



الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
المدرسة الوطنية العليا للتكنولوجيات المتقدمة
National Higher School of Advanced Technologies



Department of Electrical Engineering And Industrial Computing

**Final Year Project to Obtain the Diploma of
Engineering**

-Field -

Communication

-Specialty-

Communications Systems and Networks

- Subject -

**Integration of Machine Learning for DDoS
Attack Detection in SDN**

Realized by

BELOUZ Yacine

BOUBAKEUR Dorsaf

Members of The Jury:

KHOUMERI El Hadi
LAKHDARI Kheira
BENDOUDA Djamila
REBAI Karima
BELAHCENE Abdelkader

President
Examiner
Examiner
Supervisor
Co-Supervisor

Algiers, the 25/06/2024

Academic year 2023–2024

Abstract

Application layer Distributed Denial of Service (DDoS) attacks pose significant threats to network security and service availability. This project addresses these threats by leveraging Software-Defined Networking (SDN) and Deep Learning (DL) technologies. We developed and tested several architectures to tackle these attacks effectively. The first architecture is the 1D-CNN, which excels at capturing spatial features from the traffic data. The second is the LSTM network, designed to capture temporal dependencies and patterns over time. The third is a hybrid CNN-LSTM model, which combines the spatial feature extraction capabilities of CNN with the temporal sequence processing strengths of LSTM. Based on the evaluations of these individual architectures, we designed the YOUD ensemble model to combine their strengths. Our custom SDN topology, created using Mininet and the Ryu controller, integrates an Intrusion Detection System (IDS) powered by the YOUD model. This IDS, coupled with a Flask-based application for real-time monitoring, includes an Intrusion Prevention System (IPS) that dynamically blocks malicious IP addresses. The system is trained on an enhanced version of the CICIDS2017 dataset, ensuring a robust, adaptive, and scalable defense mechanism against application layer DDoS attacks. This project demonstrates the potential of integrating Deep Learning with SDN to significantly enhance network security.

Keywords: SDN, Deep Learning, DDoS, Security, Classification.

Résumé

Les attaques par déni de service distribué (DDoS) au niveau de la couche application constituent une menace importante pour la sécurité des réseaux et la disponibilité des services. Ce projet aborde ces menaces en utilisant les technologies de réseau défini par logiciel (SDN) et l'apprentissage profond (DL). Nous avons développé et testé plusieurs architectures pour traiter ces attaques de manière efficace. La première architecture est le réseau 1D-CNN, qui excelle à capturer les caractéristiques spatiales des données de trafic. La deuxième est le réseau LSTM, conçu pour capturer les dépendances et les motifs temporels au fil du temps. La troisième est un modèle hybride CNN-LSTM, qui combine les capacités d'extraction de caractéristiques spatiales du CNN avec les forces de traitement de séquences temporelles du LSTM. Sur la base des évaluations de ces architectures individuelles, nous avons conçu le modèle en ensemble YOUD pour combiner leurs forces. Notre topologie SDN personnalisée, créée en utilisant Mininet et le contrôleur Ryu, intègre un système de détection d'intrusion (IDS) alimenté par le modèle YOUD. Cet IDS, couplé à une application basée sur Flask pour la surveillance en temps réel, inclut un système de prévention des intrusions (IPS) qui bloque dynamiquement les adresses IP malveillantes. Le système est entraîné sur une version améliorée de la base de données CICIDS2017, garantissant un mécanisme de défense robuste, adaptatif et évolutif contre les attaques DDoS de couche application. Ce projet démontre le potentiel de l'intégration de l'apprentissage profond avec le SDN pour améliorer considérablement la sécurité des réseaux.

Mots clés : SDN, Apprentissage Profond, DDoS, Sécurité, Classification.

الملخص

تُهدد هجمات حجب الخدمة الموزعة (DDoS) على طبقة التطبيق أمان الشبكة وتوفر الخدمة. يستخدم هذا المشروع الشبكات المعرفة برمجياً (SDN) والتعلم العميق (Deep Learning) لمواجهة هذه التهديدات. لقد طورنا واختبرنا عدة معماريات لمعالجة هذه الهجمات بفعالية. المعمارية الأولى هي الشبكة العصبية التلافيفية ١ (1D-CNN)، التي تتفوق في التقاط الميزات المكانية من بيانات المرور. المعمارية الثانية هي شبكة الذاكرة طويلة الأمد (LSTM)، المصممة لالتقاط الاعتماديات الزمنية والأنماط على مر الزمن. أما المعمارية الثالثة فهي نموذج هجين يجمع بين (1D-CNN) و (LSTM)، مما يجمع بين قدرات استخراج الميزات المكانية ومعالجة تسلسل الزمن.

بناءً على تقييمات هذه المعماريات الفردية، قمنا بتصميم نموذج (YOUD) التكاملي للجمع بين نقاط القوة فيها. تتكامل طبولوجية (SDN) المخصصة التي أنشأناها باستخدام (Mininet) ووحدة التحكم (Ryu) مع نظام كشف التطفل (IDS) الذي يعمل بنموذج (YOUD). هذا النظام، المجهز بتطبيق قائم على (Flask) للمراقبة في الوقت الفعلي، يشمل نظام منع التطفل (IPS) الذي يقوم بحجب عناوين (IP) الخبيثة بشكل ديناميكي. يتم تدريب النظام على نسخة محسنة من قاعدة بيانات (CICIDS2017)، مما يضمن آلية دفاع قوية، متكيفة، وقابلة للتوسع ضد هجمات (DDoS) على طبقة التطبيق.

يبرز هذا المشروع إمكانيات دمج التعلم العميق مع (SDN) لتعزيز أمان الشبكة بشكل كبير.

الكلمات المفتاحية: (SDN)، التعلم العميق، (DDoS)، الأمان، التصنيف.

Dedication

I can't believe that I have finally reached this final step, sitting here writing this and reflecting on everything

I've been through this past year. In fact, over the last five years, it has been a long, hard, and beautiful journey where I have learned a lot, developed my personality, and changed my perspective on many things.

My mind still can't believe that this is the end of a huge chapter of my life and the beginning of another. I have grown and am now the last graduate in my family, and to be honest, they can hardly believe it either.

Without God's help and guidance, I would never be here and achieve what I've achieved. I have always carried in my heart God's love and my belief in Him. After God's guidance, there's my first love in my life, Dad, **Boubakeur Abd el Hafid**. He was such an inspiration to me, a man full of strength and wisdom who carried the responsibility of his family since he was a child and fulfilled all his duties to each one of his children. He was my biggest supporter, always encouraging me to give my best in my studies and pushing me to the end. There are no words to describe what a good father and supporter he was. I just want to say I have always been proud of you, and I will never love someone like I love you. Rest in peace, Papa.

Let's not forget my mom, **Leila Nouar**, the greatest and strongest mom ever. What should I say that would really give her credit for how much she took care of me in every stage of my life? She always had my back, and if I could say so, she's the reason I am the person I am today.

I dedicate this work to my sister, **Dr. Manel**, who is literally my second mom. Also to **Kenza** and **Sara**, thank you for being in my life. And to my precious brothers, **Aymen** and **Amir**, and my lovely **Nieces and Nephews**.

Special thanks to my cousin **Ramy**, who helped me a lot these days. And to my partner in crime, my cousin **Yasmine**, and to all my family.

Special thanks to my friends, **Chakib Hamdani**, a man with such a pure heart and strong presence in my life; I can't think of a time when I needed him and he wasn't there for me. And **Meriem**, my childhood friend, thank you for being such a rare person.

This work wouldn't have been done as perfectly as it is without my dear partner, **Yacine Belouz**. He worked so hard with such calmness and patience that I was inspired by him to give all that I can and push harder with full confidence that **"Together, we made it happen"**

And most importantly, I dedicate this work to **myself**. Through every challenge and every doubt, I have remained confident in my abilities. I have proven to myself that I can achieve what others thought was beyond my reach. Despite the surprises and the hurdles, I embraced the challenge of working in a completely new and innovative field. I poured my heart and soul into this project, and I succeeded in bringing it to fruition, just as I had envisioned. This journey has shown me the strength I possess, and for that, I am immensely proud of myself.

- Dorsaf BOUBAKEUR -

Dedication

This thesis marks the culmination of a long and transformative journey that began on my first day of school and has led me to this significant milestone. Throughout this journey, I have often imagined the arrival of this day, and now that it is here, I am filled with immense gratitude and appreciation for those who have supported and inspired me along the way.

First and foremost, I extend my deepest gratitude and appreciation to **My Parents**. Your unwavering support, encouragement, and love have been my constant source of strength. Every moment of hard work and dedication is dedicated to you. Thank you for always being there for me, from the very beginning until now.

To **My Siblings** and **Their Children**, who have grown up alongside me and treated me as a brother, your presence has been invaluable. The sibling bond we share transcends age, and I cherish every moment we've had together. You will always hold a special place in my life, and I look forward to sharing many more stages of life with you, Inshallah.

I also want to express my heartfelt appreciation to my friends. Since September 16, 2018, I have met many wonderful people who have enriched my life in ways I could never have imagined. Among them, I am particularly grateful to **Khaled Gasmi** and **Yasser Cherfaoui**, not only for their friendship but also for their unwavering support and inspiration throughout my educational journey.

A special acknowledgment goes to my colleague, **Dorsaf Boubakeur**, whose unwavering dedication and tireless effort have been the driving forces behind our success. Her commitment and collaborative spirit have not only enriched our partnership but also have been instrumental in our remarkable accomplishment that marks the end of our student journey. I am truly proud and grateful for her invaluable contributions, and it has been an honor to work alongside someone as extraordinary as her.

Lastly, I dedicate this work to **Myself**. This journey has been a testament to my perseverance, resilience, and determination. It is a reminder of my ability to overcome challenges and achieve my goals. I am proud of the person I have become and look forward to the future with hope and confidence.

-YACINE BELOUZ-

Acknowledgments

First and foremost, we express our deepest gratitude to our God for guiding us through every step of our lives and bestowing upon us the strength, patience, and wisdom needed to complete this project successfully.

We extend our heartfelt thanks to our esteemed supervisors, **Madam Rebai Karima** and **Sir Belahcene Abdelkader**. Your guidance, wisdom, and unwavering support have been instrumental in the completion of this project. Your mentorship has been a beacon of inspiration, pushing us to strive for excellence and persevere through challenges.

A special thanks to Laboratory Engineer, Sir **Cherabit Mourad**, for his invaluable assistance and support throughout this journey. Your practical insights and unwavering help have been essential to our progress.

We also want to extend our sincere thanks to the jury members **KHOUMERI El Hadi**, **BENDOUDA Djamila**, and **LAKHDARI Kheira** for accepting to evaluate our work. Your feedback and evaluation are highly appreciated.

Completing this project has demonstrated our hard work and perseverance. It has been a journey filled with intense effort, late nights, and unwavering determination, teaching us the importance of dedication, teamwork, and resilience.

We are immensely grateful for the support of our families and friends, whose encouragement and understanding have been our source of strength and motivation.

Dorsaf BOUBAKEUR and Yacine BELOUZ

Table of Contents

List of Tables	13
List of Figures	14
Acronym list	16
Introduction	18
1 Software Defined Networking Threats	19
1.1 Introduction	19
1.2 SDN	19
1.2.1 Definition	19
1.2.2 SDN Architecture	20
1.3 OpenFlow Protocol	21
1.3.1 Definition	21
1.3.2 OpenFlow Architecture	21
1.3.3 Flow Tables	23
1.3.4 Openflow Messages	25
1.3.5 Openflow Functioning	25
1.4 SDN Benefits	26
1.4.1 Enhancing Configuration	26
1.4.2 Enhancing Performance	26
1.4.3 Innovation	27
1.5 SDN Threats	27
1.5.1 DDoS Attacks	27
1.6 Cyber Security	31
1.6.1 Intrusion Detection Systems (IDSs)	31
1.6.2 Intrusion Prevention Systems (IPSs)	31

1.7	Conclusion	31
2	Deep Learning	33
2.1	Introduction	33
2.2	Learning Processes in Deep Learning	33
2.2.1	Data Collection and Preparation	33
2.2.2	Model Initialization	34
2.2.3	Forward Propagation	34
2.2.4	Loss Calculation	34
2.2.5	Backward Propagation	34
2.2.6	Optimization Algorithm	34
2.3	Overlearning	35
2.3.1	Symptoms of Overfitting	35
2.3.2	Causes of Overfitting	35
2.3.3	Techniques to Prevent Overfitting	35
2.4	Artificial Neural Networks (ANNs)	36
2.5	Convolutional Neural Networks (CNNs)	36
2.5.1	Convolution Layer	36
2.5.2	Pooling Layer	37
2.5.3	Fully Connected Layer	37
2.6	Recurrent Neural Network (RNN)	37
2.6.1	RNN Structure	38
2.6.2	RNN Limitation	39
2.6.3	Long Short Term Memory (LSTM)	39
2.7	Evaluation Metrics	40
2.7.1	Confusion Matrix	40
2.7.2	Accuracy	40
2.7.3	Precision	40
2.7.4	Recall	41
2.7.5	F1 Score	41
2.8	Conclusion	41
3	Methodology	42

3.1	Introduction	42
3.2	Project Overview : IDS for DDoS Attacks in SDN using Deep Learning	42
3.3	Dataset	44
3.3.1	Overview	44
3.3.2	Chosen Dataset	44
3.3.3	Dataset Limitations	44
3.3.4	Enhancing the Dataset	45
3.3.5	Dataset Preprocessing	48
3.4	Design of Deep Learning Models	51
3.4.1	Overview	51
3.4.2	Chosen Models	51
3.5	Model Architectures	53
3.5.1	1D-CNN Model	53
3.5.2	LSTM Model	54
3.5.3	CNN-LSTM Hybrid Model	55
3.5.4	YOUD Model : An Ensemble Approach Combining CNN, LSTM, and Hybrid CNN-LSTM	56
3.6	Enhancing YOUD Ensemble Model with Threads	58
3.7	Real-time DDoS Detection and Mitigation Framework	58
3.8	Graphical User Interface	58
3.9	Flask Application and YOUD Model	60
3.10	Conclusion	60
4	Implementation and Results	61
4.1	Introduction	61
4.2	Implementation Details	61
4.2.1	Network Topology	61
4.2.2	Real-time DDoS Detection and Mitigation Setup	63
4.2.2.1	Flow Collector	63
4.2.2.2	Feature Extractor	64
4.2.2.3	DL-based Detector	65
4.2.2.4	Mitigation Module	66
4.2.3	Implementation of Multithreading in YOUD Ensemble Model	67

4.2.4	Implementation of The Flask Application	67
4.3	Models Training and Evaluation	69
4.3.1	Hyperparameters Tuning	69
4.3.2	1D-CNN Model	71
4.3.3	LSTM Model	73
4.3.4	Hybrid CNN-LSTM Model	75
4.4	IDS Results in Real-Time with Different DL Trained Models	77
4.4.1	1D-CNN Model Results	77
4.4.2	LSTM Model Results	78
4.4.3	Hybrid CNN-LSTM Model Results	80
4.4.4	Benign Traffic Detection Performance	82
4.5	YOUD Model Results	82
4.5.1	Detection of Slow Read Attack	83
4.5.2	Interpretation	83
4.5.3	Discussion	84
4.6	Results of The YOUD Model with Flask Application	84
4.6.1	Display Explanation	84
4.6.2	Real-time Detection with Flask	85
4.7	Conclusion	86
A	Materials and Software Tools	88
1.1	Ubuntu	88
1.2	Python	88
1.3	Emulation Tool	88
1.4	Network Controllers	89
1.4.1	Ryu	89
1.4.2	OpenDayLight (ODL)	90
1.5	Packet Analyzers	90
1.6	Flow Managers	91
1.6.1	OpenFlow Manager	91
1.6.2	CICFlowMeter	91
1.7	DDoS Attack and Legitimate Traffic Generation Tools	92

1.7.1	Hping3	92
1.7.2	iPerf	93
1.7.3	SlowHTTPTest	93
1.8	TensorFlow and Keras	94
B	Traffic Generation Commands	96
2.1	Attack Generation Commands	96
2.2	Legitimate Traffic Generation Commands	96
C	Dataset Preprocessing	98
3.1	Dataset Cleaning	98
3.1.1	Stripping Whitespace from Column Names	98
3.1.2	Handling Negative Values in Numeric Columns	98
3.1.3	Removing Zero Variance Columns	98
3.1.4	Replacing Infinite Values and Removing Not a Number Numeric Data Type (NaN)	98
3.1.5	Removing Columns with Identical Values	99
3.1.6	Removing Duplicate Rows	99
3.1.7	Standardizing Column Names	99
3.1.8	Visualizing Label Distribution	99
	Bibliography	101

List of Tables

3.1	Label Counts for CICIDS2017 Dataset	45
3.2	Updated Label Counts	48
3.3	Label Counts After Dataset Cleaning	48
3.4	Label Counts After Applying SMOTE.	49
3.5	Final labels count.	49
3.6	Selected Features and Their Meanings	51
3.7	Details of the 1D-CNN Model Architecture.	53
3.8	Details of the LSTM Model Architecture.	54
3.9	Details of The Hybrid CNN-LSTM Model Architecture.	55
3.10	Summary of The YOUD Model Architecture.	56
4.1	Feature Comparison for LSTM, CNN, and CNN-LSTM Models	65
4.2	Performance Metrics of Different Models with Varying Epochs	70
4.3	Evaluation Metrics for 1D-CNN and LSTM Models with RMSProp	70
4.4	Overall Performance Metrics of the CNN Model	73
4.5	Overall Performance Metrics of the LSTM Model	75
4.6	Overall Performance Metrics of the CNN-LSTM Model	77
4.7	Detection Results of The CNN Model on Attack and Benign Traffic	77
4.8	Detection Results of The LSTM Model on Attack and Benign Traffic	79
4.9	Detection Results of The CNN-LSTM Model on Attack and Benign Traffic	80
4.10	Performance metrics of YOUD model on different attack types	83
2.1	Attacks generation commands	96
2.2	Legitimate traffic generation commands	97

List of Figures

1.1	SDN layers	20
1.2	OpenFlow architecture	22
1.3	OpenFlow forwarding workflow	23
1.4	The flow entry structure	24
1.5	Illustration of fields in the packet header.	24
1.6	The process of packet transmission with OpenFlow	26
1.7	DoS attack	28
1.8	DDoS attack types taxonomy	28
2.1	CNN architecture [1]	36
2.2	Unfolded RNN cell	38
2.3	Diagram of a memory cell of an LSTM [2]	39
3.1	Proposed IDS Development Pipeline	43
3.2	Enhancing and Integrating Generated Data Patterns for Improved Dataset	45
3.3	DDoS Attack Topology	46
3.4	Ryu Starting	46
3.5	Ping Test Results	47
3.6	Correlation Matrix of The Considered Features	52
3.7	CNN Model Architecture	54
3.8	LSTM Model Architecture	55
3.9	CNN-LSTM Model Architecture	56
3.10	Ensemble Model Architecture	57
3.11	Logical Diagram of IDS/IPS	59
4.1	SDN Framework	62
4.2	Mitigation Module Stages	67

4.3	Flask Application Plateform	68
4.4	1D-CNN Training and Validation Accuracies	71
4.5	1D-CNN Training and Validation Loss	72
4.6	1D-CNN Model Confusion Matrix	72
4.7	LSTM Model Training and Validation Accuracy	73
4.8	LSTM Model Training and Validation Loss	74
4.9	LSTM Model Training and Validation Loss	74
4.10	Hybrid CNN-LSTM training and validation accuracy	75
4.11	Hybrid CNN-LSTM training and validation loss	76
4.12	Hybrid CNN-LSTM Model Confusion Matrix	76
4.13	CNN Detection Results in Terminal	77
4.14	CNN Traffic Analysis	78
4.15	LSTM Detection Results in Terminal	79
4.16	LSTM Traffic Analysis	80
4.17	Hybrid CNN-LSTM Detection Terminal Output	81
4.18	Hybrid CNN-LSTM Traffic Analysis	81
4.19	Analysis of Benign Traffic Detection by Different Models	82
4.20	YOUD Detection Terminal Output	83
4.21	Attack Traffic Analysis by IP Address	83
4.22	Real-time detection results for interfaces s1-eth1 and s1-eth2	85
3.1	Attacks Label Counts	100

Acronym List

- **ANN** : Artificial Neural Networks
- **API** : Application Programming Interface
- **AI** : Artificial Intelligence
- **CNN** : Convolutional Neural Network
- **DDoS** : Distributed Denial of Service
- **DL** : Deep Learning
- **DNN** : Deep Neural Network
- **DoS** : Denial of Service
- **HTTP** : Hypertext Transfer Protocol
- **ICMP** : Internet Control Message Protocol
- **IDS** : Intrusion Detection System
- **IP** : Internet Protocol
- **LSTM** : Long Short-Term Memory
- **RNN** : Recurrent Neural Network
- **SDN** : Software Defined Networking
- **SYN** : Synchronize
- **TCP** : Transmission Control Protocol
- **UDP** : User Datagram Protocol

Introduction

In the ever-evolving landscape of cyber threats, Distributed Denial of Service (DDoS) attacks pose a significant challenge, particularly at the application layer. These attacks are designed to exploit specific vulnerabilities and exhaust server resources, making them both complex and subtle. Variants such as slowread, slowrange, slowheaders, range attacks, and HTML flooding necessitate advanced detection and mitigation mechanisms to ensure network security and service availability.

Software Defined Networking (SDN) provides a promising paradigm that offers centralized control and programmable network management. By decoupling the control plane from the data plane, SDN enables a more flexible and dynamic network configuration. This approach not only simplifies the management of network resources but also facilitates the deployment of new services and policies with minimal intervention. Traditional security measures often fall short in detecting and mitigating advanced application layer DDoS attacks. These attacks not only disrupt services but can also lead to significant financial and reputational damage. Developing an effective Intrusion Detection System (IDS) that can swiftly identify and respond to such threats is critical. This project aims to leverage the capabilities of Software-Defined Networking (SDN) and advanced Deep Learning (DL) techniques to create a robust and adaptive security solution.

We have developed an Intrusion Detection System (IDS) specifically tailored for application layer DDoS attacks, implemented within the Ryu controller framework on a custom topology in Mininet. We proposed and evaluated different single models using 1D Convolutional Neural Network (1D CNN) and Long Short-Term Memory (LSTM). We also developed new approaches in this project that combine CNN and LSTM networks, namely hybrid CNN-LSTM model that integrates CNN and LSTM layers sequentially, and the YOUD ensemble model, which leverages the strengths of three models : CNN, LSTM, and the hybrid CNN-LSTM. The DL models were integrated within our IDS that operates in conjunction with a Flask application, providing an interface for real-time monitoring and management. Following the detection guaranteed by the IDS, an Intrusion Prevention System (IPS) is triggered to mitigate the attack by dynamically sending flow rules to block the IP addresses of the attackers, thereby preserving the integrity and performance of the network. The solution is implemented on an SDN network with a custom topology created using Mininet and controlled via the Ryu controller.

To provide comprehensive and diverse attack scenarios for training the proposed DL-based models, the CIDS2017 dataset was selected and further enhanced with a variety of attack and benign traffic generation tools from different classes. The first chapter of this thesis defines the SDN paradigm, discusses the Open-Flow protocol, its layers, and examines associated threats, particularly focusing on DDoS attacks and their significant types. The second chapter explores the field of DL, with a specific emphasis on CNN, RNN, and LSTM architectures. The third chapter details our methodology, including an overview of the project, dataset preprocessing and enhancement, the design of DL models, the architecture of IDS and IPS, and the development of a Flask application. The fourth and final chapter presents the results of our project.

