Inter-Arrival Time
GD	Gradient Descent
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
KD	Knowledge Distillation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
NFVM	Network Function Virtualization Manager
NFVO	Network Function Virtualization Orchestrator
NOS	Network Operating System
NS	Network Slicing
NSDS	Network Slicing Dataset
NSM	Network Slice Manager
NV	Network Virtualization

OCSVM	One-Class Support Vector Machine
P4	Programming Protocol-Independent Packet Processors
PAT	Packet Arrival Time
pkt size	Packet Size
PLs	Physical Links
PNs	Physical Nodes
QoS	Quality of Service
RAN	Radio Access Network
ReLU	Rectified Linear Unit
RMS	Root Mean Squared
RTT	Round-Trip Time
SDN	Software-Defined Networking
SDNO	Software-Defined Networking Orchestrator
SETAF	Sigmoid-Exponential-Threshold Activation Function
SFC	Service Function Chaining
SGD	Stochastic Gradient Descent
Src IP	Source IP Address
Src MAC	Source MAC Address
Tanh	Hyperbolic Tangent Function
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
UE	User Equipment
VIM	Virtualized Infrastructure Manager

VLAN	Virtual Local Area Network
VM	Virtual Machine
VNFs	Virtual Network Functions
VNs	Virtual Networks

List of Symbols

Symbol	Description
$H(X)$	Entropy of the random variable X
$\sum$	Summation symbol
p(x)	Probability distribution function of the random variable x
$\log_2$	Logarithm function with base 2
$\lvert\ \rvert$	Absolute value of a quantity
μ	Mean or average value
σ	Standard deviation
K	A constant that represents the number of standard deviations from the mean
W	Weight associated with input
b	Bias term or constant offset in the equation
f(x)	The function notation, representing the output or value of the function for a given input x.
L	Represent the loss function
max	The maximum function, which returns the greater value
e	The mathematical constant e, approximately equal to 2.71828, representing the base of the natural logarithm.
/	Division operator, used to divide one value by another
y	The actual or observed value of the dependent variable
y'	The predicted or estimated value of the dependent variable
log	Natural logarithm (base e)
[]	Square brackets, used to group terms together

min	The minimum function, which returns the minimum value
H (x, y)	The joint entropy of the random variables x and y
P(x,y)	The joint probability function of x and y
$H_{norm(x,y)}$	The normalized joint entropy of the random variables x and y
nWin	The number of elements in the window
Sig(x)	The sigmoid function applied to the variable x
$\propto$	The Greek letter Alpha represents a scaling factor
Exp(x)	Exponential function applied to the variable x
d	Squared difference between two values
δ	The Greek letter Delta represents a dynamic threshold
β	The Greek letter Beta represents a scaling factor

CHAPTER ONE

General Introduction

1.1 Background and Motivation

The advent of 5G, the fifth generation of cellular technology, has introduced significant advancements in terms of data speeds, bandwidth, and latency compared to its predecessor, 4G LTE[1]. The development of 5G began in the early 2010s, and in 2018, the first official standard was released by 3GPP, defining the technical specifications for 5G networks, including frequency bands, modulation schemes, and network architecture [2]. Starting in 2019, several countries initiated the initial rollout of 5G networks in limited areas.

One of the key advantages of 5G networks lies in their ability to support a considerably higher density of connected devices. This is made possible through the implementation of advanced network slicing techniques, allowing the network to be divided into multiple virtual networks, each with distinct performance characteristics [3]. As a result, 5G networks can cater to a wide range of use cases, spanning from low-latency applications like autonomous vehicles and industrial automation to high-bandwidth applications such as 4K video streaming and virtual reality experiences [4].

Network slicing represents a new approach to managing and provisioning network resources in the era of 5G and beyond [3]. It enables network operators to partition the network into multiple virtual slices, with each slice possessing its own dedicated set of resources and functionalities, addressing the diverse requirements of different applications and users. By leveraging network slicing, each slice can be tailored to specific use cases, such as autonomous vehicles, virtual reality,

and industrial automation, accommodating varying needs for bandwidth, latency, reliability, and security [5]. Additionally, network operators can dynamically allocate resources to each slice based on the demands of the applications, ensuring efficient utilization of network resources and enhancing the overall user experience [6].

However, while NS brings about numerous benefits, including resource efficiency and improved user satisfaction, it also introduces new security challenges that must be carefully addressed [7]. One of the significant security challenges associated with NS is the occurrence of DoS/DDoS attacks [8]. DoS/DDoS attacks involve malicious attempts to disrupt the normal traffic of a targeted server, service or network by overwhelming it with an excessive flood of internet traffic originating from multiple sources [7]. These attacks can lead to significant disruptions in network services, resulting in degraded user experiences and potential revenue losses for network operators.

The utilization of NS poses a particular challenge when it comes to detecting and mitigating DoS/DDoS attacks, primarily due to the shared infrastructure among multiple slices [9]. An attack targeting a single slice has the potential to impact the entire network, resulting in service disruptions and compromised overall network performance [7]. To overcome this challenge, network operators must implement effective DoS/DDoS detection and mitigation systems capable of identifying and eliminating malicious traffic in real-time. Such systems are essential for maintaining the integrity, availability and security of network services in the face of evolving security threats.

1.2 Dissertation Scope

The scope of the dissertation work is to focus on detecting DoS/DDoS attacks in a network slicing environment. It involves a new two-level methodology for detecting these attacks, which includes using statistical methods and a DDNN model to identify and detect attacks targeting specific network slices or multiple slices simultaneously.

1.3 Related Works

In this section, the focus is on the importance of previous studies that have been conducted to detect and mitigate DoS/DDoS attacks on NS in 5G networks. As the security of network slices remains a critical concern, researchers are actively exploring new methods and techniques to enhance the security against DoS/DDoS attacks. As a result, ongoing studies continue to explore various approaches with the goal of improving DoS/DDoS detection capabilities and implementing effective mitigation strategies.

Sattar, Danish, and Ashraf Matrawy, 2019 [10] focused on achieving effective isolation between and within slices, effectively separating host hardware resources, the researchers proposed mapping each slice to a dedicated Virtual Machine (VM), particularly when multiple Virtual Network Functions (VNFs) from different slices were involved, to ensure isolation between slices. However, their research assumed a single VNF per VM, which still provided some level of isolation. Nevertheless, the study emphasized the importance of complete hardware separation to achieve isolation between slices, ensuring that

components from different slices were not allocated to the same host. For evaluation, they conducted tests using six physical servers. Among these servers, three were dedicated to managing the chipset, two servers were configured as DDoS nodes, and one server acted as a client. The client utilized ns3 to simulate the radio aspect of the 5G network. Results showed that the proposed isolation could mitigate DDoS attacks as well as increase the availability of the slices. The evaluation results included metrics such as combined response time, Round-Trip Time (RTT) between the client and the slice, and the average chip bandwidth available to the client.

Bonfim, Michel, et al., 2020 [11] introduced the FrameRTP4 framework as a solution for real-time attack detection and mitigation in 5G network slicing scenarios. The framework, based on the Software-Defined Networking (SDN) architecture, adopts a division between the data and control planes. The data plane utilizes P4, a data plane programming technology, to enable efficient processing within switches. FrameRTP4 incorporates a customizable P4 program that incorporates Service Function Chaining (SFC) to manage network slice instances. Moreover, the program includes a scalable and effective P4 table-based Access Control List (ACL) that can identify and mitigate known attacks using wildcard rules. To enhance network flow tracking, FrameRTP4 implements SFCMon, a monitoring system utilizing Bloom Filters (BFs) and sketches. In terms of the control plane, FrameRTP4 features a Python-based controller with customizable modules and a Northbound Application Programming Interface (API). This controller facilitates the lifecycle management of network slice instances and wildcard rules

within the P4 table-based ACLs, providing flexibility and control over the network slicing environment.

Moudoud, Hajar, et al., 2020 [12] proposed a hierarchical architecture and a security model aimed at predicting and detecting False Data Injection Attacks (FDIA) and DDoS attacks in 5G-enabled IoT networks. To achieve this, they monitored the behavior of individual network devices through a Markov stochastic process and integrated a range-based behavior-sifting policy into the security model. By analyzing device log files and considering activities like reading, updating, and deleting, the researchers classified device behaviors into different states. They determined threshold values for each device state by examining historical log profiles. To track the evolution of device behavior over time, the model employed a stochastic state transition matrix that captured transitions between states. The research provided empirical evidence to support the effectiveness of the proposed architecture and model in detecting and predicting FDIA and the attacks in 5G-enabled IoT networks. The researchers conducted an experiment using real log activity from an online mobile health application that recorded the activities performed by phone sensors. This utilization of real-world data further validated the practical applicability of the proposed model in securing IoT networks against potential attacks.

Thantharate, Anurag, et al., 2020 [13] developed a neural network-based "Secure5G" network slicing model to proactively detect and eliminate threats in the 5G core network. The "Secure5G" model is based on a Deep Neural Network (DNN) with 4 layers that analyze overall traffic patterns and can predict future traffic. It focuses on the main

objective of verifying connection requests for legitimacy and potential threats and taking appropriate actions such as assigning a network slice for valid requests or transferring to a quarantine slice for malicious requests. Secure5G protects against volume-based flooding attacks and attempts by hackers to exploit low-security network slice instances by masking device identities. The model's performance was evaluated using volume-based flooding and spoofing attack scenarios, achieving a detection accuracy of over 98% with their limited dataset.

Kuadey, Noble Arden Elorm, et al., 2021 [14] introduced DeepSecure as an innovative framework. DeepSecure utilized the Long Short-Term Memory (LSTM) deep learning technique to detect DDoS attacks and allocate suitable slices for legitimate User Equipment (UE) requests. The framework comprised two models: the attack detection model, which employed LSTM to classify network traffic as normal or DDoS, and the slice prediction model, which utilized LSTM to determine the most appropriate slice based on UE-specific features. To validate the effectiveness of the framework, an evaluation was conducted on the CICDDoS2019 dataset, which encompassed a diverse range of DDoS attacks. The results were remarkable, achieving an accuracy rate of 99.970% in DDoS attack detection and 98.798% accuracy in accurately predicting requested slices.

Bousalem, Badre, et al., 2022 [15] introduced a Deep Learning (DL) solution with the aim of effectively detecting and countering DDoS attacks in 5G networks. Their DL models were developed using supervised learning and Convolutional Neural Networks (CNN), and implemented using popular tools like Tensorflow, Keras, and Lucid. To

validate the effectiveness of their approach, the researchers constructed a 5G prototype and integrated OpenAirInterface, an open-source software. The integration of the northbound API with the FlexRAN SDN controller facilitated efficient management of network slicing. Additionally, OpenAirInterface successfully emulated crucial elements of cellular networks, including the Core Network (CN) and the Radio Access Network (RAN). Through their evaluation, the study's findings demonstrated the efficacy of their proposed approach, achieving an impressive detection accuracy rate of nearly 97% and a remarkably low false positive rate of less than 4% in promptly identifying DDoS attacks.

Wang, Weili, et al., 2022 [16] introduced two distributed online anomaly detection algorithms for the virtualized network slicing environment. The first algorithm made use of a decentralized One-Class Support Vector Machine (OCSVM) to detect anomalies in virtual nodes that were distributed mapped to Physical Nodes (PNs). The detection process involved solving decentralized quadratic programming problems with consensus constraints using the Alternating Direction Method of Multipliers (ADMM). In contrast, the second algorithm employed Canonical Correlation Analysis (CCA) to identify anomalies in Physical Links (PLs) by analyzing the correlation between measurements of neighboring virtual nodes. To evaluate the performance of these algorithms, they included four simulated anomaly cases that covered CPU endless loops, memory leaks, disk I/O faults, and network congestion in the PNs. Additionally, the algorithms were applied to a real-world network dataset obtained from IEEEDataPort, which provided metrics on CPU usage, memory usage, disk performance, and network activity. The effectiveness of the algorithms in detecting and identifying anomalies in

the virtualized network slicing environment was assessed based on these evaluations.

Khan, Md Sajid, et al., 2022 [17] introduced SliceSecure, a DL-based bidirectional LSTM model aimed at detecting DoS/DDoS attacks. The model capitalized on the LSTM's ability to capture long-term dependencies within recurrent neural networks. To showcase the impact of DoS/DDoS attacks on 5G network slices, the authors simulated a 5G network slice testbed utilizing Free5GC and UERANSIM. They generated datasets encompassing benign and DDoS attack traffic from the testbed and extracted pertinent features. By utilizing the newly generated dataset, which consisted of 11 features, the researchers implemented and evaluated the SliceSecure model, achieving a remarkable success rate of 99.99% in accurately classifying benign and DoS/DDoS attack traffic.

Hossain, Shajjad, et al., 2023 [18] proposed a lightweight intrusion detection scheme using DL techniques and Knowledge Distillation (KD) in the context of 5G vehicle-to-everything networks. The scheme involved training models in the cloud using DL and transferring the acquired knowledge to lightweight DL models running on Connected Autonomous Vehicles (CAVs). CNN was utilized on CAVs connected to a single network slice, and the VeReMi Extension dataset was used for training and evaluation. While deploying resource-intensive DL-based intrusion detection systems across multiple network slices in a multi-slice scenario posed challenges, the proposed scheme effectively achieved a balance between detection accuracy and security overhead, with a detection accuracy of 98%.

Bisht, Himanshu, Moumita Patra, and Sathish Kumar, 2023 [19] focused on detecting DDoS attacks in 5G network slicing. They proposed innovative algorithms for detecting and localizing these attacks by exploiting the authentication process during inter-slice handover. The system model consists of multiple network slices, some shared and others isolated, with the aim of disrupting a target slice's services. The detection mechanism involves monitoring traffic flow, specifically slice switching requests, and periodically evaluating the Average Waiting Time (AWT) parameter for anomalies. Upon detecting a potential attack based on AWT, the system proceeds to localize the attacker's devices, known as bot devices, using the average switching rate parameter and a calculated threshold value. Identified bot users are then reported to the 5G core for blocking, preventing further requests. The simulations were conducted using a Java-based simulator, and experimental results demonstrate a 91% accuracy in attack detection and 96% accuracy in identifying compromised users.

Sedjelmaci, Hichem, and Abdelwahab Boualouache, 2023[20] proposed a two-layer Federated Learning (FL)-based approach to protect the slicing of the 5G network and detect both internal and external attacks. The first FL layer consists of defense systems activated at nodes as FL clients and defense systems activated at edge servers as FL servers, while the second layer consists of defense systems activated at edge servers as FL clients and defense systems activated at Access and Mobility Management Function (AMF) as an FL server to aggregate the global training model. The approach's performance was assessed using the CSE-CIC-IDS2018 dataset, and the results indicated that the cooperative defense systems based on FL and a mean-field game demonstrated high

detection accuracy and resilience against DoS, Botnet, and poisoning attacks.

Table 1.1 provides a comprehensive overview of DoS/DDoS security approaches in a network slicing environment, summarizing their essential characteristics. It enables readers to conveniently compare these approaches and understand how they are implemented using various tools.

Table 1.1: Main Characteristics of Security Methods Against DoS/DDoS Attacks in Network Slicing

Proposed Approach	Detection Model	Implementation	Result	Dynamic Adaptive
[10]	**Mathematical**	**Slicing Testbed**	**No Attack Detection Result, RTT ($10^{-1} - 10^{0}$)ms, Bandwidth (8-12 Mbps), Response time (0-10) ms**	**No**
[11]	**ACL**	**Open-Source Framework**	**-**	**No**
[12]	**Statistical**	**Dataset**	**Detection Rate (70 -99) % based on number of attack activity, Error (40-10) %**	**No**
[13]	**DL**	**Dataset**	**Detection Accuracy 98%**	**No**
[14]	**DL**	**Dataset**	**Detection Accuracy 99.970%**	**No**
[15]	**DL**	**Open-Source Framework**	**Detection Accuracy 97%, False Positive Rate 4%**	**No**
[16]	**ML**	**Dataset**	**F-Score (95 -98) %**	**No**
[17]	**DL**	**Dataset**	**Detection Accuracy 99.99%**	**No**

[18]	**DL**	**Dataset**	**Detection Accuracy 98%**	**No**
[19]	**Mathematical**	**Dataset and Open-Source Framework**	**Detection Accuracy 91%**	**No**
[20]	**FL**	**Dataset**	**Detection Accuracy 96%**	**No**

1.4 Problem Statement

In a network slicing environment, several challenges arise when it comes to detecting DoS/DDoS attacks. The following is a summary of the key problem statements addressed in this dissertation:

1) Building effective attack detection models for DoS/DDoS attacks in network slicing is challenging due to the absence of datasets that are comprehensive, up-to-date, and designed specifically for simulating such attacks in network slicing environments.
2) DoS/DDoS attacks are a significant threat in a network slicing environment as they put both virtual network resources and underlying physical infrastructure at risk. The underlying problem lies in the fact that these attacks have the potential to target specific network slices individually or simultaneously affect multiple slices, both scenarios posing different problems.
    a) When a DoS/DDoS attack focuses on a specific network slice, the primary problem is the unavailability of that targeted slice to legitimate users. The attack overwhelms the resources allocated to the targeted slice, rendering it inaccessible and disrupting the services that rely on it.
    b) DoS/DDoS attacks on multiple slices can also be more challenging to detect, as the attack may be distributed across

multiple slices and may require a coordinated response from multiple network operators and service providers.

3) The dynamic nature of network traffic behavior poses a challenge for DoS/DDoS detection systems as it is necessary for them to be dynamically adapted in real-time to changing traffic rates in order to accurately identify potential DoS/DDoS attacks.

1.5 Dissertation Question

Despite the significant role that network slicing technology plays in modern communications, the existing solutions to its security challenges have not fully addressed all the pertinent issues. Therefore, the main goal of this dissertation is to provide comprehensive answers to the following questions:

1) What are the possible DoS/DDoS attack scenarios that can occur in a network slicing environment?
2) How does the network slicing environment get affected by DoS/DDoS attacks?
3) Is it possible for statistical analysis to detect a DoS/DDoS attack within a network slicing environment?
4) How is traffic analyzed across multiple slices to detect the DoS/DDoS attack that targets more than one slice simultaneously?
5) Is a Deep Neural Network compatible with traffic dynamics for detecting DoS/DDoS attacks?

1.6 Dissertation Objectives

The objectives of this dissertation can be described as follows:

1) Building a realistic testbed: To emulate a NS environment and create two virtual networks (or slices) on a physical network infrastructure.
2) Traffic generation and dataset creation: This involves generating normal traffic as well as DoS/DDoS attack traffic within the NS testbed and collecting this network traffic to create a comprehensive dataset that captures the characteristics of both normal and attack network traffic in an NS environment.
3) Early detection of DoS/DDoS attacks: The objective is to analyze traffic statistically in each network slice and detect suspected attack traffic when it occurs.
4) Detection of DoS/DDoS attacks across multiple slices: The main objective is to detect attacks that target one or more slices simultaneously through the use of a third-party server equipped with a detection model. This model analyzes network traffic data to detect coordinated attacks, which enhances the security of the NS environment.

1.7 Dissertation Contribution

The contributions of this dissertation can be summarized as follows:

1) Generating a dataset that simulates both normal traffic and DoS/DDoS attacks traffic in a network slicing environment as a contribution to network slicing and security.
2) Constructing an innovative methodology for detecting DoS/DDoS attacks. This methodology consists of two levels: the slice-level and the cross-level.
    a) Slice-level early attack detection by dynamic calculation of normal traffic threshold using Chebyshev's theorem. The threshold is proportional to observed traffic patterns and is used in attack detection based on joint entropy, allowing adaptation to changing conditions and reducing false positives.
    b) Cross-level deep attack detection is carried out through a third-party server that analyzes the traffic of multiple slices simultaneously. This server utilizes a DDNN model that incorporates a novel activation function called SETAF.
3) A novel Sigmoid Exponential Threshold Activation Function (SETAF) is proposed to develop DNN. SETAF combines the properties of sigmoid, exponential, and threshold activation functions to improve the performance of DNN and make the model adaptable to dynamic changes in traffic.

1.8 Dissertation Organization

This dissertation contains six chapters organized as follows:

Chapter One includes general background on 5G networks, network slicing technology, and security problems associated with

network slices. It also defines the scope of the dissertation and reviews related works, as well as the questions and problems addressed in the dissertation. Finally, it explains the main objectives of the dissertation and its contributions.

Chapter Two presents the theoretical part of the dissertation. The chapter begins by delving into the concept of network slicing and exploring its related techniques. It provides an in-depth explanation of SDN and network virtualization, highlighting their significance in the context of network slicing. Furthermore, the chapter explores various network slicing scenarios. On the other hand, addressing the crucial aspect of security in network slicing, the chapter examines the associated challenges and focuses specifically on the detection of DoS/DDoS attacks. Existing techniques for detecting these attacks are discussed, with an emphasis on statistical techniques and neural networks as potential solutions. To support the dissertation, the chapter concludes by explaining the tools and software utilized, providing insights into their functionalities and relevance to the dissertation.

Chapter Three consists of the basic steps of the proposed methodology for detecting DoS/DDoS attacks in a network slicing environment. These steps include elucidating the proposed network slicing architecture to build a real network slicing testbed, as well as the mechanism used to simulate normal traffic and attack traffic. Finally, the chapter shows how to construct our dataset based on the network slicing testbed.

Chapter Four describes the proposed methodology for detecting DoS/DDoS attacks in a network slicing environment. This methodology

consists of two levels of detection: slice-level and cross-level. The chapter covers these two levels in detail, including the steps for building a statistical analysis of traffic at the slice-level and a DDNN model to detect cross-level attacks. Finally, the chapter discusses the integration of these two levels to detect attacks in a network slicing environment.

Chapter Five describes the implementation of a dedicated test environment for network slicing and traffic generation. The chapter then proceeds to focus on the testing and evaluation phase of the proposed methodology for detecting DoS/DDoS attacks. Results obtained from the development and testing of this methodology, including metrics such as accuracy, precision, and F-score, are presented and analyzed. Finally, the chapter concludes with a discussion of the results obtained, along with a comparative analysis with previous works that closely align with the dissertation's focus.

Chapter Six presents the main conclusions derived from the dissertation work. This section provides a comprehensive summary of the key outcomes and insights obtained throughout the dissertation. Additionally, the chapter offers valuable suggestions and recommendations for future research directions. These suggestions highlight potential areas for further investigation and improvement, building upon the findings of the current work.

CHAPTER TWO

Theoretical Background

2.1 Overview

This chapter focuses on the theoretical aspects of the dissertation, providing a comprehensive explanation of NS. The fundamental concepts of NS are covered, including its architecture, the life cycle of network slices, and the basic classifications utilized in this technology. Additionally, the chapter explores associated techniques like SDNs and NV, which play a crucial role in enabling network slicing. Moving forward, the chapter delves into NS implementation scenarios, discussing various contexts and use cases where network slicing can be effectively applied. Furthermore, the chapter addresses the critical aspect of network slicing security, with a specific focus on DoS/DDoS attacks. It outlines the mechanisms employed in this dissertation to detect such attacks, encompassing a review of statistical-based detection methods and deep neural networks.

Finally in this chapter, there is an explanation of the software tools employed for constructing network slicing environments. Additionally, the performance evaluation criteria are outlined, providing insights into the metrics utilized to assess the effectiveness and efficiency of network slicing services.

2.2 Fundamentals of Network Slicing

The emergence of 5G networks has brought about a diverse range of use cases that cannot be efficiently supported by a single network design [21]. Each use case has specific requirements in terms of latency, data rate, mobility, connection density, reliability, and security. To

address this challenge, network virtualization is being explored as a solution to partition the resources of the physical network infrastructure into isolated and individually managed logical networks or "slices", as illustrated in Figure 2.1[22]. These slices encompass network resources (such as bandwidth), compute resources (processing power), and storage.

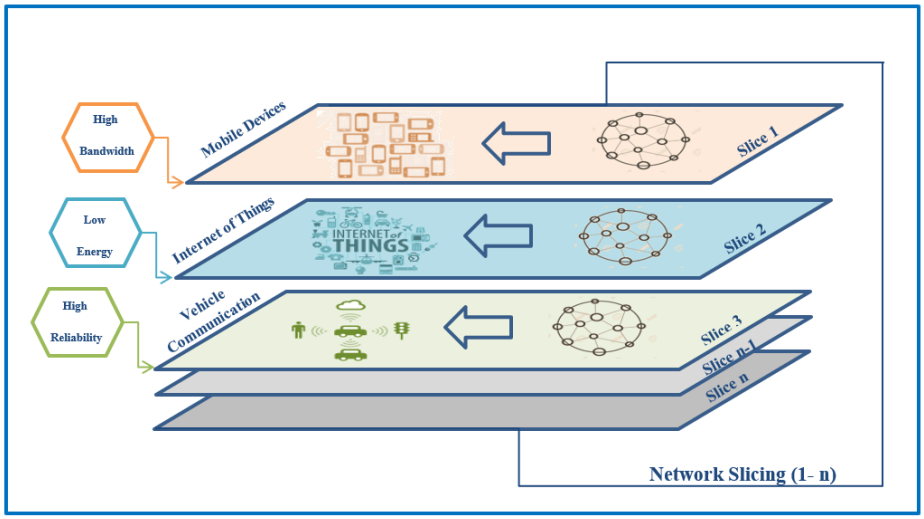


Figure 2.1: Example of Network Slices

Multiple network slices can coexist on a shared network infrastructure, where resources can be either dedicated to a single slice or shared among multiple slices [23]. Each network slice is specifically designed to cater to the resource requirements of a particular network service or function, as illustrated in Figure 2.2. The allocation of resources ensures that each network slice has the necessary resources to effectively deploy and deliver its intended network service or function.

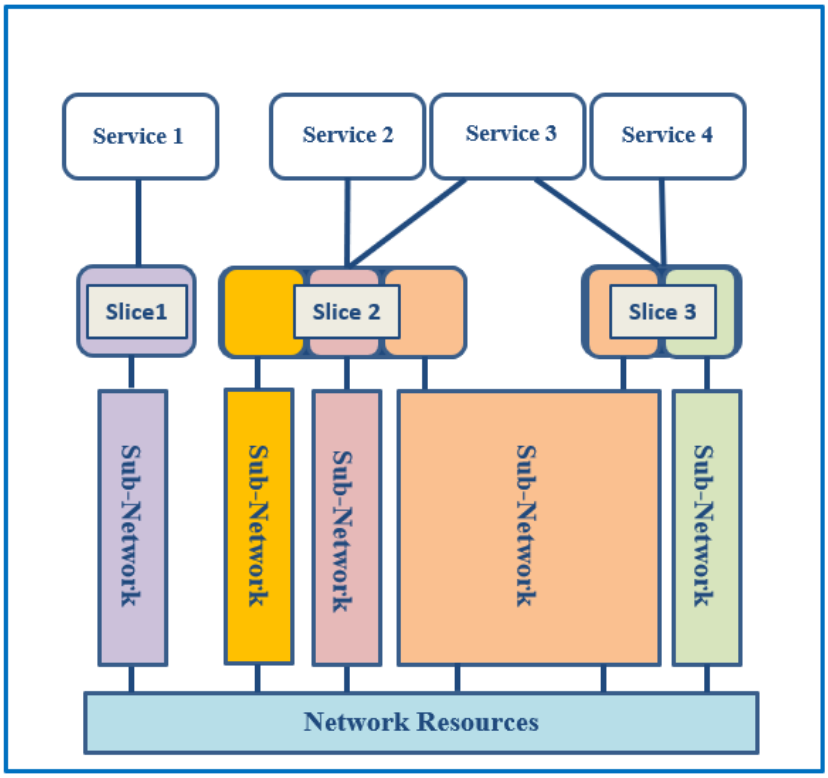


Figure 2.2: Network Slicing Concept

Once a slice is instantiated, the owners perceive it as their own network, with allocated resources at their disposal. They can independently manage the slice without affecting the performance or functionality of other slices. An early example of network slicing can be seen in the PlanetLab testbed network, where slices are leased to research projects, allowing them access to a subset of network resources [24].

Also, network slicing offers several benefits that contribute to the advancement and efficiency of modern networks [24], [25]. Here are some key benefits of network slicing:

1) Optimal resource utilization: Multiple virtual networks can coexist on a shared physical infrastructure, allowing tailored resource allocation and maximizing network efficiency [23].

2) Flexibility and scalability: Network slicing enables dynamic resource allocation and adjustment to accommodate changing demands, allowing networks to adapt quickly.
3) Isolation and security: Logically isolated slices provide enhanced security, preventing unauthorized access and ensuring the integrity and privacy of each network slice [8].
4) Cost optimization: Sharing infrastructure resources among slices reduces the need for dedicated physical infrastructure, resulting in cost savings in equipment, maintenance, and energy consumption.
5) Service agility: Network slicing enables rapid service deployment and provisioning, allowing quick response to customer demands and efficient management of services.

These fundamentals and benefits underscore the importance of network slicing in enabling efficient and customized network deployments to support various use cases in 5G networks.

2.2.1 Network Slicing Architecture

The network slicing architecture can be conceptually divided into four layers: the virtualized infrastructure layer, the network slice instance layer, the service instance layer, and the network management and orchestration layer [26]. These layers, along with their main components, are depicted in Figure 2.3[25], [27]:

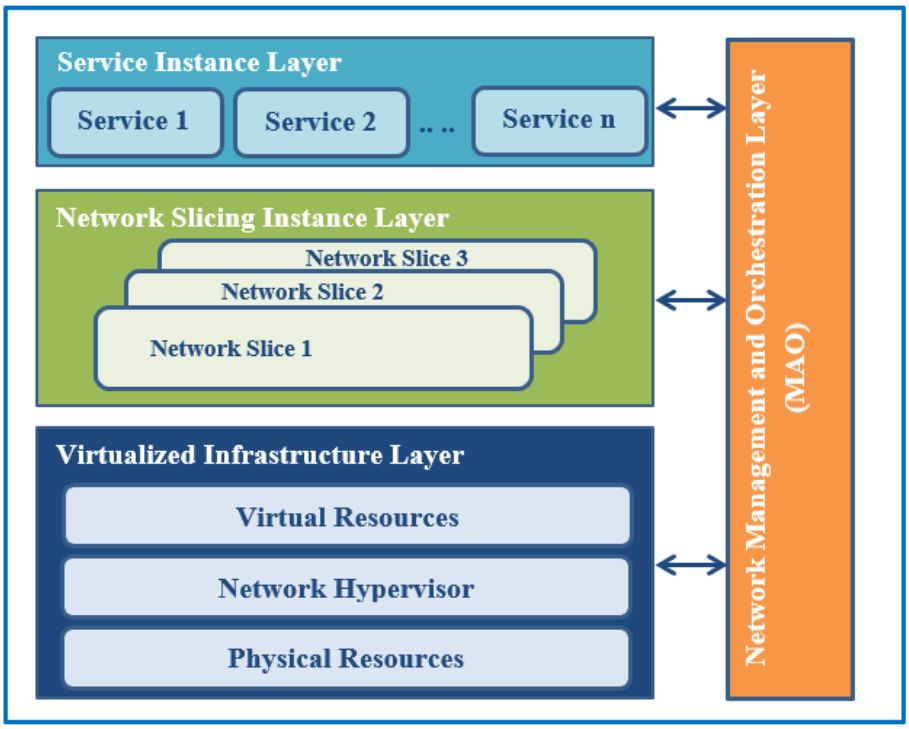


Figure 2.3: Network Slicing Architecture

1) **Virtualized Infrastructure Layer:** This layer is responsible for providing virtual instances of network resources that can be mapped to one or more network slices. Virtualization plays a central role in this layer, acting as a network hypervisor, and it is managed through the virtual infrastructure manager.
2) **Network Slice Instance Layer:** Positioned above the virtualized infrastructure layer, this layer represents the logical slices that operate on top of the virtualized infrastructure. Infrastructure resources are allocated to slices based on the specific requirements of each network slice. The management and orchestration layer handle the organization and allocation of these resources.
3) **Service Instance Layer:** This layer encompasses the different services that run on all the underlying layers. These services can be

provided by the network operator or by a third party, and they utilize the resources and capabilities offered by the network slices.

4) **Network Management and Orchestration Layer:** Considered the most crucial component of network administration, this layer comprises several sub-modules:
   - **Virtualized Infrastructure Manager (VIM):** Each VIM consists of one or more network hypervisor software, supporting the virtualization of infrastructure resources.
   - **NFV Orchestrator and Manager:** This sub-module includes the Network Function Virtualization Orchestrator (NFVO) and the Network Function Virtualization Manager (NFVM). They are responsible for orchestrating and managing network functions within the network slices.
   - **Software-Defined Networking Orchestrator (SDNO):** This sub-module, which can contain one or more SDN controllers, is specifically designed for managing network slices within the network infrastructure.

The concept of network slicing is a natural fit for the architecture of SDN [27]. Network virtualization, including network slicing, was among the first use cases proposed for SDN and has had a profound impact on the evolution of SDN technology.

SDN's architecture is well-suited to support network slicing. By abstracting network resources, SDN allows for efficient allocation and management, perfectly aligning with the goals of network virtualization [25]. The dynamic provisioning and configuration capabilities of SDN

enable the creation of network slices on-demand, optimizing the utilization of network resources.

The integration of network slicing and SDN has played a significant role in shaping the development of SDN technology [22]. It offers the advantage of coexistence with existing network infrastructure, allowing for a gradual migration to new technologies without requiring major hardware or infrastructure changes. SDN's inherent flexibility ensures a smooth evolution and deployment of network slicing within the SDN ecosystem.

In the next sections, a detailed explanation of SDN and NV will be provided, exploring their fundamental principles and highlighting how they have influenced the concept of network slicing.

2.2.2 Network Slicing Life Cycle

The network slice life cycle refers to the various stages that a network slice goes through during its existence [25], [28]. These stages typically include preparation, commissioning, operation, and decommissioning, as shown in Figure 2.4.

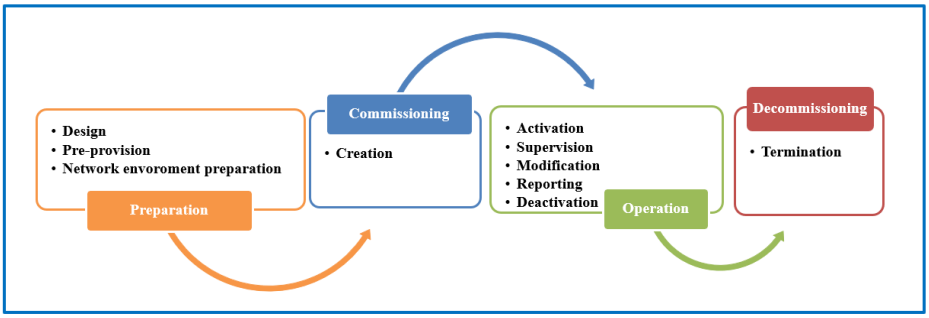


Figure 2.4: Network Slice Life Cycle

Here is an overview of each stage:

1) Preparation stage: This stage involves the evaluation of requirements for the network slice. It includes tasks such as assessing the necessary resources, defining the slice's characteristics, and preparing the network environment accordingly.
2) Commissioning stage: In this phase, the network slice is brought into operation. It includes activities such as resource allocation, configuration of network elements, and establishment of connectivity to enable the slice to function properly.
3) Operation stage: Once the network slice is commissioned, it enters the operation stage. This stage encompasses the ongoing management and monitoring of the slice to ensure its performance and availability. It involves tasks like activating the slice, supervising its operation, monitoring performance indicators, making modifications if needed, and eventually deactivating the slice.
4) Decommissioning stage: The decommissioning stage marks the end of the network slice's life cycle. During this phase, the resources and settings assigned to the slice are released, and the slice is taken out of active service. This stage may involve actions such as deallocating resources, removing configurations, and terminating connectivity associated with the slice.

2.2.3 Network Slicing Classification

Network slicing can be classified into two categories based on their use case: vertical network slicing and horizontal network slicing [25], [29].

1) Vertical Network Slicing: In vertical network slicing, all nodes within a particular network slice perform similar functions. Infrastructure resources are shared between various services and applications to improve the quality of service (QoS). Vertical network slicing separates traffic based on each service or application, allowing for dedicated resources and tailored QoS for specific services.
2) Horizontal Network Slicing: In horizontal network slicing, infrastructure resources are divided into horizontal layers, enabling devices to operate on more than one slice. Horizontal network slicing focuses on separating computing resources, providing capacity scaling. Typically, traffic travels end-to-end within a horizontal network slice, localized between the access network and the end device.

Another classification of network slicing is based on the nature of the slices: static network slicing and dynamic network slicing [25], [29].

1) Static Network Slicing: In static network slicing, the slice sets are pre-created. The devices need to specify which slice they will connect to, and the slices are already defined and available. This approach offers predetermined resource allocation and configuration.

2) Dynamic Network Slicing: In dynamic network slicing, operators define the slice design, but the allocation and optimization of resources are done dynamically based on service requirements or slice conditions. Resources can be allocated and adjusted on-demand to optimize network efficiency and meet changing service needs.

The provided classifications provide a framework for organizing network slices based on their characteristics, resource allocation, and operational adaptability. The choice between vertical or horizontal slicing, as well as static or dynamic slicing, depends on the specific requirements of the application, service needs, and desired level of resource utilization and flexibility. Our dissertation specifically concentrates on the utilization of horizontal and static slicing.

2.3 Technologies Involved in Network Slicing

Network slicing relies on the seamless integration and effective utilization of diverse technologies, which collectively enable the creation, management, and operation of distinct network slices. Several key technologies are involved in this process. Here are some of the fundamental technologies that play a crucial role in network slicing:

2.3.1 Software Defined Networking

Software Defined Networking (SDN) is an innovative networking technology that separates the control plane from the data plane, enabling greater flexibility in network control based on specific policies and

security measures [30]. Unlike traditional networks where control and data planes are implemented in static hardware appliances, SDN decouples the control logic and places it in a separate entity called the SDN controller or Network Operating System (NOS), as shown in Figure 2.5.