Chapter 1

Software Defined Networking Threats

1.1 Introduction

Traditional computer networks have long been the backbone of communication infrastructures, but they come with inherent limitations. These networks are typically characterized by their static nature, where network devices such as switches and routers are individually configured and managed. This static configuration can lead to issues such as limited scalability, inefficient resource utilization, and complex management processes.

Software-Defined Networking (SDN) emerged as a paradigm shift in network architecture, aiming to address the limitations of traditional networks. It decouples the control plane from the data plane, centralizing network control and enabling programmability, flexibility, and automation. By abstracting the underlying hardware, SDN facilitates dynamic network configuration, efficient resource allocation, and easier management through software-based controllers.

However, despite its numerous benefits, SDN is not without challenges. One significant vulnerability is its susceptibility to Distributed Denial of Service (DDoS) attacks. These attacks flood a network with an overwhelming amount of traffic, rendering it inaccessible to legitimate users. In SDN environments, the centralized control plane becomes a prime target for attackers, as compromising it can disrupt the entire network. Additionally, the controlled hosts within SDN are also vulnerable since they receive traffic based on a centralized decision maker (controller). If this decision is inappropriate, they may become targets of an attack. In this chapter, we will explore the SDN paradigm in depth, covering its concept, layers, the OpenFlow protocol, and its advantages. Then, we will examine some of its vulnerabilities and threats.

1.2 SDN

1.2.1 Definition

SDN is an emerging network architecture where the network control is detached from forwarding, allowing for direct programmability [3], this separation enables the extraction of network intelligence from the dataplane's switching devices, relocating it to controllers within the control plane. These controllers oversee the underlying network, offering a versatile and efficient platform for implementing diverse network applications and services. Consequently, switching devices can now be externally governed by software without onboard intelligence, this evolution provides a simplified programmable environment, granting external software greater ability in defining network behavior [4].

1.2.2 SDN Architecture

SDN architecture comprises three layers : an infrastructure layer, a control layer, and an application layer, positioned on top of one another [4] as depicted in figure 1.1. The communication between these layers is guaranteed by northbound and southbound Application Programming Interfaces (APIs), where the northbound API ensures communication between the application and control layers, while the southbound API manages communication between the infrastructure and control layers [5]. This section delves into a comprehensive illumination of the SDN architecture.

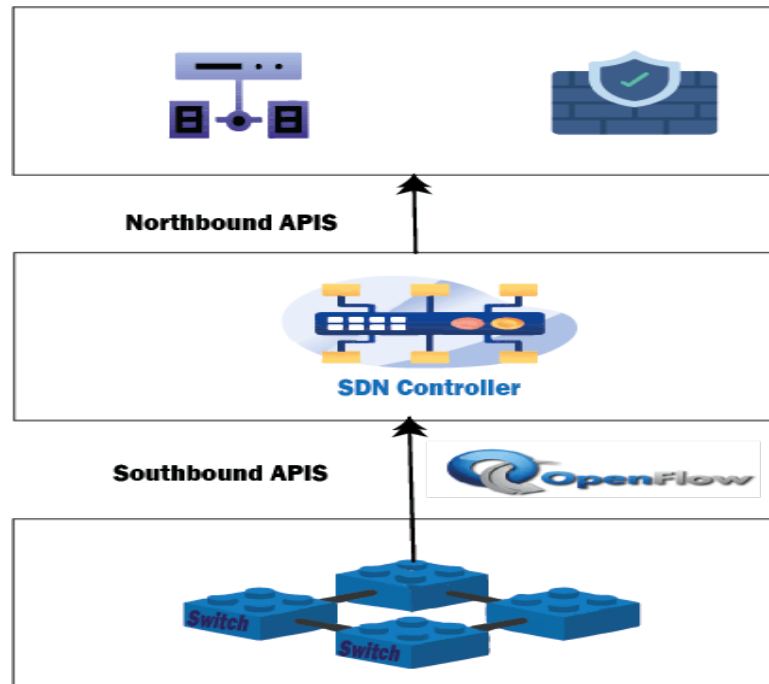


FIGURE 1.1 : SDN layers

1. Infrastructure layer :

The infrastructure layer includes devices such as switches and routers, which are in charge of gathering network status information like network topology, traffic statistics, and network usages. This data is temporarily stored on the devices and then transmitted to controllers. Furthermore, these devices use rules provided by the controllers to analyze and forward incoming packets. When a packet arrives, a switching device identifies the relevant forwarding rule and processes the packet accordingly [4]. The communication between the infrastructure and control layers is made possible by the use of OpenFlow protocol, which will be discussed further in the next section [3].

2. Control layer :

The control layer is responsible for managing and orchestrating the network, it serves as a bridge between the infrastructure layer and the application layer via its dual interfaces. The controller is the major component that accesses the infrastructure layer through the southbound interface. Specifically, it allows the switching devices to be configured programmatically by establishing packet forwarding rules using flow tables, and provides an up to date network global view. Also, it provides a northbound interface (API) that allows the interaction between control and application layers, enabling applications to request network status from switching devices and make system tuning decisions based on it, and apply these decisions by setting appropriate packet forwarding rules to switching devices using this API [4, 6].

3. Application layer :

The application layer is where the SDN applications are held, which are customized in meeting unique user demands. They reach and manage switching devices situated within the infrastructure layer leveraging the programmability guaranteed by the control layer. Specifically, software applications adjust the system configurations according to the retrieved network status information [4]. Several applications related to network security and optimization were developed including Intrusion Detection Systems (IDSs) [7] and load balancers [8].

1.3 OpenFlow Protocol

The OpenFlow protocol serves as the predominant choice for the southbound interface of Software-Defined Networking (SDN), where it delineates the separation between the data plane and the control plane [9]. Initially conceived by Stanford University, then it has evolved into a standardized protocol under the stewardship of the Open Networking Foundation (ONF) [3].

1.3.1 Definition

OpenFlow is a standardized protocol utilized in Software-Defined Networking (SDN) environments to manage communication between the control and forwarding planes of network devices. In essence, it enables a centralized controller to dictate how network traffic should be forwarded by configuring flow tables in switches and routers.

With this protocol, SDN controllers can provide a comprehensive view of a network, including network statistics, to SDN applications. These applications can then reprogram the network based on the provided information. Consequently, network reprogramming involves adding, modifying, or deleting flow rules. Thus by decoupling the control plane from the data plane, OpenFlow enhances network programmability, enabling dynamic and centralized management of network traffic flows, thereby improving network agility, scalability, and efficiency.

1.3.2 OpenFlow Architecture

The OpenFlow architecture is the actual implementation of SDN networks, primarily based on three components : the data plane, which consists of OpenFlow switches; the control plane, constituted by OpenFlow controllers; and a secure chain that allows switches to connect to the control plane. As illustrated in figure 1.2.

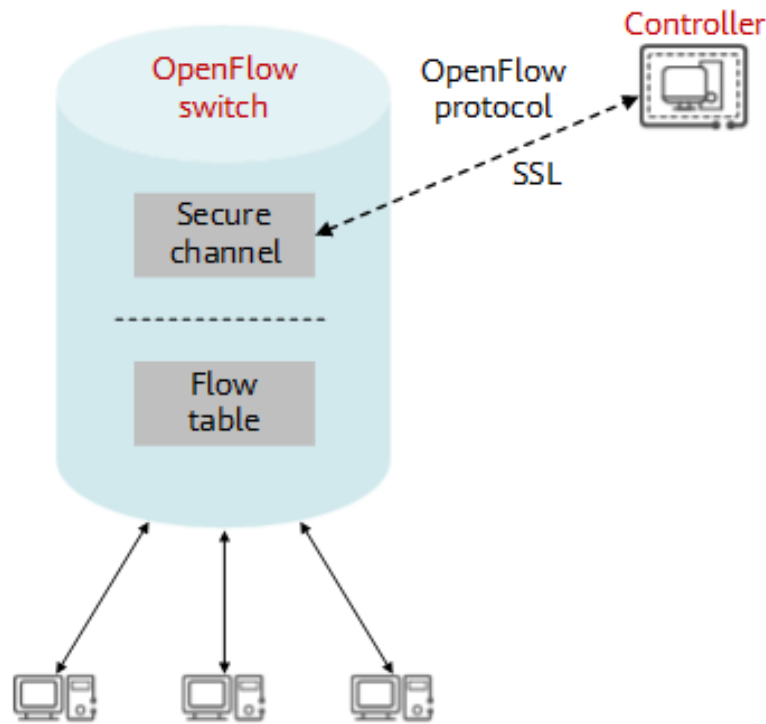


FIGURE 1.2 : OpenFlow architecture

1. OpenFlow Controllers :

An OpenFlow controller is the command center of SDN, it serves as the central intelligence of the SDN architecture, residing within the control layer to dictate data forwarding via the OpenFlow protocol. Presently, mainstream OpenFlow controllers fall into two primary categories : open-source as open daylight; RYU; ONOS, and vendor-developed commercial controllers as Huawei's iMaster NCE [5].

2. OpenFlow Secure Channel :

The OpenFlow Secure Channel is a fundamental component of Software-Defined Networking (SDN), ensuring secure communication between controllers and switches. This channel serves as a conduit for transmitting control messages and data flow instructions while guarding against potential security risks. It employs encryption protocols such as Transport Layer Security (TLS) to encrypt data transmission, thereby protecting against unauthorized access and tampering. The channel enables bidirectional communication, allowing controllers and OpenFlow switches to exchange control messages and status updates seamlessly. All messages transmitted over the channel adhere to the standardized format specified by the OpenFlow protocol, ensuring interoperability and compatibility across diverse network elements.

3. OpenFlow Switch :

As a pivotal element within the OpenFlow network framework, an OpenFlow switch primarily handles forwarding tasks at the data layer. These switches can manifest in either physical or virtualized forms, functioning as switches or routers. OpenFlow switches are categorized into distinct types based on their level of support for OpenFlow :

- **Dedicated OpenFlow Switch :** This variant represents a standard OpenFlow device dedicated solely to OpenFlow forwarding. It exclusively processes traffic in OpenFlow mode and lacks the capability to perform Layer 2 or Layer 3 forwarding.

- **OpenFlow-Compatible Switch :** In contrast, an OpenFlow-compatible switch extends support to both OpenFlow forwarding and traditional Layer 2/3 forwarding. Typically found in commercial settings, these switches offer features such as flow tables and secure channels while maintaining compatibility with conventional forwarding methods. Functionally, an OpenFlow switch directs incoming packets based on directives stored within its flow table. This table houses a collection of policy entries that dictate traffic processing rules. These entries are dynamically generated, managed, and disseminated by an external controller.

1.3.3 Flow Tables

The OpenFlow switches are composed of a pipeline of flow tables as shown in figure 1.3. The availability of multiple flow tables allows for the processing of packets in series. A flow in the table can have the action GOTO, which sends the packet to continue its matching functionality in another flow table. This new feature significantly increases the complexity of flows and the logic of matching to handle more complex scenarios. This allows for packet routing management, with the mechanism involving the accumulation of actions from each flow table to execute them at the pipeline's output.

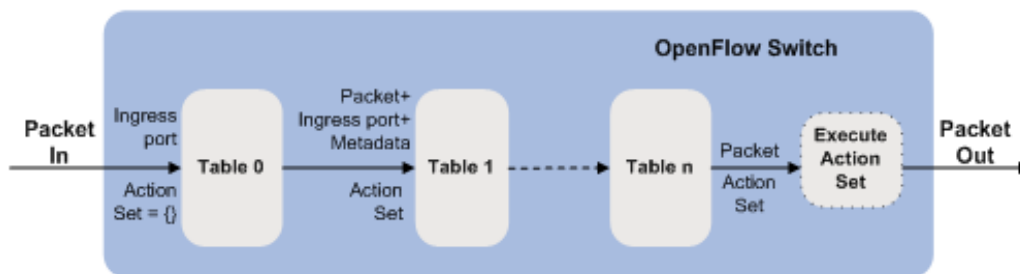


FIGURE 1.3 : OpenFlow forwarding workflow

- Flow tables in an OpenFlow switch are numbered sequentially, beginning at 0 as shown in figure 1.3.
- Packet matching starts with the first flow table, which is flow table 0.
- A flow entry can direct a packet only to a flow table with a higher number than its own.
- When a packet matches a flow entry, the OpenFlow switch updates its action set and moves the packet to the next flow table.
- In the final flow table, the switch executes all actions to modify packet contents and specify the output port for forwarding.
- If a flow table's instruction set includes an action list, the switch immediately executes those actions for a copy of the packet in that table.

Flow Entry in OpenFlow :

A flow entry is a set of rules defined in a network switch or router that determines how network traffic matching specific criteria should be handled. These rules typically include criteria such as source and destination IP addresses, ports, and protocols, along with associated actions.

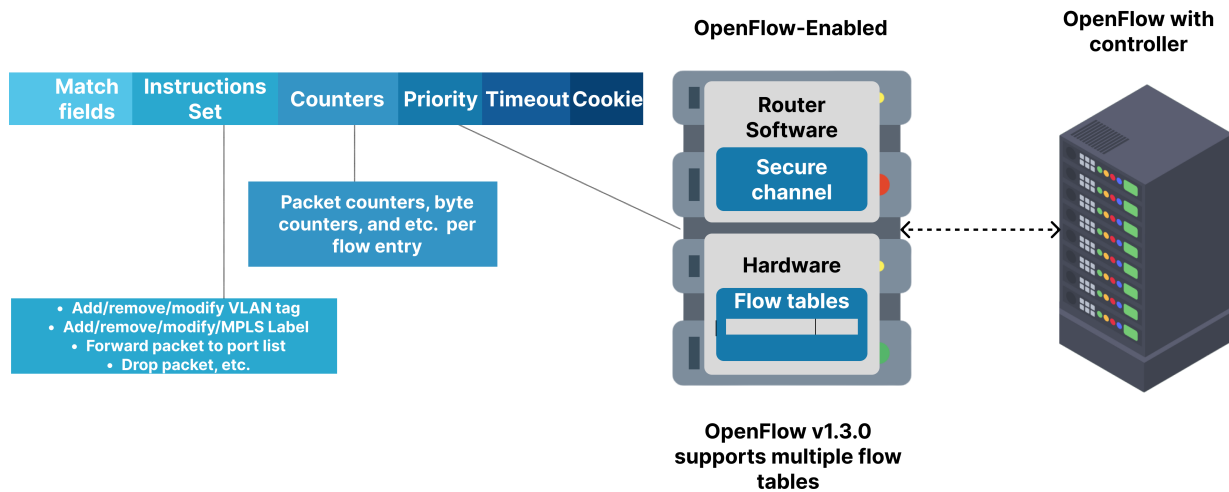


FIGURE 1.4 : The flow entry structure

The flow entry structure as illustrated in figure 1.4 is :

- **match fields** : to match against packets. On packet arrival, match its header fields with flow entries in a flow table, if any entry matches, update the counters indicated in that entry and perform indicated actions.
 - * **Packet Header** : Defines the data flow. The packet header includes various fields proposed by the protocol. Some of these fields are illustrated in Figure 1.5.

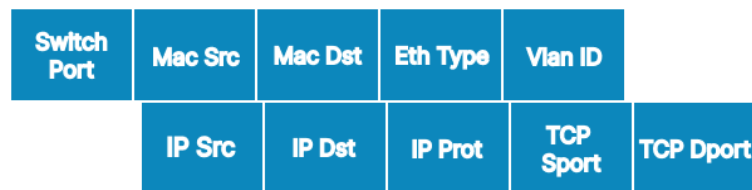


FIGURE 1.5 : Illustration of fields in the packet header.

- **Instructions set** : Specifies how packets within a flow will be processed. There are mandatory actions that every OpenFlow switch must support, as well as optional OpenFlow actions that an OpenFlow switch can support. Here are the mandatory actions :
 - * **Forward** : OpenFlow switches must support forwarding packets to physical ports as well as virtual ports.
 - * **Encapsulate and forward to controller** : Encapsulate and send the packet to a controller through the secure channel. This action is primarily used when a packet from a flow is received for the first time. The controller can decide whether a new entry should be added to the table, or if the packet should be dropped.
 - * **Drop** : Dropping all packets belonging to a flow is also an action that can be used for security reasons to halt an attack, for example.
- **Counters** : Reserved for collecting flow statistics. Counters can be maintained for each flow table, flow entry, port, and queue.

Flow table delivery :

Flow table delivery in OpenFlow can occur in proactive or reactive modes.

- * **Proactive mode** : the controller proactively delivers flow table information to OpenFlow switches.

- * **Reactive mode** : when an OpenFlow switch encounters a packet for which it has no matching flow entry, it sends a message to the controller. The controller then decides how to forward the packet, calculates the necessary flow entry, and delivers it to the switch. In this mode, the switch does not need to maintain all flow entries locally. Instead, it acquires flow entries from the controller as needed, storing them only for actively generated traffic, and removing them when they expire.

1.3.4 Openflow Messages

A secure communication link is established between an OpenFlow controller and a switch, enabling the controller to manage the switch and receive feedback. Messages transmitted over this secure channel must adhere to the format specified by the OpenFlow protocol. Typically encrypted using Transport Layer Security (TLS), the OpenFlow secure channel may also operate over plain-text TCP and port 6633 in OpenFlow versions 1.1 and later.

Three types of messages are exchanged over this channel :

- **Controller-to-Switch message** : is sent by the controller to the OpenFlow switch to manage or obtain the OpenFlow switch status.
- **Asynchronous message** : Sent by the OpenFlow switch to update the controller on network events or status changes.
- **Symmetric message** : Sent without solicitation by either the OpenFlow switch or the controller, primarily used for establishing a connection and detecting online status of the peer.

1.3.5 Openflow Functioning

The OpenFlow architecture is illustrated by the figure 1.6, which describes the process of packet transmission with OpenFlow. When a packet arrives at a switch, the switch checks if there is an entry in the flow table that matches the packet header.

Should a match be found, the switch promptly executes the associated action specified within the flow table as shown in figure 1.6. Conversely, if no matching entry is identified (1), the switch generates an asynchronous message to the controller (2), referred to as a 'Packet in' message. Subsequently, the controller, leveraging its preconfigured policies, deliberates on an appropriate action for the packet.

Following this decision-making process, the controller dispatches a new transmission rule to the switch in the form of both a 'Packet out' message and a 'Flow mod' message (3). Consequently, the switch's flow table is updated to integrate the newly installed rule, ensuring subsequent packets of similar characteristics are handled accordingly (4).

This iterative cycle of packet processing underscores the dynamic and adaptive nature of OpenFlow networks, wherein network behavior can be intelligently governed and adjusted in response to evolving traffic patterns and network conditions.

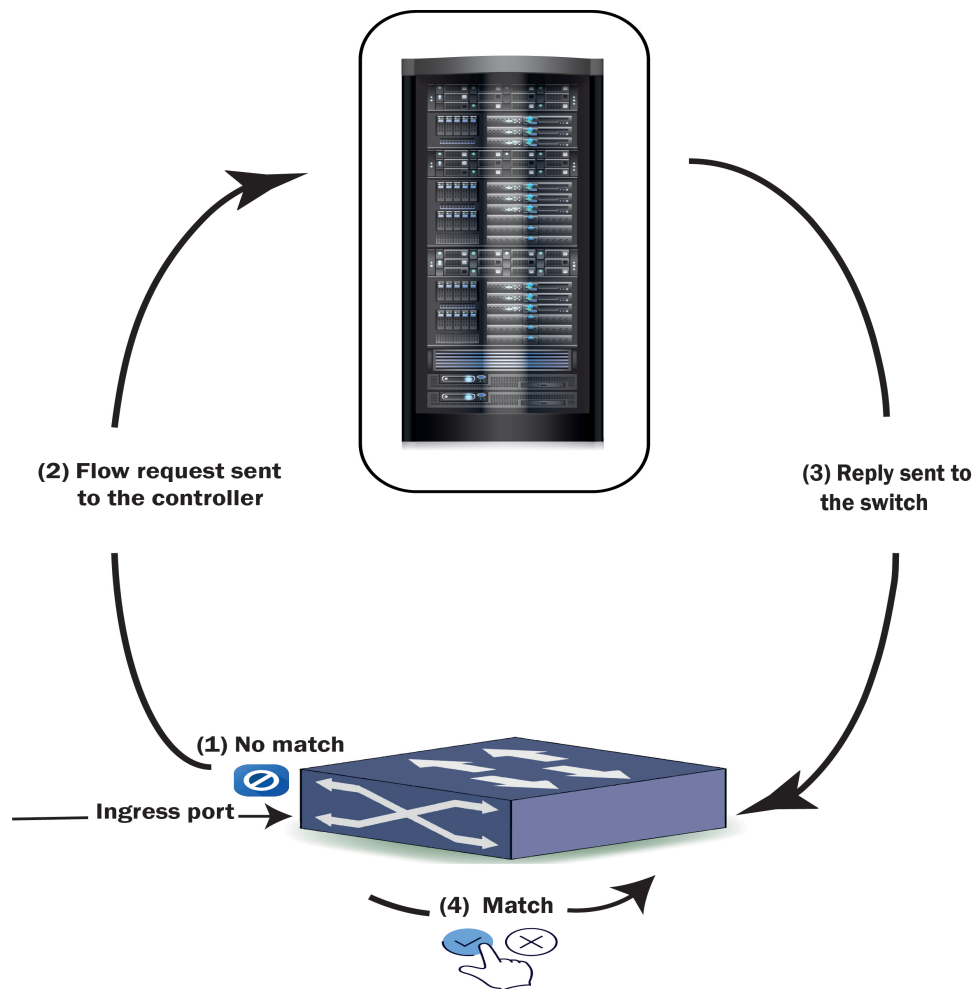


FIGURE 1.6 : The process of packet transmission with OpenFlow

1.4 SDN Benefits

1.4.1 Enhancing Configuration

Network configuration is a critical aspect of network management, particularly when integrating new equipment into an existing network. This process requires precise configurations to ensure smooth operations. However, due to the variety of network device manufacturers and configuration interfaces, current practices often involve manual intervention, which is both labor-intensive and error-prone.

SDN addresses these limitations by enabling automatic and dynamic network reconfiguration. It unifies the control plane across all network devices, including switches and routers, facilitating centralized configuration management. Through software control, network devices can be configured from a single point, allowing for programmable and dynamic optimization of the entire network based on its status [4].

1.4.2 Enhancing Performance

The arrival of SDN enhances the network performance on a global scale, by enabling centralized control with a Whole view of the network and allows feedback control by exchanging information across different layers of the network architecture. Consequently, numerous complex performance optimization challenges can be effectively addressed through well-designed centralized algorithms. This implies that innovative solutions for traditional issues like data traffic scheduling, end-to-end

congestion control, load-balanced packet routing, energy-efficient operations, and Quality of Service (QoS) support can be devised and seamlessly implemented to evaluate their impact on enhancing network performance [4].

1.4.3 Innovation

As network applications continue to evolve, future networks should prioritize fostering innovation rather than trying to precisely predict and meet the requirements of future applications. SDN promotes innovation by offering a programmable network platform that allows the implementation and experimentation of new ideas and applications in a convenient and flexible manner. The high configurability of SDN enables a clear separation among virtual networks, facilitating experimentation in a real environment. This setup allows for the progressive deployment of new ideas through a seamless transition from the experimental phase to the operational phase.

1.5 SDN Threats

SDN has revolutionized network management and operations by offering centralized control and high performance. However, this paradigm shift also introduces some challenges, particularly in terms of security vulnerabilities. Specifically, the centralized control plane of SDN that is responsible for network management is compromised to various malicious attacks due to its critical role in network operations, which may cause single point failure [10]. Additionally, the connected hosts in the data plane within the SDN network are not immune as well, as servers located in this plane are also vulnerable to exploitation. Among the myriad of threats that SDN networks face, DDoS attacks stand out as one of the most dangerous and frequent. They are aimed at disrupting network services and undermining systems integrity by overwhelming the target with a flood of malicious traffic, rendering it inaccessible to legitimate users [11].