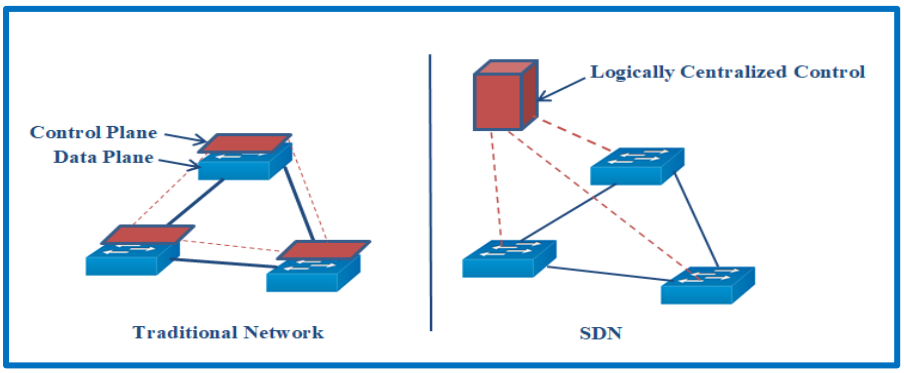


Figure 2.5: Traditional network vs SDN

The SDN architecture consists of three layers and three APIs [31]. Figure 2.6 provides a visual representation of the SDN architecture, illustrating its components.

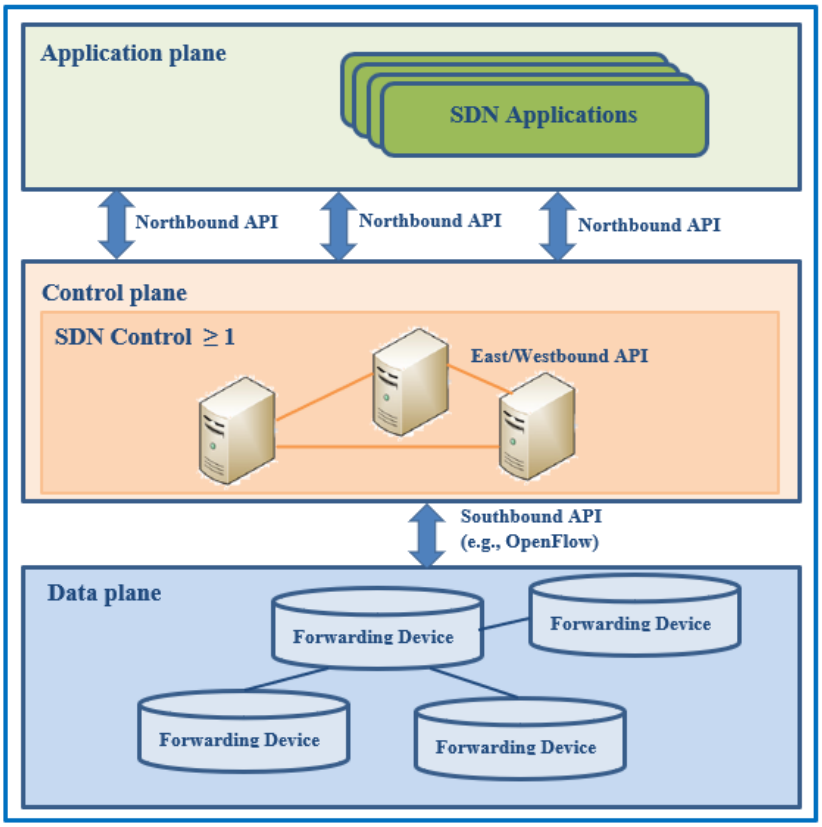


Figure 2.6: SDN Architecture

The infrastructure layer, also known as the data plane, comprises forwarding devices or network devices responsible for forwarding traffic flows based on the choices made by the control plane. The control plane, which acts as the network's brain, consists of one or more controllers [32]. It manages and controls the network by having a comprehensive view of it. The control plane can make changes to the forwarding elements, such as updating, installing, and deleting forwarding rules that guide the network's decisions and rules. It collects real-time information about the network's status, traffic, and performance to optimize the network's efficiency and make necessary adjustments.

The application plane, the final layer, encompasses network applications such as QoS, routing, and security applications. The APIs

connect the three layers, and there are two types of APIs used [30]. The open southbound API facilitates communication between the control and infrastructure layers, allowing the controller to configure, manage, and send flow forwarding decisions to the forwarding devices. OpenFlow is a commonly used SDN protocol (southbound API) for establishing communication between the controller and network forwarding devices, such as Open vSwitch. The Northbound API enables communication between SDN applications and the controller, allowing the applications to receive relevant information about network elements from the controller. In distributed SDN controller environments, the East-Westbound API is used for communication between controllers [33].

The SDN architecture, with its separate control plane and data plane, provides enhanced network control, flexibility, and programmability [30]. It enables centralized management, efficient resource allocation, and the ability to adapt to changing network requirements.

2.3.1.1 OpenFlow Protocol

The OpenFlow protocol is a crucial component of SDN that facilitates communication and control between the SDN controller and network forwarding devices, like switches and routers [34]. It defines a standardized interface and a set of messages that aid in the programming and management of network flows.

At its core, the OpenFlow protocol allows the controller to exert precise control over the behavior of network switches. It enables the

controller to define, modify, and delete flows, which determine how network traffic is processed and forwarded within the switches [30].

When a switch receives a packet, it examines the packet's header and uses the rules defined in its flow table to determine how to handle the packet. In an OpenFlow-enabled switch, the flow table is populated and updated through the OpenFlow protocol under the guidance of the SDN controller [30]. The controller can instruct the switch to match specific packet characteristics, such as source and destination addresses, ports, protocol types, and QoS markings, and apply corresponding actions, like forwarding, dropping, or modifying packets.

In the OpenFlow protocol, several key components enable communication and control between the SDN controller and network devices, as shown in Figure 2.7 [30], [35]. Here are the main components of OpenFlow:

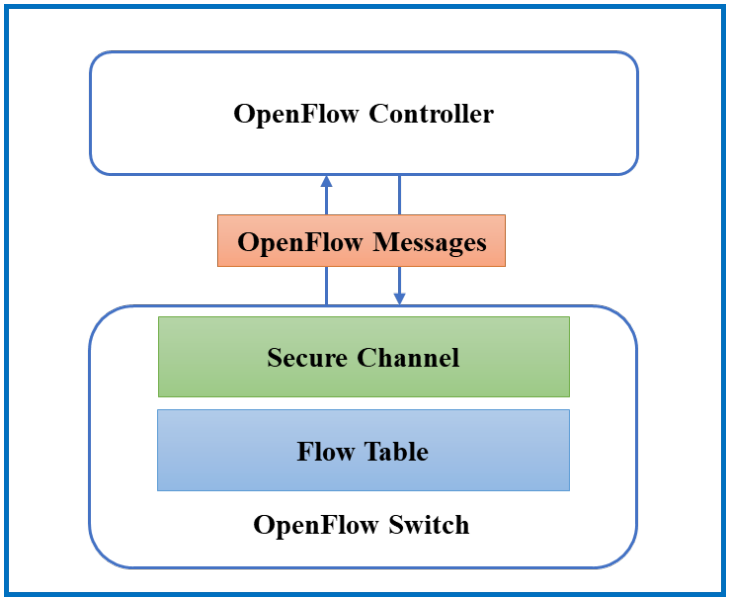


Figure 2.7: OpenFlow Components

1) **OpenFlow Switch**: An OpenFlow switch is a network device, such as a switch or a router, that supports the OpenFlow protocol [30]. It is responsible for forwarding network packets based on the instructions received from the controller. An OpenFlow switch typically consists of the following components:

- **Secure Channel:** is established as a Transmission Control Protocol (TCP) connection between the OpenFlow controller and the OpenFlow switch. The purpose of the secure channel is to ensure the confidentiality, integrity, and authentication of the communication between the two entities. To protect the mutual data exchanged between the controller and the switch from unauthorized access or tampering, encryption is applied. Transport Layer Security (TLS) is commonly used for encryption in the OpenFlow protocol [36]. TLS is a cryptographic protocol that provides secure communication over a network. It ensures that the data transmitted between the controller and the switch is encrypted, making it difficult for potential attackers to intercept and decipher the exchanged information.

  By employing TLS encryption within the secure channel, the OpenFlow protocol enhances the security of the communication between the controller and the switch [36]. It prevents unauthorized access to sensitive data and ensures that the transmitted information remains confidential and integral throughout the communication process.

- **Flow Table:** is a critical component of an OpenFlow switch [30]. It serves as a centralized database that stores a collection

of flow entries, which determine how network traffic is processed by the switch. Each flow entry in the flow table consists of three main components:

- **Match Fields:** match fields specify the criteria for matching incoming packets. These fields can include packet header information such as source/destination IP addresses, transport protocol (e.g., TCP, UDP), source/destination port numbers, VLAN tags, etc. The match fields define the conditions under which a flow entry is considered a match for an incoming packet.
- **Action:** define what actions should be taken by the switch when a packet matches a specific flow entry. These actions can include forwarding the packet to a specific port, modifying packet header fields, dropping the packet, sending the packet to the controller for further processing, or applying QoS treatments.
- **Counters:** each flow entry may have associated counters that keep track of statistics related to the packet traffic matching that entry. Counters can include packet and byte counts, which provide information about the volume of traffic associated with a particular flow entry.

2) **OpenFlow Messages:** OpenFlow protocol utilizes different types of messages for communication between the controller and the switch [30]. These messages facilitate the controller's ability to instruct the switch and receive updates from the switch. Below, the main categories of OpenFlow messages are described:

 Controller to Switch Messages:

- Packet-Out: This message is sent by the controller to instruct the switch to send a packet out to a specific port. It allows the controller to have direct control over packet forwarding.
- Flow Mod: This message is used by the controller to add, modify, or delete flow entries in the flow table of the switch. It enables the controller to define how network traffic should be processed by the switch.
- Port Mod: This message is employed by the controller to configure the behavior of a specific port on the switch, such as enabling or disabling a port or adjusting its features.
- Barrier Request/Reply: These messages are used to ensure message synchronization between the controller and the switch. The Barrier Request message is sent by the controller, and the Barrier Reply message is returned by the switch to indicate the completion of previous operations.

Asynchronous Messages:

- Packet-In: When the switch receives a packet that doesn't match any existing flow entry, it generates a Packet-In message and sends it to the controller. This message allows the controller to decide how to handle the packet, such as installing a new flow entry or forwarding it to another destination.
- Port Status: This message is sent by the switch to inform the controller about changes in the status or configuration of a port, such as link up/down events or port feature modifications.

- Flow Removed: The switch sends this message to notify the controller when a flow entry is removed from the flow table, either due to expiration or an explicit deletion.

- **Symmetric Messages:**
  - Hello: The Hello message is the first message exchanged between the controller and the switch during the connection establishment phase. It confirms the initiation of an OpenFlow session and negotiates the protocol version.
  - Echo Request/Reply: These messages are used for connectivity checking and monitoring. The controller can send an Echo Request message to the switch, which responds with an Echo Reply message to indicate that it is still alive and responsive.

3) **OpenFlow Controller:** is the central entity in the SDN architecture, responsible for managing and controlling the network [30], [33]. It establishes communication with OpenFlow-enabled switches and makes decisions about how the network behaves. The OpenFlow controller can be implemented as a standalone software application or as a component of a network operating system. It serves as the brain of the SDN network, receiving information from switches and directing them on how to handle network traffic based on specific policies and rules.

There are many OpenFlow controller software's available, including popular ones such as POX, Ryu, and OpenDaylight [37]. These controllers provide different features, programming interfaces, and levels of extensibility, allowing developers and

network administrators to choose the one that best suits their specific requirements and preferences. However, each OpenFlow controller has its own set of characteristics, such as scripting language, extension options, community support, and integration with other SDN components. Evaluating these factors can help determine which controller is most appropriate for a given SDN deployment.

2.3.1.2 Distributed SDN Controller

The concept of distributed SDN controllers is introduced to address the issues of a single point of failure and scalability in traditional centralized controller architectures [30], [33]. By distributing the control functions across multiple controllers, these challenges can be mitigated. There are different strategies for implementing distributed SDN controllers, as depicted in Figure 2.8.

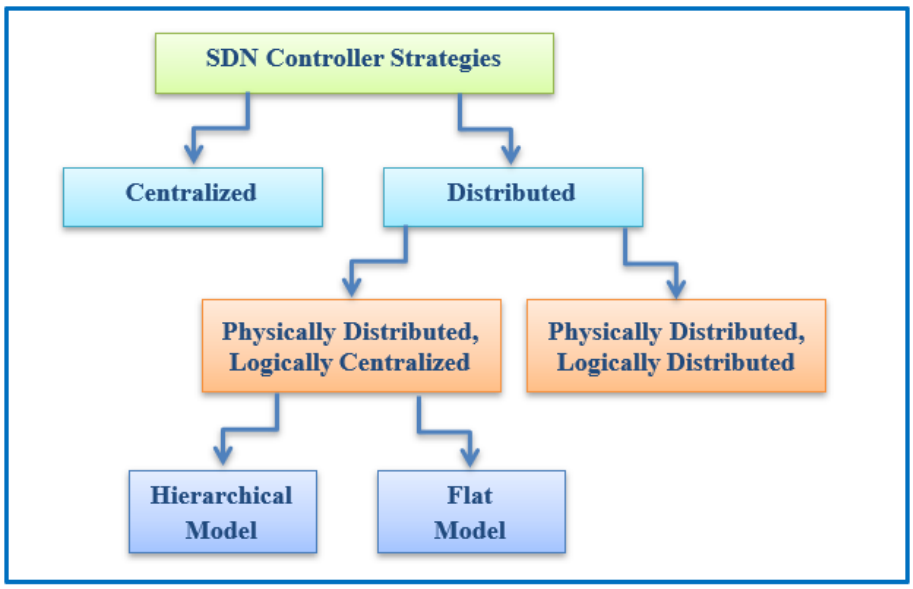


Figure 2.8: SDN Controller Strategies

To overcome the centralized control problem, researchers have proposed clustering the controllers into domains, where each domain represents a network block with its own controller [30]. This approach aims to reduce communication requests from switches to the SDN controller by shrinking the network size within each domain. This concept is known as physically and logically distributed SDN controllers, as shown in Figure 2.9.

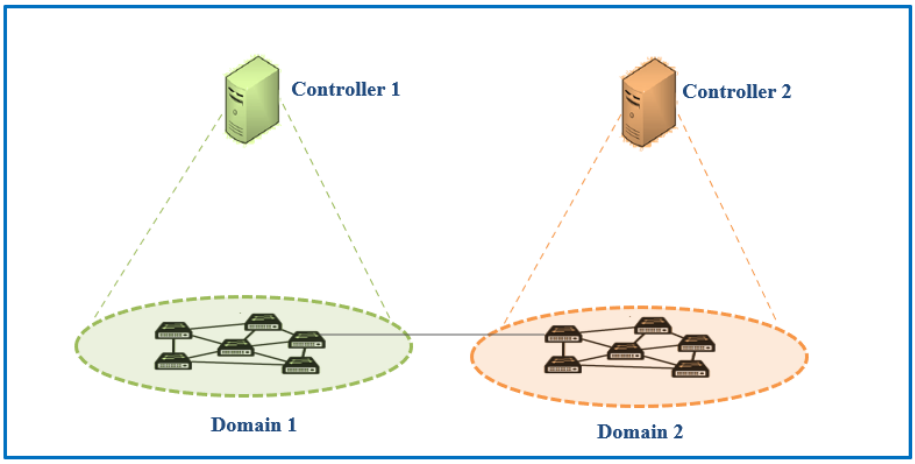


Figure 2.9: Physically and Logically Distributed SDN Controllers

In a logically centrally distributed controller architecture, multiple controllers function as if they are centrally connected to each other, although they are physically separate [33]. Communication between these controllers is established through East-Westbound APIs, allowing them to have the same control over the data while being physically distant from each other.

Figure 2.10 illustrates the commonly seen SDN distributed control architectures, which can be classified into flat and hierarchical SDN controller architectures based on the physical arrangement of the controllers [38]. In a hierarchical architecture, one or more controllers

possess the global network state, while in a flat architecture, all SDN controllers have an overview of the entire network state.

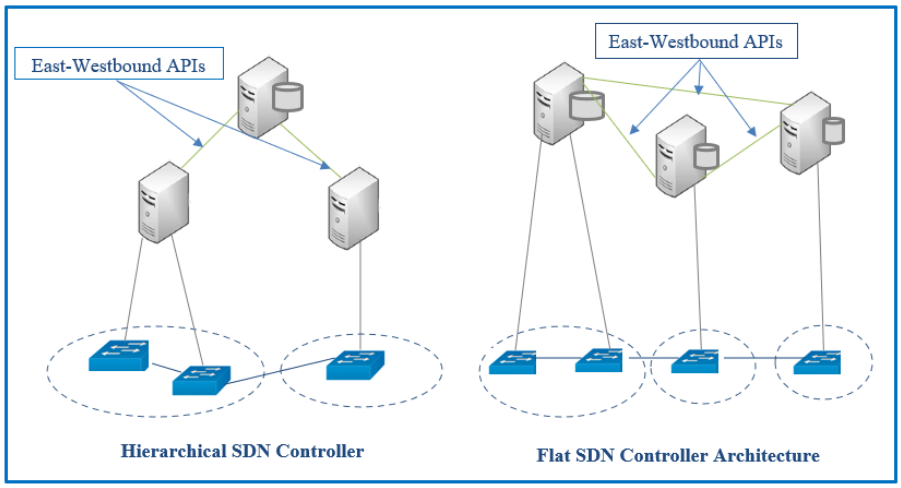


Figure 2.10: Flat and Hierarchical SDN Controller Architecture

These distributed SDN controller approaches provide fault tolerance, scalability, and more efficient network management by decentralizing control functions and dividing the network into manageable domains or clusters [38].

2.3.2 Network Virtualization

Network virtualization and network slicing are two interconnected concepts that play a significant role in enhancing the flexibility and efficiency of modern networks [39]. Network virtualization involves deploying multiple heterogeneous networks on a shared physical network infrastructure, enabling the division of responsibilities between infrastructure providers and service providers [40]. This approach aims to improve flexibility, manageability, security, and promote diversity in networking.

The architecture of network virtualization typically consists of several key components and layers that work together to create and manage virtual networks, as shown in Figure 2.11 [25].

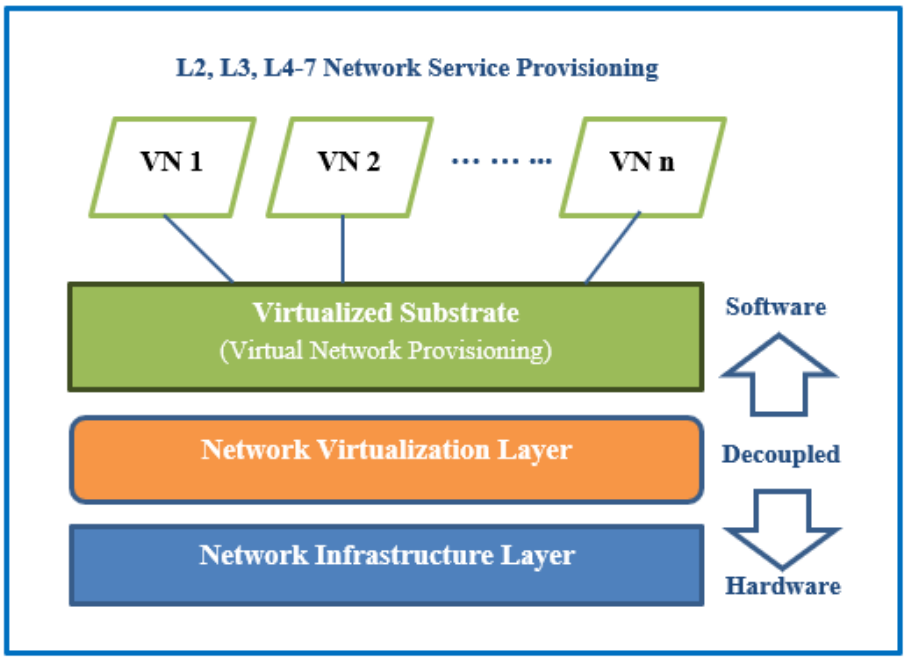


Figure 2.11: Network Virtualization Architecture

At the foundation is the infrastructure layer, which represents the underlying physical network infrastructure [41]. It includes network devices such as routers, switches, and servers, as well as the physical links (wired or wireless connections) that connect them. This layer forms the basis upon which network virtualization is built.

Sitting above the infrastructure layer is the virtualization layer, responsible for abstracting and virtualizing the underlying physical infrastructure. This layer includes components such as hypervisors, SDN controllers, and network virtualization software [42]. It provides the necessary mechanisms to create and manage virtual networks (VNs) on top of the physical infrastructure. By abstracting the physical network, the

virtualization layer enables the creation of virtual resources that can be dynamically allocated and managed.

The virtual infrastructure layer constitutes the virtual resources that are created and managed by the virtualization layer. It includes virtual routers, virtual switches, virtual firewalls, virtual load balancers, and other VNFs [25], [40]. These virtual resources form the building blocks for creating and running virtual networks. With the virtual infrastructure layer, network administrators can configure and manage the virtual resources to meet the specific requirements of the virtual networks they are creating.

VNs are logical networks that are created and operated on the virtual infrastructure. Each virtual network represents a segmented portion of the overall network, isolated from other virtual networks [39]. VNs can have their own addressing schemes, routing policies, security settings, and network services. By operating on the virtual infrastructure layer, virtual networks provide a high degree of flexibility and customization, allowing for the creation of isolated network environments tailored to specific needs [42].

NS, on the other hand, takes the concept of network virtualization further by dividing the network into separate logical slices with dedicated resources and customized characteristics [39]. It builds upon network virtualization techniques, such as virtualization technologies, VNFs, and SDN, to create and manage these slices. Network slicing enables the dynamic allocation of resources, isolation, and customization of network services for different use cases or tenants. By assigning dedicated resources to each network slice, network administrators can ensure

optimal performance and isolation for specific applications or customer requirements.

2.4 Scenarios of Implementation Network Slicing

This section will provide an explanation of the different scenarios for implementing network slicing. Each scenario will be discussed in detail, highlighting both the advantages and disadvantages associated with it.

1) Single Owner, Single Controller

In scenarios where there is only one owner and one controller, SDNs enable direct network slicing capabilities by abstracting network resources through the Northbound interface of the SDN controller [25], [43]. This facilitates efficient management and orchestration of network slices, with the SDN controller acting as the SDN Orchestrator responsible for resource allocation and coordination. The architecture is shown in Figure 2.12.

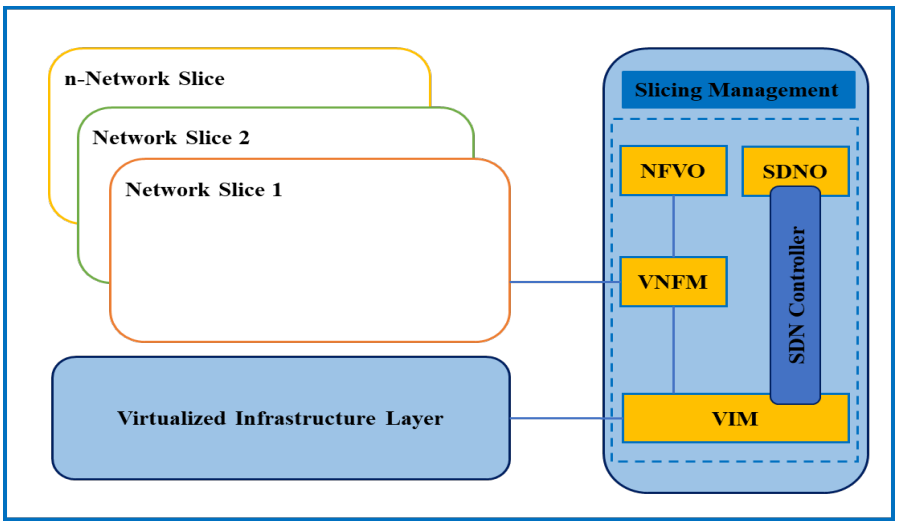


Figure 2.12: Architecture of Network Slicing Scenario with Single Controller

This solution is particularly effective in specific parts of the network, especially when there is only one owner governing the infrastructure. The centralized control provided by the single controller simplifies the management of network slices within the controlled domain. However, there are certain drawbacks to consider.

One limitation is the potential for performance issues and decreased reliability. With a single controller handling the orchestration and control of multiple network slices simultaneously, it can become a bottleneck, impacting overall network performance and responsiveness [43].

Furthermore, the presence of a single controller restricts the programmability of the networking infrastructure, especially in scenarios where multiple tenants want to deploy their own network services. Each tenant may have unique requirements and preferences for their network slices, and the centralized controller might struggle to efficiently accommodate diverse demands.

2) Single Owner, Multiple Controller

In this scenario, the SDN proxy plays a crucial role as an intermediary between the control and forwarding paths of network devices [25], [43]. Its primary function is to create an abstraction of the network forwarding path, enabling the implementation of network slicing. By leveraging the SDN protocol, the SDN proxy establishes a hardware abstraction layer that enforces predefined rules and agreements, defining the network slices and ensuring isolation

between them. Figure 2.13 provides a visual representation of the resulting architecture in this scenario.

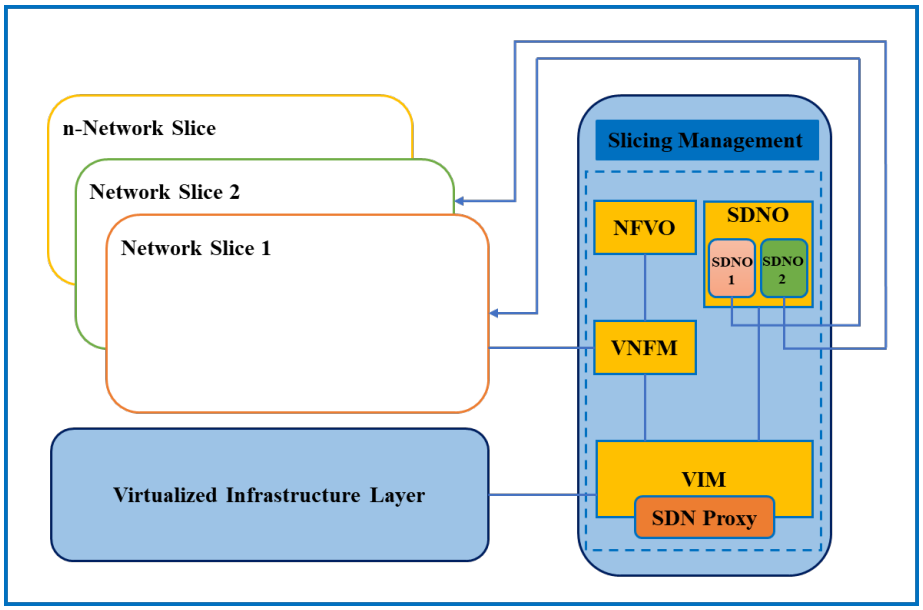


Figure 2.13: Architecture of Network Slicing Scenario with Multiple Controller

The key advantage of this scenario is the ability to divide the network infrastructure into multiple virtual networks. Typically, the owner of the infrastructure retains control over the SDN proxy [43]. This allows multiple virtual tenants to deploy their own SDN controllers on the shared infrastructure, enabling them to manage their specific network slices while maintaining isolation from other tenants.

One advantage of this scenario is the level of flexibility it provides for individual tenants. They have the option to implement their own controllers or SDN orchestrators, enabling them to have direct management of their network slices. The SDN proxy guarantees that each tenant's slice is kept separate from others. As a result, this particular scenario was utilized in the dissertation to establish a network slicing environment.

An example of a solution that implements this scenario is FlowVisor, which functions as an OpenFlow proxy [44]. FlowVisor intercepts messages between OpenFlow-enabled switches and OpenFlow controllers, enabling the desired network slicing functionality.

3) Multiple Owner, Multiple Controller

Virtualization is a crucial aspect when it comes to enabling multiple tenants to independently define their desired resource connections, regardless of the service or infrastructure provider involved [25]. While the previously mentioned architectures focus on creating network slices by dividing or isolating link resources and allowing controllers to operate on them, advanced virtualization is required to establish a mapping between physical infrastructure resources and the virtual overlay topologies desired by tenants.

This scenario aims to enhance the flexibility of programmable virtual networks [43]. An example of a network virtualization platform that facilitates this concept is OpenVirteX [45]. OpenVirteX allows tenants to specify their preferred topology and address while still enabling the infrastructure owner(s) to maintain control over their virtual SDN network.

These scenarios offer several key features, including the ability to create isolated virtual networks with topology specified by tenants, compatibility with any controller or network operating system, utilization of the entire address space, the flexibility to make runtime changes to the virtual network, and automatic recovery from physical failures [43].

These features empower tenants to define their network requirements while providing the infrastructure owner(s) with necessary control over their virtual SDN resources.

2.5 Network Slicing Security

The security challenges in network slicing arise from the diverse services it offers, each with different security requirements [46]. These challenges become more complex in multi-domain infrastructures and when resources are shared among slices with varying security policies. The deployment of network slicing in 5G systems and beyond introduces new and advanced security vulnerabilities.

To address these security issues, each network slice is designed with isolation constraints to prevent interference and ensure independent security solutions [47]. These solutions should follow basic security principles, including confidentiality, authentication, availability, integrity, and authorization [48], [49].

The security of network slicing requires each slice and its owners to independently fulfill these security requirements. Failure to address these requirements can result in exploitation by attackers and system failures.

2.5.1 Attacks on Network Slicing

NS is prone to a range of attacks that can disrupt its functionality and compromise its security requirements [50]. These attacks exploit the isolation levels established by the slice requirements. Some examples of

attacks on network slicing include interface monitoring, traffic injection, impersonation, DoS, tampering, eavesdropping, and replay attacks.

Interface monitoring attacks specifically target certain interfaces, such as the southbound and northbound interfaces of the NSM and the NFVO of the Management and Orchestration system [51]. Breaching the NSM interface can have system-wide consequences, while breaching the NFVO interface only affects elements controlled by the NSM.

An interface monitoring attack on the NSM occurs when attackers gain control over the traffic on the northbound or southbound interfaces of the NSM to uncover the system's configuration [50]. The aim is to capture snapshots of the system and identify vulnerabilities that compromise system confidentiality. Once the attackers understand the system, they can launch other malicious actions, including traffic injection, impersonation, side channels, and DoS attacks, which breach additional security requirements.

In NS, the data plane interacts with the southbound interfaces, while the control plane interacts with the northbound interface. Side-channel attacks can arise when multiple slices share resources on common hardware [46], [50]. These attacks can lead to various exploitations, such as hardware tampering, sensor errors, malware, and DDoS attacks. DDoS attacks overwhelm a targeted service or slice by flooding the network with traffic, temporarily rendering the network resources inaccessible to legitimate users.

2.5.2 DoS/DDoS Attack Points on Network Slicing

DDoS/DoS attacks can be directed at different layers of the network slicing architecture, including the infrastructure layer, network slice instance layer, service instance layer, and network management and orchestration layer [52], as shown in Figure 2.14.

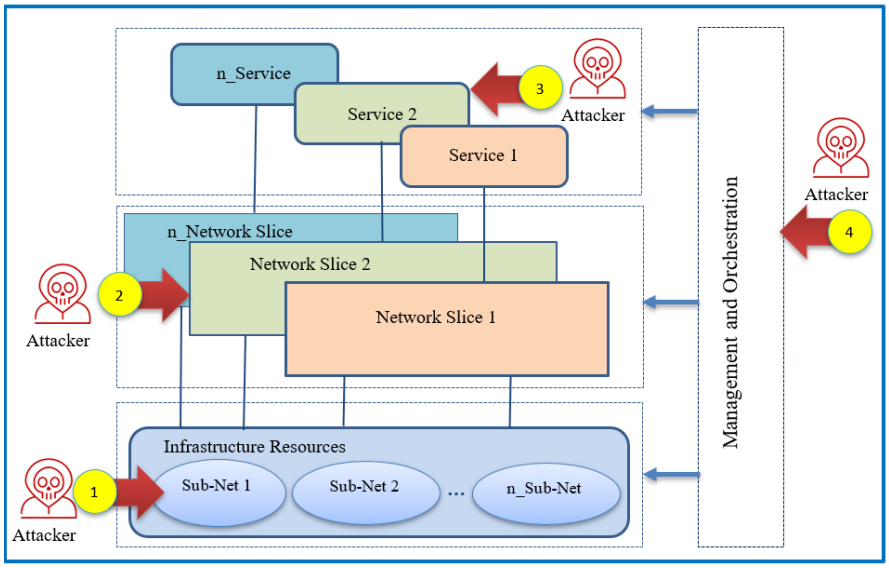


Figure 2.14: Representative Points of DoS/DDoS Attack on Network Slicing

Here's an overview of how these attack points can impact each layer:

Attack Point 1: DDoS/DoS attacks on the infrastructure layer can disrupt the underlying components that support network slices, such as routers, switches, servers, and virtualization platforms [52]. Attackers may flood the infrastructure with excessive traffic or exploit vulnerabilities in these components, leading to congestion, resource exhaustion, or system failures. As a result, the availability and performance of the infrastructure can be severely disrupted, causing service outages across multiple network slices.

Attack Point 2: DDoS/DoS attacks targeting the network slice instance layer directly affect the individual instances of network slices [53]. Attackers may inundate the network slice instances with a high volume of traffic, overwhelming the allocated resources and causing service disruptions or unavailability. These attacks deplete resources like bandwidth, CPU, and memory, negatively impacting the performance and functionality of specific network slice instances [18].

Attack Point 3: DDoS/DoS attacks on the service instance layer specifically aim at the instances of services within network slices [52], [54]. Attackers may launch a significant volume of malicious traffic or requests at specific service instances, resulting in disruptions to their normal operation. Such attacks can lead to service degradation or complete denial of service for the targeted service instances, affecting the users or applications relying on those services.

Attack Point 4: DDoS/DoS attacks targeting the network management and orchestration layer disrupt the functions responsible for managing and orchestrating network slices [52]. Attackers may overwhelm the network management and orchestration systems with a high volume of requests or malicious traffic, impacting provisioning, configuration, monitoring, or control of network slices. These attacks hinder the effective management and orchestration of network slices, adversely affecting the overall operation and performance of the network slicing infrastructure.

2.6 Attack Detection Mechanisms

Detection mechanisms encompass various methods and techniques used to identify and detect specific events, patterns, anomalies, or signals within a network or dataset. In this section, statistical methods commonly employed in this field will be explored [55]. Additionally, a comprehensive explanation of DNNs will be provided as an exemplary instance of machine learning-based methods used for detecting attacks [56].

2.6.1 Statistical-Based Detection

Statistical-based methods for detecting anomalies or identifying deviations in data include Shannon's entropy, the chi-square test, and Chebyshev's inequality [57]. Here is a brief explanation of some of these methods:

1) **Shannon's Theorem (Shannon entropy):** Shannon's theorem, developed by Claude Shannon in information theory, provides a formal definition of entropy [58]. It measures the average information required to represent an event or random variable. Mathematically, Shannon entropy is represented by Equation 2.1[58].

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i) \qquad (2.1)$$

Here, X represents a random variable, $p(x_i)$ denotes the probability of observing each value x_i of X, and the summation is

performed over all possible values of X. Shannon's entropy is commonly used to measure the unpredictability or information content of data. It can be applied in anomaly detection to identify deviations from the expected information content of observed data.

2) **Chebyshev's inequality:** This provides a probabilistic bound on the probability of a random variable deviating from its mean [59]. For any random variable X with mean μ and standard deviation σ, the inequality states that the probability of X deviating from the mean by more than k standard deviations is at most $\frac{1}{k^2}$ as shown in Equation 2.2 [59].

$$p(|X - \mu| \geq k\,\sigma) \leq \frac{1}{k^2} \qquad (2.2)$$

In this equation, |X - μ| represents the absolute difference between X and its mean μ, and k is a positive constant that determines the number of standard deviations from the mean used as a threshold. Chebyshev's inequality allows us to estimate the likelihood of large deviations from the mean. In anomaly detection, observations that fall far outside the bounds defined by Chebyshev's inequality can be considered potential anomalies [60].

2.6.2 Deep Neural-Based Detection

Deep neural-based detection, also referred to as DNN detection, is an approach within the field of machine learning that leverages artificial neural networks with multiple layers to identify and classify threats [61]. DNNs possess a remarkable capability to capture intricate relationships and dependencies within data, enabling them to discern subtle patterns

indicative of malicious activities or attacks. By undergoing extensive training on labeled datasets containing both normal and DoS/DDoS data samples, deep neural networks acquire the ability to differentiate between regular behaviors and those associated with DoS/DDoS incidents. In this section, a comprehensive explanation of deep neural networks will be provided.

2.6.2.1 Deep Neural Network

Neural networks are computational networks that mimic the complex networks of neurons found in the human brain. These networks are made up of interconnected nodes, known as artificial neurons, which are designed to emulate the behavior of real neurons [62]. The artificial neural network, depicted in Figure 2.15 alongside a biological neuron, is the most widely used type of neural network.

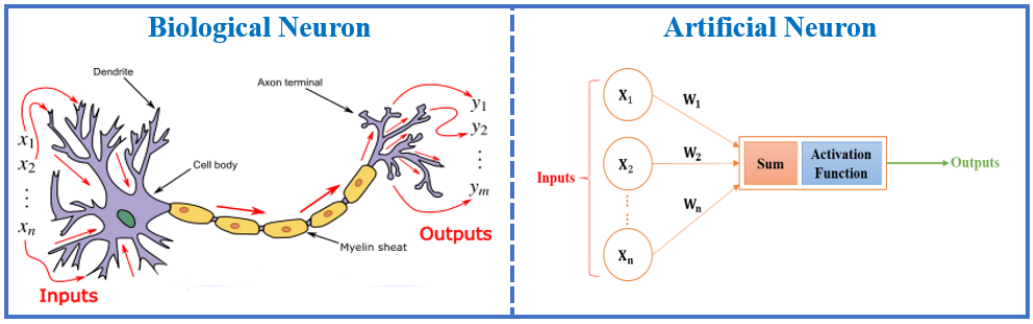


Figure 2.15: Illustration of an Artificial Neuron and a Biological Neuron

A neural network comprises interconnected layers of nodes or artificial neurons. These layers work together to process and transform input data, leading to the generation of output or predictions. Typically, a neural network consists of an input layer, one or more hidden layers, and an output layer [62]. Figure 2.16 visually represents the arrangement of

these layers within a neural network, providing a clear illustration of their organization and connectivity.

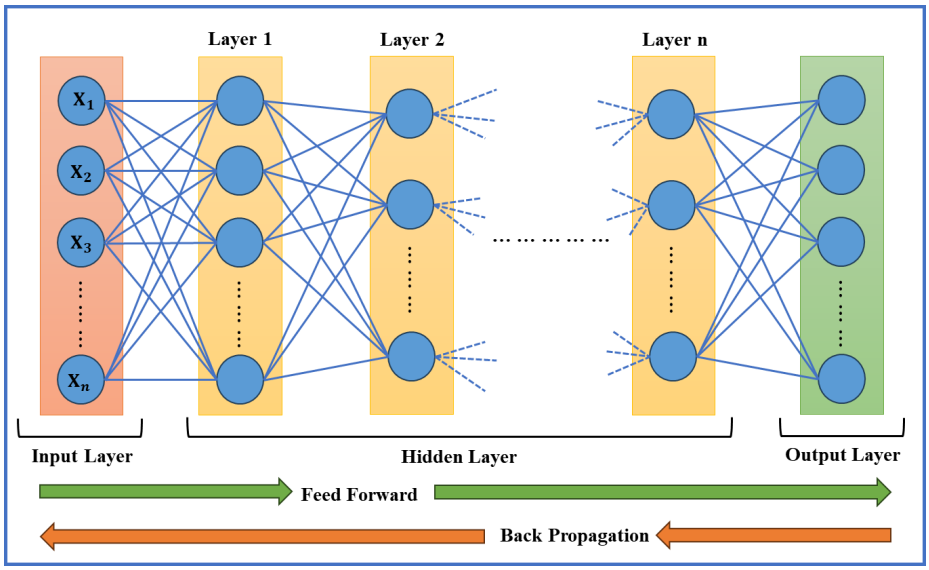


Figure 2.16: Neural Network Layer Organization and Connectivity

In neural networks, the input layer serves as the initial point of entry for data. It receives input features and transmits them to subsequent layers for processing. The hidden layers, positioned between the input and output layers, perform computations and transformations on the input data. These hidden layers are responsible for extracting and learning relevant features that enable the network to capture intricate patterns and relationships within the data [62].

When a neural network consists of more than one hidden layer, it is called a deep neural network [62]. The incorporation of multiple hidden layers enhances the network's capacity for abstraction and representation. The number and size of hidden layers can be adjusted based on the complexity of the problem and the desired level of abstraction. Deep neural networks possess the capability to handle and understand complex

data structures, making them particularly effective in solving intricate tasks.

The output layer produces the final results or predictions based on the information processed by the hidden layers [61]. The structure of the output layer depends on the nature of the problem being addressed. For example, in a classification task, the output layer may have multiple nodes, each representing a different class, and the predicted class corresponds to the node with the highest activation.

A neural network has two propagations: Feed Forward Propagation and Back Propagation. As shown in Figure 2.16. Feed Forward Propagation involves a sequential process where information flows from the input layer through the hidden layers, ultimately reaching the output layers [63]. The primary objective is to compute output results or predictions based on the input data. This process employs a perceptron classifier, which can be represented by Equation 2.3 [63].

$$y = \sum_{i=1}^{n} x_i \, w_i + b \qquad (2.3)$$

In the equation, n represents the number of neurons in the current layer, x represents the neuron data, w denotes the weights, and b represents the bias. The output results are then passed through activation functions [63](further details will be explained in the subsequent subsection).

On the other hand, Back Propagation entails the reverse propagation from the output layer to the input layer. Its primary purpose

is to facilitate the learning process of the neural network by adjusting the weights and biases. This adjustment is accomplished using optimizer and loss function methods [64](more comprehensive explanations of both will be provided in the subsequent subsection).

- **Activation Functions**

Activation functions in neural networks are inspired by the behavior of biological neurons in the human brain [63]. These functions are designed to mimic the activation potential observed in neurons. When the input reaches a certain level, known as the activation potential, the neuron becomes active. Additionally, activation functions aim to keep the output within a limited range. By introducing nonlinearity into neural networks, activation functions play a crucial role in enabling models to learn complex operations.

The most popular activation functions are Rectified Linear Unit (ReLU) Function, Hyperbolic Tangent Function (Tanh), Sigmoid Function, and Softmax Function [62], [65].

1) **ReLU Function:** is a widely used activation function in neural networks [65]. It is a non-linear function that introduces non-linearity to the decision-making process of the network, represented by Equation 2.4 [65].

$$f(x) = \max(0, x) \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad (2.4)$$

In the above equation, x denotes the input data to the activation function. The ReLU function operates by returning the

input value if it is greater than or equal to zero. If the input is negative, it "corrects" the value to zero. Positive values are left unchanged.

2) **Tanh Function:** serves as an activation function extensively utilized in recurrent neural networks for various tasks, including speech recognition and natural language processing [65]. It generates outputs within the range of -1 to 1, as indicated by Equation 2.5 [65].

$$f(x) = \frac{e^{x} - e^{-x}}{e^{-x} + e^{x}} \tag{2.5}$$

In the above equation, x represents the input data provided to the activation function. The Tanh function operates by evaluating the exponential of the input, subtracting the exponential of its negation, and subsequently dividing the result by the sum of the exponentials of both the input and its negation.

3) **Sigmoid Function:** is a widely used nonlinear activation function in neural networks' feedforward process [65]. It is a differentiable real function that outputs values between 0 and 1. Equation 2.6 represents the mathematical expression for the sigmoid function [65].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

Here, the variable x represents the input data to the activation function. The sigmoid function's distinctive S-shaped curve and its output range of 0 to 1 contribute to its significance in various applications. Particularly, within neural networks, it proves highly

beneficial for modeling probabilities and facilitating binary classification tasks [63].

4) **Softmax Function**: is a widely used activation function in the output layer of neural networks [65]. It takes a vector of real numbers as input and transforms it into a probability distribution. The Softmax function ensures that the output values range between 0 and 1, and the sum of these probabilities is always equal to 1, as represented by Equation 2.7[65].

$$f(x_i) = \frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}} \qquad (2.7)$$

In this equation, x represents the input data to the activation function. The Softmax function operates by exponentiating each element of the input vector and dividing it by the sum of the exponentiated values across all elements in the vector.

The Softmax function is particularly beneficial in scenarios involving multi-class classification problems. It generates a probability distribution over the different classes, where each output value represents the probability of the corresponding class being the correct prediction. The class with the highest probability value is considered the most probable class.

- **Loss Functions**

The loss function, also known as the cost function, is a fundamental concept in machine learning and neural networks [62], [64]. Its purpose is to quantify the discrepancy between the predicted outputs of a model and the actual values or labels associated with the training data.

When training a neural network, the model iteratively adjusts its parameters to minimize the loss function [66]. In other words, the goal is to find the set of parameter values that yield the lowest possible loss, indicating the best fit between the predicted outputs and the ground truth labels.

The choice of the loss function depends on the specific problem being tackled. Different types of problems, such as regression, binary classification, or multi-class classification, require different loss functions. The selected loss function should align with the problem's nature and the desired behavior of the model [64]. Various methods exist for defining the loss function, with some of the most commonly used ones including:

1) **Mean Squared Error (MSE):** This is a popular loss function for regression tasks [67]. It calculates the average squared difference between the predicted values and the true values, as represented by Equation 2.8 [67].

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i') \tag{2.8}$$

In this formula, y_i represents the actual value, y_i' represents the predicted value, and n represents the total number of samples or instances. The squared nature of the differences in MSE has the effect of penalizing larger errors more heavily than smaller ones. This means that the model's optimization process will prioritize reducing larger errors in order to minimize the overall loss.

2) **Mean Absolute Error (MAE):** is a loss function used in regression tasks to measure the average absolute difference between the predicted values (y_i') and the corresponding actual values (y_i). It

provides a measure of the average magnitude of the errors made by the model [67].

Mathematically, the MAE is calculated by summing the absolute differences between the predicted and actual values and dividing by the count of values (n) as represented by Equation 2.9 [67].

$$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - y_i'| \tag{2.9}$$

3) **Binary Cross-Entropy:** is a loss function commonly used in binary classification problems [66]. It measures the dissimilarity or the error between the predicted probabilities (y') and the true binary labels (y). The loss function is derived from the concept of entropy in information theory [67]. Mathematically, the Binary Cross-Entropy loss function is defined by Equation 2.10 [67].

$$L = -[y\log(y') + (1-y)\log(1-y')] \tag{2.10}$$

In this formula, y represents the true binary label (0 or 1), and y' represents the predicted probability between 0 and 1. The term y log(y') calculates the contribution to the loss when the true label is 1, and the term $(1 - y)$ log $(1 - y')$ calculates the contribution when the true label is 0.

The Binary Cross-Entropy loss function penalizes the model more when it makes incorrect predictions with high confidence [68]. It encourages the model to assign higher probabilities to the correct class and lower probabilities to the incorrect class. This makes it effective for training binary classification models,

especially when dealing with imbalanced datasets where the number of samples in each class differs significantly. In this dissertation, Binary Cross-Entropy was used as a loss function.

4) **Multi-class Cross-Entropy** Loss: is a type of cross-entropy loss function utilized to evaluate the performance of multi-class classification models [67]. It quantifies the discrepancy between the predicted probabilities y'_c and the true class labels y_c. The loss increases as the predicted probabilities deviate further from the actual labels, as shown in Equation 2.11[67].

$$L = -\sum_{c=1}^{m}(y_c \log(y'_c) + (1 - y_c)\log(1 - y'_c)) \quad (2.11)$$

In this equation, M represents the number of classes, c denotes the specific class, y_c represents the true class values, and y'_c represents the predicted class probabilities.

- **Optimizers**

In deep learning, the disparity between the predicted results and the actual values is measured by the loss function [62]. To achieve a high-accuracy model, the rate of variation in this disparity needs to be minimized. A crucial role is played by optimizers in this process as they iteratively adjust the parameters (weights and biases) of the deep learning algorithm during Backpropagation to minimize the values of the loss function [69].

Initially, arbitrary values such as zero, one, or any other number are assigned to the weights and biases. Subsequently, in the subsequent iterations, the determination of whether each parameter should

increase or decrease is made by the optimizer, aiming to reach the optimal values [62], [69].

Several optimizers are encompassed in deep learning, which aid in this optimization process [69]. In the following sections, some commonly used optimizers will be discussed.

1) **Gradient Descent (GD):** GD is a basic optimization algorithm that updates model parameters iteratively [70]. The gradient of the cost function with respect to the parameters is computed, and steps proportional to the negative gradient are taken to minimize the cost function. However, GD can be slow for large datasets since it necessitates computing gradients over the entire dataset in each iteration.
2) **Stochastic Gradient Descent (SGD) with momentum:** SGD is a variant of GD that updates the parameters using one training sample at a time [70]. It introduces momentum, which incorporates a fraction of the previous update into the current update. This inclusion accelerates convergence and smooths the update trajectory, resulting in faster training. SGD with momentum is less prone to becoming stuck in local minima compared to standard GD.
3) **Adaptive Gradient Algorithm (Adagrad):** Adagrad is an adaptive learning rate optimization algorithm. It adjusts the learning rate for each parameter based on the historical gradients [71]. Parameters with higher gradient magnitudes receive smaller updates, while parameters with lower gradients receive larger updates. Adagrad is particularly beneficial for sparse data and can automatically adapt the learning rate to different parameters.

4) **Adadelta:** is an extension of Adagrad that tackles the issue of rapidly decreasing learning rates [72]. It prevents the accumulation of squared gradients over time by maintaining a running average of gradient updates. Adadelta adapts the learning rate based on the ratio of the Root Mean Squared (RMS) gradients to the RMS parameter updates. This algorithm ensures stable and efficient optimization, especially in scenarios where manual tuning of the learning rate is challenging.
5) **Adam optimizer:** Adam, short for Adaptive Moment Estimation, combines momentum and adaptive learning rates [73]. It maintains per-parameter learning rates that are adjusted based on the first and second moments of the gradients. By utilizing exponentially decaying averages of past gradients and squared gradients to update the parameters, Adam provides fast convergence. It is widely employed due to its robustness and effectiveness. In this dissertation, the Adam optimizer was utilized.

2.6.2.2 CICIDS2017 Dataset

The CICIDS2017 dataset is a widely used benchmark for network intrusion detection. It was created by the Canadian Institute for Cybersecurity and consists of network traffic flows recorded over a period of five working days [74].

The dataset includes a total of 84 features extracted from the network traffic, providing valuable information about various aspects of network communication [74]. These features encompass attributes such

as source and destination IP addresses, port numbers, protocol types, packet sizes, time durations, and other network traffic characteristics.

The CICIDS2017 dataset is commonly employed for training and evaluating machine learning models in the field of intrusion detection. It helps researchers and practitioners assess the performance and effectiveness of different detection algorithms and techniques. Therefore, it was downloaded from [75] to be used in this dissertation.

2.6.2.3 Data Preprocessing

Data preprocessing plays a crucial role in data mining processes, serving as an important initial step [76]. It encompasses various techniques such as handling missing data, normalizing data, identifying attribute relationships, reducing outliers, and selecting or extracting significant features. However, before delving into preprocessing techniques, it is essential to understand the different types of data and how to handle them appropriately. This understanding aids in selecting suitable methods for preprocessing.

Data can be broadly classified into two types: categorical (qualitative) and numerical (quantitative) [76], as shown in Figure 2.17.

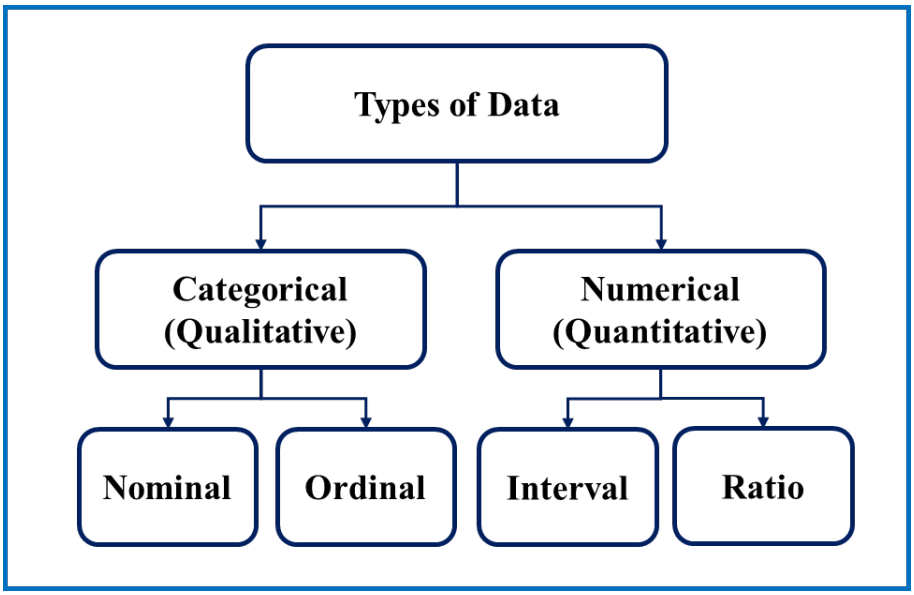


Figure 2.17: Types of data

Categorical data can be further divided into nominal and ordinal data. Nominal data assigns labels to objects solely for distinction purposes, which can be numerical or textual [76]. Examples include student ID numbers, zip codes, and car colors. On the other hand, ordinal data provides information for ordering or ranking objects, without the differences between values being necessarily meaningful. Examples include size of clothes (small, medium, large) or levels of happiness (happy, very happy, unhappy).

Numerical data consists of attributes with values derived from integers and can be categorized into two subtypes: interval and ratio data [76]. Interval data, similar to ordinal values, exhibits ordered differences between values. Examples include temperature measured in Celsius or Fahrenheit. Ratio data, on the other hand, possesses both distinctions and ratios between values, with a meaningful zero point. Examples include age, mass, and monetary quantities (e.g., the value of the Iraqi dinar in relation to the US dollar).

Additionally, attribute data can be classified as discrete or continuous [76]. Discrete data can be counted, aligning with the nominal category. Continuous data, on the other hand, is measured and corresponds to interval and ratio data.

In the context of data preprocessing, there exist several widely employed techniques for transforming and preparing data prior to its utilization in a machine learning or data analysis pipeline [77]. Here is a description of some preprocessing approaches:

1) Data Cleaning

Data in the real world often faces challenges related to incompleteness, noise, and inconsistency [76]. Incompleteness occurs when certain attributes in a dataset have null or missing values. To tackle this issue, the missing values can be addressed through the use of estimated data or imputation techniques. Noise, on the other hand, arises when the dataset contains invalid values or outliers that do not accurately represent the underlying data. Dealing with noisy data involves identifying and correcting or removing these inaccurate values or outliers to maintain the dataset's integrity. Inconsistency occurs when contradictory or conflicting data is present within specific features of the dataset, such as names or numerical values. Resolving inconsistencies requires the identification and rectification of conflicting data entries to ensure data coherence and accuracy.

2) Data Encoding

Encoding involves converting categorical variables into a numerical format that can be readily understood by machine learning

algorithms [76]. Categorical variables represent qualitative attributes like gender, color, or country. Encoding is essential as many machine learning algorithms operate on numerical inputs.

Data preprocessing commonly employs various encoding techniques. One such technique is Label Encoding, which assigns a distinct number to each category within a variable [77]. For instance, "normal" might be mapped to 0, while "attack" could be assigned 1.

3) Data Normalization

Normalization is a process used to convert the original attributes of a dataset into a more appropriate format for accurate predictive modeling, especially in deep learning [76]. Its purpose is to address issues related to large numerical values that can hinder computational efficiency and complicate the implementation process. Through normalization, a new set of values is generated, possessing desired properties that enhance the model's predictive capabilities.

One commonly employed normalization technique is Min-Max Normalization, also known as range transformation [78]. It involves scaling the data to fit within a predefined range. To perform Min-Max normalization, the minimum and maximum values of the original dataset are determined [76], [78]. These values are then used in the Equation 2.12 [78].

$$v_{new} = \frac{v - \min_x}{\max_x - \min_x} (\text{new } \max_x - \text{new } \min_x) + \min_x \qquad (2.12)$$

In this equation, v_{new} represents the result of min-max normalization, v represents the input value, min_x denotes the old minimum value, max_xrepresents the old maximum value, new $\min_x$ is the new minimum value, and new $\max_x$ represents the new maximum value [78].

By applying this formula, each value v is transformed to a new value v_{new} in such a way that it falls within the desired range defined by the new minimum and maximum values, new $\min_x$ and new $\max_x$ respectively. Min-max normalization allows for a standardized representation of the data within a specific range, facilitating comparisons and computations in machine learning algorithms [77], [78].

4) Feature Selection/Feature Extraction

Feature selection is the process of carefully choosing a subset of features from the original feature set [77]. The objective is to select the most relevant and informative features that significantly contribute to the predictive power of the model. By eliminating irrelevant or redundant features, feature selection helps reduce the dimensionality of the dataset, leading to improved model performance. Various criteria can be used for feature selection, such as analyzing the correlation with the target variable, applying statistical measures, or leveraging domain knowledge.

On the other hand, feature extraction aims to transform the original features into a new set of features [79]. The primary goal is to create a more concise representation of the data while retaining

essential information. This can be achieved by combining or deriving new features based on the original ones. Feature extraction techniques focus on capturing underlying patterns or structures present in the data, enabling a reduced-dimensional representation that still preserves crucial information relevant to the modeling task [78], [79].

2.7 Software Tools for Network Slicing Execution

This section provides an describes of the software tools employed in establishing a network slicing environment, encompassing message monitoring and analysis. In the context of this dissertation, the Mininet emulator serves as the SDN data plane, FlowVisor operates as the virtualization layer, and POX controllers are utilized to emulate the control plane of network slices. Moreover, Wireshark is employed for the analysis of communication messages.

2.7.1 Mininet

Obtaining SDN-enabled devices like switches and routers can be challenging and costly, making virtual network emulators a preferred option for researchers to conduct experiments [80]. One such emulator is Mininet, which creates virtual networks consisting of hosts, links, switches, and controllers. It allows for the execution of large networks with high performance on a single computer. Mininet utilizes lightweight virtualization mechanisms to create developable SDNs [81]. It also supports the internet topology zoo, which provides a database of real-world topologies in a graphical format.

There are several characteristics that make Mininet a preferred choice [82]:

1) Flexibility: It allows for the addition of new topologies in SDNs using programming languages and specific operating systems.
2) Scalability: It provides an environment suitable for large networks with thousands of switches on a single computer.
3) Interactivity: Network simulation and management occur in real-time, mimicking actual networks.
4) Applicability: Applications designed for the virtual environment can be executed in actual networks without modifying the source code.
5) Shareability: Developers can easily share their prototypes with collaborators, enabling them to modify and enhance their experiments.

Other network simulators exist, such as EMULAB, which employs OS-level virtualization but does not support the OpenFlow protocol [83]. The ns-3 simulator also supports OpenFlow, but only in older versions [84]. EstiNet supports the OpenFlow protocol and is used for simulating SDNs and testing the performance and functions of OpenFlow controllers [85]. However, Mininet generally offers better performance compared to these alternatives.

2.7.2 FlowVisor

The FlowVisor is an open-source network virtualization tool widely adopted in SDN environments [44]. It serves as a virtualization layer between the SDN controller and the underlying network

infrastructure, enabling secure sharing of physical network resources among multiple users or tenants while maintaining isolation and control over their respective virtual networks.

One of FlowVisor's primary functions is to partition network resources, including switches and links, into separate virtual slices [86]. Each slice operates as an independent network, allowing different users or applications to have their own isolated virtual networks within the shared physical infrastructure. FlowVisor ensures that the traffic and policies of one slice do not interfere with or impact the operation of other slices.

FlowVisor achieves this by intercepting and processing OpenFlow messages between the SDN controller and the switches [44], as shown in Figure 2.18. It enforces slice-specific policies, isolates traffic, and guarantees that one slice's traffic does not disrupt the functioning of other slices [44].

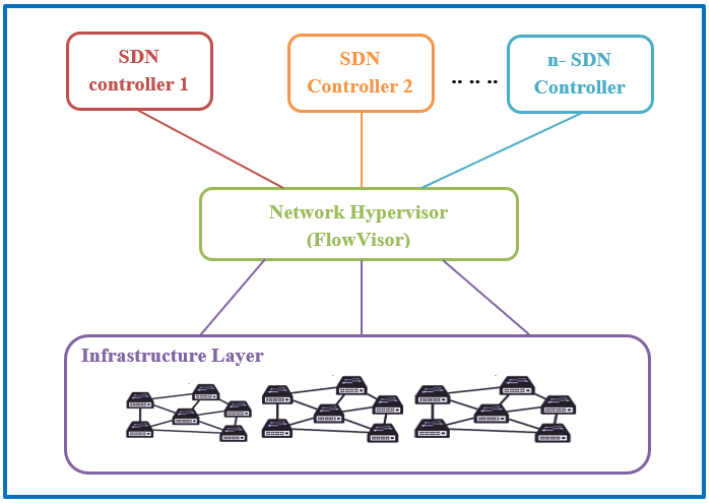


Figure 2.18: Location FlowVisor in Network Architecture

A key concept in FlowVisor is the flowspace, which defines the rules or match fields governing the classification and forwarding of network traffic within a slice [44]. Each slice in FlowVisor possesses its unique flowspace, allowing for precise control over network resources and policies.

The flowspace in FlowVisor is defined using OpenFlow match fields such as source and destination MAC addresses, IP addresses, transport protocol ports, and other packet header fields [87], [88]. Administrators can configure flowspace to specify rules and actions for different types of network traffic within a slice. In general, FlowVisor's architecture consists of two main layers: the Slicing Policies Layer and the Logic Abstraction Layer [44], as depicted in Figure 2.19.

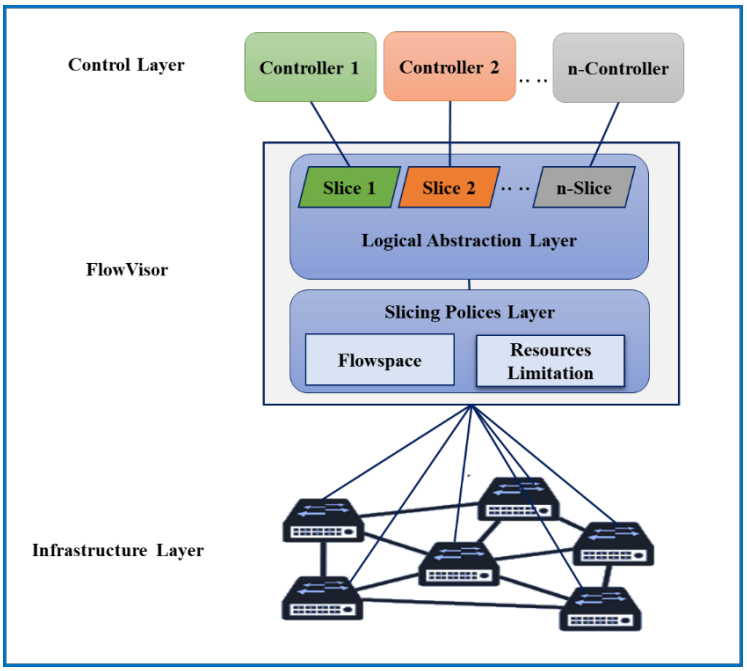


Figure 2.19: FlowVisor Architecture

The Slicing Policies Layer, located at the bottom of the architecture, is responsible for managing resource allocation within FlowVisor [44]. It

handles the allocation of link bandwidth, network topology, and forwarding entries for each network slice. This layer ensures fair resource sharing by preventing any single slice from monopolizing all the hardware resources. It also determines the optimal packet forwarding path for each slice based on predefined flow rules defined in the flowspace.

The Logic Abstraction Layer, positioned at the top of the architecture, provides a logical representation of routers and switches within FlowVisor [44]. It creates a virtualized network environment for each tenant controller, enabling customized network configurations. This layer configures and presents the sliced network to individual tenant controllers based on information obtained from the Slicing Policies layer. By abstracting the underlying physical infrastructure, the Logic Abstraction Layer enables flexible and isolated network slices for different tenants or applications.

Overall, FlowVisor's architecture consists of these two layers, each playing a crucial role in managing resource allocation, flow control, and logical abstraction to support network slicing [86].

2.7.3 POX Controller

The POX controller is an open-source SDN controller derived from the NOX controller [89]. It serves as a development platform for creating SDN controllers using the Python programming language. The POX controller is designed to be lightweight, flexible, and extensible, making it a popular choice among researchers and developers in the SDN community.

One of the key advantages of the POX controller is its simplicity [90]. It offers a minimalistic and easy-to-understand codebase, allowing developers to quickly build and customize their SDN applications. The POX controller supports the OpenFlow protocol, which enables communication and control between the controller and network switches.

The flexibility of the POX controller allows for the execution of diverse applications, such as hubs, switches, load balancers, and firewalls [89]. It enhances efficiency in managing OpenFlow devices, making it a valuable tool for SDN research and development. Due to its versatility and extensive usage in the SDN community, the POX controller has gained a strong reputation as a reliable choice for SDN projects.

While POX enjoys widespread usage as an SDN controller, it's important to recognize that there are other popular alternatives available within the SDN ecosystem [90]. Table 2.1 provides a comparison that includes the POX controller along with some commonly compared alternatives in the SDN ecosystem: OpenDaylight, Ryu, and ONOS [89], [91]–[93].

Table 2.1: The Comparison Between Some Types of Controllers

Controller	Key Features	Language	Extensibility	Protocol Support	Scalability
POX	Lightweight, flexible, and extensible	Python	Highly extensible with a simple codebase	OpenFlow	Suitable for smaller deployments
OpenDaylight	Mature and feature-rich	Java	Highly extensible with a modular architecture	OpenFlow and more	Suitable for large-scale networks with a wide range of protocols

Ryu	Simple and developer-friendly	Python	Extensible with a modular architecture	OpenFlow, NETCONF, and more	Suitable for various SDN applications with ease of development
ONOS	Highly scalable and distributed	Java	Extensible with advanced features like dynamic network slicing	OpenFlow and more	Designed for carrier-grade deployments and large-scale networks

2.7.4 Wireshark

Wireshark is a widely recognized and extensively used powerful network protocol analyzer in the field [94]. It serves as a valuable tool for observing and measuring the functionality of Mininet networks. Wireshark's primary function is to capture and analyze all network packets transmitted through network interfaces. By capturing network packets, Wireshark allows users to inspect and interpret the contents of individual packets, including their source and destination addresses, protocols used, and data payload [94]. This capability enables deep analysis of network traffic, making it useful for troubleshooting, security analysis, and performance optimization.

Wireshark supports a wide range of network protocols and provides a user-friendly interface for navigating and analyzing captured packet data. It offers various filters and search functionalities to focus on specific packets or network conditions [95]. With its extensive features and capabilities, Wireshark is an indispensable tool for network

administrators, researchers, and anyone involved in network analysis and debugging.

2.8 Performance Evaluation Metrics

The evaluation of the proposed methodology's performance in detecting attack and normal traffic can be assessed using various metrics. One commonly used metric is the binary confusion matrix, which provides a clear representation of the system's performance [76]. The binary confusion matrix consists of four elements: True Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN), as shown in Table 2.2.

Table 2.2: A Binary Confusion Matrix

		Predicated	
		Attack	**Normal**
Actual	**Attack**	TP	FN
	Normal	FP	TN

- **TP** is equal to any correctly denied, and it represents the number of attack records that identified as an attack.
- **FP** is equal to any incorrectly denied, and it represents all the normal records that predicted it as an attack.
- **FN** is equal to any incorrectly accepted, and it represents all the attack records that predict it as normal.
- **TN** is equal to any correctly accepted, and it represents all the normal records that predict it as normal.

To evaluate the system's performance, several metrics can be calculated based on the confusion matrix. The metrics commonly used in this context are Accuracy, Precision, Recall, and F1-score [96].

1) **Accuracy:** measures the percentage of correctly classified packets (both attack and normal) over the total number of packets classified by the system, as shown in Equation 2.13[76].

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (2.13)$$

2) **Recall:** also known as True Positive Rate or Sensitivity, measures the proportion of correctly detected attack (TP) over the total number of actual attacks, as shown in Equation 2.14 [76].

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.14)$$

3) **Precision:** measures the proportion of correctly detected anomalies (TP) over the total number of packets classified as anomalies (both true and false positives), as shown in Equation 2.15 [76].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.15)$$

4) **F1-score:** is the harmonic mean of Precision and Recall, providing a balanced measure of the system's performance, as shown in Equation 2.16 [76].

$$\text{F1} - \text{score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.16)$$

In addition to these metrics, the efficiency of the proposed system can be evaluated using specific measures based on the confusion matrix:

1) **Attack Detection Rate (ADR):** ADR calculates the ratio of correctly detected attack packets to the total number of attack packets, as shown in Equation 2.17 [76].

$$\mathrm{ADR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} \tag{2.17}$$

2) **False Alarm Rate (FAR):** FAR, also known as False Discovery Rate, calculates the proportion of normal packets incorrectly classified as attack packets to the total number of actual normal packets, as shown in Equation 2.18 [76].

$$\mathrm{FAR} = \frac{\mathrm{FP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.18}$$

3) **False Positive Rate (FPR):** FPR measures the proportion of normal packets incorrectly classified as attack packets to the total number of packets classified as attack packets, as shown in Equation 2.19 [76].

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2.19}$$

To achieve successful attack detection, the methodology should aim for high accuracy, low FAR, and high ADR. These measures collectively indicate the effectiveness and efficiency of the proposed methodology in detecting anomalies while minimizing false alarms and maximizing the detection of actual attacks [96].

4) **Specificity:** is a crucial performance metric in the field of attack detection within machine learning. In the context of attack detection,

binary classification tasks are common, aiming to identify instances as either normal or attack. Specificity becomes highly relevant as it measures the model's ability to accurately classify negative instances, i.e., correctly identifying normal instances and distinguishing them from attacks.

To calculate specificity, the ratio of TN to the sum of TN and FP is computed, as shown in Equation 2.20 [76].

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2.20}$$

In attack detection, specificity complements sensitivity to provide a comprehensive evaluation of the model's performance. While sensitivity focuses on the model's ability to correctly identify attacks, specificity focuses on correctly identifying normal instances.

2.9 Summary

This chapter has established the groundwork for subsequent chapters by delving into the foundations of NS and exploring associated techniques. It has provided a comprehensive overview of NS, encompassing its architecture, life cycle, and classifications. Additionally, the chapter has explained the diverse architectures employed for implementing network slicing in various scenarios. Furthermore, the chapter has addressed the crucial aspect of NS security, specifically focusing on the implications of DoS/DDoS attacks within these networks. It has also elaborated on the techniques utilized to identify and detect such attacks, ensuring the security and availability of the network.

Moreover, the chapter has discussed the software tools utilized for network analysis and dissection, along with the evaluation criteria employed to assess their performance. By establishing this foundation, the chapter has set the stage for subsequent chapters to further expand upon these topics.

CHAPTER THREE

The Environment Setup for NS and Creating NSDS

3.1 Overview

This chapter presents an overview of the proposed work, which focuses to detect DoS/DDoS attacks in a NS environment. It highlights the essential steps involved in constructing the attack detection methodology, including the proposed NS architecture for building a testbed using SDN technology and network virtualization. The chapter also explains the simulation of both normal traffic generation and the attack traffic within the network slicing environment. Finally, it discusses the construction of the dataset, utilizing the NS environment as the basis.

3.2 The Proposed Work

To enable the detection of attacks in a network slicing environment, it is necessary to generate network traffic within a network slicing testbed. Subsequently, the proposed methodology for attack detection can be applied. This section will provide a comprehensive description of the workflow, as illustrated in Figure 3.1.

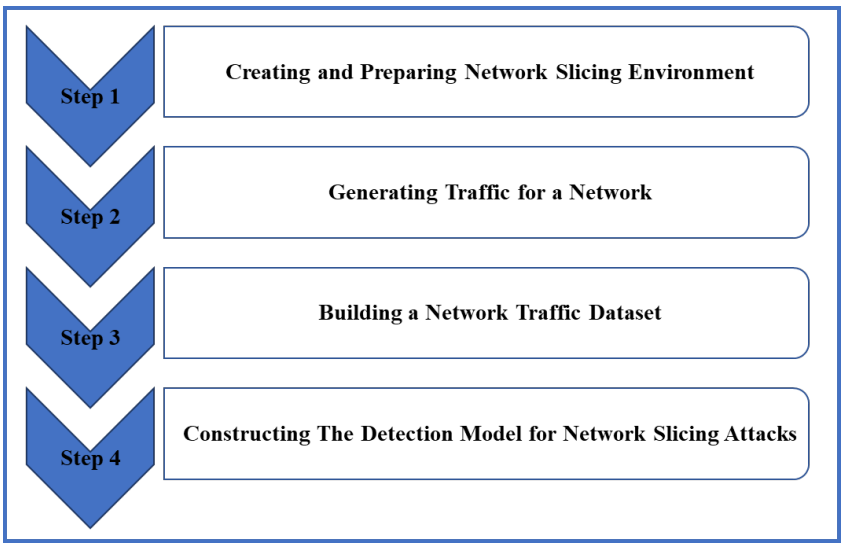


Figure 3.1: Workflow for Attack Detection in Network Slicing Environment

3.2.1 Creating and Preparing Network Slicing Environment

The concept of NS is implemented using SDN and NV technologies. Figure 2.3 illustrates the proposed network slicing architecture based on these technologies. This architecture consists of several components, each represented by a virtual machine, including the infrastructure layer, the virtualization layer, the control layer, and a third-party server.