1.5.1 DDoS Attacks

Denial of Service (DoS) attacks occur when a single attacker sends a massive amount of traffic to a target, such as a website or a network controller, to overwhelm its resources and make it inaccessible to legitimate users. This flood of traffic disrupts the normal functioning of the target, as depicted in Figure 1.7.

Distributed Denial of Service (DDoS) attacks take this concept further. In a DDoS attack, hackers take control of multiple computers, often referred to as a botnet, and use them collectively to send an even larger volume of traffic to the target. This flood of traffic is intended to disrupt the normal functioning of the target, making its services or resources inaccessible to legitimate users. DDoS attack has a devastating effect on SDN, rendering the entire network or specific sections unavailable by incapacitating the controller or disabling switches. The risk on SDN has heightened due to its reliance on centralized control and flow-tables [12]. Furthermore, they pose a significant challenge in terms of detection and effective mitigation.

Various classifications of this type of attacks are provided, some based on the attack rate and other based on the protocol used as presented in figure 1.8. In the subsequent sections, we will delve into these classifications of DDoS attacks targeting SDN infrastructures.

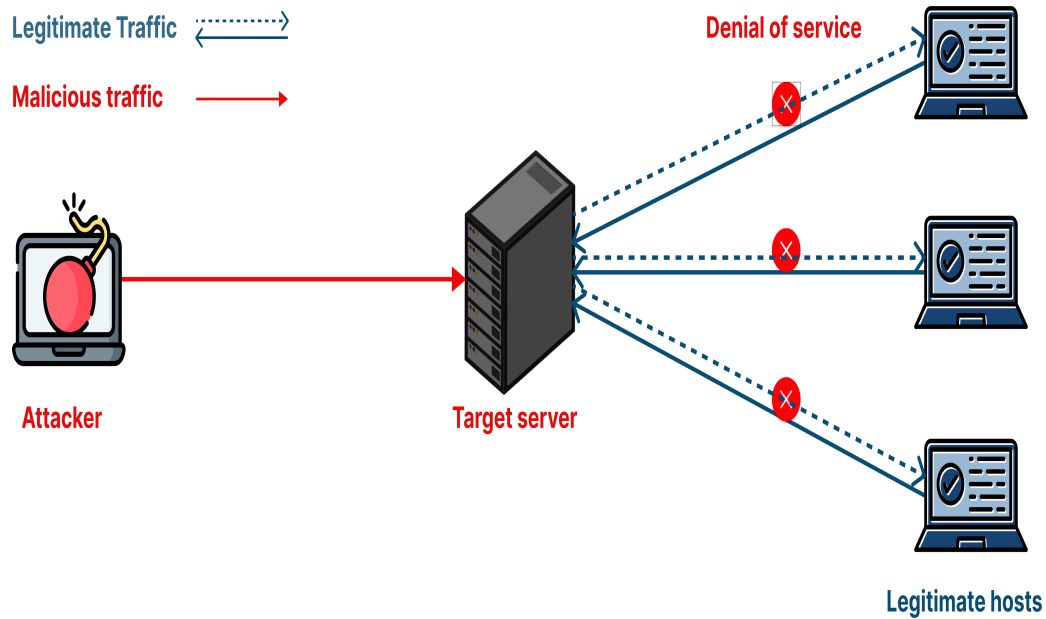


FIGURE 1.7 : DoS attack

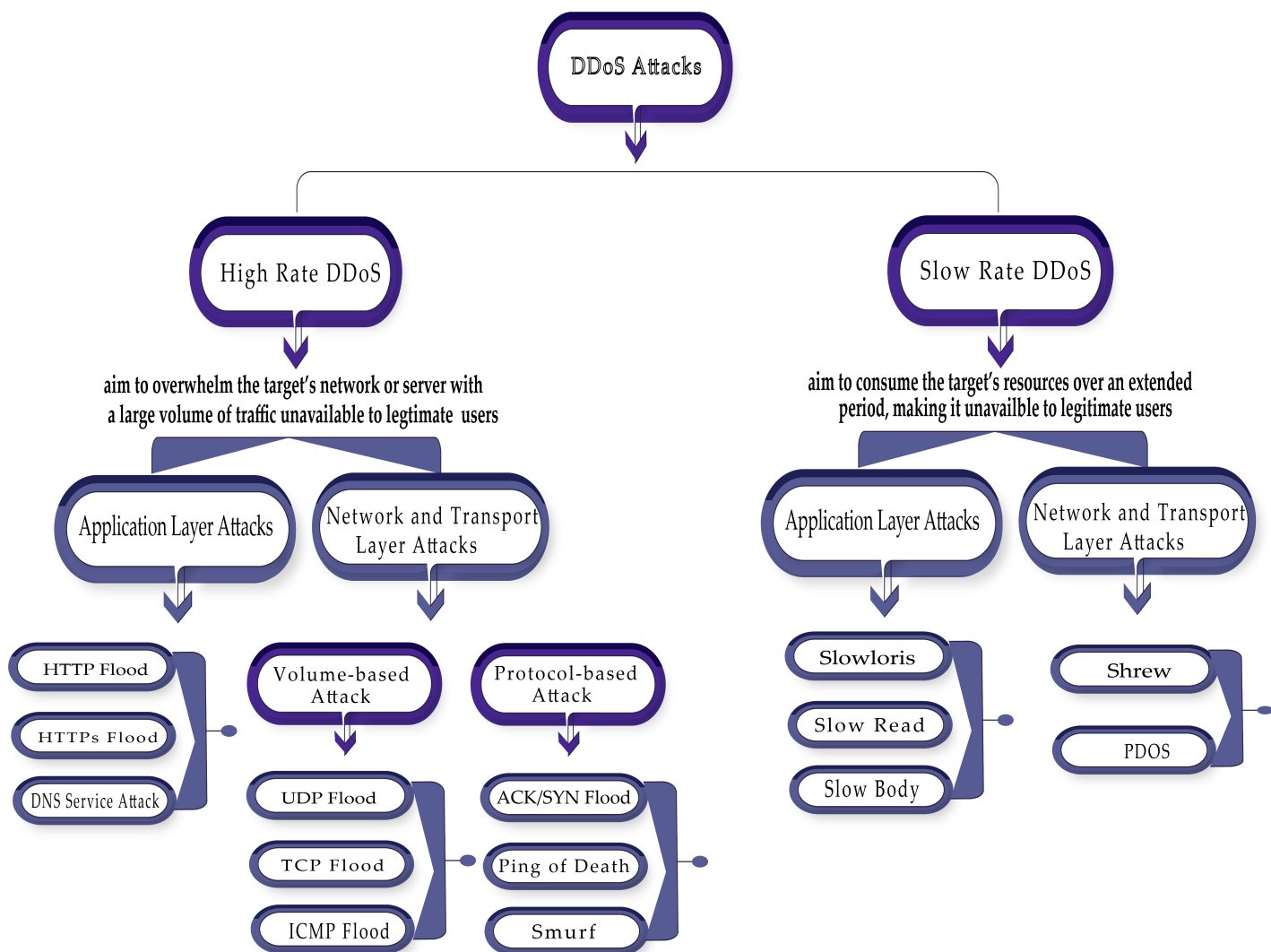


FIGURE 1.8 : DDoS attack types taxonomy

1. **High-rate DDoS:** The term "high-rate" implies that the attack generates a significant amount of data or requests per unit of time, overwhelming the target's capacity to handle incoming traffic and causing disruption or service degradation [13]. DDoS attacks on SDN have the capability to disrupt the availability of network services by aiming at various layers within SDN infrastructure, including the application, network and transport layers [14]. As it's been demonstrated in figure 1.8, High-rate DDoS attacks manifest in two main categories, each targeting specific layers of network infrastructure :

Application Layer Attacks :

The aim of application layer DDoS is to attack the application itself by targeting particular vulnerabilities or problems which make it unable to deliver content to the user. These kinds of attacks are meant for specific applications.

- **HTTP Flood :** Utilizes what appear to be legitimate HTTP GET or POST requests to attack a web server or application, And overwhelm it with a barrage of HTTP requests ("flooding" it with as many process-intensive requests as possible)[13, 15].
- **HTTPs Flood :** Similar to HTTP flood but targeting encrypted HTTPS connections, making mitigation more challenging [15].
- **DNS Service Attack :** This is an instance of a Reflection Amplification attack, where the attacker impersonates their IP address to send an information request. Subsequently, the server responds to this request, sending data to the target's IP address. Similarly, DNS Attacks target Domain Name System servers, disrupting the translation of domain names into IP addresses, thereby impeding access to websites and services.

Network and Transport Layer Attacks :

Attacks targeting the transport and network layers in SDN are often referred to as infrastructure layer attacks [16]. The network layer handles the routing and forwarding of data packets between various networks. DDoS attacks targeting this layer focus on exhausting network bandwidth resources. Meanwhile, the transport layer is tasked with transferring data across a network, offering error-checking mechanisms, and managing data flow [17].

- a. **Volume-based Attack** volume-based attacks aim to flood the target network or system with a high volume of traffic, thereby exhausting available resources.

- **UDP Flood :** This assault is recognized as a sessionless networking protocol. Here, the attacker inundates the target with numerous UDP packets across various random ports. This inundation prompts the host to continuously verify application listening ports, ultimately responding with ICMP Destination Unreachable packets. This cycle renders the target host's resources inaccessible [18].
- **TCP Flood :** In this type of attack, the assailant sends a flood of TCP SYN packets to the target server, often spoofing the source IP addresses to make it difficult to trace the origin of the attack. Then , the server keep waiting for the ACK packet to complete the TCP Handshake and establish connections. Thus, keeps the half-open connections in its memory, consuming resources as network bandwidth.
- **ICMP Flood :** Similar to the UDP flood attack, an ICMP flood Overload the target host's resources with ICMP Echo Request (ping) packets. These packets are sent rapidly, without waiting for replies. Therefore, this attack can saturate both incoming and outgoing bandwidth. The target often attempt to respond with ICMP Echo Reply packets, leading to a notable slowdown in overall system performance [18]

- b. **Protocol-based Attack**

- **SYN Flood :** It stands as one of the original DDoS attacks online, focusing on the TCP three-way handshake method. Typically, the attacker utilizes commandeered zombie hosts to Overwhelm the server with a barrage of TCP/SYN packets, varying source IP addresses or ports. This inundation triggers the creation of numerous half-open connections upon the server's response, ultimately exhausting system resources and rendering normal services unattainable [16, 13].
- **Ping of Death :** Entails the transmission of a malicious ping to a targeted computer. Typically, this ping is 32 bytes in size. The assailant then sends a packet exceeding the IP protocol's limit. Thus, this oversized or malformed packet exploits vulnerabilities in the target's networking stack, and impacts its resources. The "ping of death" attack aims to disrupt or disable the target system by overwhelming it with malicious ping packets [19].
- **Smurf Attack :** In a Smurf attack, the victim is inundated with ICMP echo-reply packets. The attacker sends packets with a spoofed source IP address targeted to the victim. Because these packets are sent to the broadcast address, they are received by all nodes within the network. Each node then responds back to the victim machine, as the source IP address is spoofed to appear as that of the victim's address [20].

2. Slow-rate DDoS :

Slow DDoS attacks operate Stealthily, gradually draining resources over an extended duration to avoid detection [21], they are characterized by employing Seemingly legitimate traffic at a slow pace, this type of DDoS leverage application layer protocols like HTTP to gradually overwhelm servers.

Dissimilar to high-rate DDoS, slow-rate DDoS attacks require minimal resources for execution, often originating from just one machine. These distinctions render slow-rate attacks more intricate and challenging to detect [22]. Slow DDoS attacks manifest in various forms, categorized according to the protocol they exploit. Specifically, they can be broadly classified into application layer attacks and network/transport layers attacks, each encompassing several subtypes.

Application layer attacks :

- **Slowloris Attack :** it operates by sending partial HTTP requests to establish connections between a single computer and a victim web server. Typically, HTTP GET requests are sent with valid HTTP headers, which the web server processes upon receipt of the complete header. Attackers exploit this behavior to execute DDoS attacks on the victim server. So they send incomplete HTTP headers, and the server continues to wait for the complete message. The attacker inundates the server with numerous such incomplete requests then maintains these connections open for as long as feasible, until it becomes overwhelmed and unable to process any further requests [23].
- **Slow Read Attack :** In a slow-read attack, the assailant transmits a legitimate HTTP request to a web server but then proceeds to read the response at an extremely low speed, if at all. By deliberately slowing down the reading process, sometimes to as little as one byte per instance, the attacker forbids the server from acquiring an idle connection timeout [23].
- **Slow Body Attack :** is another type of low-bandwidth denial-of-service (DoS) attack, similar to Slowloris. In a slow body attack, the attacker sends an HTTP POST request to a web server, but transmits the body of the request very slowly. This can exhaust server resources, as the server keeps the connection open waiting for the complete body of the request [23].

Network and Transport Layer Attacks :

- **PDoS :** Permanent Denial of Service attack entails denial of service through hardware sabotage. One prevalent method for executing a PDoS attack is known as "phlashing", where

the attacker renders a device inoperable or damages its firmware, making the device or entire system unusable. This approach exploits vulnerabilities by replacing the device's fundamental software with a corrupt firmware image.

- **Shrew attack**: Shrew attack is a low-rate DoS attack that exploits the vulnerabilities in the TCP protocol, it operates by sending small bursts of network traffic, yet still succeeds in bringing down a host.

1.6 Cyber Security

Cyber Security encompasses a broad array of technologies, processes, and practices aimed at protecting networks, devices, programs, and data from unauthorized access or damage. It is a critical aspect of the modern digital landscape, where the integrity, confidentiality, and availability of information are paramount. Its strategies include measures to combat a range of threats such as DDoS attacks.

In the realm of SDN, Cyber Security plays a crucial role in securing dynamic and programmable network environments. Specifically, its architecture supports advanced security mechanisms and provides a platform for implementing robust Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs).

1.6.1 Intrusion Detection Systems (IDSs)

IDSs are critical components in the Cyber Security arsenal. They are designed to monitor network traffic and system activities for signs of malicious behavior, policy violations, or suspicious activities. The primary goal of an IDS is to detect and alert on potential security incidents, which can then be investigated and mitigated by security personnel and sophisticated approaches.

In the context of SDN, IDSs can be highly effective due to the centralized control and visibility provided by the SDN controller. SDN-enabled IDSs can dynamically adapt to network changes and provide real-time monitoring and analysis of traffic patterns. This allows for the detection of various types of cyber threats, including DDoS attacks, by identifying unusual spikes in traffic or patterns that deviate from the norm. The programmable nature of SDN facilitates the deployment of IDSs across multiple network segments, ensuring comprehensive coverage and rapid threat detection [22].

1.6.2 Intrusion Prevention Systems (IPSs)

IPSs take the capabilities of IDSs a step further by not only detecting but also actively blocking or mitigating threats. They are designed to automatically respond to detected security incidents, such as DDoS attacks, by dropping malicious packets, blocking IP addresses, or throttling traffic to prevent network disruption.

In an SDN environment, IPSs can leverage the programmability and flexibility of the network to enforce security policies in real-time. The SDN controller can deploy IPSs rules dynamically, redirect traffic for analysis, and isolate affected segments to contain and mitigate threats. This real-time responsiveness is crucial for combating DDoS attacks, which aim to overwhelm network resources and disrupt services. By leveraging SDN's centralized management, IPSs can quickly identify and neutralize malicious traffic, ensuring that the network remains secure and operational [22].

1.7 Conclusion

In this chapter, we explored the transformative potential of Software-Defined Networking (SDN) in making networks more intelligent and manageable. SDN offers numerous advantages, including improved performance, streamlined configuration, and greater flexibility. However, alongside these benefits come significant security challenges, particularly from Distributed Denial of Service (DDoS)

attacks. As we strive to leverage the full potential of SDN, addressing these vulnerabilities becomes crucial to ensuring robust network security and maintaining the integrity and reliability of network services.

These attacks can significantly slow down or even crash networks by overwhelming them with excessive traffic. Therefore, it is imperative to build Intrusion Detection Systems (IDSs) using advanced technologies, particularly machine learning and deep learning. The next chapter will delve into these methods and their application in enhancing network security.

Chapter 2

Deep Learning

2.1 Introduction

Deep learning is a subfield of machine learning that aims at making machines capable of performing tasks by learning from data, with the use of artificial neural networks (ANNs) which are designed to mimic the perceptual processes of the human brain [24].

ANNs consist of interconnected units called neurons, which are inspired by the neurons in the human brain. Each neuron in an ANN processes input data by performing a simple computation : it applies linear regression, a mathematical operation that predicts a value based on input data. Followed by an activation function that determines whether the neuron should be activated, influencing the neuron's output. In deep learning, multiple neural nodes are utilized, in contrast to the single node typically found in traditional statistical machine learning.

This chapter introduces concepts and techniques related to deep learning, with a focus on three key types of neural networks : Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks. We will delve into their functioning and applications, providing a comprehensive understanding of how these networks operate and their significance in the field of deep learning.

2.2 Learning Processes in Deep Learning

Learning in DL refers to the iterative process through which a neural network adjusts its weights and biases based on the input data to improve its performance in predicting or classifying data points. This process is fundamental to the effectiveness of deep learning models and involves several critical steps.

2.2.1 Data Collection and Preparation

The foundation of any deep learning model is high-quality data. The data must be representative of the problem domain to ensure that the model can generalize well to new, unseen data. This preparation includes :

- **Data Collection** : Gathering raw data from various sources relevant to the task at hand. This can include images, text, numerical data, or any other type of input data.
- **Data Cleaning** : Removing noise and Divergences from the dataset, such as handling missing values, correcting errors, and removing outliers that could negatively impact model performance.
- **Data Preprocessing** : Transforming raw data into a format suitable for training. This includes normalization or standardization of numerical data, vectorization of text data, and resizing or augmenting image data.

- **Data Splitting** : Dividing the dataset into three subsets : training set (used to train the model), validation set (used to tune hyperparameters and prevent overfitting), and test set (used to evaluate the final model performance).

2.2.2 Model Initialization

Before the learning process begins, the neural network model is initialized with random weights and biases. This randomness helps in breaking symmetry and allows the model to learn diverse features during training. Proper initialization can significantly impact the speed and success of the training process.

2.2.3 Forward Propagation

During forward propagation, the input data is passed through the network layers, starting from the input layer, through the hidden layers, and finally to the output layer. Each layer performs a linear transformation followed by a non-linear activation function, transforming the input step by step. The final output is produced, which can be a probability distribution in classification tasks or a continuous value in regression tasks.

2.2.4 Loss Calculation

The loss function, also known as the cost function or error function, measures the difference between the predicted output and the actual target values. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks. The loss value provides a quantitative measure of the model's performance.

2.2.5 Backward Propagation

Backward propagation, or backpropagation, is the process of updating the model's weights to minimize the loss. This involves calculating the gradient of the loss function with respect to each weight using the chain rule of calculus. The gradients indicate the direction and magnitude of the changes needed to reduce the loss.

2.2.6 Optimization Algorithm

An optimization algorithm uses the gradients computed during backpropagation to update the model's weights. Some common optimization algorithms include :

- **Stochastic Gradient Descent (SGD)** : Updates weights using a random subset of the data (mini-batch) at each iteration, which makes it computationally efficient and helps escape local minima.
- **Momentum** : Enhances SGD by adding a fraction of the previous update to the current update, which helps in navigating ravines and speeds up convergence.
- **RMSprop (Root Mean Square Propagation)** : Adapts the learning rate for each parameter by dividing the learning rate by an exponentially decaying average of squared gradients, which helps in dealing with the vanishing gradient problem.
- **Adam (Adaptive Moment Estimation)** : Combines the advantages of both Momentum and RMSProp by maintaining an adaptive learning rate for each parameter, making it well-suited for problems with sparse gradients and non-stationary objectives.

2.3 Overlearning

Overlearning, commonly known as overfitting, occurs when a model becomes too closely aligned with the training data, capturing noise and outliers rather than general patterns. This results in an excellent performance on the training set but poor generalization to new, unseen data. Understanding and mitigating overfitting is crucial for building robust deep learning models.

2.3.1 Symptoms of Overfitting

The primary symptoms of overfitting include :

- **High Training Accuracy, Low Validation/Test Accuracy** : The model shows very high accuracy on the training data but significantly lower accuracy on the validation or test data.
- **Large Generalization Gap** : A considerable difference between training and validation/test performance indicates that the model is not generalizing well.

2.3.2 Causes of Overfitting

Overfitting can be caused by several factors :

- **Model Complexity** : Models with too many layers or parameters can learn intricate details and noise from the training data.
- **Insufficient Training Data** : A small training dataset may lead the model to memorize the data instead of learning generalizable patterns.
- **Too Many Training Epochs** : Prolonged training can cause the model to adapt too closely to the training data, losing its ability to generalize.

2.3.3 Techniques to Prevent Overfitting

Various techniques can help prevent overfitting and enhance the model's generalization capabilities :

- **Regularization** : Adds a penalty for large weights, encouraging the model to maintain smaller, more generalized weights.
 - * **L1 Regularization** (Lasso) : Adds the absolute value of the magnitude of the weights as a penalty term to the loss function.
 - * **L2 Regularization** (Ridge) : Adds the squared magnitude of the weights as a penalty term to the loss function.
- **Dropout** : Randomly drops a fraction of neurons during each training iteration, preventing the model from relying too heavily on specific neurons and encouraging redundancy.
- **Early Stopping** : Monitors the model's performance on a validation set and stops training when performance begins to degrade, preventing the model from overfitting to the training data.
- **Data Augmentation** : Increases the diversity of the training dataset by applying random transformations (rotation, scaling, flipping) to the input data. This helps the model learn more robust features and improves generalization.

2.4 Artificial Neural Networks (ANNs)

An Artificial Neural Network (ANN) is a computational model inspired by the human brain's neural networks and is the foundation of deep learning. It consists of interconnected nodes, or neurons, organized in layers : input, hidden, and output. Each neuron processes inputs through weighted connections, applies an activation function, and passes the output to the next layer. A common type of ANN is the Multilayer Perceptron (MLP), which uses multiple hidden layers. By adjusting the weights through learning algorithms like backpropagation, ANNs can recognize patterns and make predictions based on data. Over time, various architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been developed to address specific tasks, enhancing the capabilities of ANNs in areas such as image reconstruction [25] and natural language processing [26].

2.5 Convolutional Neural Networks (CNNs)

The Convolutional Neural Network (CNN) stands out as one of the most utilized techniques in Deep Learning [27], it belongs to a category of ANNs extensively applied across various domains such as radiology [28], image reconstruction [25], and classification [29]. Typically, CNNs excel in processing data structured in a grid pattern, such as images, owing to their ability to autonomously and flexibly determine spatial hierarchies of features. As depicted in figure 2.1, CNNs consist of three main building blocks [30] namely convolution, pooling, and fully connected layers, each of these blocks will be elaborated on in the subsequent subsections.

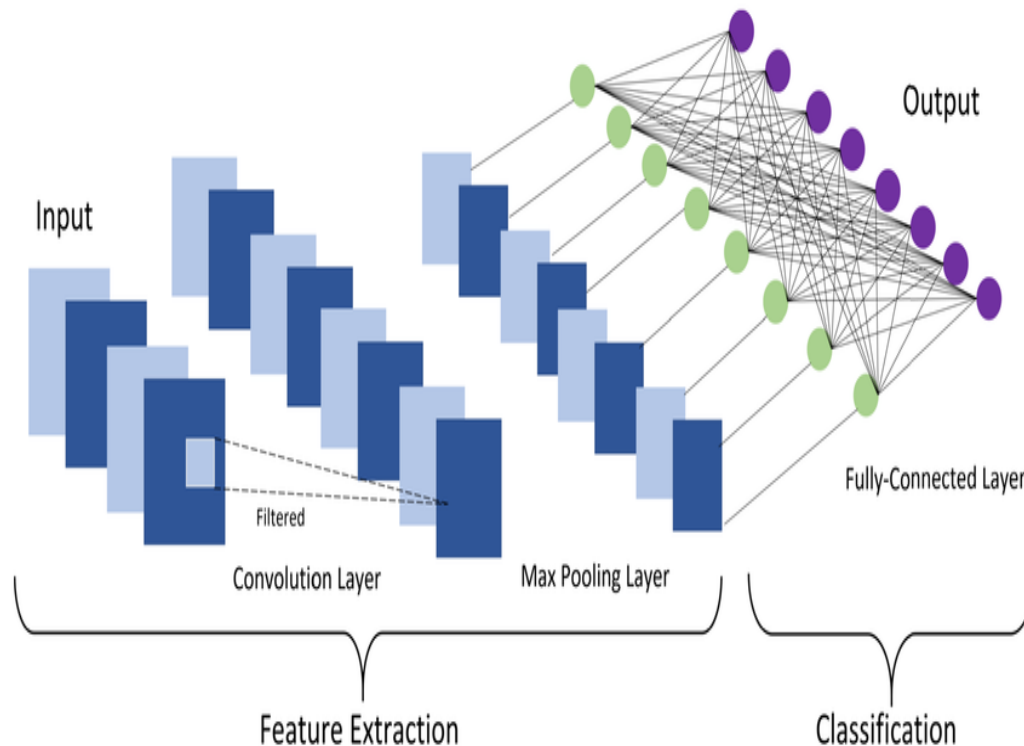


FIGURE 2.1 : CNN architecture [1]

2.5.1 Convolution Layer

The convolution layer based on weight sharing [31] serves as a cornerstone within the CNN architecture that performs feature extraction. This process typically encompasses a blend of linear and

nonlinear operations, namely the convolution operation and activation function. Convolution represents a specialized linear operation for feature extraction, it employs a small array of numbers known as a kernel or feature extractor, which is applied across the input, termed a tensor. The convolution operation computes an element-wise product between each element of the kernel and the input tensor at each location of the tensor and sum these products, this sum gives the output value in the related position of the output tensor, referred to as a feature map [31].

This operation is iterated using numerous kernels to generate an array of feature maps, each representing distinct characteristics of the input tensors. Additionally, key hyperparameters defining the convolution operation include the size and number of kernels. Also, zero padding technique is applied to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation. Summarily, during the training process in the convolutional layer, kernels are the sole parameters that are automatically learned. But, hyperparameters such as kernel size, number, padding, and stride that represents the distance between two successive kernel positions, must be predetermined before the commencement of training.

The results of the linear operation are subsequently fed through a nonlinear activation function, the prevailing choice for nonlinear activation function nowadays is the rectified linear unit (ReLU).

2.5.2 Pooling Layer

It performs a standard downsampling operation, aiming to decrease the in-plane dimensionality of the feature maps, this reduction introduces a degree of translation invariance to minor shifts and distortions while also reducing the number of learnable parameters. Notably, hyperparameters including filter size, stride, and padding are considered in pooling operations, much like in convolution operations [31]. One of the most favored forms of pooling operation is Max pooling[30] that extracts patches from the input feature maps, retaining only the maximum value within each patch and discarding the rest. Remarkably, the depth dimension of feature maps remains unaltered through this process.

Another noteworthy pooling operation is global average pooling. This method executes an extreme downsampling by transforming a feature map with dimensions height $\times$ width into a 1×1 array. It achieves this by computing the average of all elements within each feature map, while preserving the depth dimension.

2.5.3 Fully Connected Layer

Feature maps from the preceding convolutional or pooling layer are typically flattened into a one-dimensional array or vector. This vector is then fed into one or more fully connected layers, also known as dense layers [32]. In these layers, every input is linked to every output through learnable weights, also, each fully connected layer is accompanied by a non-linear activation function, such as ReLU. The activation function employed in the last fully connected layer typically differs from those used in preceding layers, its selection depends on the specific task at hand. Various activation functions are used including softmax, sigmoid, and tanh [33].

2.6 Recurrent Neural Network (RNN)

RNN is an artificial neural network designed for handling sequential or time series data. It is a deep learning structure that uses past information to improve the performance of the network on current and future inputs.

These sophisticated algorithms excel in tasks involving order or time, like language translation, natural language processing, and speech recognition. They power popular applications like Siri, voice search, and Google Translate. Just like forward-propagating CNNs, recurrent neural networks learn

from training data to enhance their performance. What sets them apart is their "memory" capability, as they leverage information from past inputs to influence both current input and output. The looping structure allows the network to store past information in the hidden state and operate on sequences.

2.6.1 RNN Structure

The RNN operates by taking an input vector X , scanning the data sequentially from left to right, and generating an output vector y . Throughout this process, at each time step, the hidden state is updated and an output is produced. What distinguishes RNNs is their parameter sharing across all time steps. This means that a consistent set of parameters (represented by U , V , W) is utilized throughout the network. Specifically, U governs the connection from the input layer X to the hidden layer h , W represents the connections between hidden layers, and V corresponds to the connection from the hidden layer h to the output layer y . This parameter sharing enables RNNs to effectively capture temporal dependencies and process sequential data more efficiently by retaining information from previous inputs in the current hidden state [34].

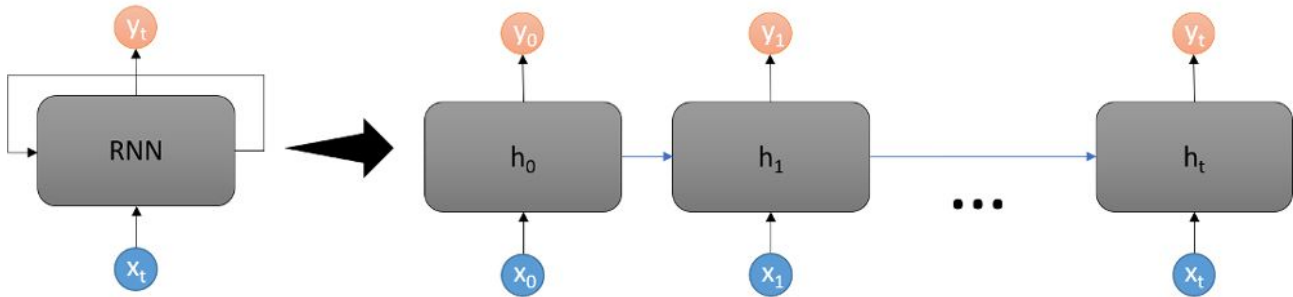


FIGURE 2.2 : Unfolded RNN cell

At each time step t , the hidden state a_t is computed based on the current input x_t :

$$a_t = f(U \cdot X_t + W \cdot a_{t-1} + b)$$

Where,

- a_t represents the output generated from the hidden layer at time step t .
- x_t is the input at time step t .
- U is the weight matrix governing the connections from the input to the hidden layer.
- W is the weight matrix governing the connections from the hidden layer to itself (recurrent connections).
- a_{t-1} is the output from the hidden layer at time $t - 1$.
- b is the bias vector for the hidden layer.
- f is the activation function.

The output at each time step t , denoted as y_t , is computed based on the hidden state output a_t using the following formula :

$$y_t = f(V \cdot a_t + c)$$

where,

- V is the weight matrix governing the connections from the hidden layer to the output layer.
- c is the bias vector for the output layer.

2.6.2 RNN Limitation

During backpropagation, gradients may diminish significantly, causing the vanishing gradient problem, or amplify excessively, leading to the exploding gradient problem as they propagate backward through time. In the vanishing gradient scenario, the gradient becomes too tiny, making it hard for the network to capture long-term relationships effectively. This might prolong the training process significantly. Conversely, the exploding gradient problem involves large gradients, which can cause numerical instability during training. This instability can lead to a loss of information over long sequences, impacting the model's ability to learn and make accurate predictions.

2.6.3 Long Short Term Memory (LSTM)

This is a popular RNN architecture, which was designed and introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem [35]. LSTM is well-suited for sequence prediction tasks and excels in capturing long-term dependencies. Its key strength, lies in its capability to understand the crucial order dependence required for solving complex problems like machine translation and speech recognition.

In contrast to traditional RNNs, which struggle with learning long-term dependencies due to a single hidden state passed through time, LSTMs resolve this issue by incorporating a memory cell. This memory cell acts as a container capable of holding information over extended periods. As a result, LSTM networks excel in understanding, long-term dependencies within sequential data.

Architecture and Working of LSTM :

The LSTM architecture comprises memory blocks within the recurrent hidden layer. These memory blocks house memory cells which allows the network to store and access information over an extended period of time. Additionally, These units have special parts called gates that help control and regulate information flow [36].

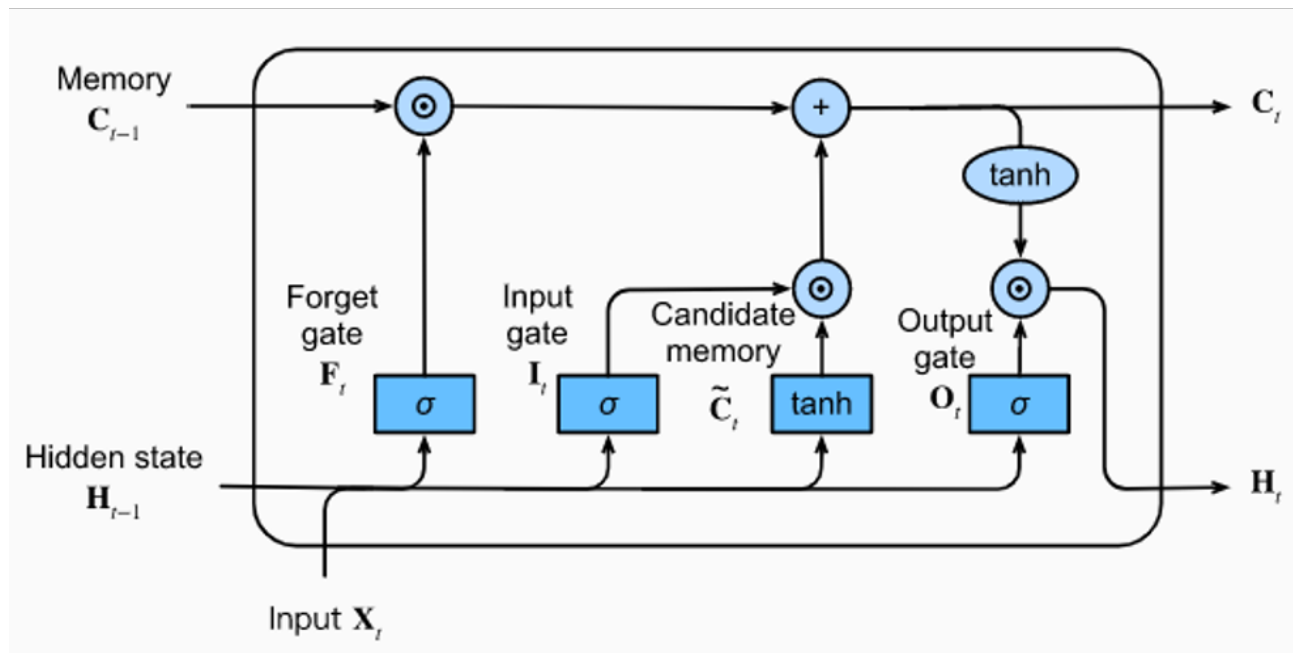


FIGURE 2.3 : Diagram of a memory cell of an LSTM [2]

The memory cell of an LSTM is composed of several gates : an input gate, an output gate and a forget gate. These gates manage how information flows within the memory cell, helping decide what to remember and

what to forget. This feature enables LSTMs to retain crucial details over lengthy sequences while disregarding less important ones. While in regular RNNs, h_t denotes the hidden state, in LSTM networks, we introduce a second state, c_t . Here, h_t signifies the neuron's short-term memory, while c_t represents its long-term memory [37].

- **Input gate :** In an LSTM, the input gate decides which new information to store in the memory cell. It uses the current input x_t and the previous state h_{t-1} to create an activation vector i_t that indicates the importance of the new information.
- **Forget gate :** The forget gate removes unnecessary information from the memory cell. It uses the current input x_t and the previous state h_{t-1} to produce an activation vector f_t , which determines what should be forgotten.
- **Output gate :** The output gate controls the LSTM's output at a given time. It uses the current input x_t and the current state of the memory cell c_t to generate an activation vector o_t , representing the LSTM's output.

2.7 Evaluation Metrics

The performance of the models was evaluated using several metrics, including confusion matrix, accuracy, precision, recall and F1 score. These metrics provide a comprehensive understanding of the DL models effectiveness.

2.7.1 Confusion Matrix

A confusion matrix is a table that describes the performance of a classification model by showing the actual versus predicted classifications. It includes :

- True Positives (TP) : Correctly predicted positive instances.
- True Negatives (TN) : Correctly predicted negative instances.
- False Positives (FP) : Incorrectly predicted positive instances.
- False Negatives (FN) : Incorrectly predicted negative instances.

The confusion matrix helps in understanding the types of errors the model is making and is crucial for calculating other metrics.

2.7.2 Accuracy

Accuracy measures the proportion of correctly classified instances out of the total instances. It is calculated as :

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

It gives an overall effectiveness of the model but can be misleading in cases of imbalanced datasets.

2.7.3 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is calculated as :

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

High precision indicates a low false positive rate.

2.7.4 Recall

Recall, also known as sensitivity or true positive rate, is the ratio of correctly predicted positive observations to all the observations in the actual class. It is calculated as :

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

High recall indicates a low false negative rate.

2.7.5 F1 Score

The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. It is calculated as :

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

It is useful when you need a balance between precision and recall.

2.8 Conclusion

In conclusion, CNNs excel in capturing spatial features from network data, RNNs with LSTM are adept at handling temporal dependencies. These architectural strengths make them well-suited for building an Intrusion Detection System (IDS) against application layer DDoS attacks. Integrating CNNs and RNNs/LSTMs into IDS frameworks promises robust defense mechanisms against evolving attack methods. Despite challenges in interpretability and scalability, ongoing research aims to refine these technologies for more effective cybersecurity solutions in the future.

Chapter 3

Methodology

3.1 Introduction

This chapter delineates the comprehensive methodology adopted for developing and evaluating an IDS aimed at detecting DDoS attacks in SDN environments using DL techniques. It begins with an overview of the project, detailing the primary objectives and the scope of the IDS system. Following this, we describe the dataset utilized for training and testing the DL models, highlighting its source, characteristics, and pre-processing steps. The rationale behind selecting specific DL models is provided, considering factors such as model complexity, training efficiency, and performance metrics.

Subsequently, we outline the process of training and evaluating these selected models, including the tuning of hyperparameters to optimize their performance. A detailed breakdown of the open-source software tools used for the development and testing of our project is provided in Appendix 4.7. The results from various models evaluations are discussed, comparing their effectiveness in detecting DDoS attacks. Furthermore, the chapter explains the development of a real-time DDoS detection and mitigation framework, integrating the trained models into a practical SDN environment.

Finally, the implementation details of the IDS system are presented, covering the technical aspects and challenges encountered during the deployment phase.

3.2 Project Overview : IDS for DDoS Attacks in SDN using Deep Learning

The primary goal of our IDS project is to enhance the security of SDN environments by developing an advanced approach capable of detecting DDoS attacks as shown in figure 3.1. Our system focuses on identifying threats targeting the SDN controller or any host within the network and mitigating the impact of these attacks by blocking the attacker's IP address, addressing the single point of failure inherent in SDN architectures.

We began by researching and identifying the most suitable dataset for training and evaluating our DL models, focusing on datasets containing a variety of network traffic data, including both benign and DDoS attacks patterns. Once the dataset was selected, we generated additional data to enhance and update the existing dataset, augmenting the network traffic samples to create a more robust training resource. We then cleaned and preprocessed the combined dataset to ensure that it was correctly formatted. Afterward, we investigated the DL algorithms that have demonstrated the highest efficacy in detecting DDoS attacks, such as CNN and LSTM. Using our preprocessed and augmented dataset, we trained various DL models, splitting the data into training and testing sets to evaluate their performance.

To assess our DL-based models, we used metrics such as accuracy, precision, recall, and F1-score, providing insights into the effectiveness of the models in correctly identifying DDoS attacks and minimizing false positives. We conducted hyperparameters tuning by adjusting parameters like the optimizer and the number

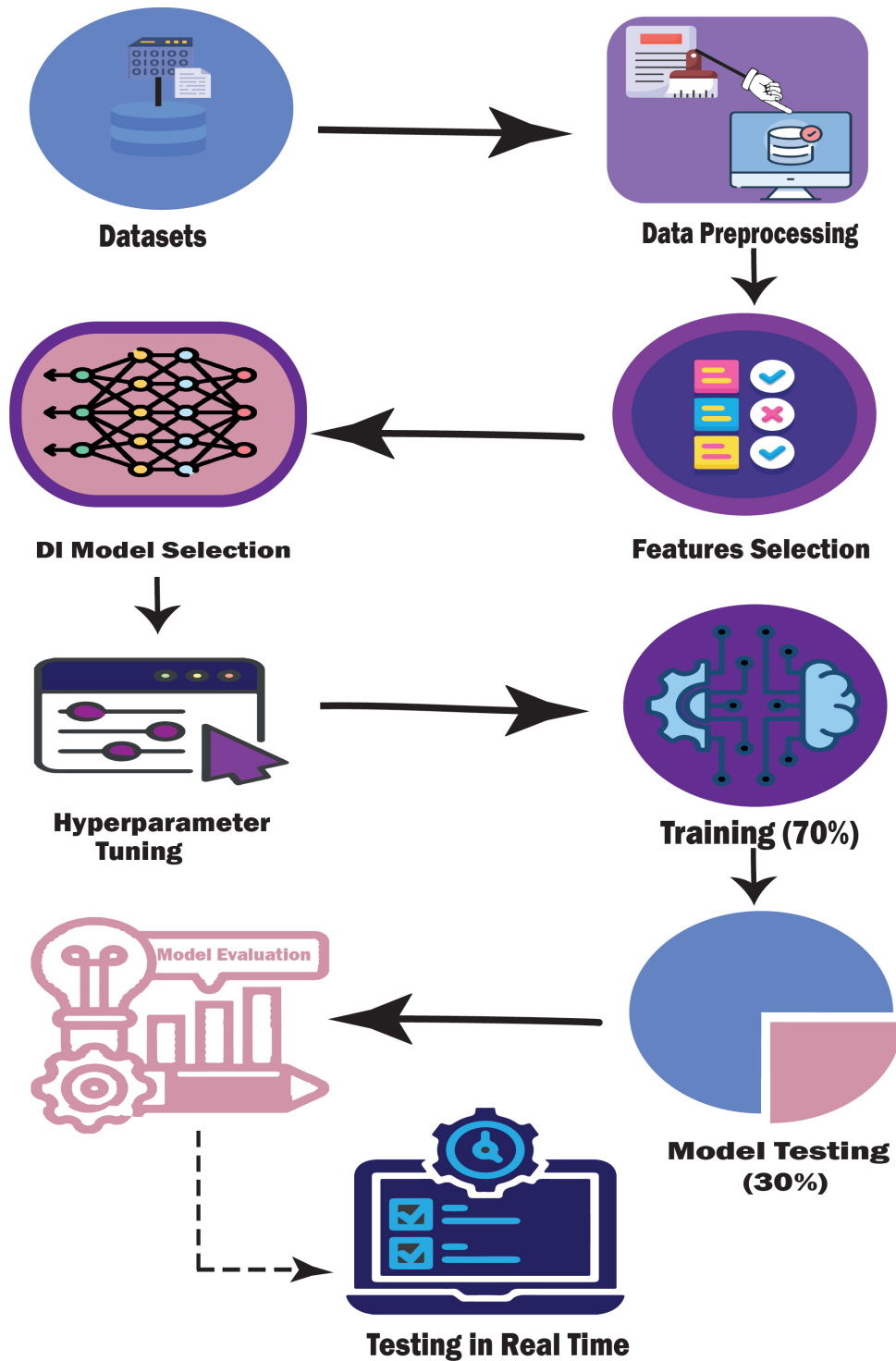


FIGURE 3.1 : Proposed IDS Development Pipeline

of epochs to enhance the models accuracy and robustness.

After training and fine-tuning our models, we integrated them into a real-time environment to test their functionality. Using simulation tools and real network traffic, we validated the IDS's capability to detect and mitigate DDoS attacks in a live SDN setup.

3.3 Dataset

3.3.1 Overview

Datasets are crucial in training DL models since they provide the examples from which these models learn patterns to make predictions on unseen data. Additionally, the quality, size, and relevance of a dataset significantly impact a model's performance and accuracy. Hence, choosing an appropriate dataset with the most significant features is essential for models training and testing considering that this choice ensures that the models learn meaningful and relevant patterns, avoid overfitting or underfitting, and perform well in real-world scenarios.

Various datasets containing DDoS attacks traffic exist, including packet-based, flow-based datasets, and those representing slow-rate and high-rate DDoS attacks. In the next subsection, we will select a suitable dataset to train and evaluate our selected DL models.

3.3.2 Chosen Dataset

To train and evaluate our DL models, we have selected one public labeled dataset named CICIDS2017 [38], from the Canadian Institute for Cybersecurity (CIC). This dataset is flow based and matches the SDN traffic representation which makes it qualified to train and test a DL based IDS in SDN. The chosen dataset encompasses both benign traffic and the latest common attacks, it was generated by simulating the behaviour of 25 users, accurately resembling real-world data in Packets Captures (PCAPs) format. Furthermore, it contains the results of network traffic analysis conducted with CICFlowMeter, featuring labeled flows based on timestamps, source and destination IPs, source and destination ports, protocols, and attack types in CSV format.

The CICIDS2017 dataset includes five days of network traffic data simulated in a controlled testbed environment, featuring a mix of up-to-date common attacks and benign traffic. Monday data contains only benign traffic, while the attacks including Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS, were executed both in the morning and afternoon on Tuesday, Wednesday, Thursday, and Friday.

For our work, we selected the Wednesday dataset, which contains application layer DoS attacks that can be considered as DDoS attacks based on the attack scenario, generated using various tools, namely DoS Slowloris, DoS Slowhttptest, DoS Hulk, and DoS GoldenEye. Specifically, the Slowhttptest tool includes four types of attacks : slow read, slow range, slow body, and slow headers. While the Hulk tool generates DoS attacks by flooding the server with a high volume of HTTP requests in a very short time, effectively overwhelming the server resources. We eliminated the Heartbleed attack from the Wednesday dataset as it does not represent a DoS attack. Additionally, we excluded the GoldenEye attack because the tool is considered risky and cannot be safely tested in our environment.

In table 3.1, a detailed illustration of the label counts for our dataset is provided.

3.3.3 Dataset Limitations

During our examination of the benchmark dataset and after attempting to train our IDS models, we encountered difficulties in detecting DDoS attacks in real-time scenarios. This prompted us to conduct deeper research into the issue. We found several common limitations in the selected dataset, such as high redundancy, outdated data, imbalanced class distribution, and lack of real-time data [39].

TABLE 3.1 : Label Counts for CICIDS2017 Dataset

Dataset	Label	Count
CICIDS2017	BENIGN	440031
	DoS Hulk	231073
	DoS slowloris	5796
	DoS Slowhttptest	5499

To address these shortcomings, we decided to generate and add novel data to the CICIDS2017 dataset. This new data is based on a real-world network scenario and recent attack patterns, aiming to provide a more comprehensive and up-to-date resource. Additionally, we generated benign traffic to provide a balanced dataset that accurately represents normal network behavior. This enhancement intends to improve our IDS ability to detect DDoS attacks in real-time.