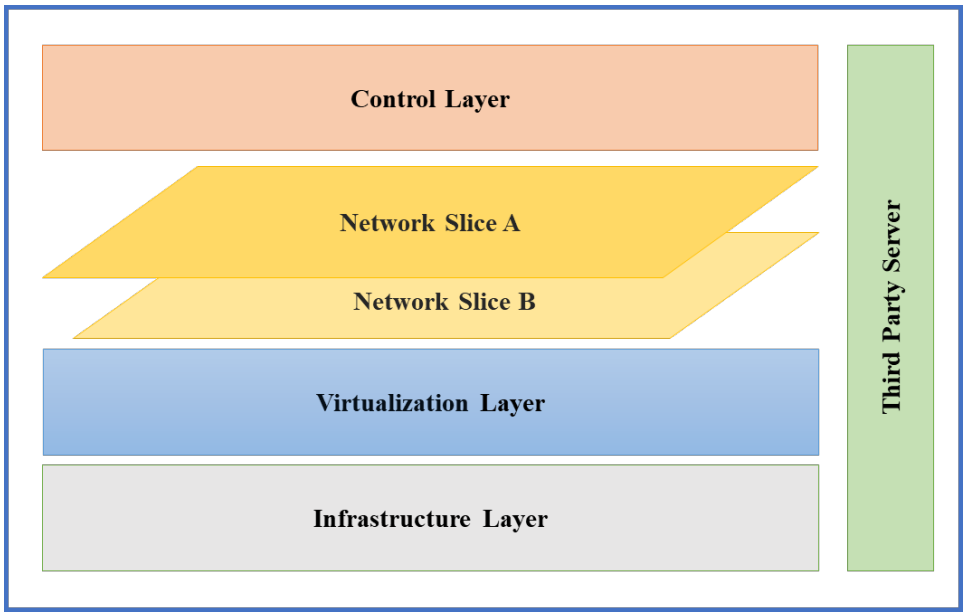


Figure 3.2: Proposed Network Slicing Environment Architecture

3.2.1.1 The Infrastructure Layer

In the proposed NS environment, the infrastructure layer is based on SDN technology, and it consists of the physical components that form the network's foundation. These components include switches that support the OpenFlow protocol and the number of hosts connected to the network. Figure 3.3 depicts these physical components within the infrastructure layer.

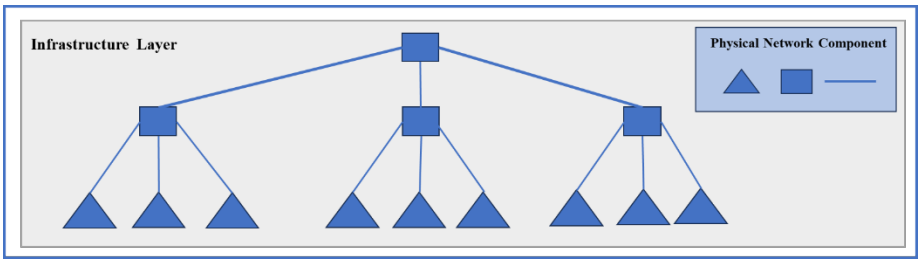


Figure 3.3: Physical Component in Infrastructure Layer

The physical components within the infrastructure layer, including OpenFlow-supported switches and hosts, establish the foundation for creating virtualized networks in the virtualization layer. These physical components are crucial for enabling the creation and operation of network slices within the NS environment based on SDN technology.

3.2.1.2 The Virtualization Layer

In the proposed environment, the virtualization layer utilizes NV technology to create virtual network components from the physical devices in the infrastructure layer. This approach enables effective resource allocation within the NS environment, resulting in optimized resource utilization. To achieve this, a specific virtualization software called FlowVisor is employed.

By leveraging FlowVisor, the proposed environment establishes two distinct network slices that are completely isolated from each other. Figure 3.4 provides a visual representation of the proposed environment, depicting the presence of the two network slices alongside their corresponding virtual network components.

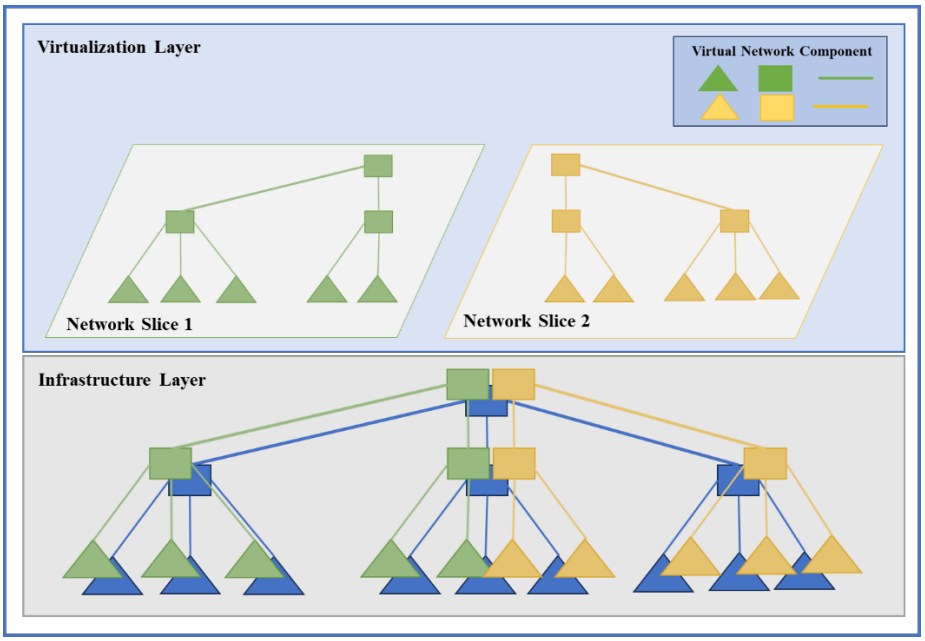


Figure 3.4: Network Slices and Virtual Components in a Network Slicing Environment

3.2.1.3 The Controller Layer

In the proposed environment, the network architecture includes the creation of two network slices. To effectively manage and control resources and services within each slice, two SDN controllers, specifically POX controllers, are employed in the Control layer. This is depicted in Figure 3.5.

Each SDN controller is assigned the responsibility of managing the virtual network components within its corresponding slice. This involves tasks such as traffic routing and enforcing network policies. The controllers achieve these functionalities through the utilization of the OpenFlow protocol, which enables centralized control and programmability in the network environment.

By employing separate SDN controllers for each slice, the proposed environment ensures independent management and control of the virtualized network components associated with each slice. This enables efficient routing of network traffic and the enforcement of appropriate network policies within each individual slice.

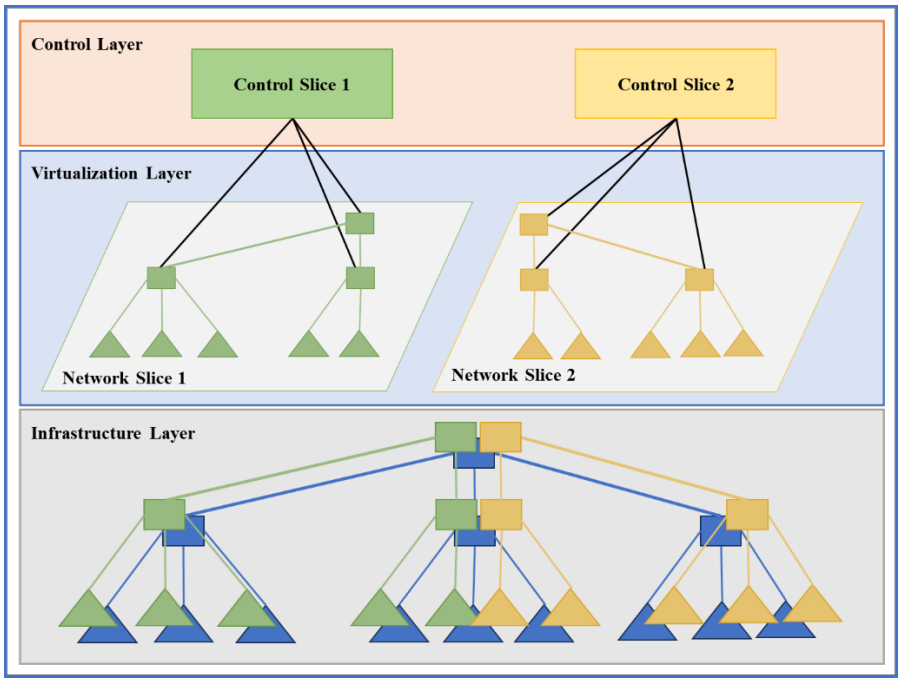


Figure 3.5: Network Slicing with Multiple Slices and SDN Controllers

3.2.1.4 The Third-Party Server

Within the proposed NS environment, a third-party server is utilized to execute cross-level detection. This server is external to the network slices and is connected to each individual slice, enabling it to receive real-time traffic parameters from every slice.

The primary purpose of this external server is to conduct cross-level detection, which involves analyzing network traffic across multiple slices. By examining traffic patterns and behaviors from different slices

simultaneously, it becomes possible to detect DoS/DDoS attacks within the NS environment.

A detailed explanation of cross-level detection and its role in detecting DoS/DDoS attacks in the NS environment will be provided in Chapter Four.

3.2.2 Generating Traffic for a Network slicing

Network traffic in a NS environment differs greatly between normal traffic and DoS/DDoS attacks. Normal traffic is generated by legitimate users and devices, while the attacks are generated by malicious actors with the intent of disrupting the network or a particular service. Thus, this section will explore the traffic characteristics involved in generating normal traffic and attack traffic.

3.2.2.1 The Normal Traffic

To simulate the behavior of normal network traffic, packets are generated with random combinations of IP addresses, source ports, destination IP addresses, and destination ports. Each source and destination IP address consists of four octets. The first octet is used to control the IP address generation to avoid getting some special addresses like a private address, broadcast address, and loopback address. Meanwhile, the source port addresses are random within the limits of 49152 - 65535, which are reserved for clients, and destination port addresses are random within the limits of 0 - 1023, which are defined as well-known ports.

Also, packet arrival times are randomly set within a specified range. In our simulation, they were set between 0.1 and 0.9 seconds. This approach is described in Algorithm 3.1, which outlines the steps for generating normal traffic.

Algorithm (3.1): Normal Packets Generation

Input: Time period
Output: Generated Normal_Packet
1. **Begin**
// Generate Source and Destination IP Packets
2. Special_addresses ← [0,10,127,254,255,169,172,192];
3. **For** IP in [1.0.0.0 - 255.255.255.255] **do**:
4. | **While** first_octet ∉ Special_addresses **do:**
5. | | src_IP ← Generate a random source IP address;
6. | | dst_IP ← Generate a random destination IP address;
7. | **End While**
8. **End For**
// Generate Source Port
9. Special_client_ports ← [49152 - 65535];
10. **While** Source_port ∈ Special_client_ports **do:**
11. | src_port ← Generate a random Source_port;
12. **End While**
// Generate Destination Port
16. Well_known_ports ← [0 – 1023]
17. **While** Destination_port ∈ Well_known_ports **do:**
18. | dst_port ← Generate a random Destination_port;
19. **End While**
20. **While** Current_time is within Time period **do:**
21. | Inter_Arrival_Time ← [0.1 – 0.9];
22. | Normal_Packet ← Generate Packet with (src_IP, src_port, dst_IP, dst_port) and random payload;
23. | Send Normal_Packet for each Inter_Arrival_Time;
24. **End While**
25. **End Algorithm**

3.2.2.2 The Attack Traffic

When simulating the behavior of DoS/DDoS attack traffic, it is essential to accurately capture its unique characteristics that differentiate

it from normal network traffic. While generating packets for source IP addresses and ports can follow a similar approach to normal traffic, there are distinct considerations for destination IP addresses and ports.

For destination IP addresses, they are randomly selected from a specific set of IPs. This random selection accurately reflects the targeted nature of DoS/DDoS attacks, where the attack traffic is often directed towards one or more specific IP addresses. By incorporating this randomness into the simulation, the targeted nature of the attacks can be accurately represented, whether it is a DoS or DDoS attack.

In addition, DoS/DDoS traffic targets specific services by directing traffic towards one or more destination port addresses. To simulate this aspect, destination port addresses are considered, as they play a crucial role in targeting and affecting the intended services.

To simulate the high packet rate of DoS/DDoS attacks, the proposed traffic generation algorithm relies on the number of botnets to determine the inter-arrival time between packets. Where the number of botnets is randomly chosen between 1-7 botnet and the value of the inter-arrival time between packets using a single botnet is 0.08 seconds. So, in the case of an attack involving multiple botnets, such as five, the inter-arrival time will be adjusted to 0.016 seconds, as shown in Algorithm 3.2. The number 0.016 is calculated using Equation (3.1).

$$\mathrm{IAT} = \frac{0.08}{\mathrm{No.\,of\_botnets}} \tag{3.1}$$

Where IAT is the inter-arrival time in seconds and No. of_botnets is the number of botnets that are running.

Algorithm (3.2): Attack Packets Generation

Input: Special_range_IP_addresses, Special_ports, Time period
Output: Generated Attack_Packet
1. **Begin**
// Generate Source IP Packets
2. Special_addresses ← [0,10,127,254,255,169,172,192];
3. **For** IP in [1.0.0.0 - 255.255.255.255] **do:**
4. | **While** first_octet ∉ Special_addresses **do:**
5. | | src_IP ← Generate a random source of IP address;
6. | **End While**
7. **End For**
// Generate Source Port
8. Special_client_ports ← [49152 - 65535];
9. **While** Source_port ∈ Special_client_ports **do:**
10. | src_port ← Generate a random Source_port;
11. **End While**
12. dst_IP ← Special_range_IP_addresses;
13. dst_port ← Special_ports;
14. num. of_botnet ← random [1-7];
15. **While** Current_time is within Time period **do:**
16. | Attack_Packet ← Generate Packet with (src_IP, src_port, dst_IP, dst_port);
17. | Inter_Arrival_Time ← 0.08/ num. of_botnet;
18. | Send Attack_Packet for each Inter_Arrival_Time;
19. **End While**
20. **End Algorithm**

3.2.3 Creation a Network Slicing Dataset

Our NSDS was generated based on the NS architecture discussed in Section 3.2.1. This dataset contains both normal and attack traffic, providing a comprehensive collection of network traffic types. The attack traffic includes four attack scenarios: DoS attacks on a single victim by a single host (DoS_SS), DoS attacks on multiple victims by a single host (DoS_MS), DDoS attacks on a single victim by multiple hosts (DDoS_SM), and DDoS attacks on multiple victims by multiple hosts (DDoS_MM). These attack scenarios were generated in each network

slice using the algorithms described in Section 3.2.2, which provide a detailed description of the traffic generation process.

In order to gather traffic information from each network slice, an application has been developed specifically for this purpose. The application operates by collecting packet header field information from packets transmitted from the switches to the controller. This approach enables the collection of the necessary data to build the NSDS.

The application is executed on each controller within the network slice, allowing it to gather the required information from the packet headers. By leveraging this application, the dataset can be constructed by capturing both normal traffic and attack traffic. The dataset encompasses various attack scenarios, including situations where attacks are launched on one or both slices simultaneously. This ensures that the recordings within the dataset accurately represent different attack scenarios and facilitate the development and evaluation of effective security measures.

The dataset we have obtained comprises 14 distinct features, which are as follows: Packet Arrival Time (PAT), Forward Inter-Arrival Time (Fwd IAT), Source MAC Address (Src MAC), Destination MAC Address (Dst MAC), Switch ID, In port, Source IP Address (Src IP), Destination IP Address (Dst IP), Source Port (Src Port), Destination Port (Dst Port), Packet Size (pkt size), Protocol, Slice Name, and Label. These features have been meticulously listed in Table 3.1, alongside detailed descriptions that provide a comprehensive understanding of each feature's role and characteristics.

Furthermore, NSDS is designed to be easily scalable and adaptable to different network architectures and configurations. Where, the application is used to extract additional traffic information from each network slice, allowing the creation of more specialized datasets tailored to specific research needs or network environments. This flexibility enables network administrators to focus on specific aspects of network traffic, such as traffic patterns or the behavior of particular protocols, and to develop more targeted security solutions.

Table 3.1: List of Features in the Created NSDS

Feature	Description
PAT	The time at which the packet arrived
Fwd IAT	The time interval between two consecutive packets in the forward direction
Src MAC	The MAC address of the source of the traffic
Dst MAC	The MAC address of the source of the traffic
Switch ID	The identifier of the switch in the network
In port	The port on the switch where the traffic entered
Src IP	The IP address of the source of the traffic
Dst IP	The IP address of the destination of the traffic
Src Port	The port number of the source of the traffic
Dst Port	The port number of the destination of the traffic
pkt size	The size of the packet in bytes
Protocol	The protocol used for the traffic (e.g., TCP, UDP)
Slice Name	The name of the network slice to which the traffic belongs
Label	The classification label for the traffic (e.g., normal, Attack)

3.2.4 Constructing The Detection Model for Network Slicing Attacks

This section extends into Chapter Four to discuss the critical workflow step, which is the innovative methodology for detecting DoS/DDoS attacks in a network slicing environment. This methodology consists of two levels: the slice-level and the cross-level.

The first level is based on statistical analysis to perform early detection of attacks at the slice-level, while the second level involves classifying traffic based on the DDNN model at the cross-level. This methodology will be discussed in detail in the next chapter.

3.3 Summary

The proposed workflow for detecting attacks in a NS environment was presented in this chapter, with a specific emphasis on the first three steps. The first step involved the construction of a NS environment based on SDN and NV, serving as the foundation for subsequent steps. In the second step, methods were proposed for generating both normal and attack traffic, essential for the development of effective security solutions in NS environments. The third step focused on the creation of a comprehensive dataset for NS, utilizing the proposed environment from the first step and the traffic generated in the second step. This dataset serves as a valuable resource for training and evaluating the detection model in the fourth step.

While the fourth step forms the basis of the proposed methodology for detecting the attacks in a NS environment, this step will be explained in detail in Chapter Four.

CHAPTER FOUR

The Innovative Methodology for DoS/DDoS Attack Detection in NS

4.1 Overview

In this chapter, the critical challenge of detecting DoS/DDoS attacks in a network slicing environment is addressed through an effective detection methodology. The methodology introduces an architecture specifically designed for detecting the attacks in a NS environment. This architecture comprises two levels of attack detection: the slice-level and the cross-level. The models and techniques used at each level are extensively explored in this chapter. Furthermore, the process of integrating these two levels to seamlessly operate within a NS environment is delved into, ensuring comprehensive detection of attacks.

4.2 Innovative Methodology

The Innovative methodology for detecting DoS/DDoS attacks in a NS environment consists of the integration of two main levels: Slice-level early attack detection and Cross-level deep attack detection.

Slice-level early attack detection involves using statistical techniques to classify the traffic within a single network slice as either normal or suspicious. This is done by analyzing the destination IP and destination port network traffic parameters within the slice and comparing them to the dynamic threshold. This statistical analysis is performed in real-time and is used to quickly detect suspected the attacks within a single slice.

Cross-level deep attack detection involves analyzing suspicious traffic across multiple network slices to detect the attacks. This is done

using a DDNN that is trained to detect the attacks in traffic across different slices.

The integration of slice-level detection and cross-level detection provides enhanced the attacks detection in a network slicing environment by detecting the attacks at multiple levels. The statistical analysis provides a fast and efficient method to analyze traffic within a single network slice, while DDNN provides a more accurate and sensitive method to detect the attacks across multiple slices. Figure 4.1 illustrates the architecture of the innovative methodology for detecting the attacks in a NS environment using the integration of slice-level detection and cross-level detection.

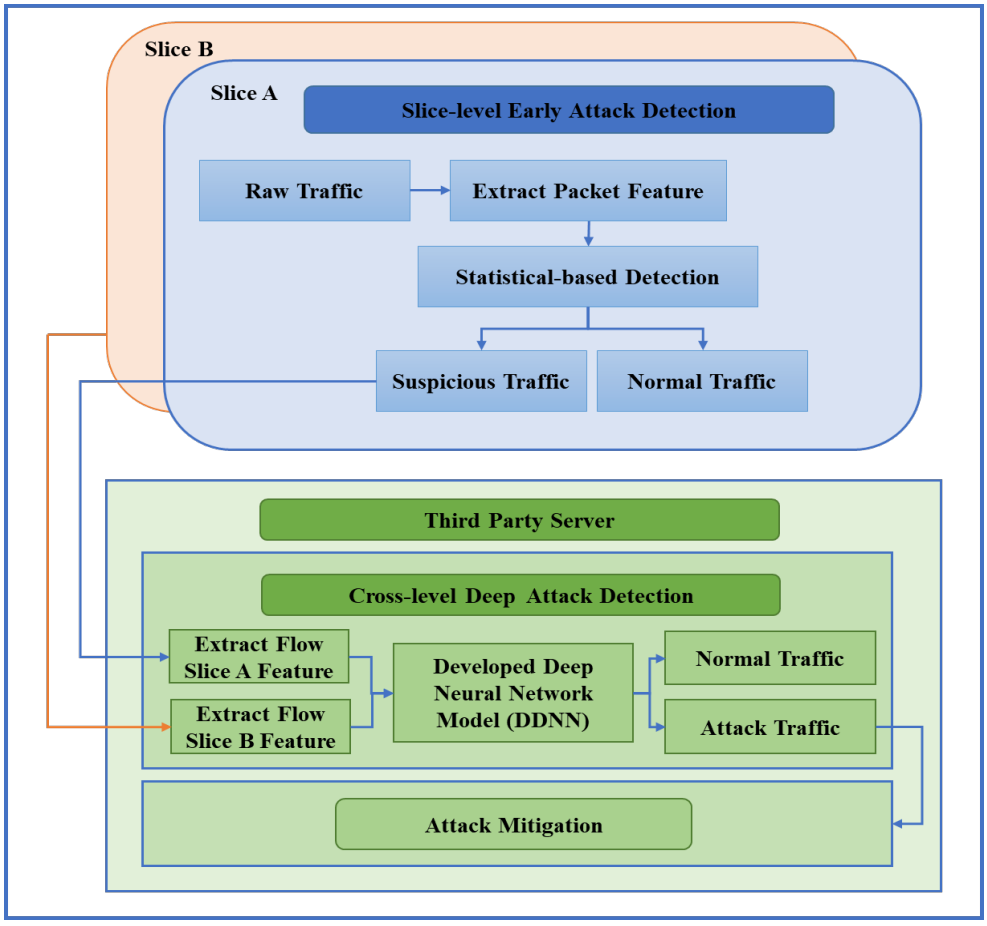


Figure 4.1: The Innovative Methodology Architecture for Detecting DoS/DDoS Attacks in Network Slicing Environment

4.2.1 Slice-Level Early Attack Detection

The slice-level early attack detection, based on statistical analysis, has been designed to operate on each network slice. It comprises four crucial steps: feature extraction from the packet header, initial threshold calculation, attack detection, and updated threshold calculation. A flowchart depicting the slice-level detection process is presented in Figure 4.2.

Overall, the slice-level detection is designed to provide a fast and efficient way of detecting suspected the attacks using the joint entropy of the destination IP and destination port within a single network slice. By analyzing the joint entropy values of the destination IP and destination port within each network slice, the system can differentiate between normal and suspicious traffic patterns. Furthermore, the slice-level detection is integrated into a larger the attacks detection system to improve detection and provide enhanced protection against attacks.

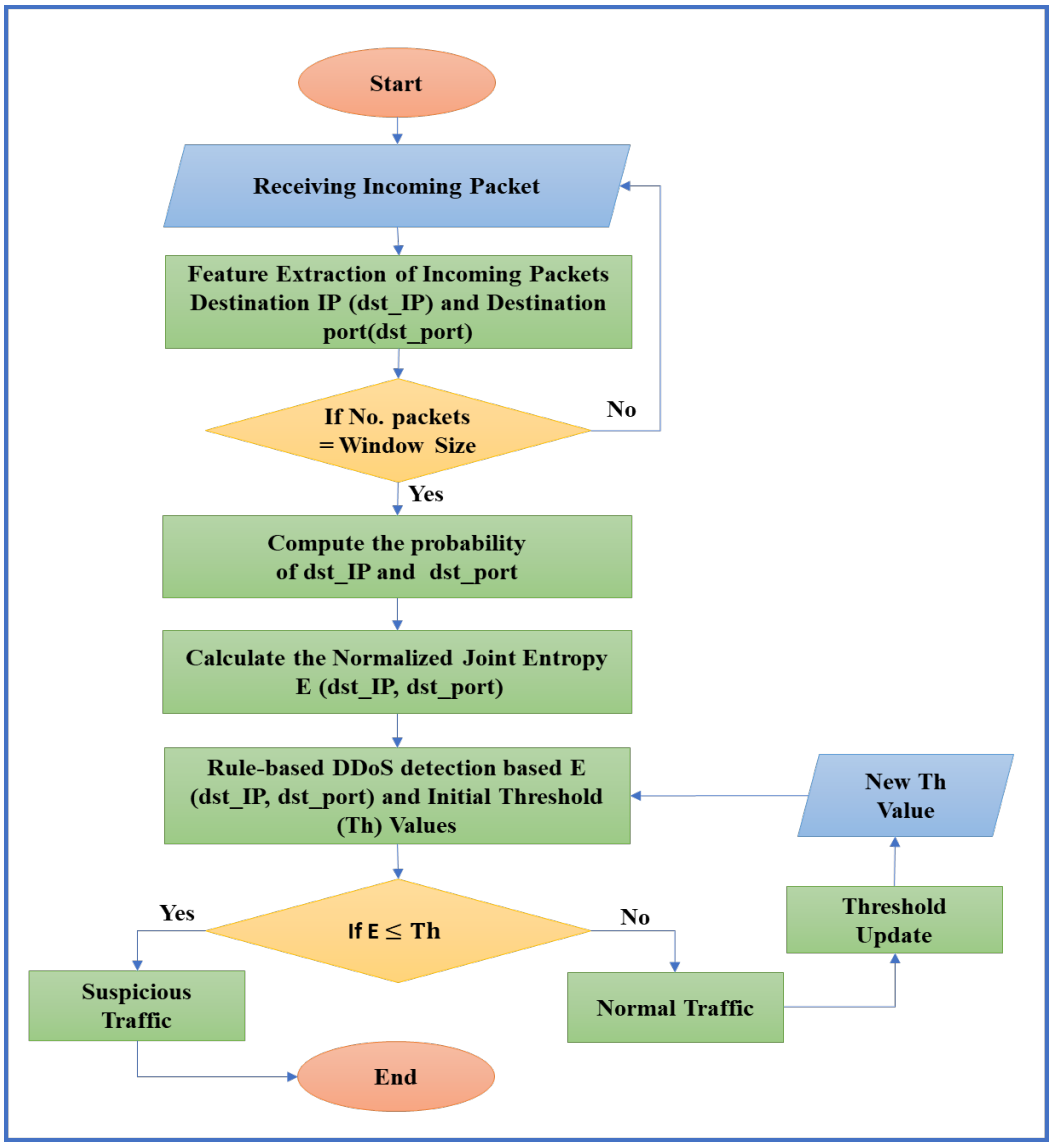


Figure 4.2: Flowchart of Slice-level Detection

4.2.1.1 Feature Extraction from the Packet Header

The statistical-based detection of suspected DoS/DDoS attacks involves collecting packet information from incoming packets within each window size of 50 packets, as described in Algorithm 4.1.

Algorithm (4.1): Feature Extraction and Dictionary Construction

Input: Incoming_Packet, Window_size
Output: Dictionary (pair [Dst_IP, Dst_port], count)
1. **Begin**
2. Clear Dictionary;
3. **For** all Incoming_Packet received by controller **do:**
4. | Collect pair [Dst_IP, Dst_port] corresponding Incoming_Packet in controller
5. **End For**
6. **For** all pair [Dst_IP, Dst_port] ∈ Window_size **do**:
7. | **If** pair [Dst_IP, Dst_port] ∉ Dictionary (pair [Dst_IP, Dst_port], Occurrences count) **do:**
8. | **Add** Dictionary (pair [Dst_IP, Dst_port], Occurrences count) = + 1;
9. | **Else**
10. | **Update** Dictionary (pair [Dst_IP, Dst_port], Occurrences count + 1);
11. | **End If**
12. **End For**
13. **Return** Dictionary;
14. **End Algorithm**

During this step, the destination IP and destination port information are extracted from the packet header to build a dictionary. If the destination IP address and destination port are new to the dictionary, they are added to the table with an initial counter value of one. Conversely, if the IP address and port already exist in the dictionary, the counter value is incremented accordingly.

The use of a window size of 50 packets enables the implementation of a "short-term" statistics-based detection approach, as explained in [97]. This approach involves analyzing network traffic data in smaller window sizes, allowing for more precise and immediate detection of suspected the attacks. Additionally, by using a smaller window size, the system can quickly detect changes in traffic patterns and respond promptly to suspected DoS/DDoS attacks.

4.2.1.2 Initial Threshold Calculation

The statistical-based detection uses Chebyshev's theorem to establish the initial threshold value for detection. This involves calculating the mean (μ) and standard deviation (σ) of the joint entropy values derived from a sample of normal traffic. By leveraging Chebyshev's theorem, the initial threshold is determined based on the calculated mean and standard deviation, allowing for effective differentiation between normal and suspected malicious traffic.

Once the mean and standard deviation are determined, Equation (4.1) is applied to set the initial threshold value (Th) based on Chebyshev's theorem:

$$\mathrm{Th} = \mu - \mathrm{K} \cdot \sigma \qquad (4.1)$$

Where μ represents the mean joint entropy value for the sample of normal traffic. σ represents the standard deviation of the joint entropy values for the sample of normal traffic, and k is a constant that determines the number of standard deviations from the mean to be included in the threshold. By setting an initial threshold value, the statistical-based detection establishes a basis for distinguishing between normal and suspicious traffic behavior.

4.2.1.3 Attack Detection

The statistical-based detection involves calculating the joint probability of the destination IP address and destination port address within a packet window using Equation (4.2) [98]. By considering the joint probability, the approach can effectively capture correlations

between the destination IP and port addresses found in incoming packets. These correlations serve as valuable indicators for identifying suspected DoS/DDoS attacks.

$$p(x_i, y_j) = \frac{\text{Number of packets with dest. IP}_i \text{ and dest. port}_j}{\text{Total number of packets}} \quad (4.2)$$

The statistical-based involves calculating the randomness of two variables: the destination IP address and destination port address, using Shannon joint entropy [59]. The calculation of Shannon joint entropy, denoted as x and y, is used to measure the level of randomness exhibited by these variables. Equation (4.3) represents the mathematical expression used to compute the Shannon joint entropy, which enables the determination of the randomness of the destination IP address and destination port address variables.

$$H(x, y) = -\sum_{i=1}^{M}\sum_{j=1}^{N} p(x_i, y_j).\ \log_2 p(x_i, y_j) \quad (4.3)$$

In the context of the Shannon joint entropy equation, H(x,y) refers to the joint entropy of two variables, x and y. While, the joint probability of x and y, represented as $p(x, y)$, is the probability that both x and y occur together.

The equation for calculating the normalized joint entropy value is not given in the previous equation. However, the calculation of normalized joint entropy involves dividing the joint entropy by the maximum possible entropy, which provides a normalized value between 0 and 1. The normalized joint entropy Equation (4.4)[59].

$$H_{norm(x,y)} = \frac{H(x,y)}{\log_2 nWin} \quad (4.4)$$

Where $H_{norm(x,y)}$ is the normalized joint entropy and nWin is number of elements in window. By applying normalization to the joint entropy calculation, the joint entropy can be compared to the threshold, and patterns of non-randomness that indicate a suspected DoS/DDoS attack can be identified.

Once a threshold value has been chosen experimentally for the joint entropy calculation, if the resulting joint entropy value is found to be lower than the threshold, it may indicate the presence of a suspected DoS/DDoS attack. This is because a lower joint entropy value indicates a lower degree of randomness or unpredictability in the observed traffic patterns, which can be a characteristic of a suspected the attack, in which case it moves to Cross-level detection. Conversely, if the joint entropy value is higher than the threshold, it may indicate that the traffic patterns are more random and less likely to be the result of the attack, in which case the threshold value is updated. Algorithm 4.2 explains the main steps of joint entropy-based suspected traffic detection.

Algorithm (4.2): Suspicious Traffic Detection

Input: Dictionary, Threshold (Th)
Output: Action
1. **Begin**
2. **Call** Feature Extraction and Dictionary Construction Algorithm;
3. **For** each Dest_IP, Dest_port **in** the dictionary **do:**
4. | **Find** joint probability distribution of flows using Equation (4.2);
5. **End for**
6. **Find** joint entropy of probability using Equation (4.3);
7. **Find** normalized joint entropy using Equation (4.4);
8. **If** normalized joint entropy value < Th
9. Action ← Forward the suspicious traffic to Cross-level detection;
10. **Else**
11. Action ← Call the Threshold Update Algorithm;
12. **End If**
13. **End Algorithm**

4.2.1.4 Threshold Updating

The update threshold is used to adjust the threshold value over time based on changes in network traffic patterns. The update threshold value can be adjusted periodically based on observed traffic patterns to adapt to changes in the network environment and reduce the false positive rate.

This is done by deleting the oldest value in the N_List normal joint entropy (LE) values and then adding the new joint entropy value at the beginning of the list. Finally, the threshold is updated by calculating the mean and standard deviation of the joint entropy values depending on the updated LE of the joint entropy using Equation (4.1). Algorithm 4.3 describes the basic steps for calculating the updated threshold.

Algorithm (4.3): Threshold Update

Input: Current_Joint_Entropy (CE), N_List_Normal_Joint_Entropy (LE)
Output: New_Threshold (N_Th)
1. **Begin**
2. Delete the oldest element in LE;
3. Updating LE ← Add CE at the end LE;
4. N_Th ← Compute threshold for Updating LE using Equation (4.1);
5. **Return** N_Th;
6. **End Algorithm**

4.2.2 Cross-Level Deep Attack Detection

The cross-level deep attack detection is designed to analyze suspicious traffic across multiple slices using a DDNN model. The construction and training of this model involve two primary stages: the pre-processing stage for the datasets and the construction of the DDNN. The block diagram presented in Figure 4.3 illustrates these stages.

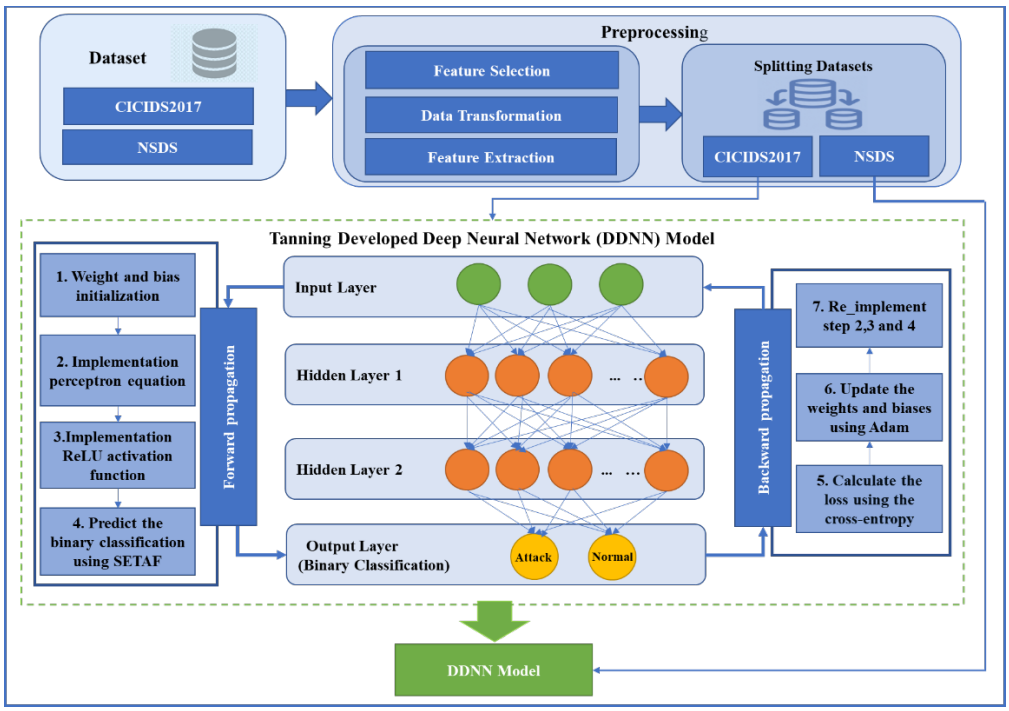


Figure 4.3: General Block Diagram for DDNN

The utilization of a DDNN in cross-level detection enhances the effectiveness of the detection in network traffic across multiple slices. This improvement in detection capability contributes to enhancing the overall security of NS.

4.2.2.1 Data Preprocessing Stage

To effectively train and evaluate a DDNN model using the CICIDS2017 dataset described in Chapter Two, and the NSDS constructed based on the NS environment as described in Chapter Three, a pre-processing task is required. This task involves four primary steps: feature selection, data transformation, feature extraction, and splitting. These steps are crucial to properly prepare the data, ensuring its suitability for the intended analysis and modeling purposes.

- **Feature Selection**

  In our work, a heuristic approach was employed, wherein features demonstrating robust classification performance while maintaining real-time operation were manually selected. Specifically, the focus was on four features commonly utilized in DoS/DDoS attack detection: Average Packet Length, Destination Port, Destination IP, and Fwd IAT Mean [99].

  These features were chosen based on their ability to provide valuable insights into the behavior of network traffic during the attack. By including these features in the analysis, relevant information that contributes to the effective detection of the attacks is captured.

- **Data Transformation**

Data transformation is indeed a critical step in preparing specific features for use in a neural network model. Its purpose is to convert the data into a format compatible with the model's requirements. One aspect of data transformation in the two datasets involves converting the IP addresses into scalar values. This conversion simplifies the representation of IP addresses, making them easier to process for the neural network model. By assigning numerical values to IP addresses, the model can effectively capture the relevant information encoded within those addresses. Additionally, the process of transforming data involves encoding the target feature (labels) into numeric values. This is typically accomplished through label encoding. By using label encoding, the model can interpret and learn from the labels more effectively, as neural networks are naturally better equipped to handle numeric values.

Furthermore, as part of the data transformation step, normalization is applied to the data. This technique scales the values of various features to a specified range, typically [0, 1]. Normalization, specifically through min-max normalization, ensures that the effect of different features on the model becomes comparable. By normalizing the data, it becomes easier to analyze and interpret the relative importance and influence of different features within the neural network model.

- **Feature Extraction**

During this step of feature extraction, new features were generated from the selected features. The first new feature involved calculating the joint entropy between the destination IP address and the destination port. This joint entropy value represents the amount of information shared by these two variables. This joint entropy value was then used as a single feature to capture the joint entropy between the IP address and the destination port.

Additionally, the entropy calculation was applied to two other features: Fwd IAT Mean and Average Packet Length. The entropy values for these features were computed for subsets of 50 records at a time. By extracting these entropy values within each 50-record subset, new features were derived. These new features provide insights into the entropy characteristics of the destination IP, destination port, "Fwd IAT Mean", and "Average Packet Length" features within each subset. This approach allows for capturing the dynamics of entropy values over smaller segments of the dataset, providing a more detailed representation of the entropy characteristics.

In other words, new features were created by calculating the joint entropy between the IP address and the destination port, as well as the individual entropy values for "Fwd IAT Mean" and "Average Packet Length". By considering subsets of 50 records, the resulting features capture the joint entropy and entropy dynamics within each subset, enhancing the representation of the dataset's entropy characteristics.

- **Splitting**

In our proposal, the datasets are divided into a training dataset and a testing dataset, which is a crucial step in model training and evaluation. During the training phase, the CICIDS2017 published dataset will be used to train our model. Subsequently, the model will be tested using the NSDS to assess its ability to generalize to unseen data and obtain an unbiased estimate.

4.2.2.2 Developed Deep Neural Network Stage

This section will delve deeper into the DDNN model. Its architecture will be presented, and the custom activation function that has been proposed will be highlighted.

- **Model Architecture**

The DDNN model for DoS/DDoS detection consists of four layers, which are explained below:

1) **Input layer:** represents the input data initialization layer, and the cross-level detection is based on three nodes. Represents the number of pre-processed dataset features.
2) **The first hidden layer:** This layer is located between the input and output layers and consists of 128 neurons. During forward propagation, the input data is first multiplied by a weight matrix of size (3, 128), where 3 is the magnitude of the input vector. Then a bias vector of magnitude 128 is added to the multiplication result. Finally, the ReLU activation function is applied to the resulting vector, giving the output of the layer.

3) **Second hidden layer:** This layer is also located between the input and output layers and consists of 64 neurons. During forward propagation, the output of the first hidden layer is used as the input. The input data is first multiplied by a weight matrix of size (128, 64), where 128 is the size of the output vector from the previous layer. Then a bias vector of size 64 is added to the multiplication result. Again, the ReLU activation function is applied to the resulting vector, giving the output of the layer.
4) **Output Layer:** This layer is the final layer of the network and consists of a single neuron, which produces the results, in binary classification it produces the normal packets or attack packets. During forward propagation, the output of the second hidden layer is used as the input. The input data is first multiplied by a weight vector of size (64, 1), where 64 represents the size of the output vector from the previous layer. Then a bias term of magnitude 1 is added to the multiplication result. Finally, the SETAF activation function is applied to the resulting scalar value, which sets the output to the range [0, 1]. A detailed explanation will be provided for the SETAF function.

In the DDNN architecture described, the input layer nodes are fully connected to all the nodes in the next hidden layer and the subsequent layers. This means that each node in the input layer is connected to every node in the next hidden layer, creating a dense or fully connected graph. Figure 4.4, which represents the DDNN architecture, visualizes this connectivity pattern.

After defining the architecture, the next step is to set the loss function to cross-entropy, the optimizer to "Adam," and the evaluation metric to accuracy. These choices enable the model to train and optimize its parameters based on the gradients of the loss function, with the goal of achieving improved performance. The accuracy metric is used to evaluate the model's performance during the training process, providing a measure of how well it predicts the correct outcomes.

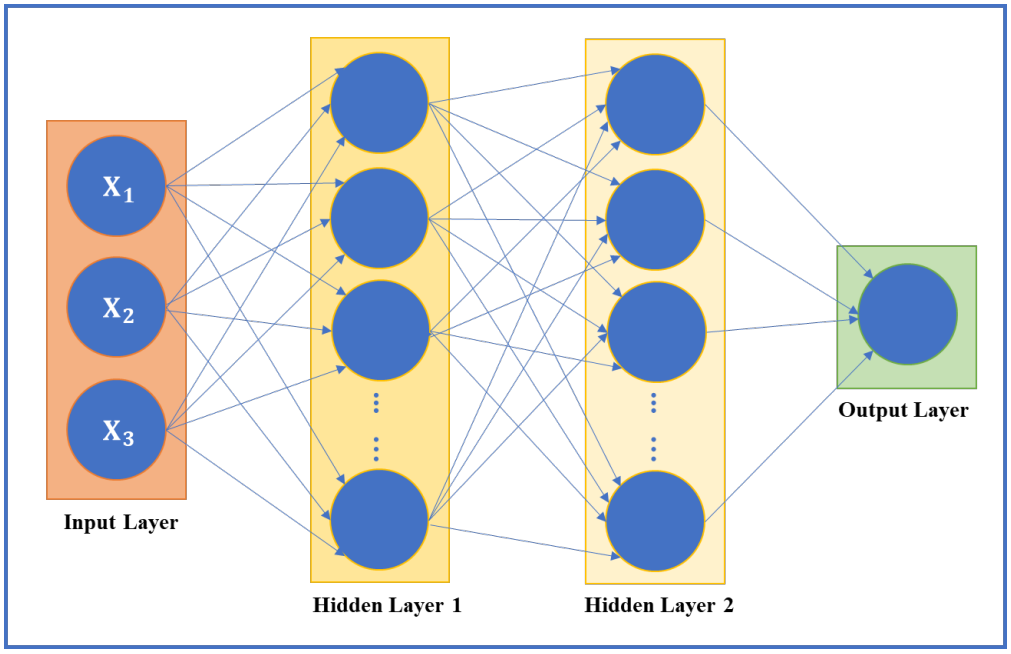


Figure 4.4: The Developed Deep Neural Network Architecture

- **Mathematical Formulation of SETAF**

The Sigmoid-Exponential-Threshold Activation Function (SETAF) is proposed as a modified version of the activation function commonly used in DNN. By integrating the properties of sigmoid, exponential, and threshold activation functions, SETAF has the

potential to improve the network's performance. Here are the mathematical steps to derive the SETAF equation:

First, let's start with the sigmoid function Sig(x), which is commonly used as an activation function in deep neural networks. It maps input values to a range between 0 and 1, providing a smooth and bounded activation response, as shown in Equation (4.5).

$$\mathrm{Sig(x)} = \frac{1}{1 + e^{-x}} \tag{4.5}$$

Through Equation (4.6), the sigmoid function is scaled by an alpha $\propto$ factor.

$$\mathrm{Sig(x)} = \frac{\propto}{1 + e^{-x}} \tag{4.6}$$

Secondly, Equation (4.7) represents the exponential function.

$$\mathrm{Exp(x)} = \mathrm{e^{d}} \tag{4.7}$$

Third, the squared distance function or the squared difference function is used to represent the squared difference or the squared distance between x and the threshold value, as shown in Equation (4.8).

$$\mathrm{d} = (\mathrm{x} - \delta)^2 \tag{4.8}$$

By multiplying Equation (4.8) by -1 to create a negative value for the exponential function and substituting Equation (4.8) into Equation (4.7), we can obtain a value between 0 and 1 using Equation (4.9).

$$\mathrm{Exp(x)} = \mathrm{e^{-(x-\delta)^2}} \tag{4.9}$$

The exponential component is also scaled by the beta β factor in Equation (4.10).

$$\mathrm{Exp}(x) = \beta\, e^{-(x-\delta)^2} \tag{4.10}$$

Finally, Equation (4.11), known as the SETAF equation, is derived by combining Equation (4.6) and Equation (4.10) together.

$$\mathrm{SETAT} = \frac{\propto}{1 + e^{-x}} + \beta\, e^{-(x-\delta)^2} \tag{4.11}$$

The equation can be simplified as shown in Equation (4.12).

$$\mathrm{SETAF} = \propto \mathrm{Sig}(x) + \beta\, e^{-(x-\delta)^2} \tag{4.12}$$

The goal of the SETAF activation function is to improve the efficiency of deep neural networks by providing a versatile and adjustable activation response. Here are some benefits of using SETAF for deep neural networks.

1) Enhanced Non-Linearity: SETAF combines sigmoid, exponential, and threshold components to introduce increased non-linearity in the activation function.
2) Flexibility and Adaptability: SETAF offers flexibility by allowing adjustments to parameters such as ∝, β , and δ. This adaptability enables customization of the activation function to meet specific requirements of the problem domain.
3) Capturing Exponential Relationships: SETAF incorporates an exponential component to capture exponential growth or decay patterns in the data. This enables the activation function to handle

non-linear relationships that may not be effectively represented by a simple sigmoid function.

Thus, SETAF contributes to improved performance of DNN, enabling better modeling of complex data relationships and patterns.

4.3 Integration of Slice-level and Cross-level Detection for Network Slicing Security

In the context of detecting DoS/DDoS attacks within a network slicing environment, the innovative architecture involves integrating two levels of detection: slice-level and cross-level, as mentioned earlier. Slice-level detection is performed on each network slice separately, while cross-level detection is performed by a third-party server.

To combine slice-level detection and cross-level detection for a network slicing environment, the following steps are followed, which are shown in Figure 4.5.

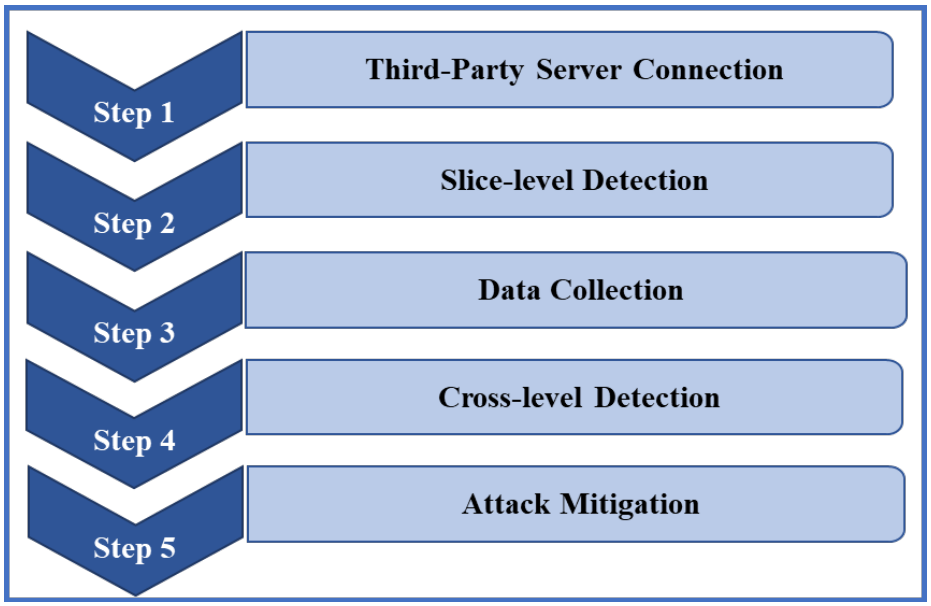


Figure 4.5: Workflow for Integrating Slice-level and Cross-level Detection

- **Step 1: Third-Party Server Connection**

The third-party server establishes a reliable connection with each network slice controller in the NS environment. This connection allows the server to receive data from the network slices for analysis.

- **Step 2: Slice-level Detection**

Traffic analysis is conducted individually for each network slice using joint entropy-based statistical analysis. Based on a dynamic threshold, the traffic is classified as normal or suspicious. If the traffic is flagged as suspicious, the relevant traffic parameters include the joint entropy value and the threshold used. Additionally, entropy values are calculated for both a IAT and the packet length within the same window. These parameters are then sent to a third-party server for further analysis, utilizing cross-level detection techniques.

- **Step 3: Data Collection**

The third-party server collects the network traffic parameters from each network slice for analysis.

- **Step 4: Cross-level DDoS Detection**

The collected parameters are fed into a DDNN model specifically designed for the detection. By analyzing the collected parameters, the DDNN model can identify patterns indicative of the attacks that span across multiple network slices.

- **Step 5: Attack Mitigation**

  If the DDNN model detects the attack based on the analyzed data, the third-party server generates an alert or notification to inform the network administrator. Upon receiving the alert, the network administrator can take appropriate measures to mitigate the attack.

By integrating slice-level detection and cross-level detection, the innovative architecture enables the detection of the attacks in a NS environment, providing enhanced security and protection for the network infrastructure. Algorithm 4.4 describes the methodology for detecting the attacks in a NS environment in detail.

Algorithm (4.4): DoS/DDoS Attack Detection in Network Slicing

Input: Network_slices (list of network_slice controllers)
Output: Alert (Boolean value indicating if an attack is detected)
1. **Begin**
// Construct third party server connections
2. **For each** controller **in** network_slice controllers:
3. | Establish a reliable connection between server and controller
4. **End For**
// Slice-level Detection
5. **For** each slice in Network_slices:
6. | slice_traffic_parameters ← controller
7. | Perform Joint Entropy-based Statistical Analysis (slice_traffic_parameters)
8. | slice_level detection parameters ← Joint Entropy-based Statistical Analysis
9. | **If** Suspicious:
10. | | Features_Extraction ← Features Extraction from slice_traffic_parameters
11. | | Send Features_Extraction and Slice_level detection parameters to Server
12. | **End If**
13. **End For**
// Server Data Collection
14. Collect parameters from each network slice for analysis.
// Cross-level Detection
15. Analyze Collected parameters with DDNN Model
//Attack Mitigation
16. **If** Attack Detected:
17. Alert = true
18. **Else:**
19. Alert = false
20. **End If**
21. **End Algorithm**

4.4 Summary

The chapter primarily discussed the architecture of an innovative methodology for detecting DoS/DDoS attacks in a NS environment. The architecture included two levels of attack detection. The first level focused on slice-level detection, where statistical techniques were used to analyze traffic and detect suspicious activity within each network slice. The second level, known as cross-level detection, involved the use of a DDNN to analyze suspicious traffic across multiple slices. Additionally, the

chapter demonstrated the integration process of this detection architecture into a NS environment. The integration was achieved by implementing the first level on each network slice, while the second level was implemented on a third-party server.

In the next chapter, the focus will shift toward the practical implementation of this architecture and the evaluation of the innovative methodology's performance in detecting the attacks within NS environment.

CHAPTER FIVE

The Implementation, Results and Evaluation

5.1 Overview

In the previous chapter, a detailed explanation of the proposed methodology for detecting DoS/DDoS attacks was provided. In this chapter, the focus will be placed on the step-by-step implementation of both a NS environment and the proposed methodology for detecting the attacks. The implementation will cover both the slice-level and cross-level detection aspects of the methodology in network slicing environment. Following the implementation, the performance of the methodology will be evaluated to assess its effectiveness and efficiency in detecting the attacks.

Furthermore, a comparative analysis will be conducted to compare the proposed methodology with other existing works in the field. In this analysis, valuable insights will be gained into how the methodology performed in comparison to related approaches, with a particular emphasis on its unique contributions and distinguishing advantages.

5.2 Implementing the Proposed Methodology

This section describes the implementation of the NS testbed environment and the testing steps of the proposed methodology on a Windows 10 Pro x64 PC. The PC used for this purpose is equipped with an Intel Core i7-8750H CPU running at a clock speed of 2.20GHz and 32GB of RAM. In addition, the Python programming language will be used on the PyCharm platform from JetBrains software.

5.2.1 Implementing the Network Slicing Environment

The proposed NS environment was implemented using VMware Workstation 15.5.7. This involved creating four virtual machines, three of which were specifically configured to run Ubuntu 14.04 LTS, while the fourth virtual machine was configured to run Ubuntu Server 20.04.1 LTS. Figure 5.1 visually presents these virtual machines and their associated components, providing an overview of the network slicing environment setup.

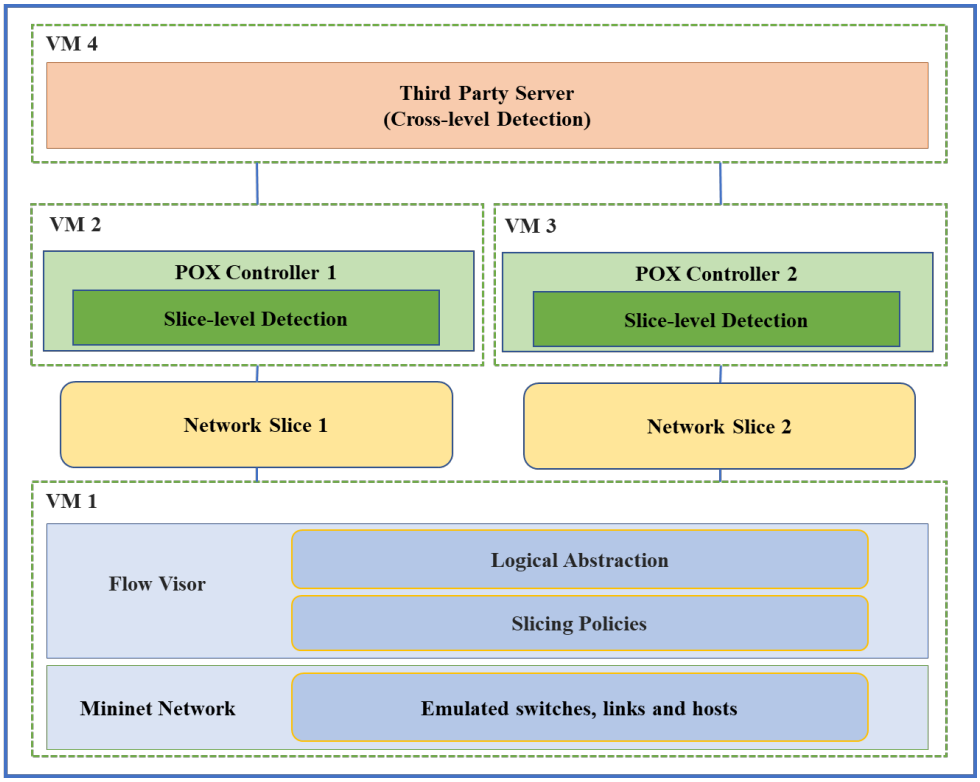


Figure 5.1: Visualization of Virtual Machines and Components for Proposed Network Slicing

The first virtual machine (VM1), named vSDN, served as a representation of the infrastructure and virtualization layers. It included a Mininet emulator with Open vSwitch and FlowVisor to facilitate network slicing. The second and third virtual machines (VM2, VM3) were

dedicated to the control layer, with each hosting a POX controller. Slice-level detection was applied to these controllers. Lastly, the fourth virtual machine (VM4) functioned as a third-party server, where cross-level detection was carried out. Thus, the proposed NS Environment was built, as shown in the Figure 5.2.

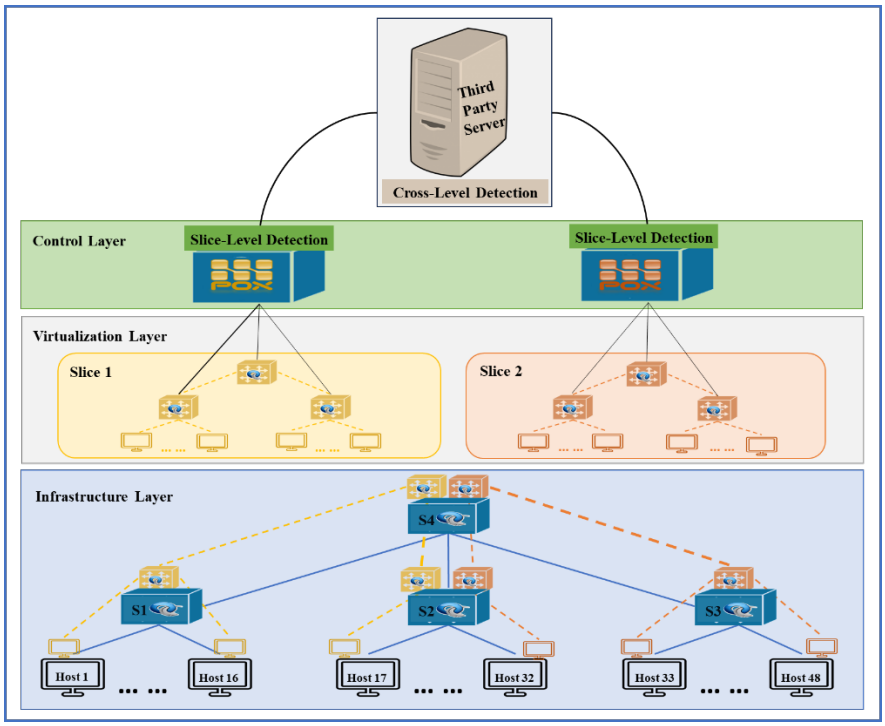


Figure 5.2: Proposed Network Slice Topology

5.2.1.1 The Infrastructure Layer

The infrastructure layer of the NS environment was successfully implemented by leveraging the capabilities of the Mininet emulator (version 2.3.1b1) installed on the VM1. This powerful emulator provided an emulated environment for network components, enabling the construction of the infrastructure layer. To facilitate the implementation, Open vSwitch (version 2.0.2) was utilized as a flexible switch supporting various networking protocols, including the OpenFlow protocol.

The network infrastructure was designed with 4 Open vSwitches and 48 hosts with IP addresses ranging from 10.0.0.1 to 10.0.0.48, which were configured using a Python script code. By executing this script within the Mininet network emulator, the infrastructure components were created seamlessly as shown in Figure 5.3.

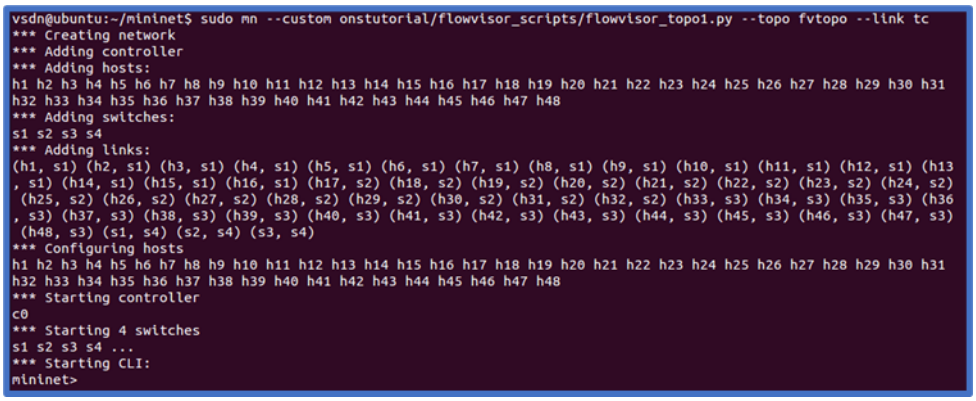


Figure 5.3: Creation of Infrastructure Components using Mininet Network Emulator.

The combined utilization of Mininet and Open vSwitch played crucial roles in simulating network infrastructure components within the network slicing environment.

5.2.1.2 The Virtualization Layer

In the network virtualization layer, a significant advancement was achieved by installing the FlowVisor network hypervisor (version 1.4.0) on VM1, which already had the Mininet emulator installed. This integration introduced powerful capabilities for network slicing, enabling the effective division of the network topology. As depicted in Figure 5.4, two distinct network slices, namely slice1, and slice2.

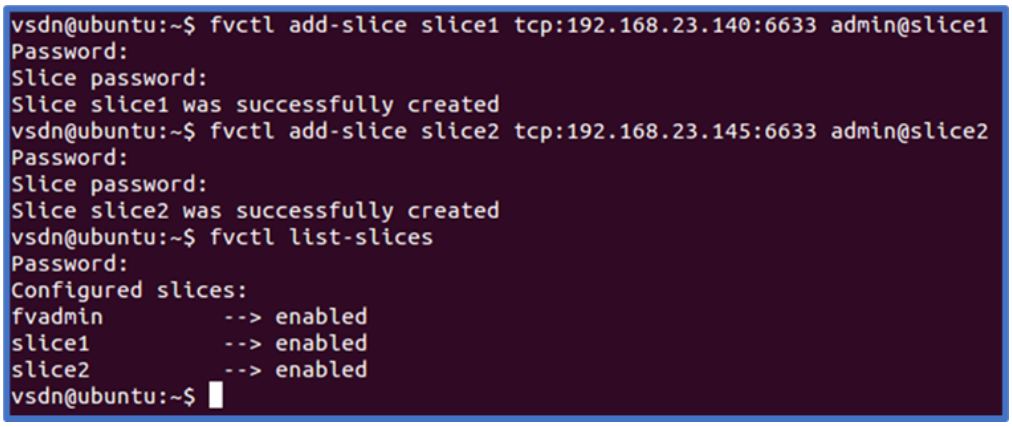


Figure 5.4: Creation of Two Distinct Network Slices, Slice 1 and Slice 2

To create these network slices, the "add-slice" command was utilized. This command, specific to FlowVisor, enabled the creation of individual slices within the virtualized network environment. Each slice operates as an independent virtual network, with its own set of resources and isolated network connectivity.

In this setup, each network slice was associated with its own controller, communicating over the TCP protocol. The controller was identified by an IP address and port, enabling separate management and control of each slice. Additionally, the contact information for the person responsible for each slice was provided as "slice1@adim" and "slice2@admin". These contact details indicate the responsible administrators or individuals overseeing the management and operation of each respective slice.

Furthermore, the "list-slices" command was used to obtain a list of the existing slices within the network virtualization environment.

Once the network slices were successfully created using FlowVisor, the next step involved configuring flowspace for each slice.

To achieve this, the switch ports were mapped to the corresponding flowspace, ensuring that traffic entering or leaving those ports conformed to the specified rules and policies. Within each slice, the flowspace was defined by associating it with the slice name, establishing a unique flowspace configuration tailored to the requirements of that particular slice. To create a flowspace, been used the add-flowspace command for example:

“fvctl add-flowspace dpid1 1 1 any slice1=7”

- "fvctl": This is the command to interact with FlowVisor using the "fvctl" tool.
- "add-flowspace": This specific command is used to add a flowspace configuration.
- "dpid1": This parameter specifies the DPID (Data Path ID) of the switch to which the flowspace applies. In this case, "dpid1" represents the unique identifier of the switch.
- "1 1": These parameters specify the priority and the flowspace ID, respectively. In this case, the priority is set to 1, and the flow space ID is set to 1.
- "any": This match condition indicates that the flowspace matches any incoming network traffic.
- "slice1=7": This part of the command associates the flowspace with a specific slice named "slice1" and assigns it a permission level of "7". The permission level determines the access rights of the slice, such as read, write, and delegate permissions. The value "7" indicates that the slice has full permissions to read, write, and delegate.

The Figure 5.5 provides an illustrative representation of the process involved in adding flowspace.

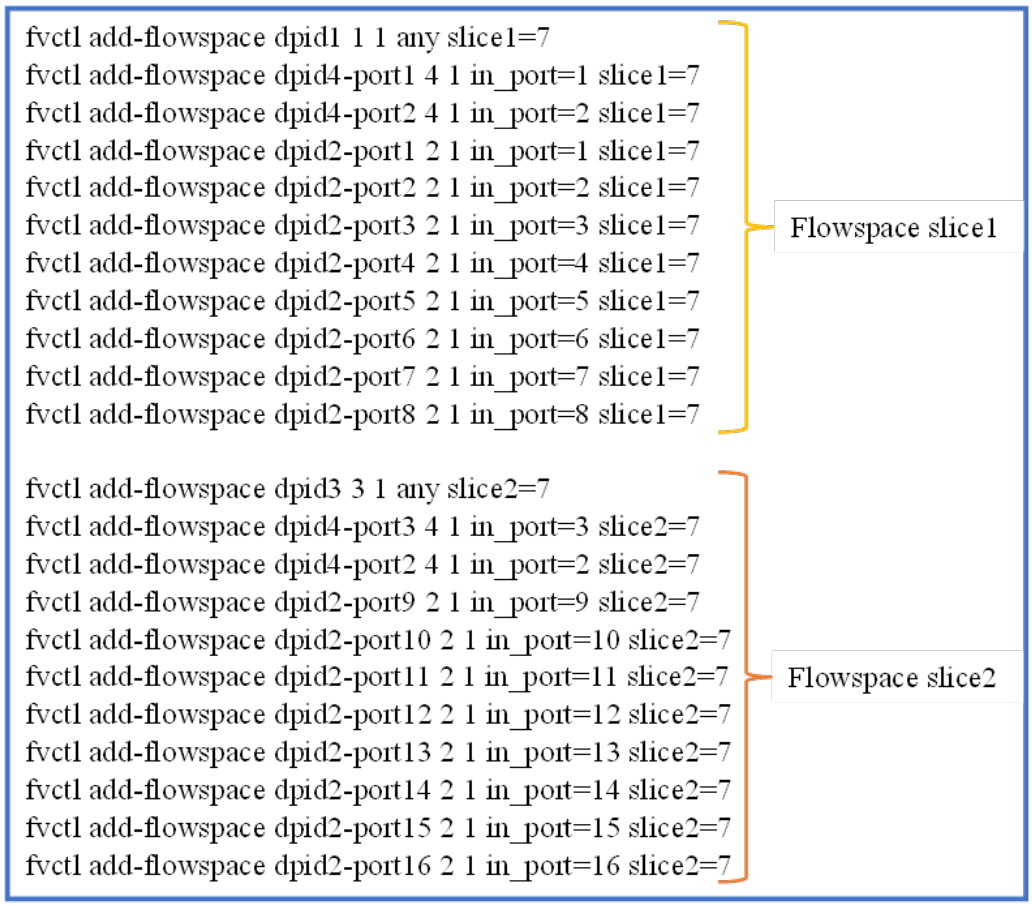


Figure 5.5: Process of Adding Flowspace in FlowVisor

The "list-flowspace" command can be used to obtain and display the current flowspace setups in FlowVisor. When this command is executed, it generates a record of the flowspace entries that have been established in FlowVisor. This record provides information about the existing flowspace configurations, as depicted in Figure 5.6.

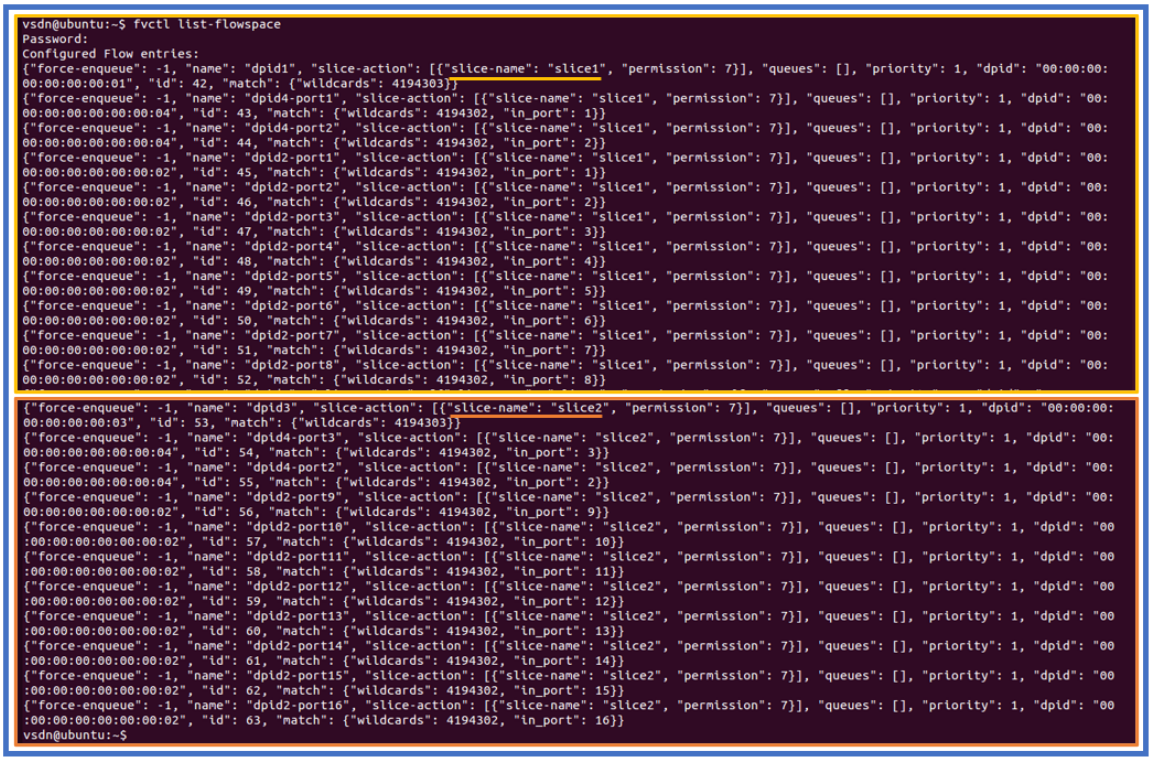


Figure 5.6: List the Current Flowspace in FlowVisor

After successfully adding the flowspace of each network slice, two virtual networks (two slices) were created from the same physical infrastructure, as shown in Figure 5.2.

5.2.1.3 The Control Layer

Within the proposed architecture, the control layer is composed of VM2 and VM3. Each of these virtual machines incorporates the installation of a POX controller. Furthermore, every POX controller is connected to a specific network slice, as shown in Figure 5.7.

By leveraging the OpenFlow protocol, the controllers establish connections to actively control and manage traffic flow within their assigned network slices.

VM2: POX Controller (Slice 1)	poxcontroller20@ubuntu:~/pox$ ./pox.py forwarding.l3_learning_E11 POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al. WARNING:version:Support for Python 3 is experimental. INFO:core:POX 0.7.0 (gar) is up. INFO:openflow.of_01:[00-00-00-00-00-04 1] connected INFO:openflow.of_01:[00-00-00-00-00-01 2] connected INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
VM3: POX Controller (Slice 2)	poxcontroller2@ubuntu:~/pox$ ./pox.py forwarding.l3_learning_E12 POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al. WARNING:version:Support for Python 3 is experimental. INFO:core:POX 0.7.0 (gar) is up. INFO:openflow.of_01:[00-00-00-00-00-03 1] connected INFO:openflow.of_01:[00-00-00-00-00-02 2] connected INFO:openflow.of_01:[00-00-00-00-00-04 3] connected

Figure 5.7: Connection of POX Controllers to Specific Network Slice Components

The figure illustrates this setup, where the POX controller in VM2 is connected to virtual switches S1, S2, and S4, representing slice1. Similarly, the POX controller in VM3 is connected to switches S2, S3, and S4, representing slice2. These connections enable controllers to perform slice-level detection, efficiently manage traffic, enforce rules, and make dynamic decisions based on network conditions.

5.2.1.4 The Third-Party Server

The VM4 has been installed with Ubuntu Server 20.04.1 LTS and is serving as a third-party server. It has been assigned the IP address 192.168.23.188 and is listening on port 128. While this server implements cross-level detection, the initial step involves establishing reliable socket-based communication using Python programming with the controllers associated with each network slice, as depicted in Figure 5.8.

```
securityserver@ubuntu:~$ sudo python3 security_server.py
Server started on 192.168.23.188 port 128
Client connected from: ('192.168.23.140', 50306)
Received data: Hello, server!
Client connected from: ('192.168.23.145', 35498)
Received data: Hello, server!
```

Figure 5.8: Server Communication with each Controller Slice

The process of establishing reliable communication with the controllers for each network slice using Python socket programming began by following the implementation steps outlined below:

1) Assigning IP address and port to a server socket to enable the slice controller to connect to the server.
2) Defining the IP address and port of the server in the client socket on each controller for the purpose of establishing communication between the network slices and the server.
3) In this step, the server is ready to listen to requests from the controllers.

5.2.2 Implementing the Traffic Generation for a Network Slicing

To generate traffic for both normal and DoS/DDoS attack scenarios, a script was implemented using the Scapy tool in the Python programming language. This script utilized Algorithms 3.1 and 3.2 to simulate normal and attack traffic, respectively. These algorithms are described in section 3.2.2.

In the scenario of normal traffic generation, the process of packet generation included randomly selecting IP addresses and ports for both the source and destination from within the given topology. This approach is demonstrated in Figure 5.9, where the random selection of IP addresses and ports is illustrated.

Figure 5.9: Normal Traffic Generation

Regarding the attack traffic generation, Figure 5.10 showcased the creation of packets with random spoofed source IP addresses and random source port addresses. The destination IP addresses and port addresses were defined within a specific range chosen by the attacker for targeting during the attack.

The figure illustrates the generation of attack traffic targeting four IP addresses within the range of 10.0.0.10 to 10.0.0.14. The attack is specifically directed towards well-known ports 80 (HTTP) and 53 (DNS) for these IP addresses.