3.3.4 Enhancing the Dataset

In our efforts to address the limitations of the CICIDS2017 dataset, we adopted a proactive approach. Acknowledging the dynamic nature of DDoS attacks, we employed previously utilized attack tools, namely slowhttptest and Hulk, to generate a diverse array of DDoS attacks within our SDN custom topology simulation. Each type of attack present in the dataset was meticulously reproduced, ensuring comprehensive coverage of the known threats. Additionally, we generated benign traffic using hping3 for SYN and UDP, creating realistic normal traffic patterns. These newly generated instances were then seamlessly integrated, preserving the original labeling scheme of the CICIDS2017 dataset as explained in figure 3.2.

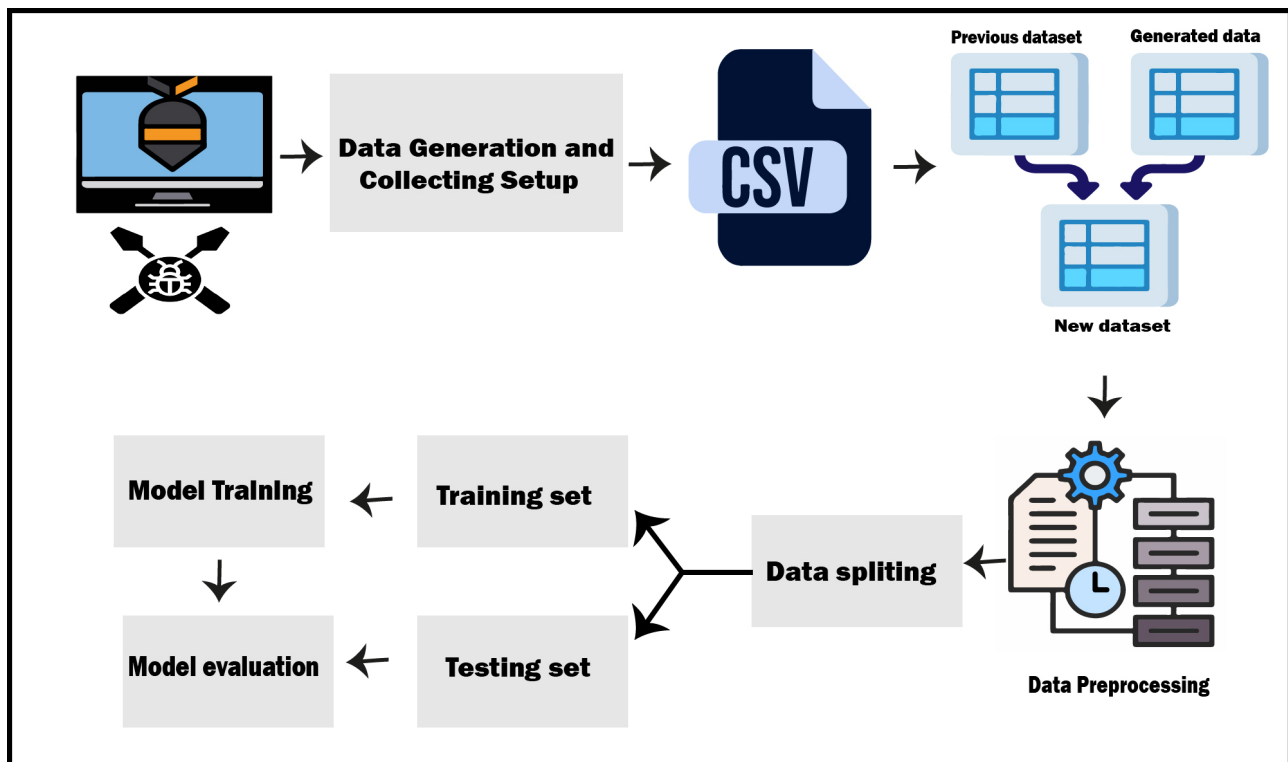


FIGURE 3.2 : Enhancing and Integrating Generated Data Patterns for Improved Dataset

This concatenation not only overcomes the dataset constraints but also ensures a robust representation of contemporary DDoS threats by enriching the dataset with a spectrum of attack types from potent and prevalent tools. This proposed solution was implemented through the following steps.

Topology Generation in Mininet :

We created a simple topology using Mininet, featuring one Open Virtual Switch (OVS) and two hosts to simulate both DDoS attacks and legitimate traffic. The specific command to create this topology and configure it to wait for a remote controller is :

```
1 sudo mn --topo single,2 --controller=remote
```

As a result of executing this command, we obtained the following output illustrated in figure 3.3 indicating the creation of the network topology.

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

FIGURE 3.3 : DDoS Attack Topology

Next, we started the Ryu controller with the command :

```
1 ryu-manager ryu.app.simple_switch_13.py
```

The `simple_switch_13.py` application acts as a fundamental example of the southbound API functionality in Ryu. It utilizes OpenFlow 1.3 to establish communication between the controller and the network devices (switches). The execution of the Ryu controller with the `simple_switch_13.py` application initializes a basic learning switch using OpenFlow 1.3, allowing the controller to manage network switches and control packet forwarding dynamically. The `simple_switch_13.py` application handles packet-in messages, learns source MAC addresses, and forwards packets accordingly. The output in figure 3.4 confirms the successful initialization of the Ryu controller, ready to manage the network environment.

```
$ ryu-manager simple_switch_13.py gui_topology/gui_topology.py
loading app simple_switch_13.py
loading app gui_topology/gui_topology.py
loading app ryu.controller.ofp_handler
loading app ryu.app.rest_topology
loading app ryu.app.ws_topology
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of Switches
creating context switches
instantiating app None of DPSet
creating context dpset
instantiating app simple_switch_13.py of SimpleSwitch13
instantiating app gui_topology/gui_topology.py of GUIServerApp
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.rest_topology of TopologyAPI
instantiating app ryu.app.ws_topology of WebSocketTopology
instantiating app ryu.app.ofctl_rest of RestStatsApi
(2464735) wsgi starting up on http://0.0.0.0:8080
```

FIGURE 3.4 : Ryu Starting

Following this setup, we tested the connectivity within our network by using the `pingall` command in Mininet as showed in figure 3.5 to ensure that all the nodes could communicate effectively.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> 
```

FIGURE 3.5 : Ping Test Results

Attacks generation :

After installing the DoS/DDoS attacks tools, namely slowhttptest and Hulk, we initiated attacks on the generated topology. slowhttptest executes slow HTTP attacks by keeping multiple connections open with incomplete requests. While Hulk conducts high-speed HTTP attacks, flooding the server with requests. In our setup, the first host serves as the victim server, while the second host acts as the attacker.

First, we verified the IP addresses of the hosts and the switch interfaces they are connected to by using specific commands. To obtain the IP addresses of the hosts, we used :

```
1 h1 ifconfig
2 h2 ifconfig
```

To determine the interfaces that are connected to the hosts on the switch, we employed :

```
1 sudo ovs-vsctl show
```

These commands provided the necessary information to initiate the attacks and monitor the setup accurately.

Second, we started a server on the first host (h1) in Mininet using the following command :

```
1 sudo python3 -m SimpleHTTPServer 80 &
```

Next, we started to launch the DoS attacks from the second host (h2) using various commands for each of the DoS attack tools mentioned above, these commands are listed in Appendix 2.1.

Each attack was simulated separately and saved in a separate CSV file with the same label as the original dataset. Specifically, at the same time as the attack was launched, we executed a python script that employs CICFlowMeter tool to sniff network traffic from the switch's interface connected to the victim host, this tool allows real-time packet sniffing on the targeted interface and converts the Packet Captures (PCAPs) into a flow-based CSV file containing more than 80 features, in the same way as the CICIDS2017 dataset was created. This approach enabled us to obtain new datasets containing the different types of attacks with the same features as the original dataset, facilitating concatenation.

Legitimate Traffic Generation :

We needed to improve our training outcomes for our DL models, which required expanding our base dataset with additional benign traffic. This enhancement allows our models to learn and train on a larger dataset, ultimately improving their detection capabilities and overall performance. So as we added attack flows, we also incorporated benign packets using hping3 tool. Details are provided in Appendix 2.2.

Datasets Concatenation :

After obtaining the new datasets representing different DoS/DDoS attacks and benign traffic, we concatenated them with the original one. The resulting dataset reflects an updated distribution of labels, which is depicted in table 3.2.

This updated dataset now includes both recent attacks patterns and legitimate traffic maintaining the same features set as the original, ensuring consistency and compatibility.

By integrating these new data patterns, we have enhanced our dataset to reflect the current threat landscape

TABLE 3.2 : Updated Label Counts

Dataset	Label	Count
CICIDS2017	BENIGN	587597
	DoS Hulk	263172
	DoS slowloris	35592
	DoS Slowhttptest	97322

more accurately. Hence, it will be very useful to train our models, providing them with comprehensive and up-to-date information to improve their ability to detect and mitigate a wide range of DoS and DDoS attacks effectively.

3.3.5 Dataset Preprocessing

The CICIDS2017 dataset indeed required thorough preprocessing to make it suitable for model training. In addition to null and negative values, we encountered several other challenges. These included duplicate values, columns with identical values, infinite values, and pairs of columns with identical values, among others. Addressing these issues was crucial to ensure the integrity and effectiveness of the training of our DL models.

Dataset Cleaning :

The data cleaning process was essential to ensure the reliability and accuracy of the dataset used for training our DL models for DDoS attacks detection and mitigation. It consists of standardizing data and removing outliers. The detailed steps are cited in Appendix 3.1.

After dataset cleaning, the labels distribution became imbalanced, as shown in table 3.3. Addressing this issue is the next step in ensuring the dataset's integrity.

TABLE 3.3 : Label Counts After Dataset Cleaning

Dataset	Label	Count
CICIDS2017	BENIGN	218437
	DoS Hulk	105798
	DoS slowloris	22328
	DoS Slowhttptest	65275

To address the imbalanced data issue post-cleaning, we applied the SMOTE technique to oversample the BENIGN, Slowhttptest, and Slowloris class labels. As a result, we achieved a balanced dataset, conducive to more reliable analysis and models training, the outcome of this operation is represented in table 3.4.

Handling Attack Labels :

To streamline the classification process into a binary classification problem, we aggregated multiple types of DoS/DDoS attacks into a unified 'ATTACK' category. This approach simplifies the classification task while maintaining the integrity of the dataset for meaningful analysis.

```

1 # Consolidate DDoS attack types into 'ATTACK' category
2 df["label"].replace(["DoS Hulk", "DoS Slowhttptest", "DoS
   ↳ slowloris", "ATTACK", inplace=True)
```

TABLE 3.4 : Label Counts After Applying SMOTE.

Dataset	Label	Count
CICIDS2017	BENIGN	327000
	DoS Hulk	105798
	DoS Slowloris	110842
	DoS Slowhttptest	110842

Both BENIGN and ATTACK classes are balanced as illustrated in table 3.5 due to data generation and applying SMOTE technique to augment the minority classes instances. This balanced distribution mitigates the issue of class imbalance, enabling our models to learn from a more representative dataset and improving their ability to accurately classify instances from both classes.

TABLE 3.5 : Final labels count.

Dataset	Label	Count
CICIDS2017	BENIGN	327000
	ATTACK	327482

Data Shuffle :

Shuffling the dataset randomly mixes up the order of its rows. This helps to remove any patterns or biases in the data that could affect our analysis. After shuffling, resetting the row indexes ensures that they go back to being in order.

```

1 #shuffle the Dataframe
2 df = df.sample(frac=1, random_state=42).reset_index(drop=True
  ↳ )

```

Feature Selection :

In our final dataset, we managed to gather a diverse set of features for training our models. However, to achieve optimal results and enhance the learning process, it's essential to identify the most significant features that can effectively distinguish between legitimate traffic and attack instances. To accomplish this, we require reliable methods to rank these features based on their importance, prioritizing those that contribute the most to the models ability to differentiate between normal and malicious traffic. In our feature selection approach, we employed both the Chi-square Test and the Information Gain Ratio (IGR) methods [?, 11] to identify the most relevant features for models training and evaluation.

1. Chi-square Test :

It is a statistical method that assesses the dependency between categorical variables, which makes it suitable for feature selection in classification tasks. We set the parameter k to 35, indicating our objective of selecting the top 35 features. The SelectKBest function from the scikit-learn library was employed to perform the feature selection based on their chi-square scores.

```

1 Selected Features by Chi-square Test: [feature1, feature2
  ↳ , ..., feature35]
2 Chi-square Test Scores for each feature: [score1, score2,
  ↳ ..., scoreN]

```

2. Information Gain Ratio (IGR) :

In addition to the Chi-square test technique, we employed the Information Gain Ratio (IGR) method that evaluates the usefulness of a feature by considering both its information gain and intrinsic value. We first discretized the continuous features using the KBinsDiscretizer from scikit-learn to facilitate the calculation of IGR. Subsequently, we computed the IGR scores for all the features, providing insights into their predictive power and relevance to the target variable. We selected the top-ranked 35 features identified through the IGR method.

```

1      Selected Features by IGR: [feature1, feature2, ...,
      ↪ feature35]
2      IGR Scores for each feature: [score1, score2, ..., scoreN
      ↪ ]

```

Finally, we combined the selected features from both the Chi-square test and IGR methods to create the final features set for training our models. This comprehensive approach ensured the inclusion of the most informative features, enhancing the models predictive performance.

```

1      selected_features = [item for item in selected_features_IGR if
      ↪ item in selected_features_chi2]

```

Final Set of features :

Specifically, the selected features for the analysis were determined by the intersection of the features chosen by both Chi-Square test and the Information Gain Ratio (IGR) methods. These techniques identified a set of key features that are illustrated in table 3.6. This comprehensive features set captures various aspects of the data, ensuring robust input for subsequent analysis and models development.

TABLE 3.6 : Selected Features and Their Meanings

Feature	Meaning
bwd_pkt_len_max	Maximum packet length in the backward direction.
fwd_pkts_s	Number of forward packets per second.
flow_pkts_s	Number of packets per second for the entire flow.
flow_iat_min	Minimum inter-arrival time between packets in the flow.
flow_iat_mean	Mean inter-arrival time between packets in the flow.
pkt_len_var	Variance of packet lengths.
bwd_iat_max	Maximum inter-arrival time between packets in the backward direction.
idle_max	Maximum idle time between packets.
idle_std	Standard deviation of idle times between packets.
flow_iat_max	Maximum inter-arrival time between packets in the flow.
fwd_iat_max	Maximum inter-arrival time between packets in the forward direction.
bwd_iat_std	Standard deviation of inter-arrival times between packets in the backward direction.
idle_mean	Mean idle time between packets.
fwd_iat_std	Standard deviation of inter-arrival times between packets in the forward direction.
fwd_blk_rate_avg	Average rate of blocked forward packets.
idle_min	Minimum idle time between packets.
fwd_iat_tot	Total inter-arrival time for packets in the forward direction.
bwd_iat_tot	Total inter-arrival time for packets in the backward direction.
init_bwd_win_byts	Initial window size of backward traffic in bytes.

3.4 Design of Deep Learning Models

3.4.1 Overview

The selection of DL algorithms is crucial to build our IDS to detect DDoS attacks. These attacks generate complex patterns in network traffic that can overwhelm traditional detection methods [40]. DL excels in automatically learning and identifying these intricate patterns, making it well-suited for DDoS detection. Furthermore, DoS and DDoS attacks tactics are constantly evolving and changing; DL models can be trained on diverse datasets, allowing them to learn from new attack patterns and adapt their detection capabilities accordingly.

3.4.2 Chosen Models

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs); can identify subtle anomalies and complex relationships within network traffic that traditional methods might miss. By training on a comprehensive dataset that includes diverse attack signatures, DL models enhance the accuracy and

speed of DDoS detection, enabling real-time responses to mitigate threats.

Three different DL approaches exist for detecting DDoS attacks. The first is the single model approach, which uses one DL algorithm to make predictions and has been widely applied in SDN environments. Furthermore, to enhance their performance, ensemble and hybrid models are employed. Where the ensemble learning approach combines several DL classifiers to improve overall accuracy, using methods like average voting, majority voting, and optimally weighted majority voting [14]. These models are more robust to noise and overfitting compared to single models. On the other hand, the hybrid model approach stacks two types of deep neural networks within a single framework, leveraging their combined strengths to solve complex problems.

Through our research, we have concluded from previous work [14, 41, 42, 13] that 1D-CNN is known by its ability of detecting relationship between neighboring features, enabling the extraction of more deeper features. Before designing our model, we started by the visualization of correlation matrix between the selected features (Fig 3.6), which highlights relationships between the features of our dataset. We can observe that some features are strongly correlated, as indicated by warmer colors. The 1D-CNN can utilize these correlations to better understand the underlying patterns in the data, enhancing its ability to detect complex dependencies and interactions between features. This capability allows the 1D-CNN to excel in tasks such as DDoS attacks detection, where identifying nuanced patterns in correlated features is crucial for accurate and effective threat identification.

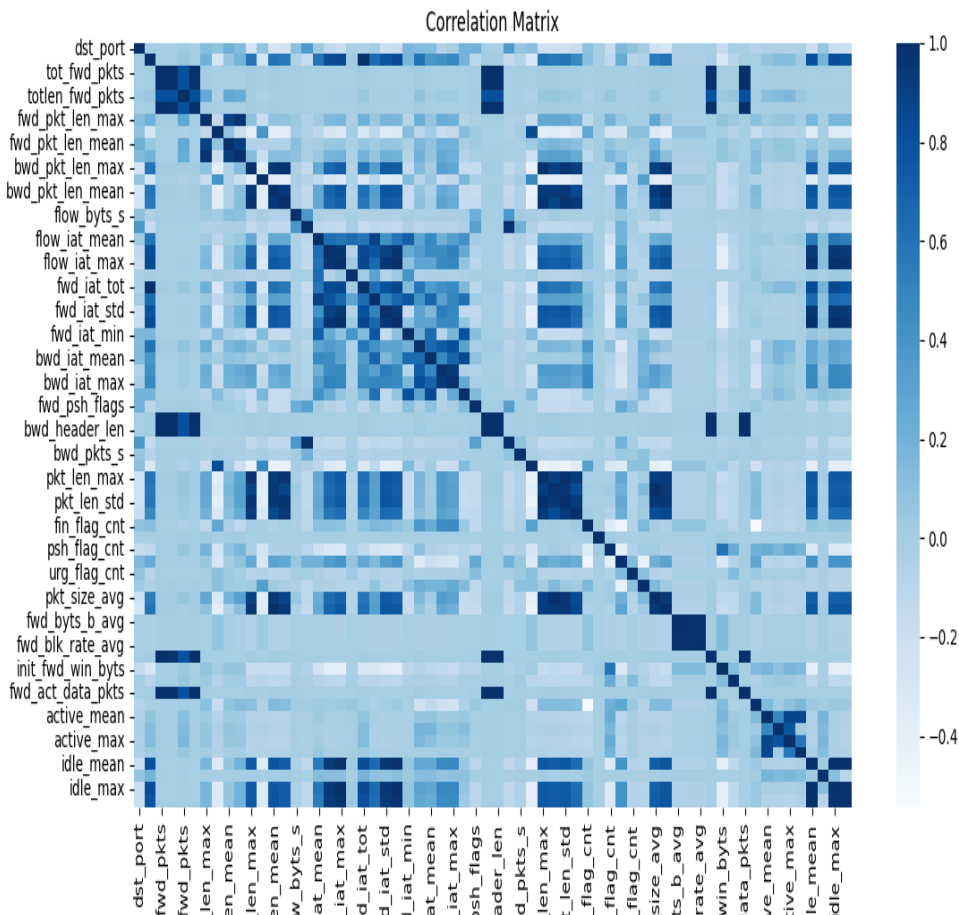


FIGURE 3.6 : Correlation Matrix of The Considered Features

Furthermore, LSTM captures temporal dependencies and long-range relationships within sequential data. Unlike traditional neural networks, LSTMs are designed to remember information for extended periods,

making them highly effective in tasks involving time-series data or any data where the order of inputs matters. This is particularly relevant for feature-based datasets like CICIDS2017, which contains flows captured in time series. In the context of DDoS attacks detection, LSTMs can analyze these sequences of network traffic data, identifying patterns and anomalies over time. This temporal awareness allows LSTMs to detect subtle and evolving attacks strategies, complementing the spatial dependency detection capabilities of 1D-CNNs. Together, these models provide a comprehensive approach to identify and mitigate threats in SDN based networks, leveraging both spatial and temporal patterns within the data.

We incorporated 1D-CNN and LSTM into our project, leveraging their strengths with three approaches : using them as separate single models, combining them into a hybrid model, and integrating these models (single 1D-CNN, LSTM, and hybrid 1D-CNN-LSTM) using the ensemble technique. These approaches will be presented in the following sections.

3.5 Model Architectures

3.5.1 1D-CNN Model

The 1D-CNN model architecture illustrated in 3.7 is designed for an IDS to detect DDoS attacks by identifying complex patterns in network traffic. It includes convolutional layers for feature extraction and pooling layers for dimensionality reduction. Each component is fine-tuned to ensure the model's robustness and accuracy in real-world scenarios. Detailed layer information is provided in table 3.7. After extensive hyperparameter tuning, we arrived at this optimal architecture that employ regularization methods namely BatchNormalization, Dropout, and L2 to avoid overfitting.

TABLE 3.7 : Details of the 1D-CNN Model Architecture.

Layer Type	Details
Conv1D	64 filters, kernel size 3, ReLU activation
BatchNormalization	Normalizes the output
MaxPooling1D	Pool size 2
Dropout	Rate 0.3
Conv1D	128 filters, kernel size 3, ReLU activation
BatchNormalization	Normalizes the output
MaxPooling1D	Pool size 2
Dropout	Rate 0.3
Flatten	Converts 2D output to 1D feature vector
Dense	128 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	64 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	1 unit, sigmoid activation

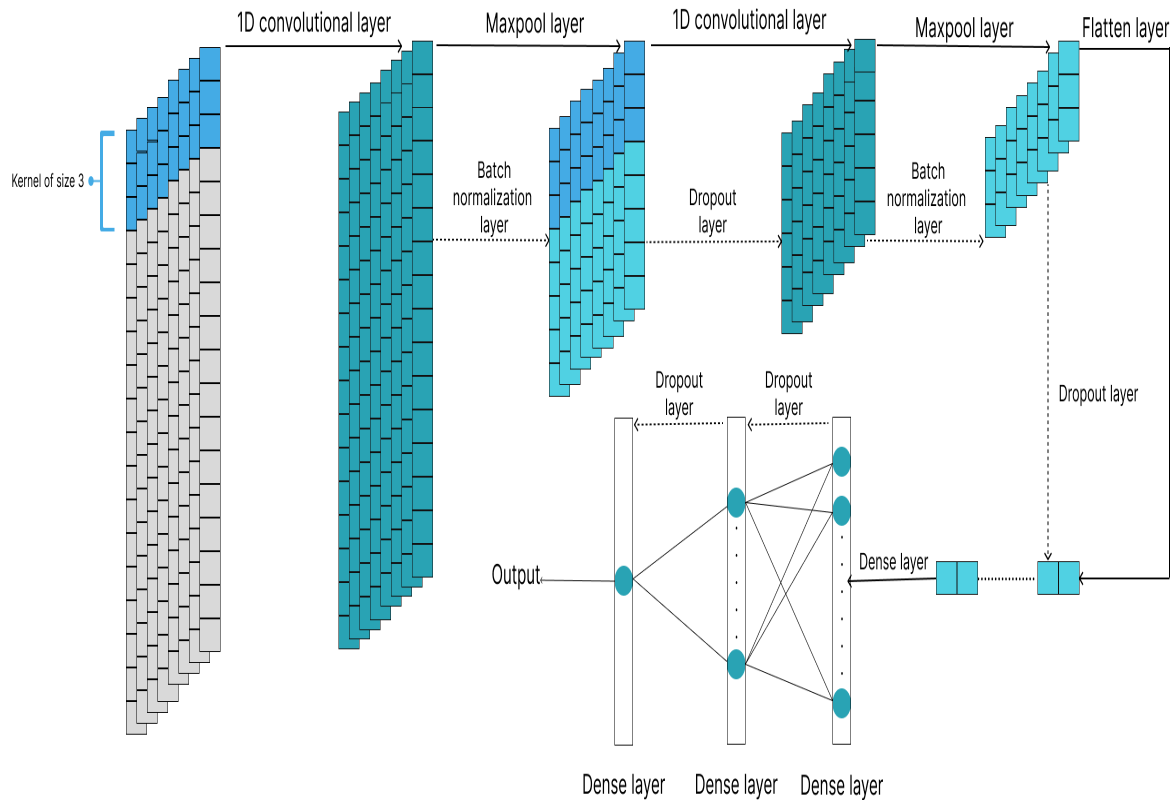


FIGURE 3.7 : CNN Model Architecture

3.5.2 LSTM Model

Our LSTM model for IDS is specifically designed to leverage the sequential nature of network traffic data to enhance detection performance. By capturing temporal dependencies in network traffic, the model effectively identifies patterns indicative of malicious activity. This capability allows the IDS to distinguish between normal and abnormal traffic more accurately. The model architecture (Fig 3.8) includes multiple LSTM layers that process the data sequentially, normalization layers to ensure stable training, dropout layers to prevent overfitting, and dense layers for final classification. These components work together to provide a robust solution for detecting different network attacks. After extensive hyperparameter tuning, the specifics of the best architecture are detailed in table 3.8.

TABLE 3.8 : Details of the LSTM Model Architecture.

Layer Type	Details
LSTM	64 units, ReLU activation, return sequences
BatchNormalization	Normalizes the output
Dropout	Rate 0.4
LSTM	32 units, ReLU activation
BatchNormalization	Normalizes the output
Dropout	Rate 0.4
Dense	64 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	32 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	1 unit, sigmoid activation

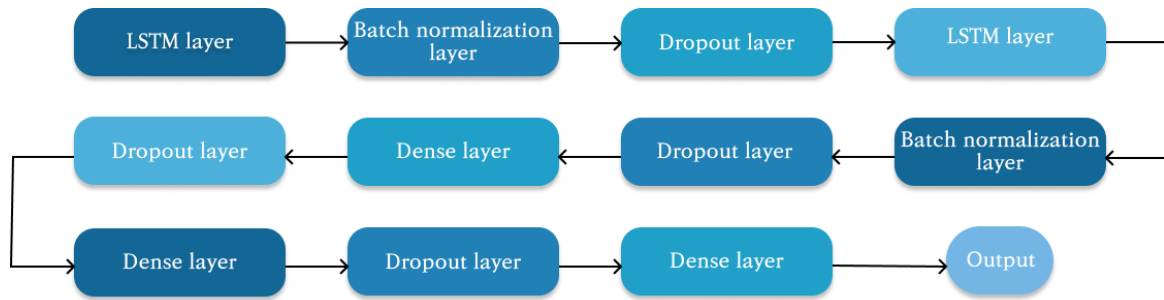


FIGURE 3.8 : LSTM Model Architecture

3.5.3 CNN-LSTM Hybrid Model

The architecture of this model is designed to effectively detect DDoS attacks, it combines convolutional layers for feature extraction and LSTM layers for sequence learning. Convolutional layers help in capturing spatial features from the input data, while LSTM layers are adept at understanding temporal dependencies, making the model robust in detecting sequential patterns in network traffic. To enhance training stability and generalization, the model includes normalization layers to standardize the input, dropout layers to prevent overfitting, and dense layers for final decision-making. The detailed layer information is outlined in table 3.9.

TABLE 3.9 : Details of The Hybrid CNN-LSTM Model Architecture.

Layer Type	Details
Conv1D	64 filters, kernel size 3, ReLU activation
BatchNormalization	Normalizes the output
MaxPooling1D	Pool size 2
Dropout	Rate 0.3
Conv1D	128 filters, kernel size 3, ReLU activation
BatchNormalization	Normalizes the output
MaxPooling1D	Pool size 2
Dropout	Rate 0.3
TimeDistributed	Flattens the convolutional output
LSTM	64 units, ReLU activation
Dense	128 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	64 units, ReLU activation, L2 regularization (0.001)
Dropout	Rate 0.5
Dense	1 unit, sigmoid activation

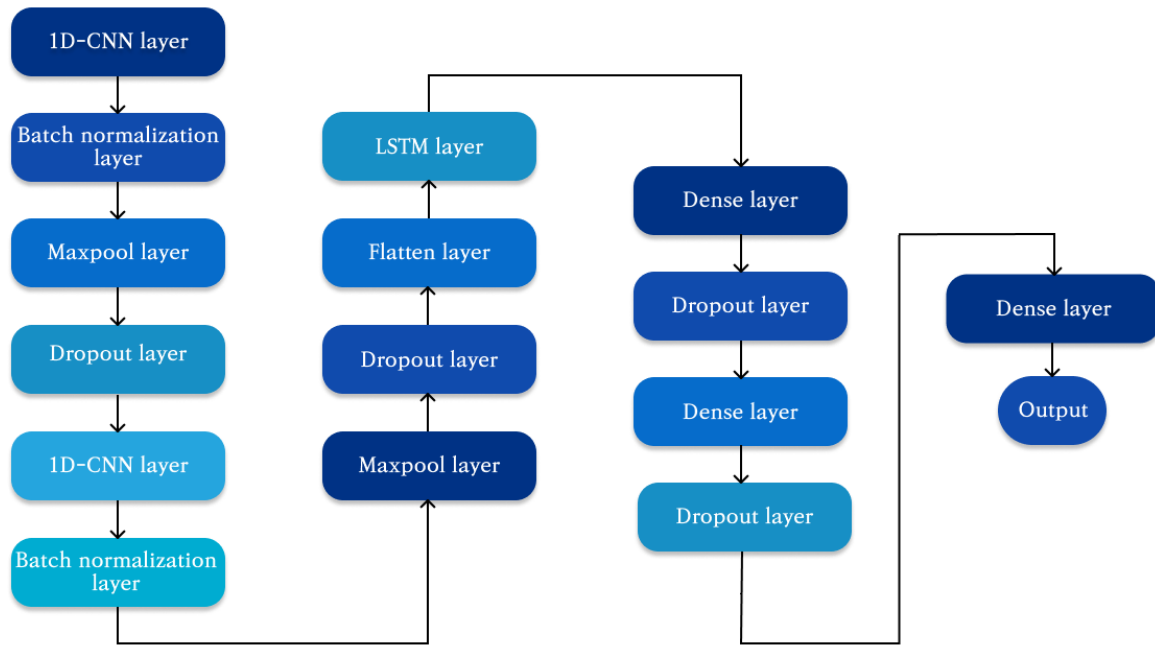


FIGURE 3.9 : CNN-LSTM Model Architecture

3.5.4 YOUD Model : An Ensemble Approach Combining CNN, LSTM, and Hybrid CNN-LSTM

In this section, we proposed a new ensemble learning approach for detecting and classifying network traffic anomalies. It integrates CNN, LSTM, and hybrid CNN-LSTM models to leverage their strengths in feature extraction and sequence modeling. Each model is independently trained, and their predictions are combined using a majority voting mechanism (Fig 3.10). The detailed layer information is presented in table 3.10.

TABLE 3.10 : Summary of The YOUD Model Architecture.

Component	Details
LSTM Model	Load from JSON, 64 units and 32 units LSTM layers, ReLU activation, BatchNormalization, Dropout, Dense layers with L2 regularization
CNN Model	Load from JSON, Conv1D layers with 64 and 128 filters, ReLU activation, BatchNormalization, MaxPooling1D, Dropout, Flatten, Dense layers with L2 regularization
Hybrid Model	Load from JSON, Conv1D layers, BatchNormalization, MaxPooling1D, TimeDistributed Flatten, LSTM layer, Dense layers with L2 regularization
Ensemble Process	Preprocess and scale input features, reshape for each model, predict with LSTM, CNN, and Hybrid models, aggregate predictions using majority voting

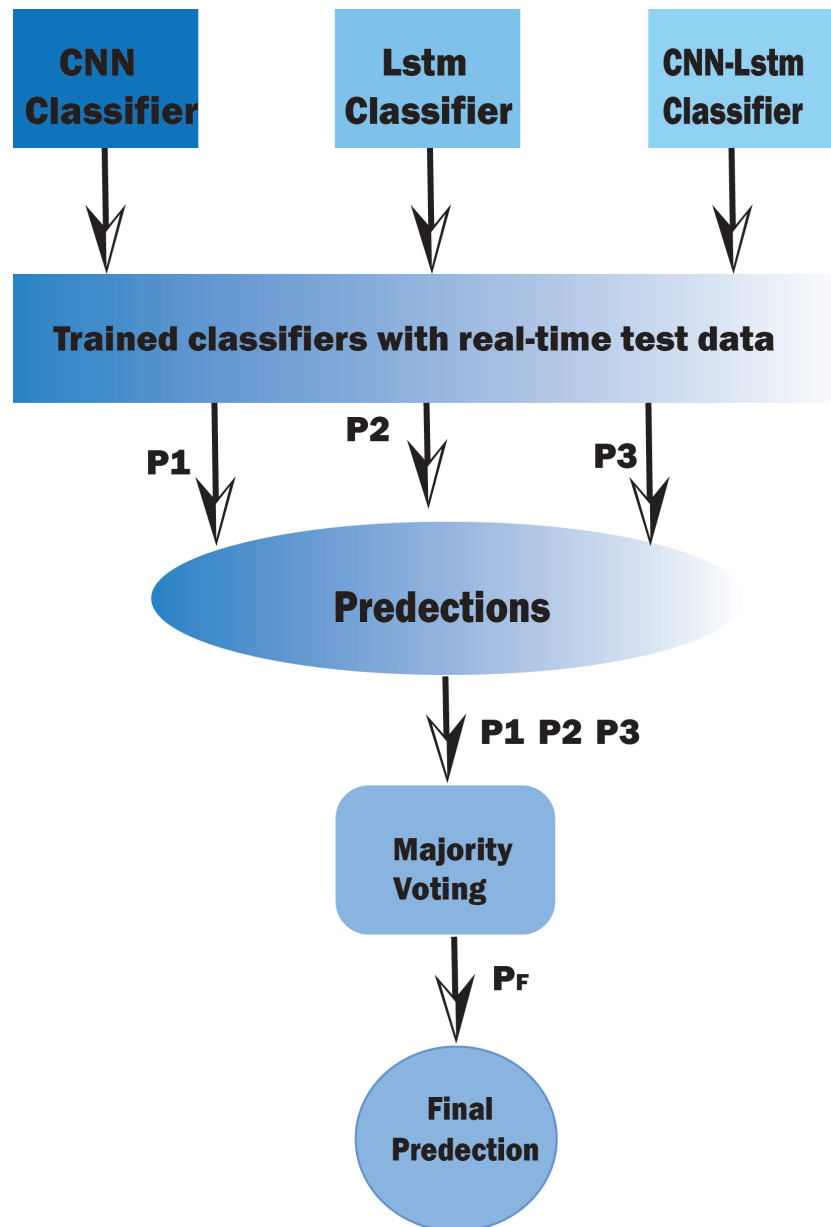


FIGURE 3.10 : Ensemble Model Architecture

3.6 Enhancing YOUD Ensemble Model with Threads

To improve the efficiency and effectiveness of our YOUD ensemble model, we implemented multithreading. This approach allows to monitor multiple network interfaces concurrently, ensuring real-time anomaly detection and response to network traffic. By adding threads to our program, we can handle large volumes of data from various interfaces simultaneously, enhancing the overall performance of the IDS.

Multithreading In IDS

Multithreading is a programming technique that enables multiple threads to run concurrently within a single process. Each thread can execute independently, enabling parallel processing and more efficient utilization of system resources. In the context of our IDS, multithreading offers several benefits :

- **Concurrent Monitoring** : Each network interface is monitored by a separate thread, allowing for simultaneous data collection and analysis.
- **Improved Performance** : By distributing the workload across multiple threads, the IDS can process data more efficiently, reducing the detection latency.
- **Scalability** : Multithreading facilitates the addition of new network interfaces without significant changes to the core program, making the IDS scalable to larger networks.
- **Real-Time Detection** : With multiple threads, the IDS can analyze traffic in real-time, quickly identifying and mitigating potential threats.

3.7 Real-time DDoS Detection and Mitigation Framework

The implementation of a DL-based IDS in SDN (Fig 3.11) can be divided into three key steps : attack identification, reporting, and mitigation [7]. During the attack identification step, the IDS module collects network traffic in real-time using the CICFlowMeter tool, which converts network traffic from Packet Captures format to a set of flow-based features and saves the network data as a CSV file. This CSV file serves as an input for the DL-based models to detect DDoS attacks in the SDN system. Next, in the reporting step, the IDS module announces the presence of attack traffic, providing details about the interface from which the attack patterns originate. Finally, in the mitigation step, based on the alert and report received from the IDS module, a flow rule is sent to the Ryu controller. This rule is then applied to the OVS-based switches to block the IP address from which the corrupted network traffic emerges.

3.8 Graphical User Interface

We developed a user-friendly Flask-based web interface for our IDS and IPS, integrating essential features for streamlined network security management.

Python, HTML, CSS, and JavaScript were used to build the web application that manages tasks such as traffic sniffing initiation, flow analysis, and real-time attack detection using our YOUD model. This setup aids network administrators and cybersecurity professionals in monitoring and responding to network threats efficiently.

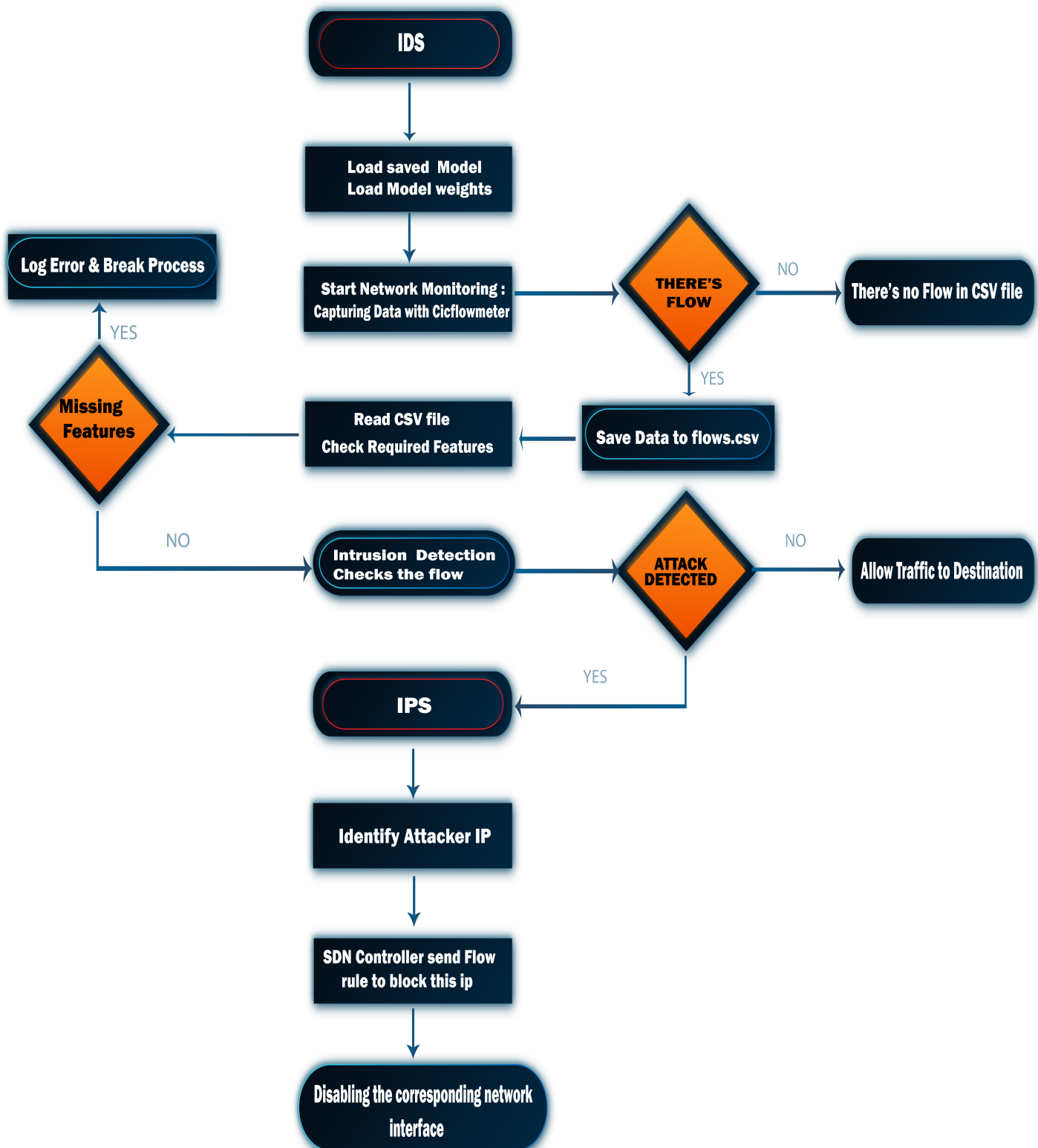


FIGURE 3.11 : Logical Diagram of IDS/IPS

3.9 Flask Application and YOUD Model

The final YOUD model, that is an ensemble of CNN, LSTM, and hybrid CNN-LSTM models, was implemented with multithreading to monitor multiple network interfaces simultaneously. This approach ensures efficient real-time detection and mitigation of DDoS attacks. The Flask application is designed to start monitoring network interfaces at the click of a button. The application uses multithreading to handle simultaneous monitoring tasks for different network interfaces, ensuring that our YOUD model can analyze traffic in real-time without performance bottlenecks.

The implementation of our system in the Flask application is based on several critical components that work together to ensure efficient detection and mitigation of DDoS attacks :

1. **Model Loading** : We load the pre-trained YOUD model components (CNN, LSTM, and CNN-LSTM) using TensorFlow. Each model is loaded using its respective JSON and weights files.
2. **Multithreading** : We use threading to monitor multiple network interfaces concurrently. Each thread is responsible for capturing traffic, processing it through the YOUD model, and displaying the results.
3. **Flow Rule Modification** : When an attack is detected, the application sends a flow rule to the Ryu controller to block traffic from the malicious IP address, thereby mitigating the attack.
4. **User Interface** : The Flask application provides a clean and intuitive interface for starting the sniffing process, viewing attack history, and resetting the messages. The detected results, including IP addresses, attack counts, and benign traffic percentages, are displayed on the web page.

3.10 Conclusion

In our methodology, we began by carefully selecting and preparing our dataset for training our models, focusing on data enhancement, cleaning, and preprocessing. We then strategically chose DL models : 1D-CNN, LSTM, and a hybrid CNN-LSTM to develop our IDS. These models were optimized to detect application layer DDoS attacks effectively.

Additionally, we introduced the YOUD ensemble model, which combines predictions from all three models using a majority voting mechanism. To enhance scalability and real-time performance, multithreading was integrated into the YOUD model within a Flask-based application framework.

The next chapter will detail our implementation process and present the results of our comprehensive testing and evaluation, providing insights into the performance and reliability of our IDS approach in practical applications.

Chapter 4

Implementation and Results

4.1 Introduction

This chapter presents the results of our study on application layer DDoS attacks detection and mitigation using various DL models. We begin by highlighting the results of training and evaluation of 1D-CNN, LSTM, and hybrid CNN-LSTM models. Then, we explore an ensemble method called YOUD that combines the predictions of the 1D-CNN, LSTM, and CNN-LSTM models to enhance detection accuracy.

Additionally, we demonstrate the application of our models in real-time DDoS detection and mitigation, analyzing the system's response to live attacks. Lastly, we deploy our YOUD model through a multithreaded, user-friendly Flask application and conduct thorough testing of the final implemented approach.

4.2 Implementation Details

In this section, we detail the implementation of our SDN setup using Mininet and the Ryu controller, focusing on a custom topology created through Mininet's API. We will integrate the IDS developed earlier that is designed to detect application layer DDoS attacks within this network. Network traffic will be captured and transformed using CICFlowMeter, and subsequently passed to our application for DDoS detection. Upon detecting a DDoS attack, the application will mitigate it by installing appropriate flow rules to manage and control the malicious traffic. All the tools used in this implementation are detailed in the "Material and Software Tools" section in Appendix 4.7.

4.2.1 Network Topology

We created a custom topology using Mininet consisting of six hosts. Where three hosts are connected to one Open Virtual Switch (OVS), and the remaining three are connected to another. Additionally, the two switches are interconnected to ensure connectivity between all the hosts. In the first switch, we have designated one host as attacker and another as the victim. While the second switch includes one additional attacker, as depicted in figure 4.1.

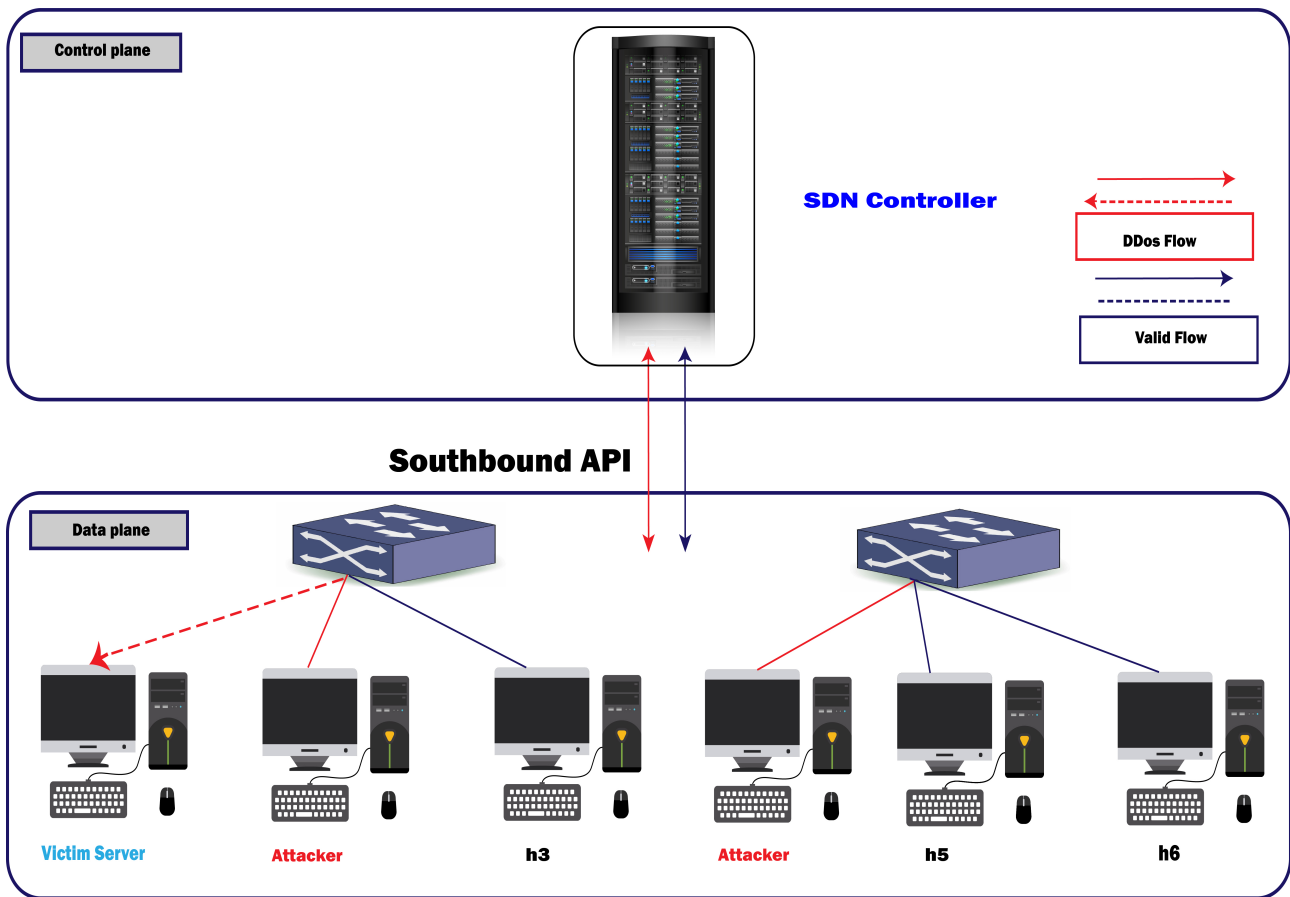


FIGURE 4.1 : SDN Framework

This topology was specifically designed to integrate and test our proposed approach for detecting and mitigating DDoS attacks. It was created using the predefined Python API provided by Mininet.