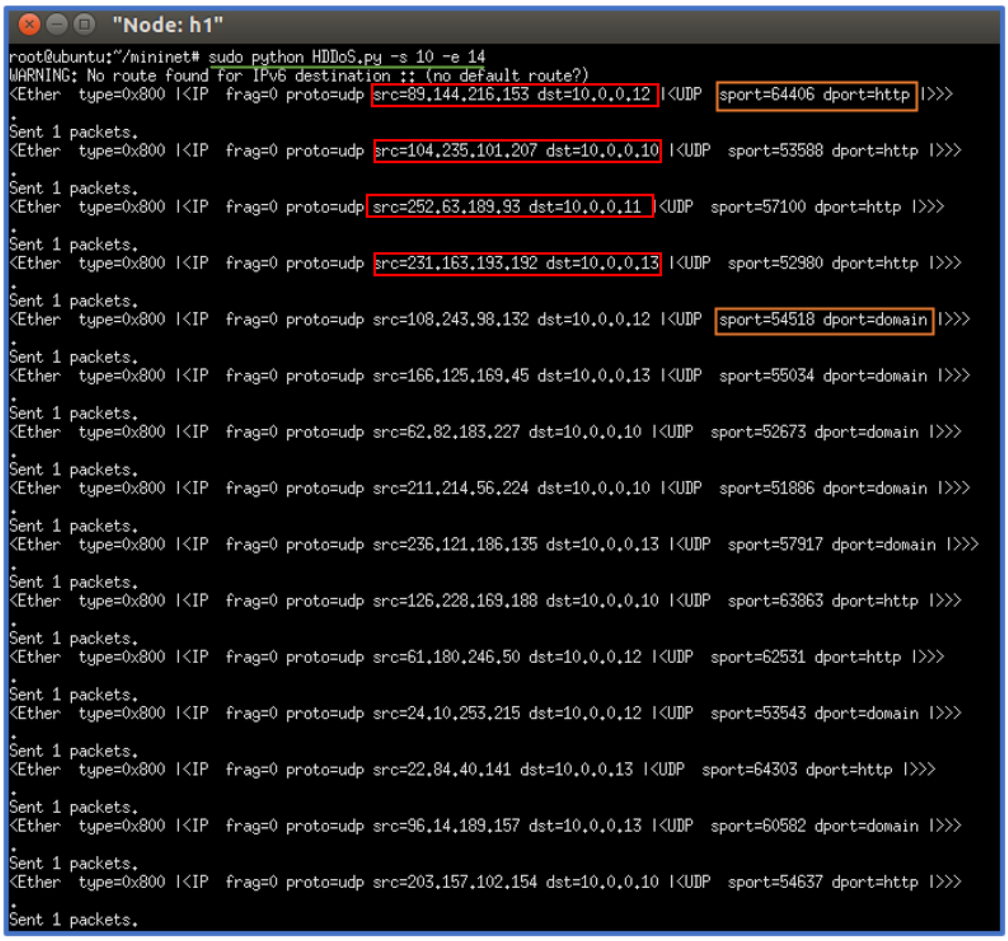


Figure 5.10: DoS/DDoS Attack Traffic Generation

5.2.3 Result of Creation NSDS

In a network slicing environment, to create the dataset, an application was utilized on each network slice controller to monitor the traffic and collect relevant parameters. The application was responsible for extracting traffic information from incoming packets to the network slice's controller. It was integrated with the l3_learning module to enable efficient traffic monitoring within the network slice.

By extracting traffic information from the packets, the application was able to capture and record parameters such as source IP addresses, destination IP addresses, packet length, protocol, and other relevant

details. These parameters were then stored in a CSV file, forming the dataset. The Figure 5.11 presents the parameters obtained from the controller of each network slice, showcasing the results of the monitoring process.

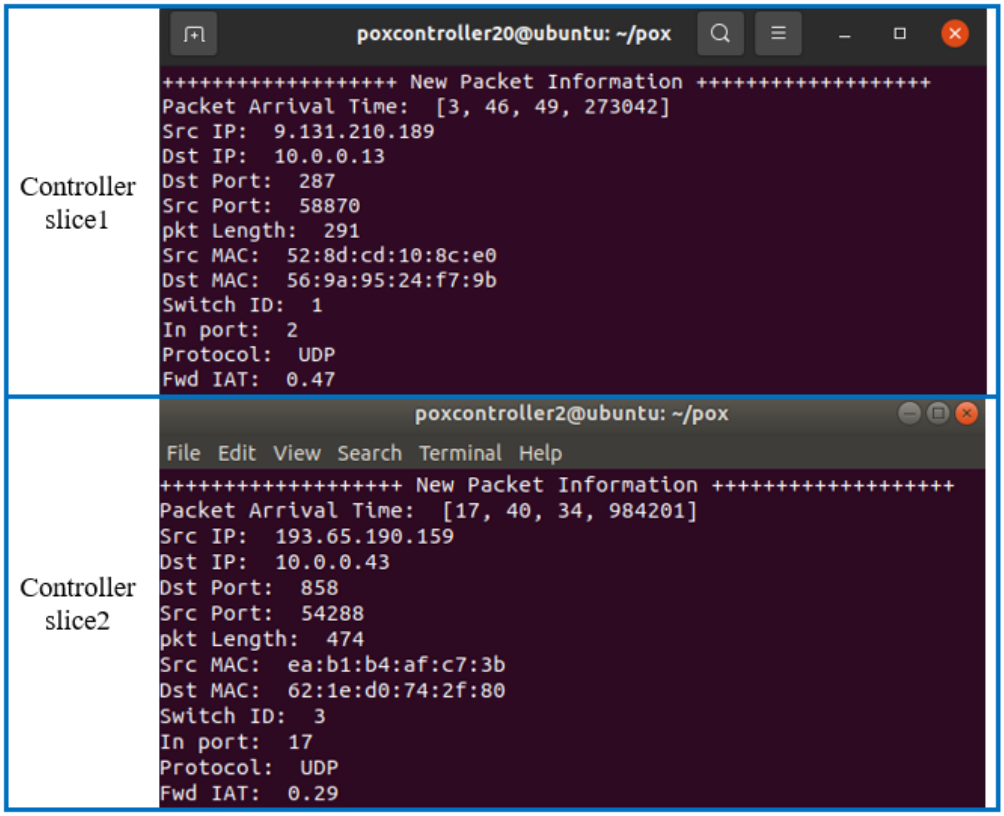


Figure 5.11: Capture of Parameters Obtained from Network Slice Monitoring Application Controllers

The NSDS dataset consisted of 14 features and included both normal and attack traffic scenarios. During the attack, specific hosts within each network segment were targeted. In Slice 1, the attack impacted hosts with IP addresses ranging from 10.0.0.15 to 10.0.0.18. Similarly, in Slice 2, the attack specifically focused on hosts 10.0.0.46 and 10.0.0.47. The dataset included various attack scenarios, such as DoS_SS,

DoS_MS, DDoS_SM, and DDoS_MM. A sample representation of the generated NSDS is provided in Figure 5.12.

The First Records of the Dataset

	Packet Arrival Time	Fwd IAT	Src IP	Dst IP	Dst Port	Src Port	pkt Length	Src MAC	Dst MAC	Switch ID	In port	Protocol	Slice Name	Label
2	[12, 4, 23, 17906]	0.82	7.25.227.30	10.0.0.40	55301	153	180	d6:1e:87:e0:b8:37	f2:f4:4a:d0:5b:98	3	17	UDP	Slice 2	Normal
3	[12, 5, 2, 138802]	0.18	217.41.110.53	10.0.0.42	61794	104	199	d6:1e:87:e0:b8:37	c2:74:4a:ac:0a:a4	3	17	UDP	Slice 2	Normal
4	[21, 43, 11, 361556]	0.88	150.36.140.189	10.0.0.1	54959	678	425	aa:30:a1:d9:f0:68	8a:7a:f9:2a:9a:40	4	2	UDP	Slice 1	Normal
5	[21, 42, 47, 191540]	0.12	18.137.87.131	10.0.0.14	51181	915	278	aa:30:a1:d9:f0:68	06:61:ef:ca:d7:7e	2	5	UDP	Slice 1	Normal
6	[21, 43, 25, 214457]	0.12	85.220.81.142	10.0.0.8	53259	894	222	aa:30:a1:d9:f0:68	52:ee:fe:f2:c8:f1	4	2	UDP	Slice 1	Normal
7	[21, 41, 55, 886256]	0.43	220.219.235.170	10.0.0.19	63267	820	226	02:e8:a2:7f:33:2a	6e:42:87:d1:f2:31	2	2	UDP	Slice 1	Normal
8	[21, 41, 54, 314367]	0.75	22.204.47.55	10.0.0.18	56145	638	101	02:e8:a2:7f:33:2a	b2:5c:63:ce:4a:46	2	2	UDP	Slice 1	Normal
9	[21, 43, 2, 16641]	0.64	160.171.91.94	10.0.0.14	56786	980	407	aa:30:a1:d9:f0:68	06:61:ef:ca:d7:7e	4	2	UDP	Slice 1	Normal
10	[21, 41, 21, 391554]	0.18	184.41.167.159	10.0.0.19	62408	288	330	02:e8:a2:7f:33:2a	6e:42:87:d1:f2:31	2	2	UDP	Slice 1	Normal
11	[21, 41, 10, 619498]	0.21	63.255.45.110	10.0.0.21	56100	883	326	02:e8:a2:7f:33:2a	6e:88:00:13:96:db	2	2	UDP	Slice 1	Normal
12	[21, 42, 42, 962416]	0.64	27.248.195.158	10.0.0.10	52639	439	497	aa:30:a1:d9:f0:68	2e:44:79:0b:4a:d9	4	2	UDP	Slice 1	Normal
13	[21, 42, 4, 696298]	0.2	74.17.37.1	10.0.0.22	58434	634	298	02:e8:a2:7f:33:2a	26:88:f0:03:64:8d	2	2	UDP	Slice 1	Normal
14	[21, 41, 38, 220531]	0.45	123.31.33.28	10.0.0.19	51290	524	186	02:e8:a2:7f:33:2a	6e:42:87:d1:f2:31	2	2	UDP	Slice 1	Normal
15	[21, 43, 15, 43183]	0.53	164.81.116.21	10.0.0.17	55449	73	437	aa:30:a1:d9:f0:68	02:e8:a2:7f:33:2a	2	5	UDP	Slice 1	Normal
16	[21, 41, 7, 56196]	0.71	17.96.111.247	10.0.0.20	63458	339	314	02:e8:a2:7f:33:2a	aa:30:a1:d9:f0:68	2	2	UDP	Slice 1	Normal
17	[21, 41, 29, 672838]	0.48	77.157.168.175	10.0.0.23	50923	996	427	02:e8:a2:7f:33:2a	96:85:7a:67:9d:aa	2	2	UDP	Slice 1	Normal
18	[12, 4, 28, 425988]	0.16	70.67.28.244	10.0.0.39	52025	426	258	d6:1e:87:e0:b8:37	56:12:d5:02:ab:2a	3	17	UDP	Slice 2	Normal
19	[21, 42, 58, 42673]	0.45	224.8.177.11	10.0.0.13	59761	491	372	aa:30:a1:d9:f0:68	b2:9b:f1:42:c7:d6	2	5	UDP	Slice 1	Normal
20	[21, 41, 57, 415070]	0.92	184.205.37.66	10.0.0.18	64458	484	248	02:e8:a2:7f:33:2a	b2:5c:63:ce:4a:46	2	2	UDP	Slice 1	Normal

The Last Records of the Dataset

7301	[11, 53, 40, 496402]	0.00064	83.159.122.59	10.0.0.47	61512	53	8	5a:28:92:cf:d9:32	22:a3:c8:d5:1a:69	3	14	UDP	Slice 2	DDoS
7302	[21, 23, 2, 903552]	0.00064	95.30.194.123	10.0.0.17	55631	53	8	42:3d:8c:12:26:19	02:e8:a2:7f:33:2a	1	3	UDP	Slice 1	DDoS
7303	[11, 54, 3, 25456]	0.00064	83.78.249.52	10.0.0.46	57146	80	8	5a:28:92:cf:d9:32	d6:42:0a:25:73:ac	3	14	UDP	Slice 2	DDoS
7304	[21, 22, 32, 17185]	0.08	222.123.60.125	10.0.0.15	59160	80	8	8a:7a:f9:2a:9a:40	ee:8b:37:da:73:76	1	2	UDP	Slice 1	DDoS
7305	[21, 24, 31, 360993]	0.016	249.219.230.139	10.0.0.18	59982	80	8	42:3d:8c:12:26:19	b2:5c:63:ce:4a:46	1	3	UDP	Slice 1	DDoS
7306	[21, 21, 47, 966300]	0.016	90.30.247.164	10.0.0.16	65066	80	8	8a:7a:f9:2a:9a:40	ae:85:d5:75:36:58	1	2	UDP	Slice 1	DDoS
7307	[11, 53, 36, 874380]	0.08	130.247.211.96	10.0.0.47	64667	53	8	5a:28:92:cf:d9:32	22:a3:c8:d5:1a:69	3	14	UDP	Slice 2	DDoS
7308	[21, 21, 56, 479785]	0.0032	62.246.150.199	10.0.0.15	56268	53	8	8a:7a:f9:2a:9a:40	ee:8b:37:da:73:76	1	2	UDP	Slice 1	DDoS
7309	[21, 25, 36, 811878]	0.0032	202.162.101.225	10.0.0.16	50418	53	8	e6:70:5c:84:cb:d5	ae:85:d5:75:36:58	1	5	UDP	Slice 1	DDoS
7310	[11, 53, 34, 269852]	0.00064	147.192.154.162	10.0.0.46	62474	53	8	5a:28:92:cf:d9:32	d6:42:0a:25:73:ac	3	14	UDP	Slice 2	DDoS
7311	[21, 22, 26, 26447]	0.016	24.235.143.64	10.0.0.16	51868	53	8	8a:7a:f9:2a:9a:40	ae:85:d5:75:36:58	1	2	UDP	Slice 1	DDoS
7312	[21, 22, 10, 700487]	0.016	244.210.108.91	10.0.0.17	55434	80	8	8a:7a:f9:2a:9a:40	02:e8:a2:7f:33:2a	1	2	UDP	Slice 1	DDoS
7313	[21, 22, 34, 24147]	0.016	229.112.189.69	10.0.0.17	57474	80	8	8a:7a:f9:2a:9a:40	02:e8:a2:7f:33:2a	1	2	UDP	Slice 1	DDoS
7314	[21, 26, 13, 546189]	0.016	59.44.232.210	10.0.0.16	57238	53	8	e6:70:5c:84:cb:d5	ae:85:d5:75:36:58	1	5	UDP	Slice 1	DDoS
7315	[11, 54, 41, 234911]	0.00064	175.14.130.86	10.0.0.47	65370	80	8	5a:28:92:cf:d9:32	22:a3:c8:d5:1a:69	3	14	UDP	Slice 2	DDoS
7316	[11, 53, 31, 360664]	0.08	63.137.207.244	10.0.0.47	61764	53	8	5a:28:92:cf:d9:32	22:a3:c8:d5:1a:69	3	14	UDP	Slice 2	DDoS
7317	[21, 22, 2, 343214]	0.00064	154.111.226.45	10.0.0.16	55049	53	8	8a:7a:f9:2a:9a:40	ae:85:d5:75:36:58	1	2	UDP	Slice 1	DDoS

Figure 5.12: Sample of NSDS

5.2.4 Implementation Result of Detection Model for Network Slicing Attacks

In this section, practical steps for implementing DoS/DDoS detection at the slice-level and cross-level will be covered.

5.2.4.1 Slice-level Detection

The dual objective of the slice-level detection approach was to enhance the early detection rate of DoS/DDoS attacks on network slices and reduce the occurrence of false positives. To assess the efficacy of this approach, two crucial parameters were taken into consideration: the DR and the FPR. The detection rate represented the percentage of accurately identified the attacks by the system, while the false positive rate indicated the percentage of normal traffic incorrectly identified as attack.

In the initial experiment conducted, normal network traffic was generated. This process involved the calculation and storage of the joint entropy value for normal traffic in a vector of length 10. Subsequently, multiple initial thresholds were computed using a K value ranging from 2 to 14, based on Chebyshev's theorem. The outcomes of this implementation were shown in Figure 5.13, while Figure 5.14 depicted the correlation between the K value and the calculated initial threshold.

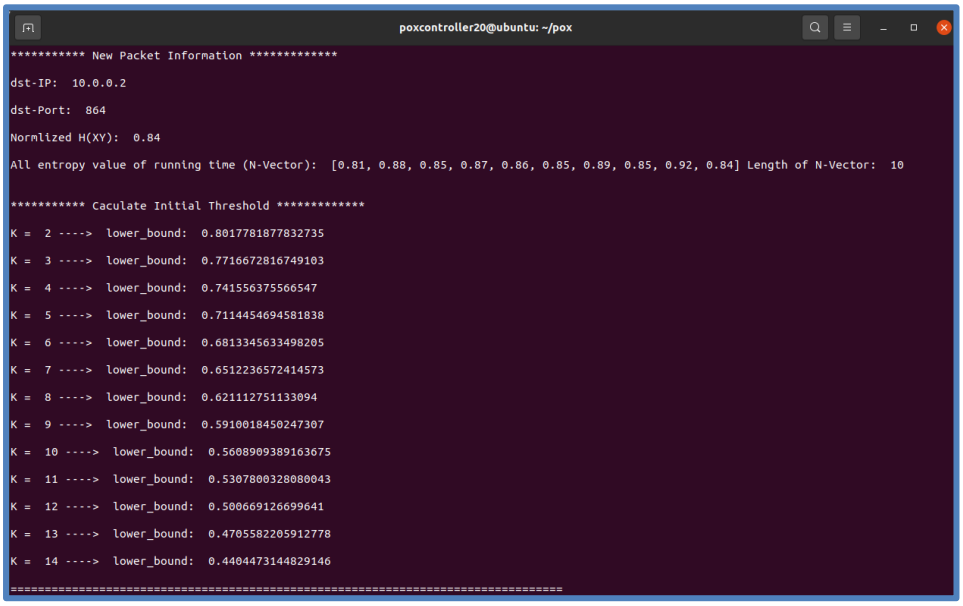


Figure 5.13: Implementing Initial Threshold Calculation in the POX Controller

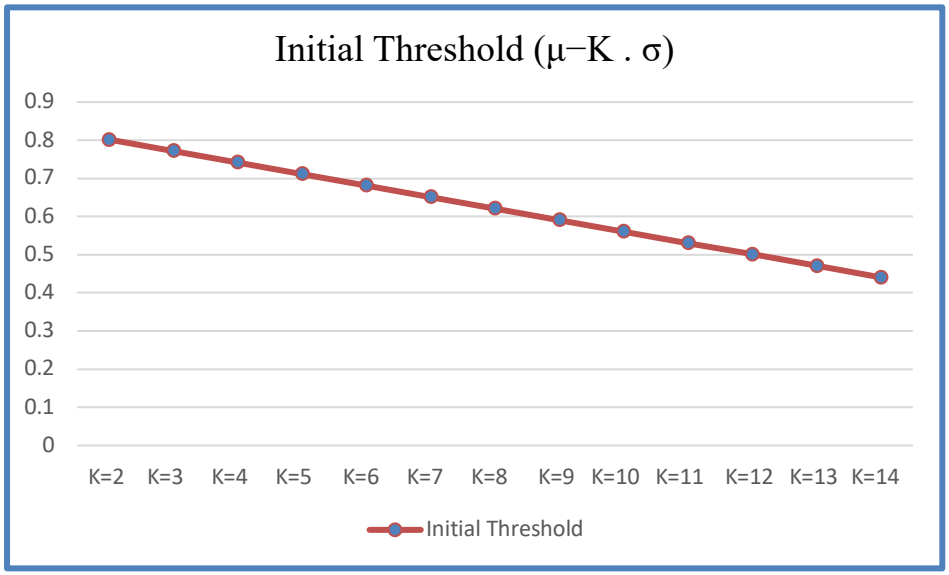


Figure 5.14: Analyzing Initial Thresholds Based on K Values

After calculating the initial thresholds in the first experiment, the effectiveness of these thresholds in detecting DoS/DDoS attacks was evaluated in the second experiment. The slice-level detection was tested by detecting the attack traffic.

During the experiment, the DR and FPR were computed using the initial threshold limits determined in the first experiment. The outcomes of this experiment are presented in Figure 5.15, showing the correlation between the DR, FPR, and the K value used in the threshold calculation.

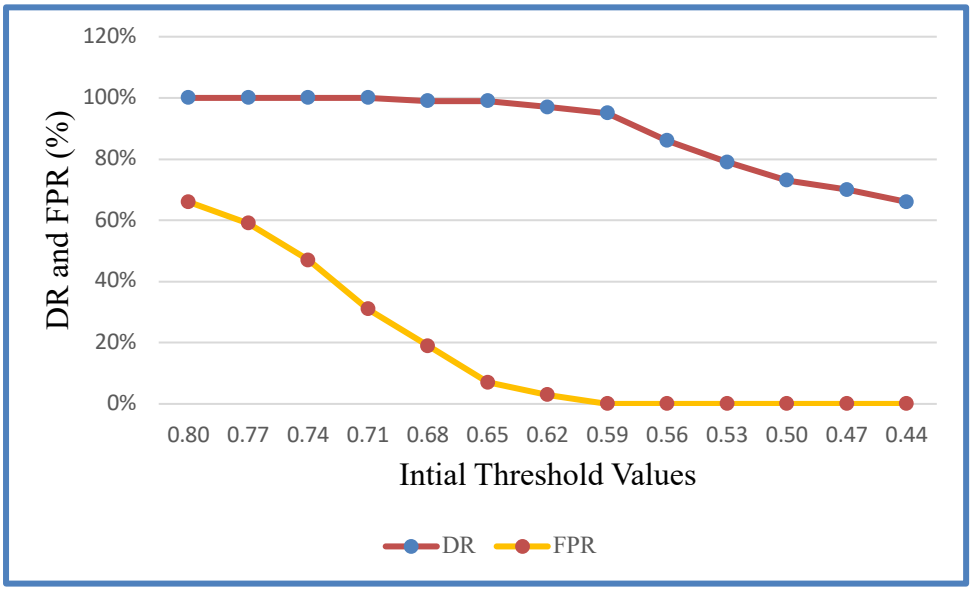


Figure 5.15: Examining the Impact of Different Thresholds on DR and FPR

The main objective of the experiment was to determine a suitable threshold value for slice-level detection that could effectively detect attacks early while minimizing the false positive rate. Based on the results, it was concluded that a K value of 7 was the most effective choice for setting the initial threshold value to 0.65 in the system. By setting this initial threshold value, the system can achieve a high DR of up to 99% and a FPR of less than 7%, which is handled during cross-level detection, as explained in the Section 5.2.4.3.

After setting the ideal K value for threshold calculation and determining the value of the initial threshold, the detection efficiency at the slice-level for slice1 was tested. In the third experiment, the joint entropy value was calculated for multiple windows under normal traffic conditions. The experiment aimed to evaluate the behavior of joint entropy and determine how to update the threshold accordingly. The results, including the values of the joint entropy and the approach for updating the threshold, were presented in Figure 5.16.

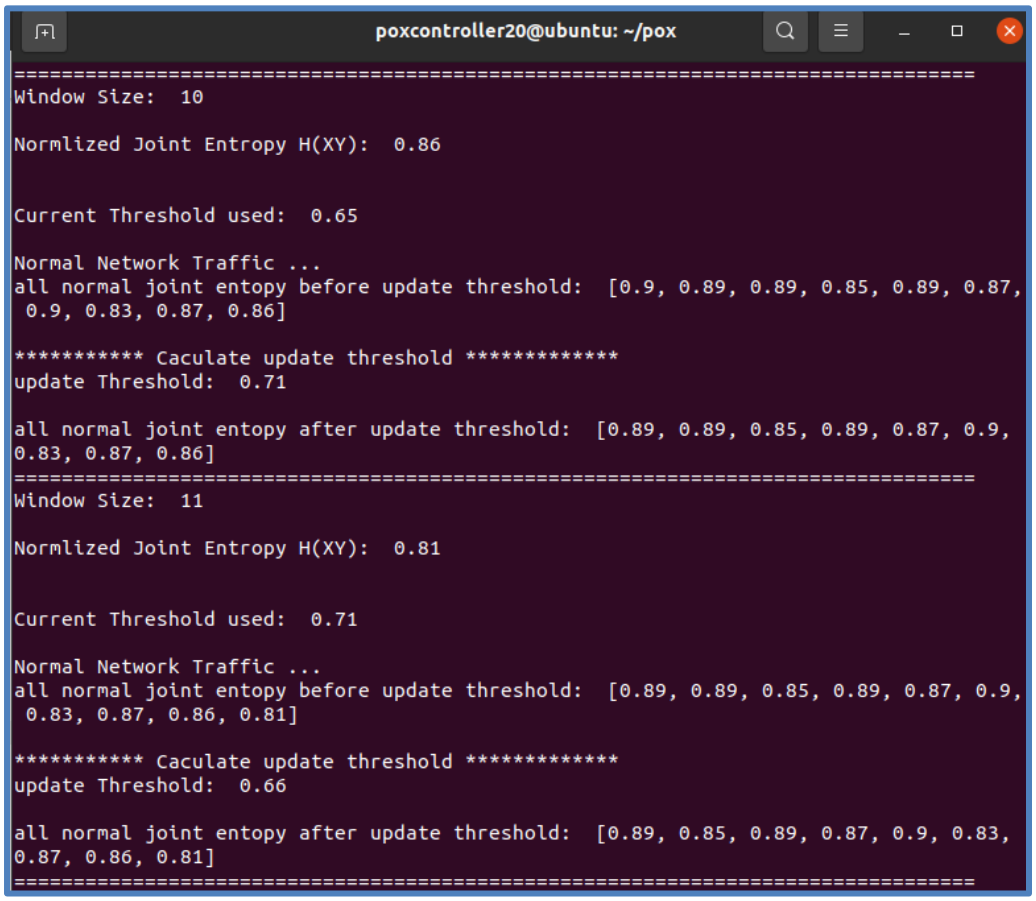


Figure 5.16: Joint Entropy Values and Threshold Update for Multiple Windows under Normal Traffic

Also, the calculation of joint entropy values was performed for multiple windows under DoS/DDoS attack traffic conditions. The attack was focused on four hosts (10.0.0.10 - 10.0.0.14) within slice1, targeting ports 80 and 53. The primary objective of this analysis was to evaluate the behavior of joint entropy and the pre-established threshold value in effectively detecting the attacks. Figure 5.17 presents the resulting joint entropy values, offering valuable insights into the effectiveness of the detection mechanism employed. The joint entropy values were observed to range between 0.51 and 0.52, while the threshold was set at 0.66. The effectiveness of this detection method was evident as the joint entropy

values fell below the threshold, indicating a low level of randomness in the traffic during the specified window size.

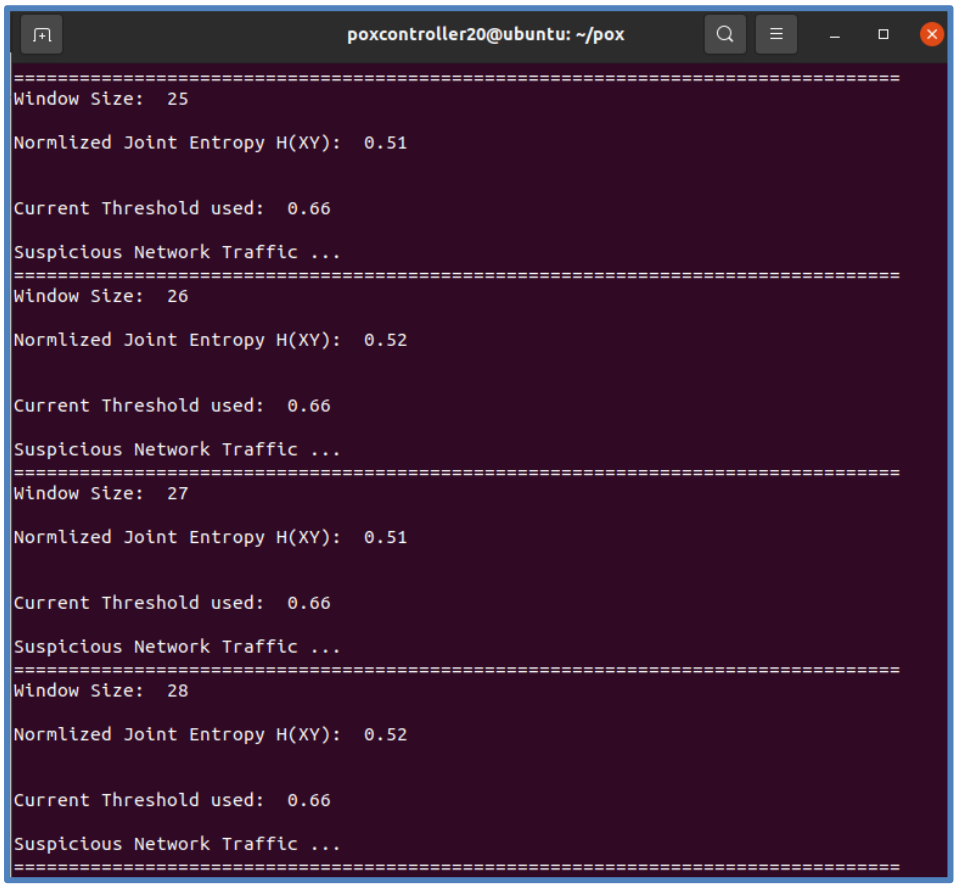


Figure 5.17: Joint Entropy Calculation and the Attack Detection on Multiple Windows

While Figure 5.18 illustrates the contrasting joint entropy values of normal traffic and the attacks. This visual representation offers valuable insights into the disparities in joint entropy levels observed between these two traffic conditions.

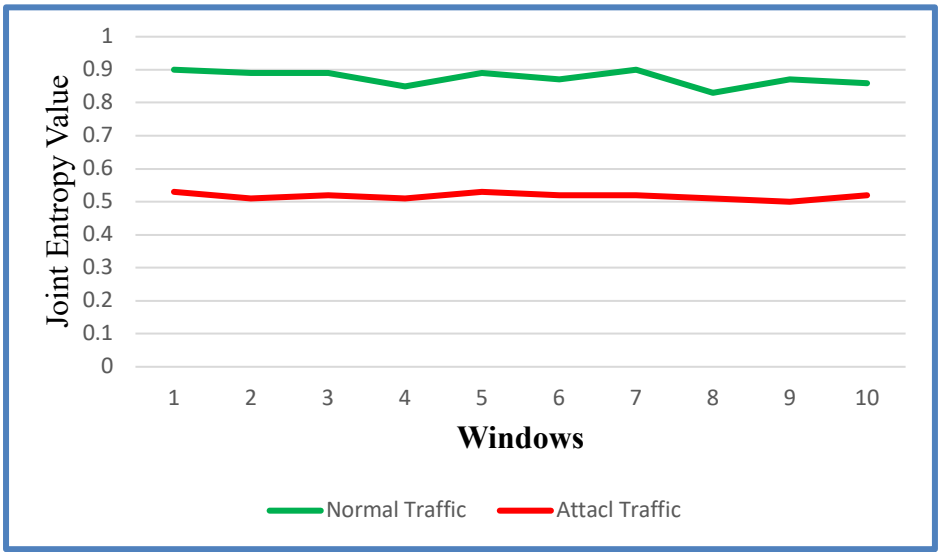


Figure 5.18: The Joint Entropy Values of Normal Traffic and DoS/DDoS Attacks

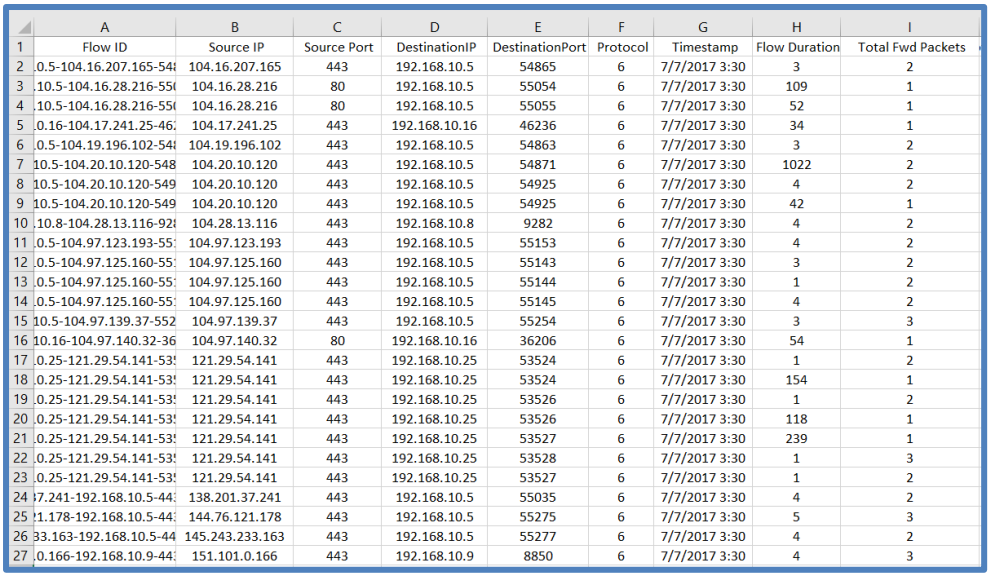
With slice-level detection, attacks were detected in less than one second. However, a decrease in accuracy is observed due to the presence of normal traffic directed toward the destination IP addresses and target ports which is classified as attack traffic. Therefore, a complementary approach is adopted by combining cross-level detection and slice-level detection. This combined approach aims not only to address accuracy limits but also to detect attacks targeting multiple chipsets simultaneously. The implementation and description of this approach are provided in the next subsection.

5.2.4.2 Cross-level Detection

In this section, the detailed results of building the DDNN model will be presented. This includes pre-processing operations on datasets, as well as the steps involved in building and evaluating the model for detecting the attacks across multiple slices.

Data Preprocessing

In the proposed model, two datasets were utilized for training and evaluation. The first dataset, CISIDS2017, which consisted of 84 features. A sample of this dataset is displayed in Figure 5.19. Meanwhile, the second dataset, referred to as the NSDS, was specifically created for this dissertation. A sample of the NSDS was illustrated in Figure 5.12.



	A	B	C	D	E	F	G	H	I
1	Flow ID	Source IP	Source Port	DestinationIP	DestinationPort	Protocol	Timestamp	Flow Duration	Total Fwd Packets
2	0.5-104.16.207.165-54	104.16.207.165	443	192.168.10.5	54865	6	7/7/2017 3:30	3	2
3	10.5-104.16.28.216-55	104.16.28.216	80	192.168.10.5	55054	6	7/7/2017 3:30	109	1
4	10.5-104.16.28.216-55	104.16.28.216	80	192.168.10.5	55055	6	7/7/2017 3:30	52	1
5	0.16-104.17.241.25-46	104.17.241.25	443	192.168.10.16	46236	6	7/7/2017 3:30	34	1
6	0.5-104.19.196.102-54	104.19.196.102	443	192.168.10.5	54863	6	7/7/2017 3:30	3	2
7	10.5-104.20.10.120-548	104.20.10.120	443	192.168.10.5	54871	6	7/7/2017 3:30	1022	2
8	10.5-104.20.10.120-549	104.20.10.120	443	192.168.10.5	54925	6	7/7/2017 3:30	4	2
9	10.5-104.20.10.120-549	104.20.10.120	443	192.168.10.5	54925	6	7/7/2017 3:30	42	1
10	10.8-104.28.13.116-92	104.28.13.116	443	192.168.10.8	9282	6	7/7/2017 3:30	4	2
11	0.5-104.97.123.193-55	104.97.123.193	443	192.168.10.5	55153	6	7/7/2017 3:30	4	2
12	0.5-104.97.125.160-55	104.97.125.160	443	192.168.10.5	55143	6	7/7/2017 3:30	3	2
13	0.5-104.97.125.160-55	104.97.125.160	443	192.168.10.5	55144	6	7/7/2017 3:30	1	2
14	0.5-104.97.125.160-55	104.97.125.160	443	192.168.10.5	55145	6	7/7/2017 3:30	4	2
15	10.5-104.97.139.37-552	104.97.139.37	443	192.168.10.5	55254	6	7/7/2017 3:30	3	3
16	10.16-104.97.140.32-36	104.97.140.32	80	192.168.10.16	36206	6	7/7/2017 3:30	54	1
17	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53524	6	7/7/2017 3:30	1	2
18	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53524	6	7/7/2017 3:30	154	1
19	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53526	6	7/7/2017 3:30	1	2
20	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53526	6	7/7/2017 3:30	118	1
21	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53527	6	7/7/2017 3:30	239	1
22	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53528	6	7/7/2017 3:30	1	3
23	0.25-121.29.54.141-53	121.29.54.141	443	192.168.10.25	53527	6	7/7/2017 3:30	1	2
24	7.241-192.168.10.5-44	138.201.37.241	443	192.168.10.5	55035	6	7/7/2017 3:30	4	2
25	1.178-192.168.10.5-44	144.76.121.178	443	192.168.10.5	55275	6	7/7/2017 3:30	5	3
26	33.163-192.168.10.5-44	145.243.233.163	443	192.168.10.5	55277	6	7/7/2017 3:30	4	2
27	0.166-192.168.10.9-44	151.101.0.166	443	192.168.10.9	8850	6	7/7/2017 3:30	4	3

Figure 5.19: Sample from CICIDS2017 Dataset

These datasets underwent pre-processing operations, and the initial step involved manually selecting features. The four features were chosen: "Average Packet Length", "Fwd IAT Mean", "Destination Port" and "Destination IP", as well as Label.

A Python program was used to perform additional preprocessing operations. Firstly, categorical variables were converted using the "LabelEncoder" from the scikit-learn library [100]. This conversion

transformed the categorical variable "Label" into numeric labels. Where 0 indicates normal traffic while 1 indicates attack traffic.

Next, the 'DestinationIP' column was converted into a human-readable format. This involved using the `ipaddress` module to translate the IP address strings into integer representations. By applying the lambda function to each value in the 'DestinationIP' column, the IP addresses were converted to integers.

Then, the numerical variables "Average Packet Length," "Fwd IAT," "Destination Port," and "DestinationIP" were normalized using the `MinMaxScaler` from the scikit-learn library [100]. This normalization process adjusted the values of these variables to a standard range, typically between 0 and 1. The normalized values were then mapped back to their respective columns. After these preprocessing steps, the two datasets, CICIDS2017 and NSDS, are shown in Figure 5.20 and Figure 5.21 respectively.

	A	B	C	D	E
1	**Packet Length Mean**	**Fwd IAT Mean**	**DestinationIP**	**DestinationPort**	**Label**
2	0.00309784	2.50E-08	0.751588887	0.837224562	0
3	0.00309784	0	0.751588887	0.840108649	0
4	0.00309784	0	0.751588887	0.840123909	0
5	0.00309784	0	0.751588889	0.705548434	0
6	0.00309784	2.50E-08	0.751588887	0.837194043	0
7	0.00309784	8.52E-06	0.751588887	0.83731612	0
8	0.00309784	3.33E-08	0.751588887	0.838140145	0
9	0.00309784	0	0.751588887	0.838140145	0
10	0.00309784	3.33E-08	0.751588888	0.141640725	0
11	0.011702952	3.33E-08	0.751588887	0.841619362	0
12	0.011702952	2.50E-08	0.751588887	0.841466764	0
13	0.011702952	8.33E-09	0.751588887	0.841482024	0
14	0.011702952	3.33E-08	0.751588887	0.841497284	0
15	0.006324757	1.25E-08	0.751588887	0.843160593	0
16	0	0	0.751588889	0.552493438	0
17	0	8.33E-09	0.751588892	0.816761277	0
18	0	0	0.751588892	0.816761277	0
19	0	8.33E-09	0.751588892	0.816791796	0
20	0	0	0.751588892	0.816791796	0

Figure 5.20: Sample of the Preprocessed CICIDS2017 Dataset

	A	B	C	D	E
1	**Fwd IAT**	**Dst IP**	**Dst Port**	**pkt size**	**Label**
2	0.299551713	1.00E+00	0	0.6	0
3	0.599743836	9.20722E-09	0.960680679	0.31	0
4	0.879923151	0	0.838491385	0.834	0
5	0.119436439	3.14984E-09	0.780780276	0.54	0
6	0.119436439	1.70E-09	0.812522913	0.428	0
7	0.429634966	4.36E-09	0.965400831	0.436	0
8	0.749839898	4.12E-09	0.856608212	0.186	0
9	0.639769452	3.14984E-09	0.866399853	0.798	0
10	0.179474864	4.36E-09	0.952279115	0.644	0
11	0.209494076	4.85E-09	0.855920811	0.636	0
12	0.639769452	2.18E-09	0.803052059	0.978	0
13	0.199487672	5.09E-09	0.891573995	0.58	0
14	0.449647775	4.36E-09	0.782445313	0.356	0
15	0.529699007	3.88E-09	0.845976415	0.858	0
16	0.709814281	4.60361E-09	0.968318465	0.612	0
17	0.479666987	5.33E-09	0.776839179	0.838	0
18	0.159462056	9.20722E-09	0.793672858	0.5	0
19	0.449647775	2.91E-09	0.911844678	0.728	0
20	0.919948767	4.11902E-09	0.983594036	0.48	0

Figure 5.21 Sample of the Preprocessed NSDS Dataset

Finally, a Python program was used to extract new features from existing ones. The program calculated the joint entropy between the IP

address and destination port, resulting in a new feature called "Joint IP and Port". Additionally, it calculated the entropy of "Fwd IAT Mean" and "Average Packet Length", resulting in two more new features named "H Fwd IAT Mean" and "H Average Packet Length". The resulting datasets, which includes these new features, is shown in Figure 5.22 and Figure 5.23 as a sample.

	A	B	C	D
1	**H Packet Length Mean**	**H Fwd IAT Mean**	**Joint IP and Port**	**Label**
2	0.362890549	0.494568303	0.971650589	0
3	0.61140809	0.777354634	0.865340297	0
4	0.992912647	0.961888159	0.179917993	0
5	0.985825294	1	0	0
6	0.992912647	1	0	0
7	0.992912647	1	0	0
8	0.985825294	1	0	0
9	0.804024174	0.829528908	0.363099916	0
10	0.559820186	0.51685419	0.297936399	0
11	0.734954311	0.653098035	0.641326767	0
12	0.964563236	0.148755804	0	0
13	0.943301178	0.197437624	0	0
14	0.905189337	0.173156166	0	0
15	0.926451395	0.221597345	0	0
16	0.942363299	0.829528908	0.62835121	0
17	0.831450267	1	0.708275632	0
18	0.90909944	0.964563236	0.518002625	0
19	0.914013888	0.564719286	0	0
20	0.95038853	0.699015375	0	0

Figure 5.22: CICIDS2027 Dataset Feature Extraction Results: Joint IP and Port, H Fwd IAT Mean, and H Average Packet Length

	A	B	C	D
1	**H Packet Length Mean**	**H Fwd IAT Mean**	**Joint IP and Port**	**Label**
2	0.930013763	0.916503872	0.810085084	0
3	0.978586174	0.926323937	0.927000914	0
4	0.915737879	0.934867572	0.810085084	0
5	0.944289646	1	0.810085084	0
6	0.985724116	0.907960237	0.927000914	0
7	0.961616119	0.916503872	0.917816564	0
8	0.975892003	0.919277996	0.942149593	0
9	0.957172348	0.934867572	0.902667885	0
10	0.971448232	0.957503089	0.927000914	0
11	0.975892003	0.934867572	0.942149593	0
12	0.990167887	0.934867572	0.942149593	0
13	0.947340235	0.984410424	0.942149593	0
14	0.961616119	0.946185331	0.869150506	0
15	0.985724116	0.934867572	1	0
16	0.985724116	0.943411206	0.951333942	0
17	0.983029945	0.92782163	0.942149593	0
18	0.978586174	0.939139389	1	0
19	0.971448232	0.900914296	0.951333942	0
20	0.947340235	0.889596537	0.893483535	0

Figure 5.23: NSDS Dataset Feature Extraction Results: Joint IP and Port, H Fwd IAT Mean, and H Average Packet Length

- **Developed Deep Neural Network**

The DDNN model was constructed using a training dataset consisting of 4515 records. Table 5.1 provides a sample of the building process, showcasing the steps taken during the 100 epochs of model training.

While Figure 5.24 illustrates the outcomes of the loss function for both the training and validation phases of the model. When the loss value is lower, particularly below 0.02 as observed in this case, it signifies strong performance in accurately predicting the desired output. This information provides valuable insights into the model's effectiveness and its ability to minimize the discrepancy between predicted and actual values.

Table 5.1: Steps of Training DDNN Model

Steps of Training Model (Building Model)
Epoch 1/100 198/198 [==========] - 1s 2ms/step - loss: 0.1479 - accuracy: 0.9745 - val_loss: 0.0438 - val_accuracy: 0.9908 Epoch 2/100 198/198 [==========] - 0s 2ms/step - loss: 0.0418 - accuracy: 0.9892 - val_loss: 0.0298 - val_accuracy: 0.9911 Epoch 3/100 198/198 [==========] - 0s 2ms/step - loss: 0.0347 - accuracy: 0.9900 - val_loss: 0.0251 - val_accuracy: 0.9915 Epoch 4/100 198/198 [==========] - 0s 2ms/step - loss: 0.0316 - accuracy: 0.9902 - val_loss: 0.0226 - val_accuracy: 0.9922 . . . . Epoch 98/100 198/198 [==========] - 0s 2ms/step - loss: 0.0164 - accuracy: 0.9953 - val_loss: 0.0127 - val_accuracy: 0.9963 Epoch 99/100 198/198 [==========] - 0s 2ms/step - loss: 0.0167 - accuracy: 0.9951 - val_loss: 0.0118 - val_accuracy: 0.9963 Epoch 100/100 198/198 [==========] - 0s 2ms/step - loss: 0.0163 - accuracy: 0.9946 - val_loss: 0.0130 - val_accuracy: 0.9952

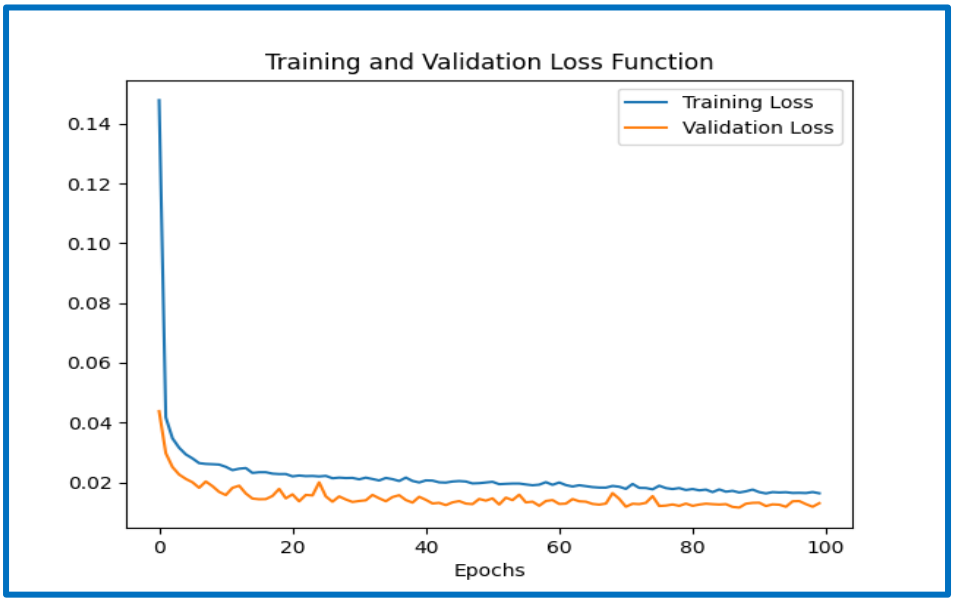


Figure 5.24: Training and Validation Loss Function

Moreover, Figure 5.25 presents the accuracy results for both the training and validation phases of the model. In this case, an accuracy value exceeding 0.99 indicates exceptional performance in accurately predicting the desired output.

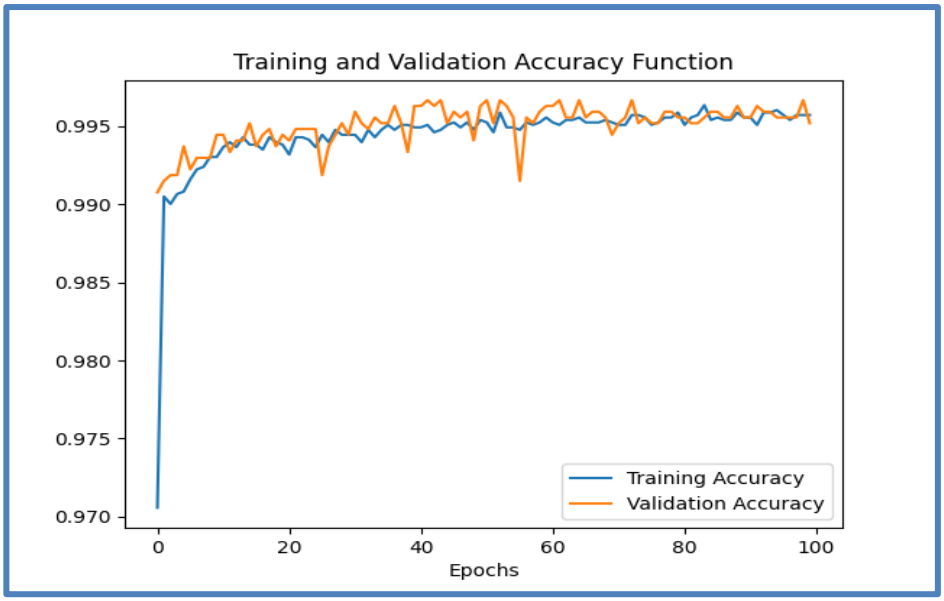


Figure 5.25: Training and Validation Accuracy

The binary classification (Normal and Attack) is clarified in Table 5.2 as a confusion matrix for testing data.

Table 5.2: Confusion Matrix for Binary Classification of Testing Data

		Predicted	
		Normal	Attack
Actual	Normal	1936	19
	Attack	4	2556

Figure 5.26 showcases the outcomes of accuracy, recall, precision, F-score, and specificity for the test data. Furthermore, a mean squared error (MSE) value of 0.00509 was recorded.

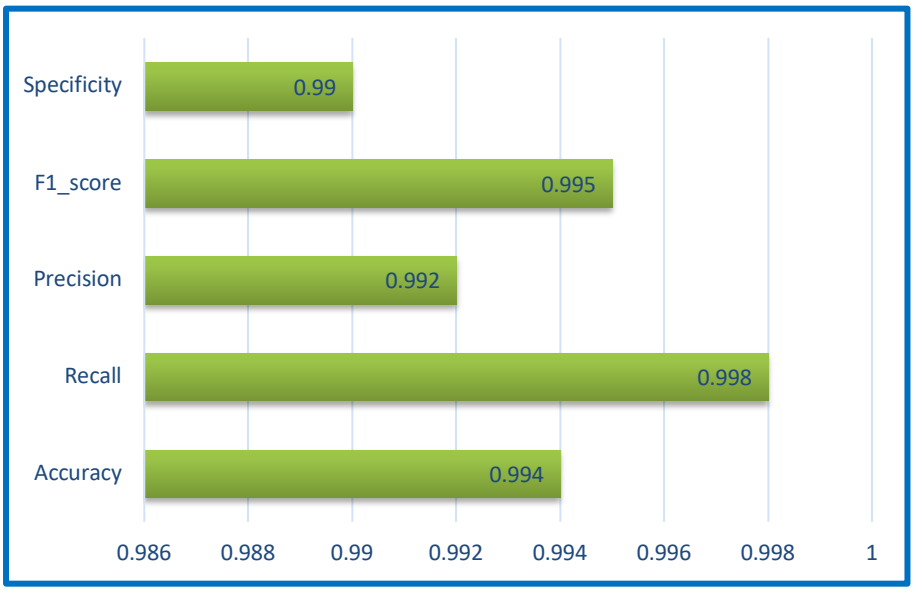


Figure 5.26: Evaluation Results for Binary Classification of Testing Data

5.2.4.3 Integration of Slice-Level and Cross-Level Detection for Network Slicing Security

In this section, an explanation will be given of how to implement slice-level and cross-level detection integration to detect DoS/DDoS attacks. The integration process begins by generating attack traffic and targeting one or a specific group of hosts in each network slice. Figure 5.27 shows attack traffic captured by the Wireshark tool, using spoofed IP addresses that are displayed under the source column to target destination IP addresses (10.0.0.10, 10.0.0.11, 10.0.0.42, 10.0.0.43, etc.).

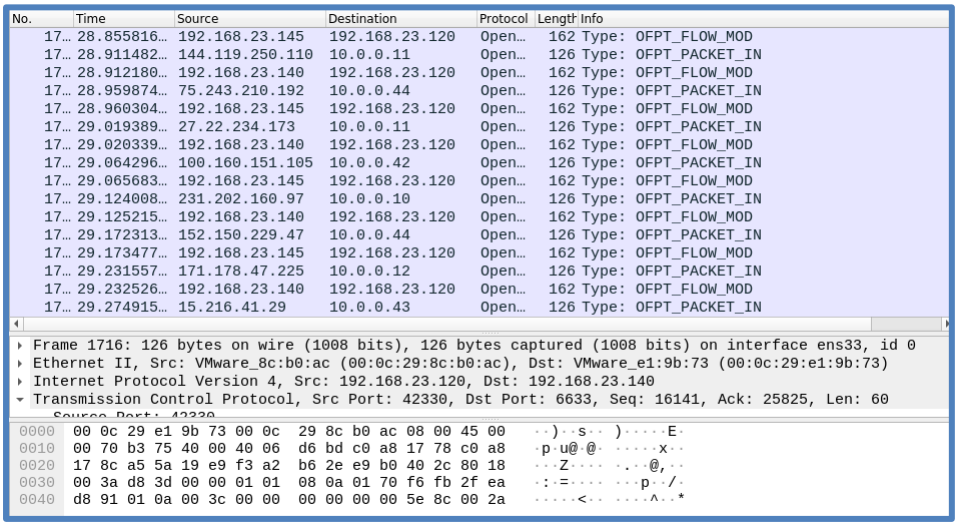


Figure 5.27: Analysis of Attack Traffic Captured by Wireshark: Spoofed IP Addresses and Targeted Destinations

Through the slice controller, early detection is performed at the slice-level. In Figure 5.28, early detection of suspicious traffic is shown in Slice 1 and Slice 2. After that, the parameters of the suspicious traffic were sent to the third-party server.

Furthermore, the joint entropy and threshold value used for slice-level detection, along with the entropy IAT and entropy Packet Length, were transmitted to the third-party server to conduct cross-level detection.

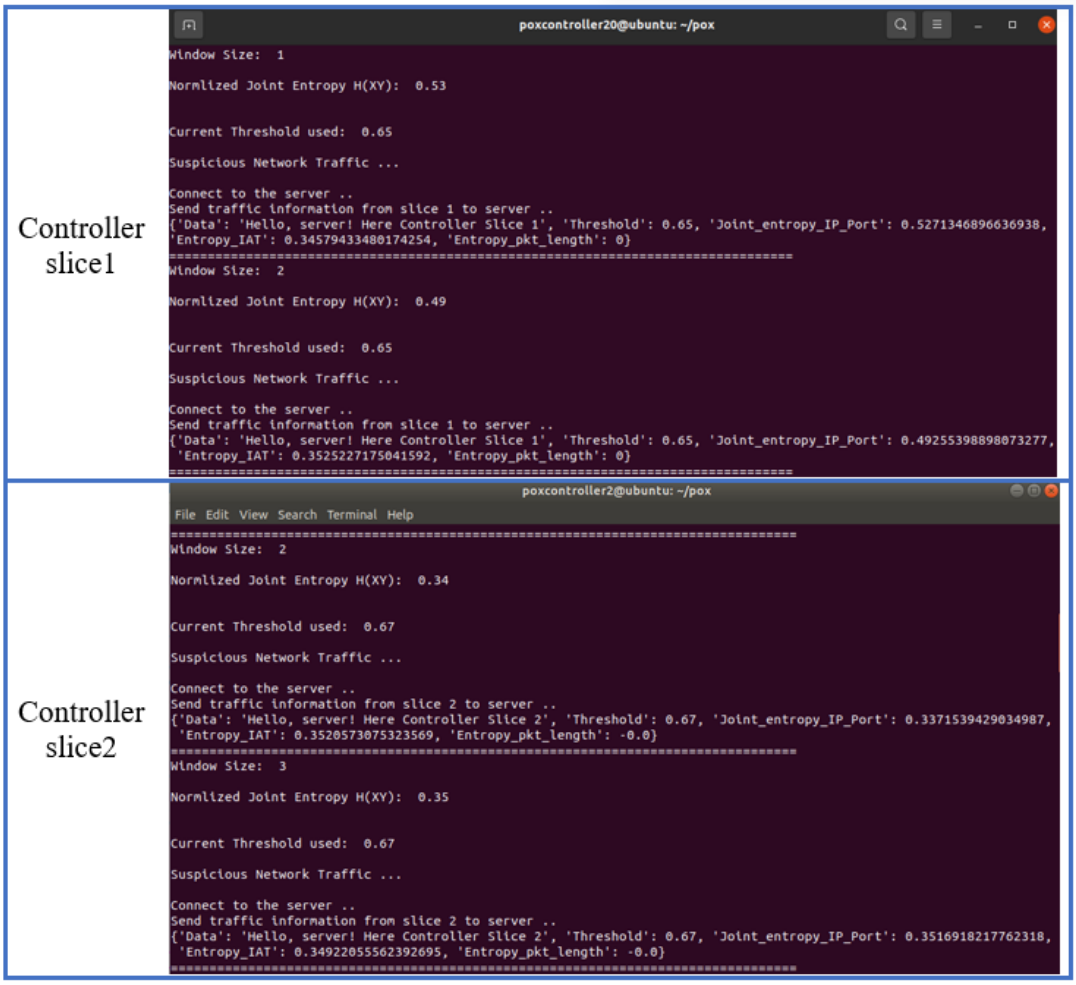


Figure 5.28: Early Detection of Suspicious Traffic at Slice1 and Slice2 with Transmission of Traffic Parameters to Third-Party Server

Figure 5.29 represents a capture by the Wireshark tool of the suspicious traffic information received by the server from each network slice controller.

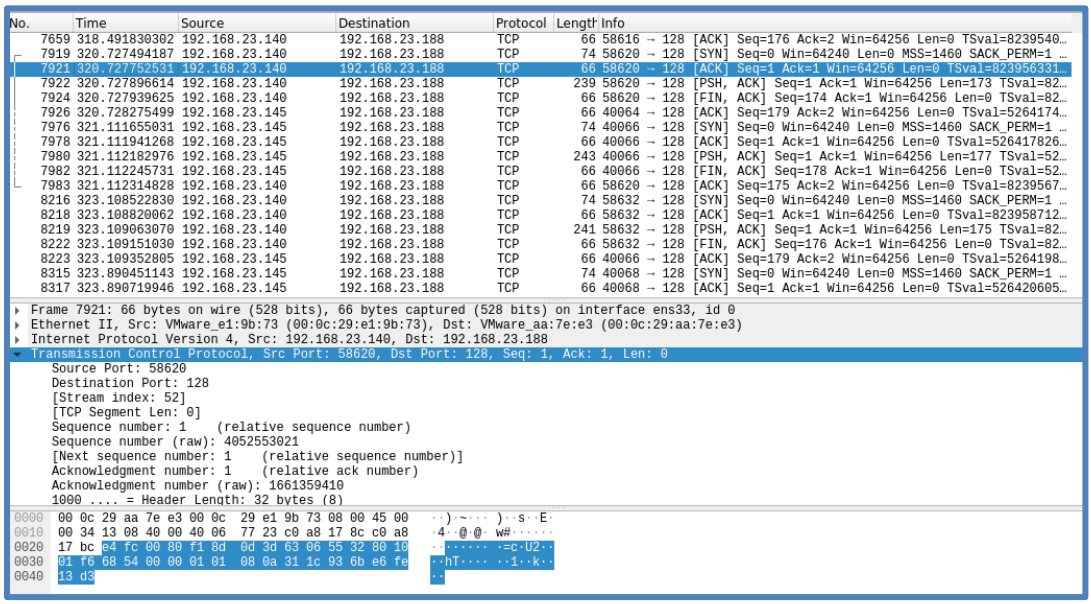


Figure 5.29: Wireshark Capture of Suspicious Traffic Information from Network Slice Controllers

Finally, the third-party server receives the suspicious traffic information, which is then utilized in the DDNN model for traffic classification. The DDNN model determines whether the traffic should be categorized as normal, attack, or targeting one or more slices. Figure 5.30 illustrates the detection of an attack on both slices using the DDNN model.

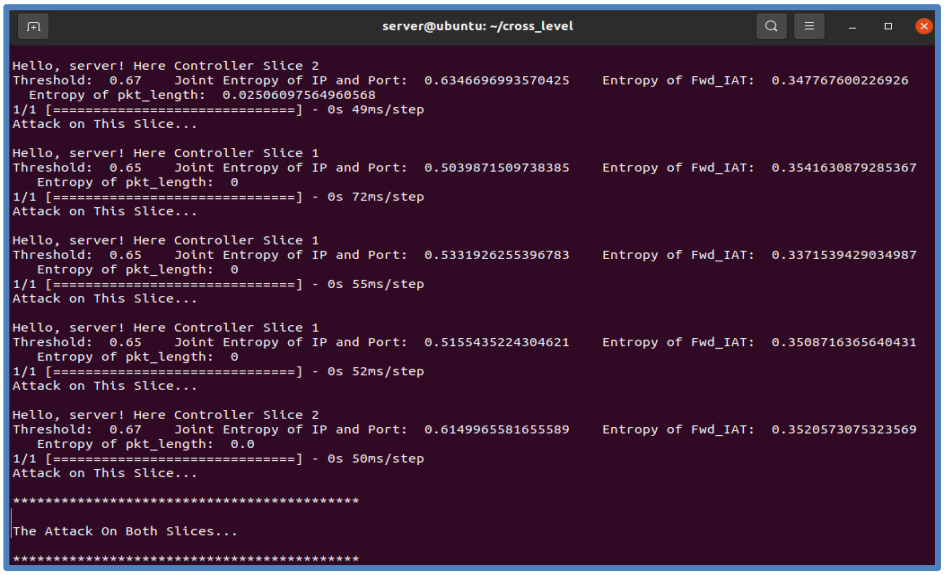


Figure 5.30: Cross-Level Detection: The Attack Traffic on Both Slices

On the other hand, Figure 5.31 and Figure 5.32 depict the detection of an attack targeting only one slice, specifically slices 1 and 2, respectively.

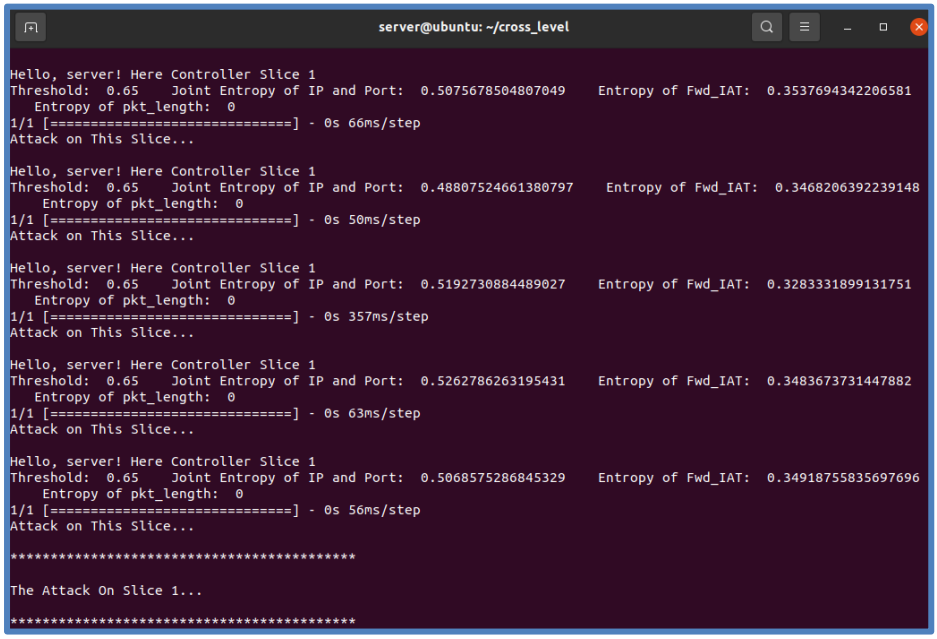


Figure 5.31: Cross-Level Detection: The Attack Traffic on slice1

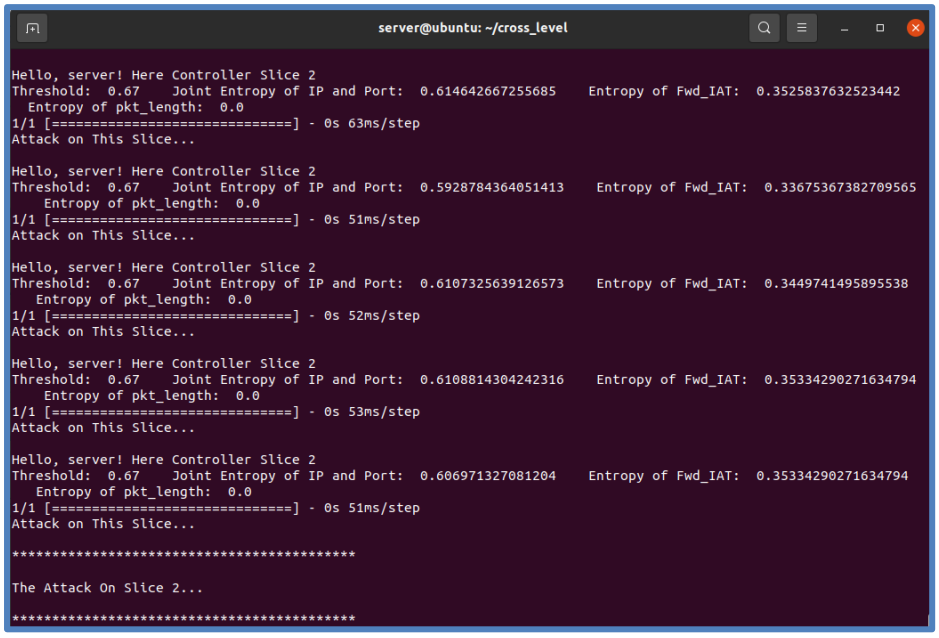


Figure 5.32: Cross-Level Detection: The Attack Traffic on slice2

5.3 Evaluating the Proposed Methodology

The performance of the DDNN model was assessed through a comparative analysis with the standard DNN model that employed the sigmoid activation function. Both models were trained on the same dataset, utilizing the training and test data splits that were defined. The training process for both models encompassed an equal number of epochs and identical configurations, including batch size, optimizer, and loss function. Figure 5.33 illustrates the performance analysis of both the standard DNN Model and the DDNN Model, focusing on the training and validation accuracy and loss.

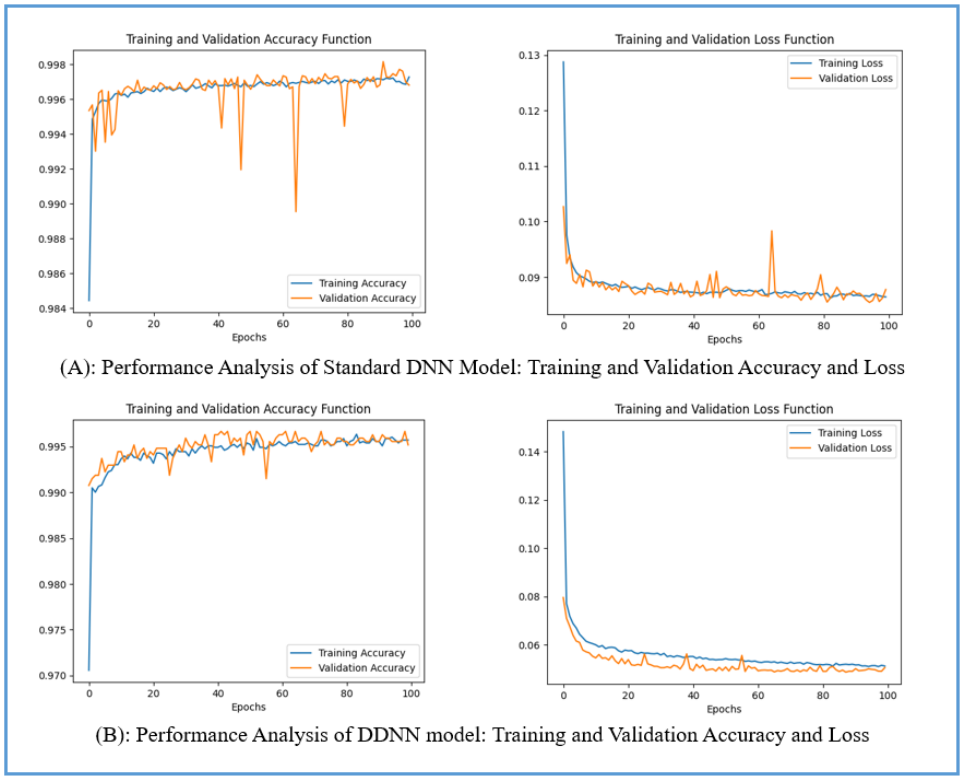


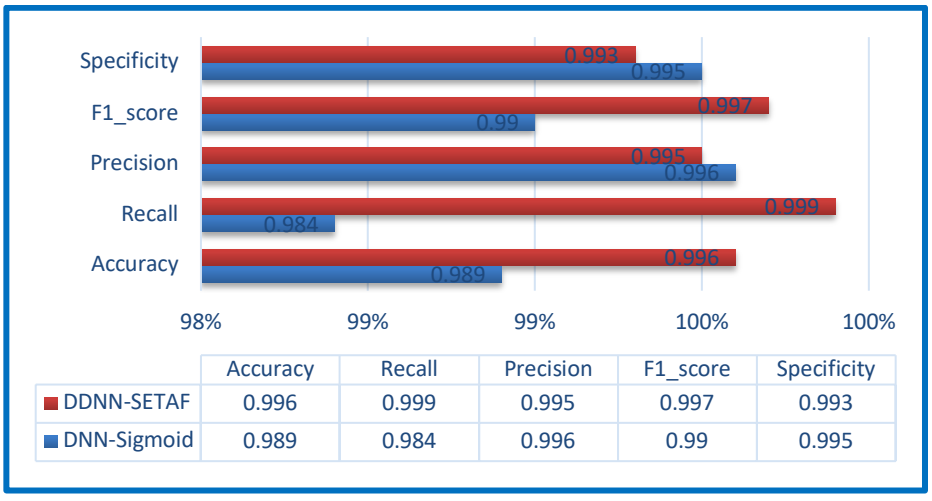
Figure 5.33: Visualizing Performance: Standard DNN Model and DDNN Model

Figure 5.33 (A) illustrates the training and validation accuracy and loss when using the standard activation function, namely the sigmoid function. In contrast, Figure 5.33 (B) presents the training and validation

accuracy and loss when employing a custom activation function called SETAF. Both figures demonstrate a notable decrease in the training and validation loss function, specifically from 0.09 to 0.06. This reduction signifies an improvement in the model's ability to minimize the disparity between predicted and actual values, leading to more precise predictions. Lower loss values generally indicate enhanced performance in capturing the underlying patterns and characteristics of the data.

Comparing the two figures, it can be inferred that the utilization of the custom activation function SETAF yields a more substantial decrease in the loss function compared to the standard sigmoid activation function. This suggests that SETAF has the potential to enhance the model's learning capacity and improve its predictive capabilities.

Following the training phase, the performance of both models was evaluated on the test data, and various performance metrics such as accuracy, precision, recall, specificity and F1-score were compared. These performance measures are presented in Figure 5.34.



	Accuracy	Recall	Precision	F1_score	Specificity
DDNN-SETAF	0.996	0.999	0.995	0.997	0.993
DNN-Sigmoid	0.989	0.984	0.996	0.99	0.995

Figure 5.34: Comparison of Performance Metrics: Test Data Evaluation for Standard DNN Model and DDNN Model

Further details into the training and testing of the two models can be obtained by referring to Table 5.3, where essential details and information about their performance and evaluation are provided.

Table 5.3: Implementation Metrics Comparison: Standard DNN Model vs DDNN Model

Model	Features	Training Time	MSE	Confusion Matrix
DNN-Sigmoid	Packet Length Mean, Destination IP, Destination Port, Fwd IAT Mean	1719.0747 seconds	0.0107	$\begin{bmatrix} 97266 & 452 \\ 1967 & 126060 \end{bmatrix}$
DDNN-SETAF	Packet Length Mean, Destination IP, Destination Port, Fwd IAT Mean	997.8193 seconds	0.0030	$\begin{bmatrix} 97106 & 612 \\ 85 & 127942 \end{bmatrix}$

Furthermore, the dataset's performance was assessed following the extraction of new features by analyzing its performance on both the standard DNN model and the DDNN model. This evaluation utilized the CICIDS2017 and NSDS feature extraction datasets, and the outcomes are depicted in Figure 5.35. The analysis primarily concentrates on the training and validation accuracy and loss.

Figure 5.35: Performance of CICIDS2017 and NSDS Feature Extraction Dataset: Standard DNN Model vs DDNN Model

Subsequently, after completing the training phase, the performance of both models was assessed on the test dataset derived from the NSDS feature extraction dataset. A comprehensive comparison was conducted using multiple performance metrics, encompassing accuracy, precision, recall, specificity, and F1-score. The findings of these performance measures are showcased in Figure 5.36.

For more comprehensive information on the training and testing of the two models, please refer to Table 5.4, which includes crucial details and evaluations of their performance.

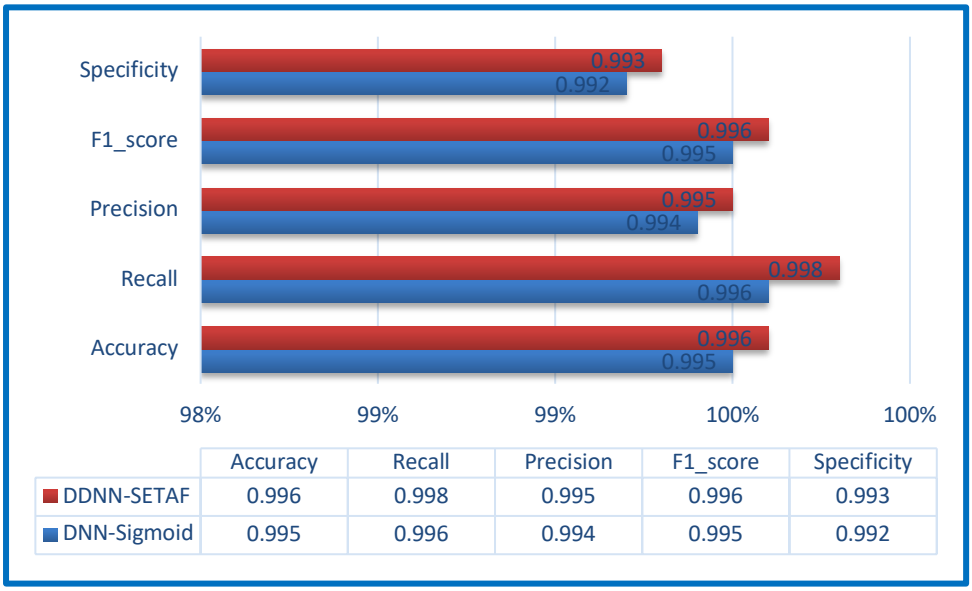


Figure 5.36: Comparison of Performance Metrics: Test Data Evaluation for Standard DNN Model and DDNN Model using NSDS Feature Extraction Dataset

Table 5.4: Comparison of Implementation Metrics: Standard DNN Model vs DDNN Model using CICIDS2017 and NSDS Feature Extraction Dataset

Model	Features	Training Time	MSE	Confusion Matrix
DNN-Sigmoid	H Packet Length Mean, H Fwd IAT Mean, Joint IP and Port	35.2323 seconds	0.0048	$\begin{bmatrix} 1941 & 14 \\ 8 & 2552 \end{bmatrix}$
DDNN-SETAF	H Packet Length Mean, H Fwd IAT Mean, Joint IP and Port	33.1226 seconds	0.0037	$\begin{bmatrix} 1943 & 12 \\ 5 & 2555 \end{bmatrix}$

5.4 The Performance of Network Slicing Under Attack

In the context of emulation network slicing, two performance metrics were measured: CPU utilization and RAM network utilization. Figure 5.37 illustrates the measurement of CPU and RAM network utilization in an emulation network slicing environment both before and under attack scenarios.