```

1 from mininet.topo import Topo
2 from mininet.node import OVSSwitch, RemoteController
3 from mininet.net import Mininet
4 from mininet.cli import CLI
5 from mininet.log import setLogLevel
6
7 class MyTopo(Topo):
8
9     def build(self):
10
11         # Add hosts and assign IP addresses
12         Host1 = self.addHost('h1', ip='10.0.0.1/24')
13         Host2 = self.addHost('h2', ip='10.0.0.2/24')
14         Host3 = self.addHost('h3', ip='10.0.0.3/24')
15         Host4 = self.addHost('h4', ip='10.0.0.4/24')
16         Host5 = self.addHost('h5', ip='10.0.0.5/24')
17
18         # Add switches
19         leftSwitch = self.addSwitch('s2', cls=OVSSwitch, protocols='
↪ OpenFlow13')
20         rightSwitch = self.addSwitch('s1', cls=OVSSwitch, protocols='
↪ OpenFlow13')
21
22         # Add links
23         self.addLink(Host1, rightSwitch)
24         self.addLink(Host2, rightSwitch)
25         self.addLink(rightSwitch, leftSwitch)

```

```

26         self.addLink(Host3, leftSwitch)
27         self.addLink(Host4, leftSwitch)
28         self.addLink(Host5, leftSwitch)
29
30 topos = { 'mytopo': (lambda: MyTopo()) }
31
32 def startNetwork():
33     topo = MyTopo()
34     net = Mininet(topo=topo, switch=OVSSwitch, controller=None, autoSetMacs=
35     ↪ True)
36     net.addController('c0', controller=RemoteController, ip='127.0.0.1',
37     ↪ port=6633)
38     net.start()
39     CLI(net)
40     net.stop()
41
42 if __name__ == '__main__':
43     setLogLevel('info')

```

Listing 4.1 : Custom Topology Python Script

Executing the custom topology described earlier in Mininet is straightforward, requiring just a single command. Specifically, by saving the python script defining the topology, we launched Mininet and instantiated the custom network by the following command :

```

1 sudo mn --custom /home/yacinebelouz/Desktop/mininet/custom/MyTopo.py
   ↪ --controller=remote --topo mytopo

```

4.2.2 Real-time DDoS Detection and Mitigation Setup

Implementing an IDS for real-time DDoS detection involves a multi-step process. This process includes collecting data from network interfaces, extracting the necessary features from the collected data, and then feeding the prepared data into our pre-trained DL detection models. While, the IPS system operates by sending a flow rule to the Ryu controller based on the IDS detection to block the IP address of the attacker in the network. This flow rule is sent through the REST API of the Ryu controller and is applied to the underlying infrastructure.

4.2.2.1 Flow Collector

The Flow Collector is responsible for gathering network traffic data in real-time. This component monitors the network and collects raw data packets, which are then aggregated into flow records. Each flow record typically contains information such as source and destination IP addresses, ports, protocol, and the number of packets and bytes transferred. The key functions of the Flow Collector are :

- **Packet Capture :** We continuously capture packets from the network interfaces using CICFlowMeter. This tool not only captures packets but also extracts flow features in real-time. Simultaneously, tools like Wireshark and tcpdump were used to visualize packet data, ensuring that the flow data is being correctly captured and aggregated.
- **Data Storage as CSV File :** We temporarily store flow records in a CSV file for further processing and feature extraction. Storing flow data in a structured format like CSV facilitates manipulation and analysis through data processing libraries such as Pandas.

```

1 subprocess.run(['cicflowmeter', '-i', 's1-eth1', '-c', 'flows.csv'],
2               ↪ timeout=run_duration)
3

```

Listing 4.2 : Flow Data Collection Module

4.2.2.2 Feature Extractor

The Feature Extractor processes flow records to extract relevant features required for detecting DDoS attacks. These features are statistical and temporal metrics that characterize network behavior. Feature extraction transforms raw flow data into a format suitable for the detection algorithms.

- **Feature Calculation :** CICFlowMeter computes features such as packet count, byte count, flow duration, and protocol distribution from flow records.
- **Normalization :** We normalize features to ensure that they are on a comparable scale, which improves the performance of the detection algorithms.

```

1 from sklearn.preprocessing import MinMaxScaler
2 scaler = MinMaxScaler()
3         X_test = scaler.fit_transform(X_test)
4         X_test_reshaped = X_test.reshape(X_test.shape[0], X_test
5         ↪ .shape[1], 1)

```

Listing 4.3 : Scaling extracted features

- **Real-time Processing :** We perform these calculations in real-time to provide timely input to the detection module.

The features used for the LSTM, CNN, and CNN-LSTM models show a high degree of similarity, ensuring each model captures essential aspects of network traffic, aiding in the accurate detection of DDoS attacks. These features were extracted during data preprocessing for each model. After collecting flow data in real-time with CICFlowMeter, these features are compared to the flow features extracted by CICFlowMeter. If there is a resemblance, the analysis of the flow begins; otherwise, an error is flagged.

Common features such as *bwd_pkt_len_max*, *fwd_pkts_s*, *flow_pkts_s*, *flow_iat_min*, and *pkt_len_var* provide crucial information about packet lengths, inter-arrival times, and flow statistics. These features help the models identify abnormal patterns in network traffic, which are indicative of DDoS attacks.

The inclusion of *idle_max*, *idle_std*, *flow_iat_max*, *fwd_iat_max*, and *bwd_iat_std* ensures that the models consider variations in idle times and inter-arrival times, which can signal an ongoing attack. Additionally, features like *fwd_blk_rate_avg* and *init_bwd_win_byts* (used exclusively in the LSTM model) contribute in capturing the initial window size of backward traffic, further enhancing detection capabilities.

TABLE 4.1 : Feature Comparison for LSTM, CNN, and CNN-LSTM Models

Feature	LSTM	CNN	CNN-LSTM
bwd_pkt_len_max	✓	✓	✓
fwd_pkts_s	✓	✓	✓
flow_pkts_s	✓	✓	✓
flow_iat_min	✓	✓	✓
flow_iat_mean	✓	✓	✓
pkt_len_var	✓	✓	✓
bwd_iat_max	✓	✓	✓
idle_max	✓	✓	✓
idle_std	✓	✓	✓
flow_iat_max	✓	✓	✓
fwd_iat_max	✓	✓	✓
bwd_iat_std	✓	✓	✓
idle_mean	✓	✓	✓
fwd_iat_std	✓	✓	✓
fwd_blk_rate_avg	✓	✓	✓
idle_min	✓	✓	✓
fwd_iat_tot	✓	✓	✓
bwd_iat_tot	✓	✓	✓
init_bwd_win_byts	✓		

4.2.2.3 DL-based Detector

The DL-based Detector is the core of our IDS, leveraging DL algorithms to identify potential DDoS attacks. In our setup, we developed a different types of DL models trained on a comprehensive dataset of labeled network traffic to distinguish between benign and malicious traffic. These models are designed to provide robust detection capabilities.

- **Models Training :** The DL models are trained on historical network traffic data, which is meticulously labeled as either 'ATTACK' or 'BENIGN'. The training process incorporates techniques such as early stopping and learning rate scheduling to prevent overfitting and optimize models performance. The dataset used for training includes a diverse range of attack types and normal traffic patterns, ensuring the models ability to generalize to real-world scenarios.
- **Real-time Inference :** For real-time DDoS detection, the trained models are loaded from a JSON configuration and applied to data captured in real-time by CICFlowMeter. This process involves continuously capturing packets from the network interfaces, extracting relevant features, and normalizing these features to ensure they are on a comparable scale. The normalized features are then fed into the selected DL models for prediction, allowing the IDS to identify and respond to potential attacks promptly.
- **Loading and Compiling the Models :** The trained models are loaded and compiled with the appro-

appropriate optimizer, loss function, and performance metrics to ensure they are ready for real-time inference. Below is a snippet demonstrating the loading and compilation process :

```

1 with open('Model.json', 'r') as json_file:
2     loaded_model_json = json_file.read()
3 loaded_model = keras.models.model_from_json(loaded_model_json)
4 loaded_model.load_weights('Model.weights.h5')
5 loaded_model.compile(optimizer='adam', loss='binary_crossentropy',
    ↪ metrics=['accuracy'])

```

Listing 4.4 : Loading and Compiling the CNN Model

- **Prevention Display:** The IDS features a prevention display that provides real-time notifications about the network status. For each IP address, the IDS indicates whether the traffic is identified as an 'ATTACK' or 'BENIGN'. The display updates continuously as new data is analyzed, ensuring that the network status is always current.

4.2.2.4 Mitigation Module

The mitigation module (Fig 4.5) in our project is designed to dynamically block malicious traffic by modifying the network's flow rules in response to detected attacks. Once a potential attack is identified, the mitigation module included in the IPS swiftly acts to mitigate its impact on the network. The IDS extracts the IP address of the attacker, ensuring that threats are promptly addressed to maintain network integrity and performance. The functioning of this module can be explained through the following steps :

- **Automated Response :** The mitigation module employs automated responses to detected threats, ensuring that actions are taken promptly to minimize network disruption.
- **Flow Rule Modification :** The core function of the mitigation module is to modify flow rules on the network switches to block traffic from identified malicious IP addresses. The function constructs a flow rule specifying the conditions under which the traffic should be blocked. This includes :
 - **dpid :** The Data Path ID (DPID) of the switch where the rule will be applied.
 - **table_id :** The table within the switch where the rule will be placed.
 - **priority :** The priority of the rule, determining its precedence over other rules.
 - **match :** The conditions that match the traffic to be blocked, specifically targeting the malicious IP address and Ethernet type.
 - **instructions :** The actions to be taken on matching traffic, in this case, to drop the packets.

The function sends this flow rule to the Ryu controller using its REST API endpoint. If the rule is successfully added, it confirms that traffic from the specified IP address is blocked; otherwise, it logs the failure for further analysis.

- **Adaptive Response :** The mitigation module continuously monitors the effectiveness of the applied flow rules and adapts them as necessary. This adaptive approach ensures ongoing protection against evolving threats.

The system's effectiveness and reliability are crucial for maintaining network security and preventing disruption caused by DDoS attacks.

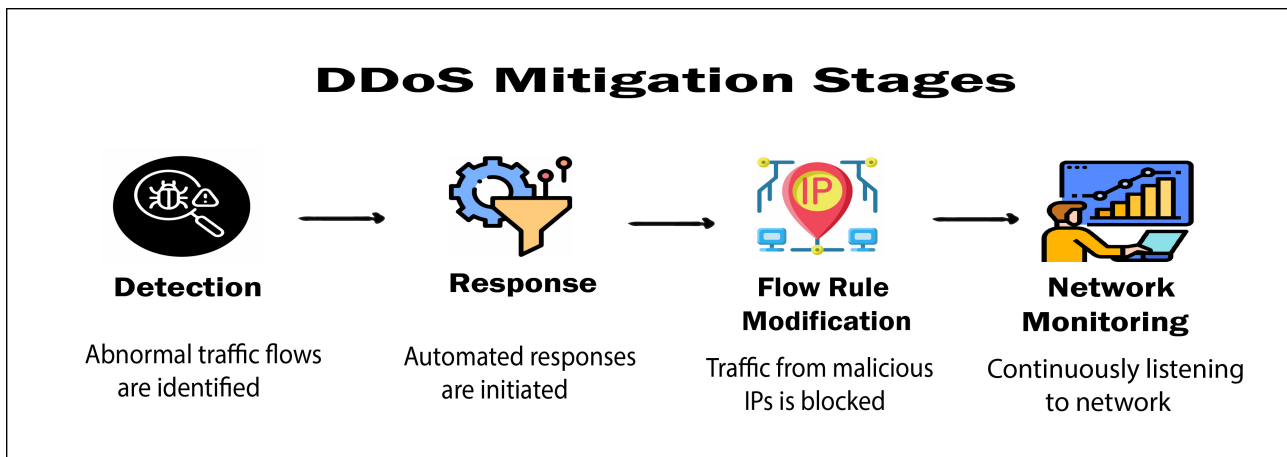


FIGURE 4.2 : Mitigation Module Stages

4.2.3 Implementation of Multithreading in YOUD Ensemble Model

In our implementation, each network interface is monitored by a separate thread. This allows for concurrent processing and analysis of network traffic across multiple interfaces. The following code snippet demonstrates how we implemented multithreading in our YOUD ensemble model :

Listing 4.5 : Multithreading in YOUD Ensemble Model

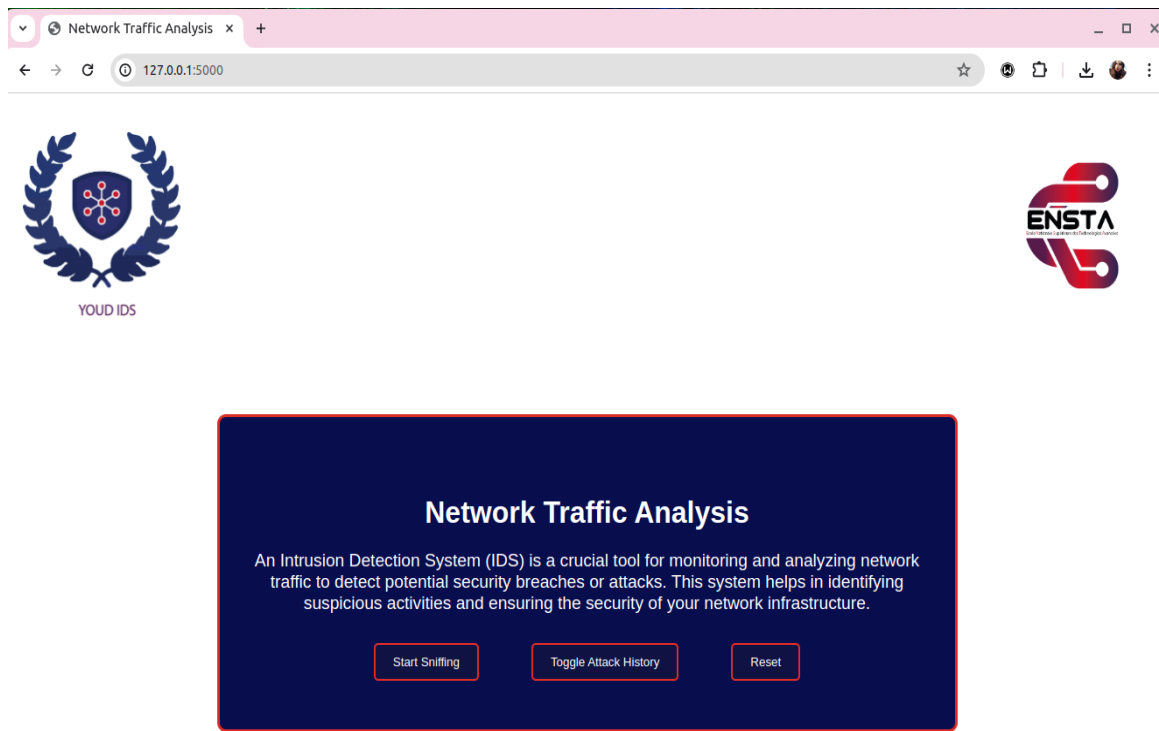
```