By comparing the CPU and RAM network utilization metrics before and under attack, can gain insights into the performance degradation caused by the attack. High CPU utilization during an attack may indicate increased processing requirements to handle the attack traffic or computational overhead of security mechanisms. Similarly, increased RAM network utilization suggests higher memory demands due to the attack's effects on data processing and storage.

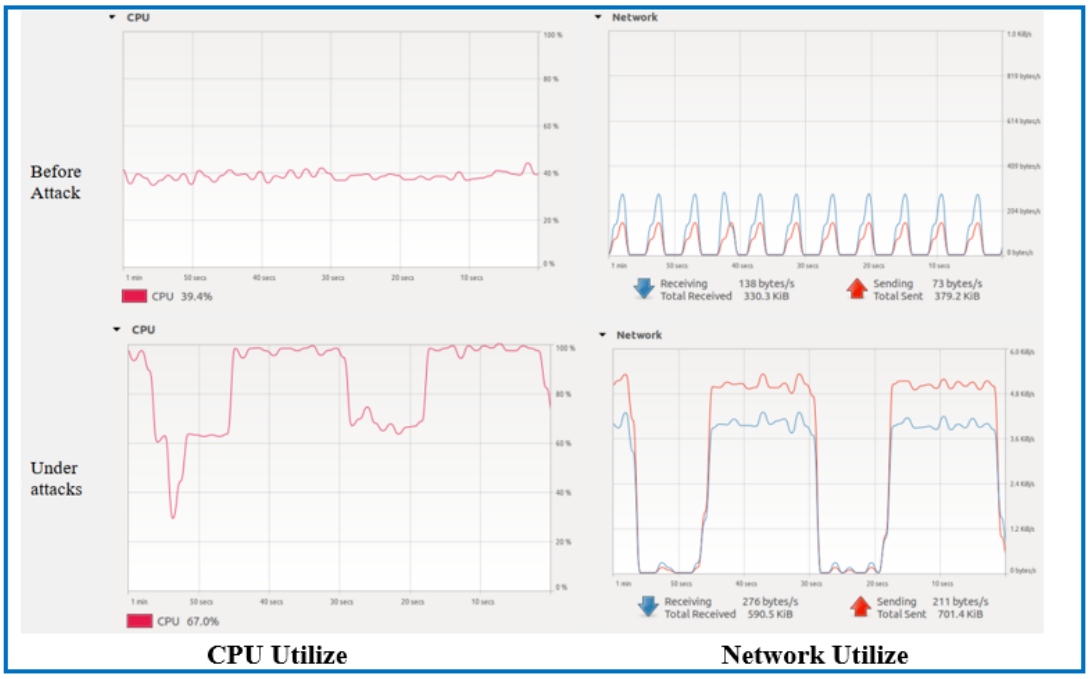


Figure 5.37: CPU and RAM of Network Slicing Before and Under Attack

Figure 5.38 illustrates the measurement of link latency between host 2 and host 3 in two scenarios: before and during an attack. Link latency refers to the duration it takes for a packet to traverse a link between two hosts.

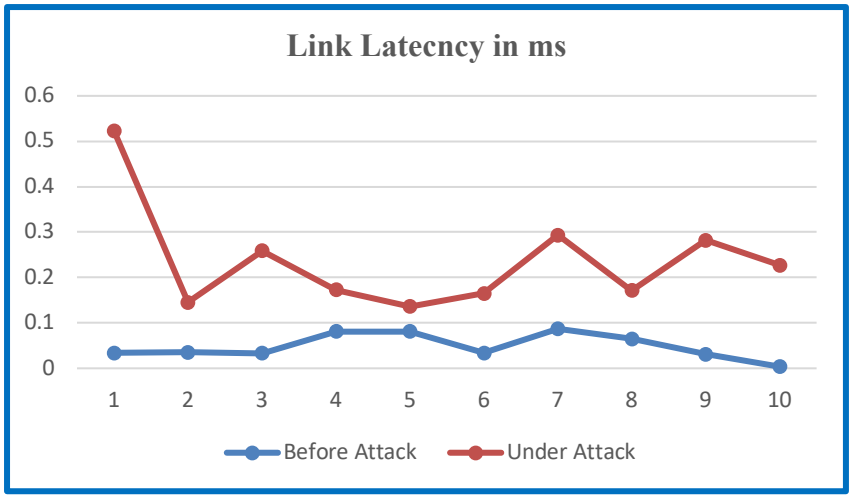


Figure 5.38: The Link Latency at Network Slicing

To measure the link latency, the ping command was utilized in the Mininet Command-Line Interface, as shown in Figure 5.39. This figure outlines the specific steps and procedures involved in accurately measuring the link latency.

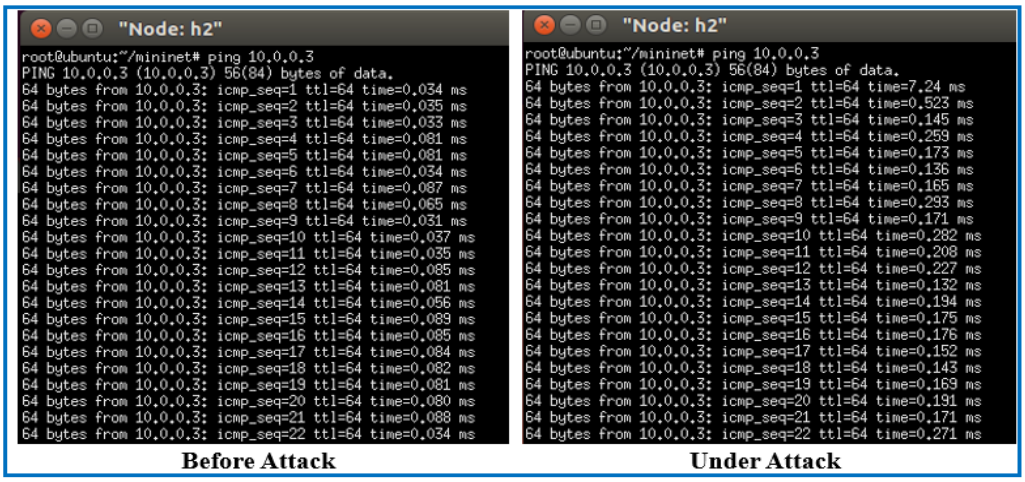
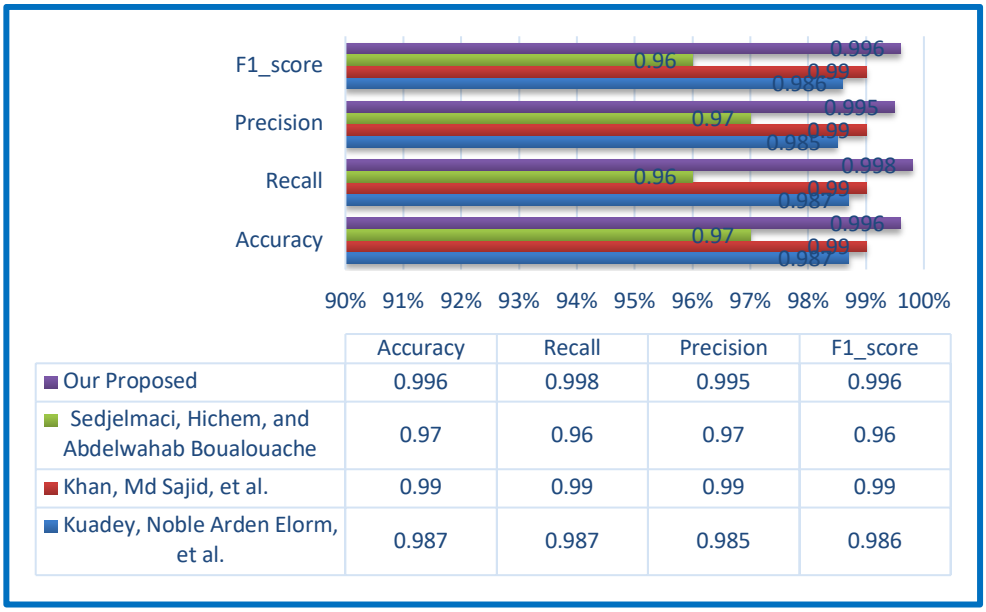


Figure 5.39: The Link Latency Calculation by PING Command Between Host 1 and Host 2

By comparing the link latency measurements before and during an attack, the impact of the attack on link performance can be assessed. An increase in link latency during an attack suggests potential network congestion, longer packet processing times, or limitations in network resources caused by the attack activity. These measurements provide valuable insights into the effects of attacks on network responsiveness and aid in the development of strategies to mitigate latency-related issues, thus enhancing the overall performance and security of NS deployments.

5.5 The Comparisons of the Proposed Model Against Related Works

When compared to some other related works, the proposed methodology for detecting DoS/DDoS attacks in a NS environment achieved better results and higher accuracy, as shown in Figure 5.40.



	Accuracy	Recall	Precision	F1_score
Our Proposed	0.996	0.998	0.995	0.996
Sedjelmaci, Hichem, and Abdelwahab Boualouache	0.97	0.96	0.97	0.96
Khan, Md Sajid, et al.	0.99	0.99	0.99	0.99
Kuadey, Noble Arden Elorm, et al.	0.987	0.987	0.985	0.986

Figure 5.40: Performance Comparison: Our Proposed Methodology vs. Related Work

The proposed methodology exhibited superior performance, despite using a smaller set of network traffic features for DoS/DDoS detection compared to other studies, as demonstrated in Table 5.5. This success can be attributed to the efficiency of the step-by-step development process, which was meticulously designed and implemented. By carefully selecting and integrating underlying network traffic features, the methodology achieved superior results, outperforming other methods.

Table 5.5: Comparison of the Proposed Methodology with Recent Works

Authors	Dataset	Network	Detection Methodology	No. of Features
Kuadey, Noble Arden Elorm, et al. 2021, [14]	CICDDoS 2019 dataset	Unknown	One level on each slice	9
Khan, Md Sajid, et al.2022 , [17]	Their own dataset	Testbed using Free5GC and UERANSIM	One level on each slice	11
Sedjelmaci, Hichem, and Abdelwahab Boualouache. 2023, [20]	CSE-CIC-IDS-2018 dataset	Unknown	Two level on each slice	84
Our Proposed	CICIDS2017 and NSDS datasets	Network Slicing Testbed using Mininet FlowVisor and pox controller	Two level (slice-level and Cross-level)	3

5.6 Summary

In this chapter, the proposed methodology for detecting DoS/DDoS attacks in a network slicing environment has been implemented and evaluated. The step-by-step implementation of a network slicing environment using Mininet, FlowVisor, and POX was demonstrated. The

implementation and evaluation of the proposed methodology for detecting the attacks at two levels, namely, the slice level and the cross-level, were highlighted, which ensured the comprehensive detection of attacks in a network slicing environment. Furthermore, a comparison was made between the proposed methodology and similar works, with an emphasis on the implementation and efficiency of detection. The strengths and advantages of the proposed methodology were then emphasized, indicating its superiority over other works.

While promising results have been shown with the proposed methodology, future developments have been suggested to further enhance its capabilities. These potential areas for improvement will be explored and discussed in the next chapter, with the aim of refining and developing the proposed methodology.

CHAPTER SIX

Conclusions and Future Works

6.1 The Conclusions

In this dissertation, an innovative methodology was presented for the detection of DoS/DDoS attacks in NS environment. The methodology consisted of a two-level architecture that aimed to achieve efficient and accurate attack detection. At the first level, the detection took place at the slice-level, where statistical analysis was utilized. The joint entropy was employed for this purpose, and a dynamically changing threshold was used to adapt to the varying network traffic conditions. The second level involved cross-level detection, which utilized a DDNN model. In this phase, an activation function called SETAF was created and applied in the output layer of the DDNN model.

To assess and validate the proposed methodology, it was implemented and tested in NS testbed environment. The experiments conducted provided valuable insights into the effectiveness and performance of the methodology. Consequently, the main conclusions drawn from this work are presented below:

1) Through the combination of joint entropy and DDNN, the proposed methodology has achieved a balanced solution that offers both speed and accuracy in detecting DoS/DDoS attacks. This balance is crucial for effective security measures in NS environments as it enables prompt response to attacks while maintaining reliable detection capabilities.
2) The use of joint entropy for slice-level detection facilitated early detection. Through joint entropy analysis, the methodology was able to detect suspicious traffic in less than a second. Furthermore,

this analysis allowed for the detection threshold to be dynamically adjusted in response to real-time changes in the network environment.

3) By incorporating a DNN model in the cross-level detection phase, the methodology experienced a significant boost in the accuracy of attack detection. The developed DNN model, employing the SETAF, played a pivotal role in providing a robust and precise assessment of attack patterns and characteristics.
4) A crucial role was played by the SETAF activation function, which was employed in the output layer of the DNN model to enhance the model's detection ability. This activation function effectively distinguished attacks from normal network traffic, as evidenced by the low MSE value of 0.0037. As a result, there was a significant improvement in the overall accuracy value of 0.996 for attack detection.
5) By extending the detection mechanism beyond individual slices, the methodology gained a holistic understanding of the network's behavior and traffic patterns. This enabled the identification of coordinated attacks that might span multiple slices or impact the overall network performance.
6) Reducing the number of features in the training dataset to three not only helped avoid high dimensionality but also contributed to reducing the complexity of the DDNN model in cross-level detection. This simplification reduced the time required to train the model to 33.1226 seconds while maintaining high levels of performance.

7) By implementing cross-level detection on a third-party server, we have eliminated the computational burden from the NS infrastructure. The cross-level detection process usually requires large computational resources to analyze and process network traffic data. Therefore, using a third-party server for this task removes the computational workload from the NS infrastructure. This ensures that the NS environment can focus on its core functions and operations without being overwhelmed by the computational requirements of cross-level detection.

Finally, it was concluded that the proposed methodology was effective in detecting attacks within network slicing environments. By incorporating slice-level statistical analysis and employing DDNN-based cross-level detection, the effectiveness, and value of the methodology in enhancing network security in the context of network slicing were demonstrated. These findings make a significant contribution to the development of network security practices in NS environments.

6.2 The Future Works

In this section, some future works are proposed to enhance the detection and mitigation of DoS/DDoS attacks in NS environments, as follows:

1) Further improve the proposed methodology for detecting low-rate DoS/DDoS attacks: Continuously enhance the methodology by improving slice-level statistical analysis techniques and enhancing the performance of DDNN-based cross-level detection through

exploring new algorithms, advanced machine learning techniques, or emerging technologies to enhance the accuracy and efficiency of the methodology.

2) Propose mechanisms to mitigate the impact of the attack and protect the network slices: by applying bandwidth restrictions to users who pose a threat. This approach involves identifying users or devices that exhibit suspicious or malicious behavior and limiting their bandwidth allocation to prevent them from overwhelming the NS.
3) Dynamically adjust resource allocation to mitigate attacks: Integrate mechanisms to dynamically adjust resource allocation within affected network segments when attacks are detected. By expanding the slice's resources, such as CPU, memory, or bandwidth, this adaptive resource allocation can help mitigate the impact of attacks on network slice services and maintain the QoS for legitimate users.

By considering these recommendations, future development efforts can further refine and optimize the proposed methodology, enhance its scalability and adaptability, and incorporate resource allocation adjustments for effective attack mitigation. These advancements will contribute to the ongoing improvement of network security in network slicing environments.

REFERENCES

[1] B. G. Gopal and P. G. Kuppusamy, "A comparative study on 4G and 5G technology for wireless applications," *IOSR Journal of Electronics and Communication Engineering*, vol. 10, no. 6, pp. 2278–2834, 2015.

[2] Q. V. Khanh, N. V. Hoai, L. D. Manh, A. N. Le, and G. Jeon, "Wireless communication technologies for IoT in 5G: Vision, applications, and challenges," *Wirel Commun Mob Comput*, vol. 2022, pp. 1–12, 2022.

[3] S. M. A. Kazmi, L. U. Khan, N. H. Tran, and C. S. Hong, *Network slicing for 5G and beyond networks*, vol. 1. Springer, 2019.

[4] Z. Nadir, T. Taleb, H. Flinck, O. Bouachir, and M. Bagaa, "Immersive services over 5G and beyond mobile systems," *IEEE Netw*, vol. 35, no. 6, pp. 299–306, 2021.

[5] P. Rost *et al.*, "Customized industrial networks: Network slicing trial at hamburg seaport," *IEEE Wirel Commun*, vol. 25, no. 5, pp. 48–55, 2018.

[6] P. Rost *et al.*, "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Communications magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[7] C. De Alwis, P. Porambage, K. Dev, T. R. Gadekallu, and M. Liyanage, "A Survey on Network Slicing Security: Attacks, Challenges, Solutions and Research Directions," *IEEE Communications Surveys & Tutorials*, 2023.

[8] Z. Kotulski et al., "On end-to-end approach for slice isolation in 5G networks. Fundamental challenges," *in 2017 Federated conference on computer science and information systems (FedCSIS)*, IEEE, 2017, pp. 783–792.

[9] M. J. K. Abood and G. H. Abdul-Majeed, "Classification of network slicing threats based on slicing enablers: A survey," *International Journal of Intelligent Networks*, vol. 4, pp. 103–112, 2023.

[10] D. Sattar and A. Matrawy, "Towards secure slicing: Using slice isolation to mitigate DDoS attacks on 5G core network slices," in *2019 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2019, pp. 82–90.

[11] M. Bonfim, M. Santos, K. Dias, and S. Fernandes, "A real-time attack defense framework for 5G network slicing," *Softw Pract Exp*, vol. 50, no. 7, pp. 1228–1257, 2020.

[12] H. Moudoud, L. Khoukhi, and S. Cherkaoui, "Prediction and detection of FDIA and DDoS attacks in 5G enabled IoT," *IEEE Netw*, vol. 35, no. 2, pp. 194–201, 2020.

[13] A. Thantharate, R. Paropkari, V. Walunj, C. Beard, and P. Kankariya, "Secure5G: A deep learning framework towards a secure network slicing in 5G and beyond," in *2020 10th annual computing and communication workshop and conference (CCWC)*, IEEE, 2020, pp. 852–857.

[14] N. A. E. Kuadey, G. T. Maale, T. Kwantwi, G. Sun, and G. Liu, "DeepSecure: Detection of distributed denial of service attacks on 5G network slicing—Deep learning approach," *IEEE Wireless Communications Letters*, vol. 11, no. 3, pp. 488–492, 2021.

[15] B. Bousalem, V. F. Silva, R. Langar, and S. Cherrier, "Ddos attacks detection and mitigation in 5g and beyond networks: A deep learning-based approach," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 1259–1264.

[16] W. Wang, C. Liang, Q. Chen, L. Tang, H. Yanikomeroglu, and T. Liu, "Distributed online anomaly detection for virtualized network slicing environment," *IEEE Trans Veh Technol*, vol. 71, no. 11, pp. 12235–12249, 2022.

[17] M. S. Khan, B. Farzaneh, N. Shahriar, N. Saha, and R. Boutaba, "SliceSecure: Impact and Detection of DoS/DDoS Attacks on 5G Network Slices," in *2022 IEEE Future Networks World Forum (FNWF)*, IEEE, 2022, pp. 639–642.

[18] S. Hossain, A. Boualouache, B. Brik, and S.-M. Senouci, "A Lightweight 5G-V2X Intra-slice Intrusion Detection System Using Knowledge Distillation," *A Lightweight 5G-V2X Intra-slice Intrusion Detection System Using Knowledge Distillation*, 2023.

[19] H. Bisht, M. Patra, and S. Kumar, "Detection and Localization of DDoS Attack During Inter-Slice Handover in 5G Network Slicing," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, IEEE, 2023, pp. 798–803.

[20] H. Sedjelmaci and A. Boualouache, "When two-layer federated learning and mean-field game meet 5g and beyond security: Cooperative defense systems for 5g and beyond network slicing," *IEEE Transactions on Network and Service Management*, 2023.

[21] A. Napolitano, A. Giorgetti, K. Kondepu, L. Valcarenghi, and P. Castoldi, "Network slicing: an overview," in *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, IEEE, 2018, pp. 1–4.

[22] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.

[23] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Resource sharing efficiency in network slicing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 909–923, 2019.

[24] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[25] S. S. Mahdi and A. A. Abdullah, "Survey on Enabling Network Slicing Based on SDN/NFV," in *International Conference on Information Systems and Intelligent Applications*, Springer, 2022, pp. 733–758.

[26] M. A. Habibi, B. Han, and H. D. Schotten, "Network slicing in 5G mobile communication architecture, profit modeling, and challenges," *arXiv preprint arXiv:1707.00852*, 2017.

[27] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[28] K. Abbas, T. A. Khan, M. Afaq, and W.-C. Song, "Network slice lifecycle management for 5g mobile networks: An intent-based networking approach," *IEEE Access*, vol. 9, pp. 80128–80146, 2021.

[29] Q. Li, G. Wu, A. Papathanassiou, and U. Mukherjee, "An end-to-end network slicing framework for 5G wireless communication systems," *arXiv preprint arXiv:1608.00572*, 2016.

[30] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[31] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.

[32] C. N. Tadros, B. Mokhtar, and M. R. M. Rizk, "Software Defined Network-Based Management Architecture for 5G Network," in *Paradigms of Smart and Intelligent Communication, 5G and Beyond*, Springer, 2023, pp. 171–195.

[33] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *computer networks*, vol. 121, pp. 100–111, 2017.

[34] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[35] Y. Li, D. Zhang, J. Taheri, and K. Li, "SDN components and OpenFlow," *Big Data Softw. Defin. Networks*, vol. 12, pp. 49–67, 2018.

[36] B. Agborubere and E. Sanchez-Velazquez, "Openflow communications and tls security in software-defined networks," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2017, pp. 560–566.

[37] C. Prabha, A. Goel, and J. Singh, "A survey on sdn controller evolution: A brief review," in *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, IEEE, 2022, pp. 569–575.

[38] F. Bannour, S. Souihi and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 333-354, 2017.

[39] U. Khan, Latif, et al., "Network slicing: Recent advances, taxonomy, requirements, and open research challenges", *IEEE Access*, pp.36009-36028, 2020.

[40] M. Kist, J. F. Santos, D. Collins, J. Rochol, L. A. DaSilva, and C. B. Both, "Airtime: End-to-end virtualization layer for ran-as-a-service in future multi-service mobile networks," *IEEE Trans Mob Comput*, vol. 21, no. 8, pp. 2701–2717, 2020.

[41] M. Veeraraghavan, T. Sato, M. Buchanan, R. Rahimi, S. Okamoto, and N. Yamanaka, "Network function virtualization: A survey," *IEICE Transactions on Communications*, vol. 100, no. 11, pp. 1978–1991, 2017.

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

[43] F. Granelli, "Network slicing," in *Computing in Communication Networks*, Elsevier, 2020, pp. 63–76.

[44] R. Sherwood *et al.*, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[45] Y. Zhang, J. Qin, X. Li, and S. Zhang, "Improvement of virtual network mapping based on OpenVirteX," in *Journal of Physics: Conference Series*, IOP Publishing, 2021, p. 032011.

[46] V. A. Cunha *et al.*, "Network slicing security: Challenges and directions," *Internet Technology Letters*, vol. 2, no. 5, p. e125, 2019.

[47] Z. Kotulski *et al.*, "Towards constructive approach to end-to-end slice isolation in 5G networks," *EURASIP J Inf Secur*, vol. 2018, pp. 1–23, 2018.

[48] F. Salahdine, Q. Liu, and T. Han, "Towards secure and intelligent network slicing for 5g networks," *IEEE Open Journal of the Computer Society*, vol. 3, pp. 23–38, 2022.

[49] W. Shalitha, and M. Liyanage., "The role of security orchestrator in network slicing for future networks," *Journal of Communications and Networks*, vol. 25, no. 3, pp. 355-369, 2023.

[50] C. Mohammed, et al., "A comprehensive survey on the E2E 5G network slicing model," *IEEE Transactions on Network and Service Management,* (2020), pp.49-62.

[51] S. Zhang, Y. Wang, and W. Zhou, "Towards secure 5G networks: A Survey," *Computer Networks*, vol. 162, p. 106871, 2019.

[52] R. F. Olimid and G. Nencioni, "5G network slicing: A security overview," *IEEE Access*, vol. 8, pp. 99999–100009, 2020.

[53] A. Jain, T. Singh, S. K. Sharma, and V. Prajapati, "Implementing Security in IoT Ecosystem Using 5g Network Slicing and Pattern Matched Intrusion Detection System: A Simulation Study.," *Interdisciplinary Journal of Information, Knowledge & Management*, vol. 16, no. 2021, 2021.

[54] B. Dzogovic, B. Santos, I. Hassan, B. Feng, N. Jacot, and T. Van Do, "Zero-Trust cybersecurity approach for dynamic 5g network slicing with network service mesh and segment-routing over IPv6," in *2022 International Conference on Development and Application Systems (DAS)*, IEEE, 2022, pp. 105–114.

[55] M. Nooribakhsh and M. Mollamotalebi, "A review on statistical approaches for anomaly detection in DDoS attacks," *Information Security Journal: A Global Perspective*, vol. 29, no. 3, pp. 118–133, 2020.

[56] S. D. Pande and A. Khamparia, "A review on detection of DDOS attack using machine learning and deep learning techniques," *Think India Journal*, vol. 22, no. 16, pp. 2035–2043, 2019.

[57] B. A. Khalaf, S. A. Mostafa, A. Mustapha, M. A. Mohammed, and W. M. Abduallah, "Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods," *IEEE Access*, vol. 7, pp. 51691–51713, 2019.

[58] P. A. Bromiley, N. A. Thacker, and E. Bouhova-Thacker, "Shannon entropy, Renyi entropy, and information," *Statistics and Inf. Series (2004-004)*, vol. 9, pp. 2–8, 2004.

[59] E. B. Manoukian, *Mathematical nonparametric statistics*. Taylor & Francis, 2022.

[60] L. D. Tsobdjou, S. Pierre, and A. Quintero, "An online entropy-based DDoS flooding attack detection system with dynamic threshold," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1679–1689, 2022.

[61] J. Kim, N. Shin, S. Y. Jo, and S. H. Kim, "Method of intrusion detection using deep neural network," in *2017 IEEE international conference on big data and smart computing (BigComp)*, IEEE, 2017, pp. 313–316.

[62] K.-L. Du and M. N. S. Swamy, *Neural networks and statistical learning*. Springer Science & Business Media, 2013.

[63] N. Gupta, P. Bedi, and V. Jindal, "Effect of activation functions on the performance of deep learning algorithms for network intrusion detection systems," in *Proceedings of ICETIT 2019: Emerging Trends in Information Technology*, Springer, 2020, pp. 949–960.

[64] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[65] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[66] J. T. Barron, “A general and adaptive robust loss function,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4331–4339.
[67] S. Jadon, “A survey of loss functions for semantic segmentation,” in *2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB)*, IEEE, 2020, pp. 1–7.
[68] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” *Adv Neural Inf Process Syst*, vol. 31, 2018.
[69] M. G. M. Abdolrasol *et al.*, “Artificial neural networks based optimization techniques: A review,” *Electronics (Basel)*, vol. 10, no. 21, p. 2689, 2021.
[70] E. Yazan and M. F. Talu, “Comparison of the stochastic gradient descent based optimization techniques,” in *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, IEEE, 2017, pp. 1–5.
[71] A. Lydia and S. Francis, “Adagrad—an optimizer for stochastic gradient descent,” *Int. J. Inf. Comput. Sci*, vol. 6, no. 5, pp. 566–568, 2019.
[72] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
[73] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
[74] D. Stiawan, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto, “CICIDS-2017 dataset feature analysis with information gain for anomaly detection,” *IEEE Access*, vol. 8, pp. 132911–132921, 2020.
[75] IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity, “https://www.unb.ca/cic/datasets/ids-2017.html.”
[76] K. Thearling, “An introduction to data mining,” *Direct Marketing Magazine*, pp. 28–31, 1999.
[77] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*, vol. 72. Springer, 2015.
[78] S. Patro and K. K. Sahu, “Normalization: A preprocessing stage,” *arXiv preprint arXiv:1503.06462*, 2015.
[79] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, “A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction,” *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 56–70, 2020.
[80] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” in *2014 IEEE Colombian conference on communications and computing (COLCOM)*, Ieee, 2014, pp. 1–6.
[81] M. Großmann and S. J. Schuberth, “Auto-Mininet: Assessing the Internet topology zoo in a software-defined network emulator,” *Messung, Mellierung un Bewertung von Rechensystemen (MMBnet)*, vol. 7, pp. 1–10, 2013.
[82] U. Prabu and V. Geetha, “Towards the Implementation of Traffic Engineering in SDN: A Practical Approach,” *in Inventive Systems and Control: Proceedings of ICISC 2023*, Springer, 2023, pp. 155–161
[83] M. Hibler *et al.*, “Large-scale virtualization in the emulab network testbed,” in *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008.
[84] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.
[85] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, “EstiNet openflow network simulator and emulator,” *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.

[86] B, Andreas, et al., “Pairing SDN with network virtualization: The network hypervisor placement problem,” *In 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 198-204, 2015.

[87] M. T. Kurniawan, I. Moszardo, and A. Almaarif, “Network Slicing on Software Defined Network Using Flowvisor and POX Controller To Flowspace Isolation Enforcement,” in *2022 10th International Conference on Smart Grid (icSmartGrid)*, IEEE, 2022, pp. 29–34.

[88] S. S. Mahdi and A. A. Abdullah, “Implementation of Network Slicing for Multi-controller Environment Based on FlowVisor,” in *International Conference on New Trends in Information and Communications Technology Applications*, Springer, 2022, pp. 173–188.

[89] S. Kaur, J. Singh, and N. S. Ghumman, “Network programmability using POX controller,” in *ICCCS International conference on communication, computing & systems, IEEE*, sn, 2014, p. 70.

[90] L. Zhu *et al.*, “SDN controllers: A comprehensive analysis and performance evaluation study,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020.

[91] S. Badotra and J. Singh, “Open Daylight as a Controller for Software Defined Networking.,” *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.

[92] S. Asadollahi, B. Goswami, and M. Sameer, “Ryu controller’s scalability experiment on software defined networks,” in *2018 IEEE international conference on current trends in advanced computing (ICCTAC)*, IEEE, 2018, pp. 1–5.

[93] P. Berde *et al.*, “ONOS: towards an open, distributed SDN OS,” in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[94] V. Ndatinya, Z. Xiao, V. R. Manepalli, K. Meng, and Y. Xiao, “Network forensics analysis using Wireshark,” *International Journal of Security and Networks*, vol. 10, no. 2, pp. 91–106, 2015.

[95] S. Suri and V. Batra, “Comparative study of network monitoring tools,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 1, no. 3, pp. 63–65, 2010.

[96] P. Natesan, P. Balasubramanie, and G. Gowrison, “Improving attack detection rate in network intrusion detection using adaboost algorithm with multiple weak classifiers,” *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, vol. 9, no. 8, pp. 2239–2251, 2012.

[97] S. Oshima, T. Nakashima, and T. Sueyoshi, “Early DoS/DDoS detection method using short-term statistics,” in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, IEEE, 2010, pp. 168–173.

[98] M. A. Aladaileh, M. Anbar, I. H. Hasbullah, and Y. K. Sanjalawe, “Information theory-based approaches to detect DDoS attacks on software-defined networking controller a review,” *Int. J. Educ. Inf. Technol*, vol. 15, pp. 83–94, 2021.

[99] D. Javaheri, S. Gorgin, J.-A. Lee, and M. Masdari, “Fuzzy logic-based DDoS attacks and network traffic anomaly detection methods: Classification, overview, and future perspectives,” *Information Sciences,* vol. 626, pp. 315-383, 2023.

[100] E. Bisong and E. Bisong, “Introduction to Scikit-learn,” *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pp. 215–229, 2019.

نموذج الشبكة العصبية العميقة المطور (DDNN) الذي تم تدريبه واختباره باستخدام مجموعتين مختلفتين من البيانات: مجموعة بيانات CICIDS2017 المعروفة وNSDS. يتضمن نموذج DDNN وظيفة تنشيط جديدة تسمى SETAF، والتي تجمع بين عناصر من الوظائف السيني والأسية إلى جانب العتبة الديناميكية.

أثناء تنفيذ المحاكي، تم إنشاء اختبار تقطيع الشبكة باستخدام محاكي شبكة Mininet لمحاكاة البنية التحتية. كان FlowVisor بمثابة طبقة المحاكاة الافتراضية على مكونات البنية التحتية، مما يتيح إنشاء شريحتين للشبكة. تم التحكم في كل شريحة شبكة بواسطة وحدة تحكم POX مخصصة. على مستوى الشريحة، تم إجراء التحليل الإحصائي بواسطة وحدات تحكم فردية لكل شريحة شبكة. علاوة على ذلك، تم تنفيذ نموذج DDNN على خادم طرف ثالث، والذي تم ربطه بكل شريحة شبكة. قدمت نتائج المحاكي التي تم الحصول عليها من اختبار تقطيع الشبكة تأكيدًا قويًا لفعالية المنهجية في اكتشاف الهجمات، مما حقق معدل دقة اكتشاف ملحوظًا قدره 0.99 خلال 49 مللي ثانية. وبالتالي، تثبت نتائج هذه الأطروحة بوضوح أن المنهجية فعالة في اكتشاف الهجمات والحفاظ على أمان تقطيع الشبكة في شبكة الجيل الخامس.

الخلاصة

يؤدي تقطيع الشبكة (NS) دورًا أساسيًا في بنية شبكة الجيل الخامس لأنه يتضمن تقسيم البنية التحتية المادية للشبكة إلى شبكات أو شرائح افتراضية متعددة، كل منها مصمم لتلبية المتطلبات المحددة لمختلف التطبيقات أو الصناعات أو مجموعات المستخدمين. تعمل كل شريحة شبكة كشبكة منطقية مستقلة لها مجموعتها الخاصة من الموارد والتكوينات وخصائص الأداء. في حين أن تكنولوجيا NS تسمح بتخصيص موارد الشبكة بكفاءة ومرونة بناءً على احتياجات حالات الاستخدام المختلفة، إلا أنها لا تخلو من التحديات، خاصة في مجال الأمن.

أحد المخاوف الأمنية الرئيسية المتعلقة بـ NS هو خطر هجمات رفض الخدمة (DoS) وهجمات رفض الخدمة الموزعة (DDoS). تتضمن هذه الهجمات إغراق الشبكة أو الخدمة بسيل من حركة المرور من مصادر متعددة، مما يجعلها غير متاحة للمستخدمين الشرعيين. في بيئة NS، يمكن أن تؤدي هجمات DoS/DDoS إلى تعطيل توفر شرائح الشبكة وأدائها، مما لا يؤثر فقط على الشريحة المستهدفة ولكن من المحتمل أن يؤثر على الشرائح الأخرى التي تشترك في نفس البنية التحتية المادية.

تركز هذه الأطروحة في المقام الأول على اكتشاف الهجمات داخل بيئة NS، وينقسم العمل إلى ثلاثة أجزاء رئيسية. يتضمن الجزء الأول إنشاء بيئة NS باستخدام شبكات محددة برمجيًا ومحاكاة الشبكة الافتراضية، والتي تتضمن شريحتين للشبكة مع وحدات تحكم مخصصة لكل شريحة. ويركز الجزء الثاني على إنشاء مجموعة بيانات متخصصة تسمى مجموعة بيانات تقطيع الشبكة (NSDS)، المصممة خصيصًا لـ NS. تقوم مجموعة البيانات هذه بتكرار سيناريوهات حركة مرور الشبكة العادية بشكل فعال بالإضافة إلى حركة مرور هجوم DoS/DDoS داخل بيئة NS. أخيرًا، يدور جوهر الأطروحة حول منهجية مبتكرة تستخدم NSDS للكشف عن الهجمات من خلال التقنيات الإحصائية والشبكات العصبية العميقة، مما يتيح الكشف الدقيق عن الهجمات داخل بيئة NS.

تتكون بنية المنهجية المبتكرة من مستويين للكشف عن الهجمات: مستوى الشريحة والمستوى المتقاطع. على مستوى الشريحة، تُستخدم تقنيات التحليل الإحصائي المستندة إلى الإنتروبيا المشتركة والعتبات الديناميكية للكشف المبكر عن الهجمات داخل كل شريحة من شرائح الشبكة. بالانتقال إلى الكشف عبر المستويات، تعمل البنية على توسيع الكشف عبر شرائح متعددة من الشبكة. على هذا المستوى، يتم اكتشاف الهجمات باستخدام

جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة بابل
كلية تكنولوجيا المعلومات
قسم شبكات المعلومات

اكتشاف هجمات DoS/DDoS على مستوى الشرائح في الشبكة ذات الشرائح عن طريق تطوير شبكة عصبيه عميقه

أطروحة

مقدمة إلى مجلس كلية تكنولوجيا المعلومات في جامعة بابل كجزء من متطلبات الحصول على درجة الدكتوراه فلسفة في تكنولوجيا المعلومات / شبكات المعلومات

من قبل

سؤدد صفاء مهدي هندي

باشراف

ا.م.د. الحارث عبد الكريم عبد الله

١٤٤٥هـ

٢٠٢٣ م