1  import threading
2
3  # Define the function to monitor network traffic
4  def monitor_interface(interface):
5      # Code to capture and analyze traffic on the specified interface
6      pass
7
8  # List of network interfaces to monitor
9  interfaces = ['s1-eth1', 's1-eth2', 's1-eth3', 's2-eth1', 's2-eth2', '
    ↪ s2-eth3', 's2-eth4']
10
11 # Create and start a thread for each interface
12 threads = []
13 for interface in interfaces:
14     thread = threading.Thread(target=monitor_interface, args=(
    ↪ interface,))
15     thread.start()
16     threads.append(thread)
17
18 # Wait for all threads to complete
19 for thread in threads:
20     thread.join()

```

4.2.4 Implementation of The Flask Application

The Flask application serves as the front-end interface for real-time monitoring and detection of DDoS attacks using the YOUD model. The application includes several interactive elements to facilitate user interaction and provide real-time feedback on network security status.



Darsaf & Yacine

FIGURE 4.3 : Flask Application Plateform

User Interface Elements :

The key components of the Flask application are as follows :

- **Start Sniffing Button** : This button initiates the IDS monitoring process across the specified network interfaces. When the user clicks this button, the Flask application starts capturing and analyzing network traffic in real-time, utilizing the trained YOUD model to detect potential threats.
- **Toggle Attack History Button** : This button allows users to view and hide the attack log. The attack log records all detected threats, including the source IP addresses, and the number of attack flows. This feature enables users to review past detections and gain insights into the network's security events.
- **Reset Button** : This button clears the current log of detected threats, allowing the user to start fresh with new detections. This is particularly useful for monitoring sessions where the user needs to reset the detection data and start a new analysis without the interference of previous logs.
- **Logo** : The application prominently displays a logo in the top-left corner of the web page, reinforcing the branding and purpose of the IDS system. The logo enhances the visual appeal of the interface and provides an immediate visual identifier for the application.

Backend Processing and Multithreading :

The Flask application is designed to handle multiple network interfaces concurrently, ensuring comprehensive coverage and real-time responsiveness. The following elements outline the backend processing :

1. **Model Loading** : The YOUD model, including LSTM, 1D-CNN, and hybrid CNN-LSTM models, is loaded into the Flask application for DDoS attack detection.
2. **Traffic Capture and Analysis** : The application uses 'cicflowmeter' to capture network traffic and analyze it with the YOUD model to identify threats.
3. **Multithreading** : The application employs multithreading to monitor multiple network interfaces concurrently, ensuring efficient and real-time threat detection.
4. **Data Visualization** : Real-time visual feedback is provided, showing traffic analysis results by IP address, benign and attack traffic percentages, and detailed attack information.
5. **Flow Rule Modification** : Upon detecting an attack, the application modifies flow rules via the network controller to block malicious IP addresses, mitigating ongoing attacks.

4.3 Models Training and Evaluation

In our approach, we begin by ensuring that the dataset is appropriately divided into training and testing sets, maintaining the original class distribution. We employ a stratified shuffle split with a 70-30 ratio, meaning 70% of the data is allocated for training and 30% for testing. This method ensures that each class is proportionally represented in both the training and testing sets, which is crucial for training an unbiased model. The input data is then reshaped to include an additional dimension, aligning with the required input format for each model.

After training, we evaluate the models performances on the testing set. Metrics such as accuracy, confusion matrix, and classification report are used to assess the models effectiveness in distinguishing between benign and attack traffic. Visualizations, including loss and accuracy curves provide insights into the models training dynamics and classification performance.

4.3.1 Hyperparameters Tuning

In optimizing our models for detecting DDoS attacks in SDN environments, a thorough hyperparameters tuning process was conducted. This involved fine-tuning parameters like number of epochs and optimizer selection within each model including 1D-CNN, LSTM, and hybrid CNN-LSTM models. By meticulously adjusting these settings, we aimed to maximize the models performances in accurately identifying DDoS threats. The resulting ensemble model called YOUD, formed through a majority voting system, is capitalized on the optimized configurations of each model, culminating in a robust and effective defense mechanism against DDoS attacks within SDN infrastructures.

1. Number of Epochs :

The table 4.2 presents the performance metrics of the three models (1D-CNN, LSTM, and CNN-LSTM) across different epochs (50, 75, and 100). The evaluation metrics include accuracy, precision, recall, and F1 score. Notably, the results are recorded for the Adam optimizer.

1D-CNN Model

TABLE 4.2 : Performance Metrics of Different Models with Varying Epochs

Model	Epochs	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
1D-CNN	50	92%	87.36%	98.42%	92.56%
	75	93.98%	89.04%	96.87%	92.81%
	100	92.92%	87.37%	98.42%	92.56%
LSTM	50	94.26%	95%	95%	95%
	75	95%	95%	95%	94%
	100	98.46%	98%	98%	98%
CNN-LSTM	50	93.72%	96.20%	88.99%	92.44%
	75	92.34%	94.02%	90.2%	92%
	100	92.12%	92.3%	86.95%	91.2%

The 1D-CNN model achieved its highest performance at 75 epochs with an accuracy of 93.98%, precision of 89.04%, recall of 96.87%, and F1 score of 92.81%. However, its performance was relatively consistent across different epochs.

LSTM Model

The LSTM model demonstrated the best overall performance at 100 epochs, achieving an accuracy of 98.46%, precision, recall, and F1 score of 98%. This indicates its strong capability in accurately identifying both attack and benign traffic.

CNN-LSTM Model

The CNN-LSTM model showed competitive performance but did not outperform the LSTM model. Their best metrics were at 50 epochs with an accuracy of 93.72%, precision of 96.20%, recall of 88.99%, and F1 score of 92.44%.

2. Optimizers

The results presented previously are obtained using Adam optimizer. Hence, we would like to evaluate our DL models learning using another optimizer. We have chosen RMSProp optimizer known by its ability to adapt learning rates dynamically for each parameter.

TABLE 4.3 : Evaluation Metrics for 1D-CNN and LSTM Models with RMSProp

Metric	CNN with RMSProp	LSTM with RMSProp
Accuracy	79.16%	90.50%
Precision	85.13%	88.76%
Recall	82.88%	89.57%
F1 Score	73.75%	90.62%

The table 4.3 presents the evaluation metrics for both the best 1D-CNN model and the optimal LSTM model using the RMSProp optimizer. Overall, the metrics suggest that the RMS optimizer is not as effective for the 1D-CNN model or even the LSTM model, resulting in lower accuracy and precision. This proves that the Adam optimizer is the best choice for training our models.

4.3.2 1D-CNN Model

Training and Validation Accuracy :

The model underwent training for 75 epochs with the use of Adam optimizer, yielding optimal results. As depicted in figure 4.4, the training and validation accuracy curves of the 1D-CNN model are represented. The training accuracy shows a consistent upward trajectory, stabilizing at high values from around the 40th epoch, indicating effective learning from the training data. In contrast, the validation accuracy exhibits significant fluctuations throughout the training process, suggesting variability in the model's performance on unseen data. Despite these fluctuations, the overall trend shows improvement, with the validation accuracy converging to values slightly higher than the training accuracy in the final epochs, indicating good generalization capabilities.

Notably, The pronounced spikes and drops in validation accuracy, particularly between the 0th and 40th epochs, could be due to the inherent complexity of the validation dataset. Nevertheless, the general upward trend in both training and validation accuracy indicates the model's increasing effectiveness in classifying data over time.

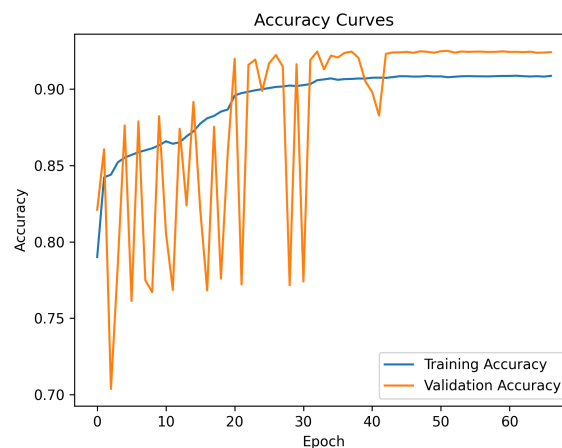


FIGURE 4.4 : 1D-CNN Training and Validation Accuracies

Training and Validation Loss :

As illustrated in figure 4.5, the training and validation loss curves of the 1D-CNN model are delineated. The training loss exhibits a consistent downward trajectory, indicating successful learning throughout the training epochs. This gradual decrease suggests that the model effectively minimizes errors on the training dataset.

The validation loss shows more pronounced fluctuations compared to the training loss, which can be attributed to variability in the validation data batches. Despite these fluctuations, the overall trend of the validation loss is downward, closely mirroring the training loss trajectory. This indicates that the model not only learns from the training data but also generalizes well to unseen data, despite some variability during training.

From around the 40th epoch onwards, both the training and validation losses stabilize, with the validation loss settling into a lower range. This stabilization indicates that the model has reached a point where further training does not significantly reduce the loss, implying convergence and effective learning.

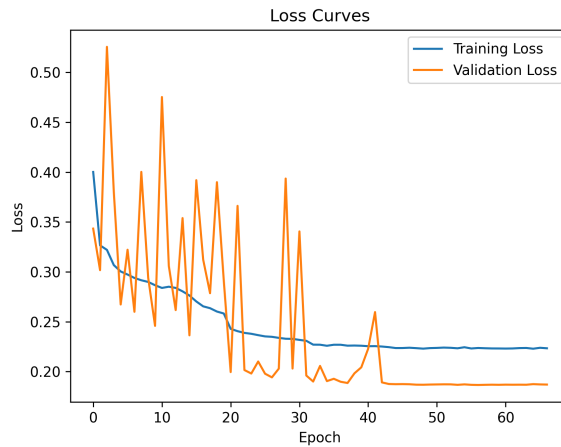


FIGURE 4.5 : 1D-CNN Training and Validation Loss

Confusion Matrix and Model Performance :

The confusion matrix presented in figure 4.6 provides a clear representation of the CNN model's accuracy in distinguishing between benign and attack classes. The confusion matrix includes the following components :

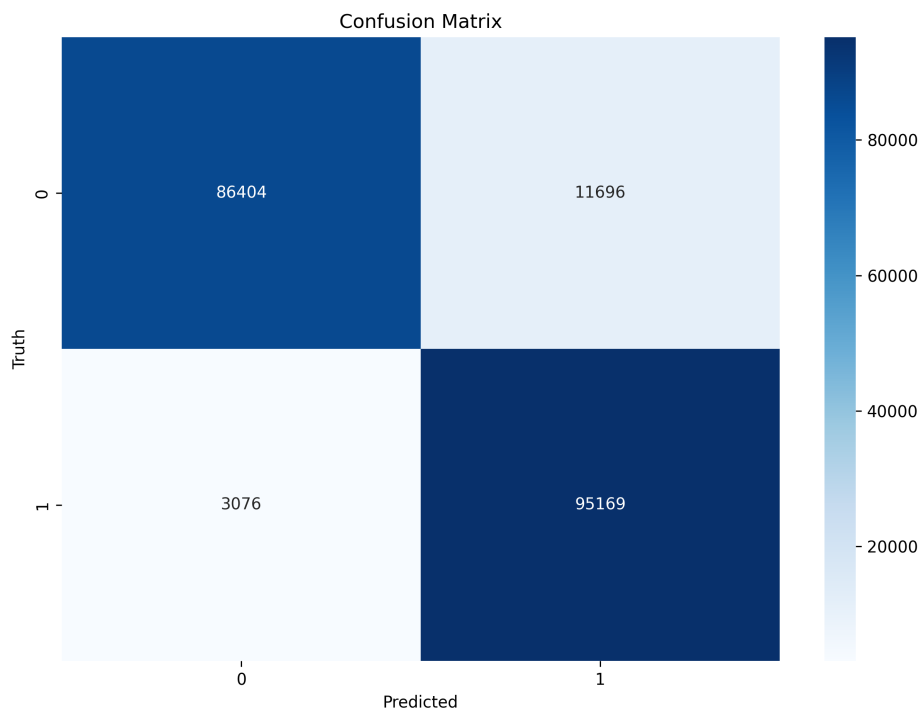


FIGURE 4.6 : 1D-CNN Model Confusion Matrix

Interpretation :

- **True Positives (TP)** : Instances where the model correctly predicts the positive class. In the matrix, this is the bottom-right cell with a value of 95,169. This means that the model correctly identified 95,169 positive instances.

- **True Negatives (TN)** : Instances where the model correctly predicts the negative class. This is shown in the top-left cell with a value of 86,404. This indicates that the model accurately identified 86,404 negative instances.
- **False Positives (FP)** : Instances where the model incorrectly predicts the positive class when it is actually negative. This is represented by the top-right cell with a value of 11,696. This means that the model incorrectly classified 11,696 negative instances as positive.
- **False Negatives (FN)** : Instances where the model incorrectly predicts the negative class when it is actually positive. This is shown in the bottom-left cell with a value of 3,076. This means that the model mistakenly classified 3,076 positive instances as negative.

The final evaluation metrics for the CNN model are summarized in table 4.4. These results indicate the model's effectiveness in distinguishing between the two classes.

TABLE 4.4 : Overall Performance Metrics of the CNN Model

Model	Metric	Value
CNN	Accuracy	93.98%
	Precision	89.04%
	Recall	96.87%
	F1 Score	92.81%

In summary, the CNN model demonstrates strong accuracy, precision, recall, and F1 Score, indicating its high effectiveness in correctly classifying both classes.

4.3.3 LSTM Model

Training and Validation Accuracy :

The training and validation accuracy curves for the LSTM model are shown in the figure 4.7. The curves show that both training and validation accuracies increased steadily and eventually converged to a high value, indicating that the model generalized well to unseen data.

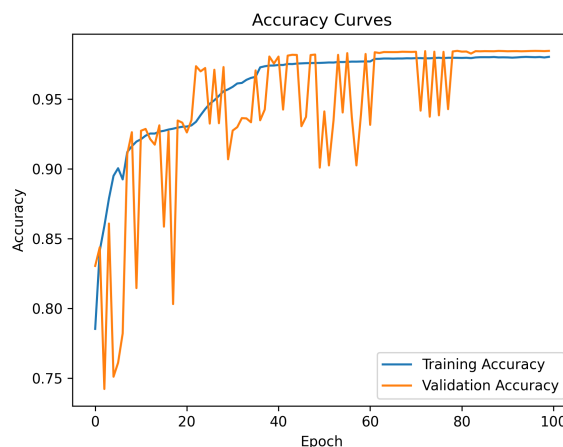


FIGURE 4.7 : LSTM Model Training and Validation Accuracy

Training and Validation Loss :

The training and validation loss curves for the LSTM model are shown in the figure 4.8. The loss curves demonstrate a significant reduction in loss over the training epochs, which suggests effective learning and minimal overfitting. The validation loss fluctuates but overall trends downwards, aligning closely with the training loss.

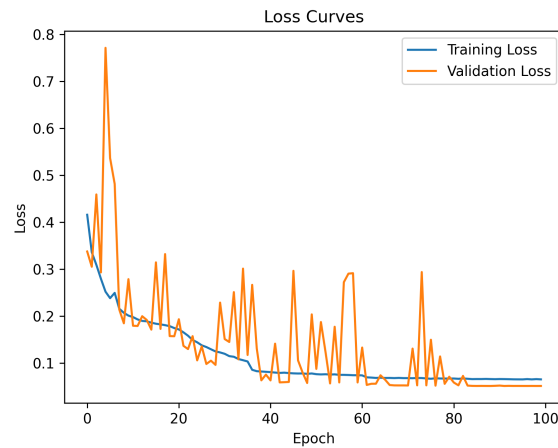


FIGURE 4.8 : LSTM Model Training and Validation Loss

Confusion Matrix and Model Performances :

The confusion matrix presented in figure 4.9 is specified for the LSTM model used for DDoS attack detection, it provides a clear representation of the model's accuracy and performances in distinguishing between benign and attack traffic.

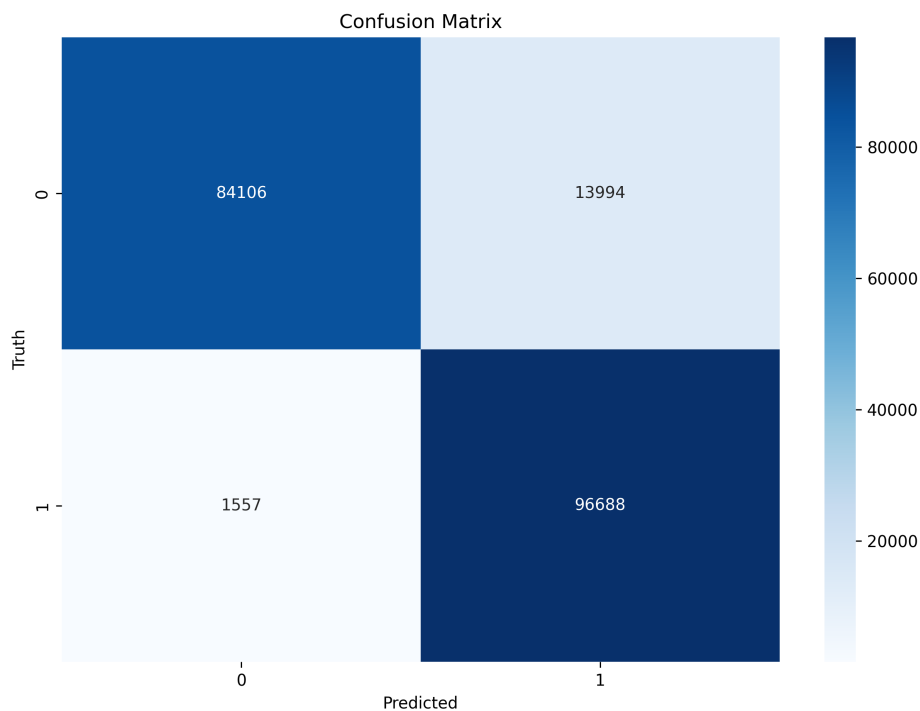


FIGURE 4.9 : LSTM Model Training and Validation Loss

The final evaluation metrics for the LSTM model are summarized in Table 4.5. The model achieved an accuracy of 98.46%, with precision, recall, and F1-score values of 98%. These results indicate the model's high effectiveness in distinguishing between benign and attack traffic.

TABLE 4.5 : Overall Performance Metrics of the LSTM Model

Model	Accuracy	Precision	Recall	F1-Score
LSTM	98.46 %	98%	98%	98%

4.3.4 Hybrid CNN-LSTM Model

Training and Validation Accuracy :

As depicted in figure 4.10, the training accuracy of the hybrid CNN-LSTM shows a consistent upward trajectory, stabilizing at high values from around the 15th epoch, indicating effective learning from the training data. While the validation accuracy exhibits significant fluctuations throughout the training process. However, despite these fluctuations, the overall trend shows improvement, with the validation accuracy converging to values slightly higher than the training accuracy in the final epochs, indicating good generalization capabilities.

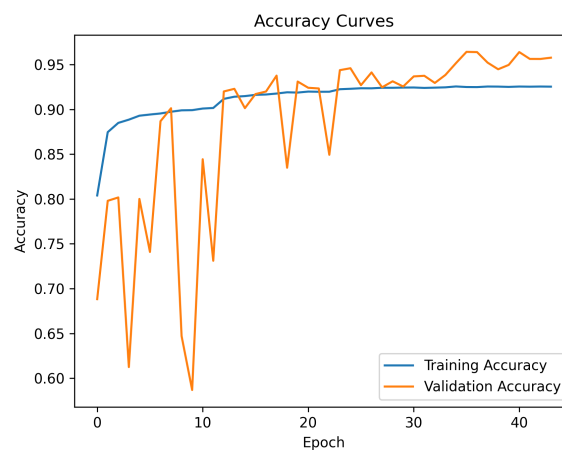


FIGURE 4.10 : Hybrid CNN-LSTM training and validation accuracy

Training and Validation Loss :

As showed in figure 4.11, the training and validation loss curves of the hybrid CNN-LSTM model are represented. The training loss exhibits a consistent downward trajectory, indicating successful learning throughout the training epochs. This gradual decrease suggests that the model effectively minimizes errors on the training dataset.

The overall trend of the validation loss is downward, closely mirroring the training loss trajectory. This indicates that the model not only learns from the training data but also generalizes well to unseen data, despite some variability during training.

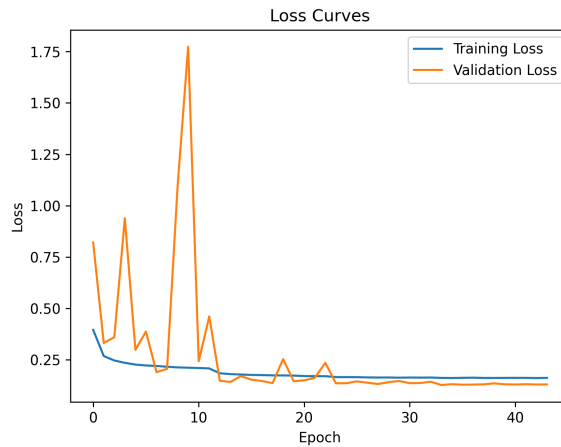


FIGURE 4.11 : Hybrid CNN-LSTM training and validation loss

Confusion Matrix and Model Performances :

The confusion matrix presented in figure 4.12 provides a clear representation of the hybrid CNN-LSTM model's accuracy and performances in distinguishing between the two classes. Overall, it demonstrates strong performance, with a high number of correct classifications and relatively few misclassifications.

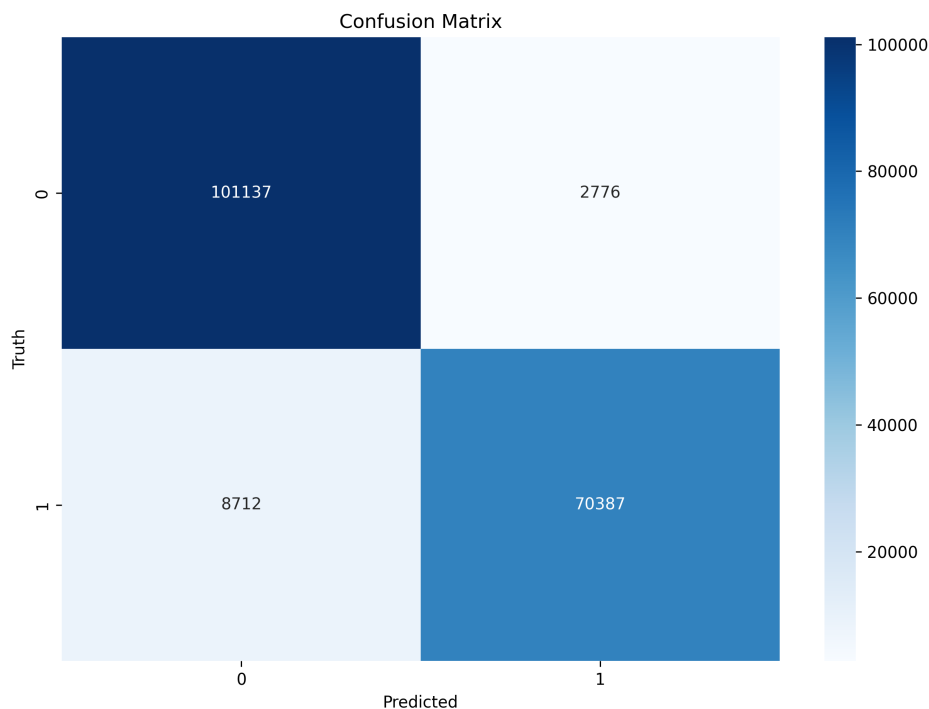


FIGURE 4.12 : Hybrid CNN-LSTM Model Confusion Matrix

The final evaluation metrics for the hybrid CNN-LSTM model are summarized in table 4.6. These results indicate its effectiveness in distinguishing between benign and malicious traffic.

In summary, the hybrid CNN-LSTM model demonstrates strong accuracy, precision, recall, and F1 Score, indicating its high effectiveness in correctly classifying both classes.

TABLE 4.6 : Overall Performance Metrics of the CNN-LSTM Model

Model	Metric	Value
CNN-LSTM	Accuracy	93.72%
	Precision	96.20%
	Recall	88.99%
	F1 Score	92.44%

4.4 IDS Results in Real-Time with Different DL Trained Models

In this section, we will explore the real-time detection capabilities of our IDS using various DDoS attack tools. The primary objective of our IDS is to monitor network traffic in real-time and detect potential security threats. We will examine the real-time results and effectiveness of CNN, LSTM, and hybrid CNN-LSTM models in correctly detecting DDoS attacks in SDN based networks. In our test scenario, the host with IP address **10.0.0.1** was attacked by the host with IP address **10.0.0.2** for CNN and Hybrid models. For LSTM and YOUNG models, the host with IP address **10.0.0.1** was attacked by the host with IP address **10.0.0.4**.

4.4.1 1D-CNN Model Results

The results of the CNN model's detection of DDoS attacks and benign traffic are displayed in table 4.7.

TABLE 4.7 : Detection Results of The CNN Model on Attack and Benign Traffic

Attack Type	Percentage of Attack (%)	Percentage of Benign (%)
Hulk	84.93	15.07
Slow Body	77.51	22.49
Slow Read	93.17	6.67
Slowloris (slow Headers)	90.17	9.83
Range	96.99	3.01
Benign Traffic	9.83	90.17

Detection of Range Attack :

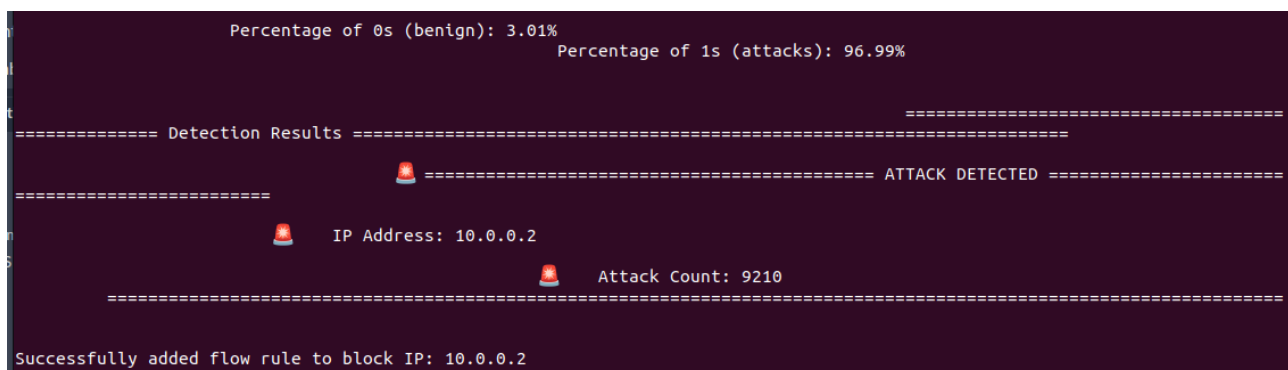


FIGURE 4.13 : CNN Detection Results in Terminal

The detection results are displayed in the terminal (4.13), where the system identified and flagged traffic originating from **10.0.0.2** as an attack :

- The percentage of mistaken detections, where attack traffic was incorrectly identified as benign, is **3.01%**.
- The percentage of correct detections, where attack traffic was accurately identified, is **96.99%**. It accurately detected and counted **9210** attack flows.
- The output shows the successful addition of a flow rule to block the malicious IP address, thereby mitigating the ongoing attack.

Interpretation :

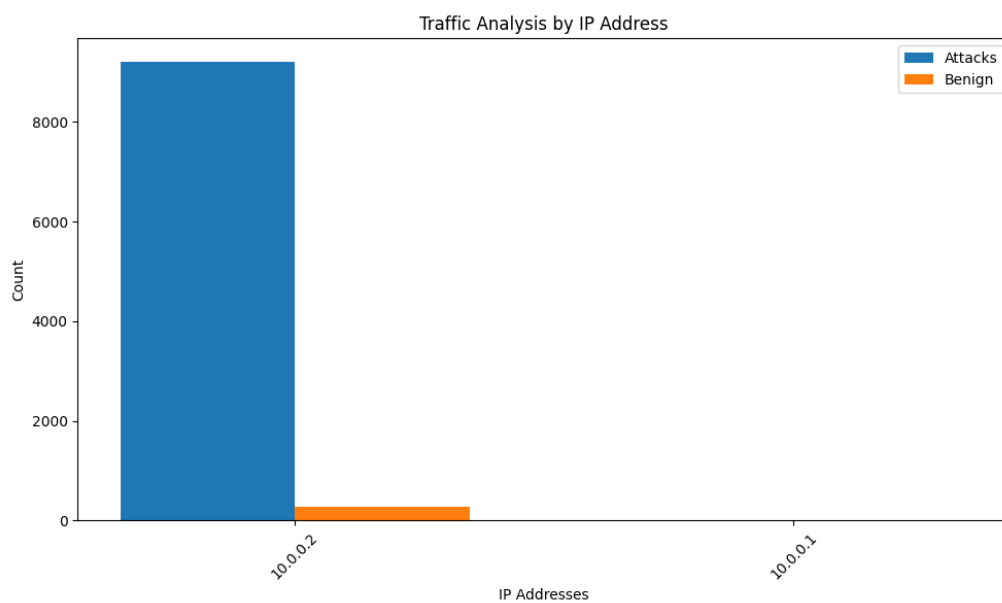


FIGURE 4.14 : CNN Traffic Analysis

The figure 4.14 illustrates the traffic analysis based on IP addresses, displaying the count of times Slowhttp-test Range attack was correctly detected by the CNN model as well as the instances that were mistakenly identified as benign traffic. The bar chart shows the distribution of detected traffic, highlighting that the CNN model successfully identified the majority of the attack traffic originating from IP address **10.0.0.2**, which was aimed at the target IP address **10.0.0.1**.

4.4.2 LSTM Model Results

The detection results of the LSTM model for DDoS attacks and legitimate traffic are outlined and summarized in table 4.8.

TABLE 4.8 : Detection Results of The LSTM Model on Attack and Benign Traffic

Attack Type	Percentage of Attack (%)	Percentage of Benign (%)
Hulk	86.89	13.11
Slow Body	85.37	14.63
Slow Read	93.45	6.55
Range	94.83	5.17
Slowloris (slow Headers)	86.87	13.13
Benign Traffic	4.76	95.24

Detection of Range Attack :



FIGURE 4.15 : LSTM Detection Results in Terminal

The detection results are displayed in the terminal (4.15), where the IDS identified and flagged traffic originating from IP address **10.0.0.4** as an attack :

- The percentage of erroneous detections, where attack traffic was falsely identified as benign, is **5.17%**.
- The percentage of correct detections, where attack traffic was perfectly identified, is **94.83%**. Specifically, it accurately detected and counted **8216** attack flows.
- The output confirms that a flow rule has been successfully added to block the malicious IP address, effectively mitigating the ongoing attack.

Interpretation :

The figure 4.16 highlights the traffic analysis during a Slowhttptest range attack based on IP addresses, displaying the count of attack and benign traffic detected by the LSTM model.

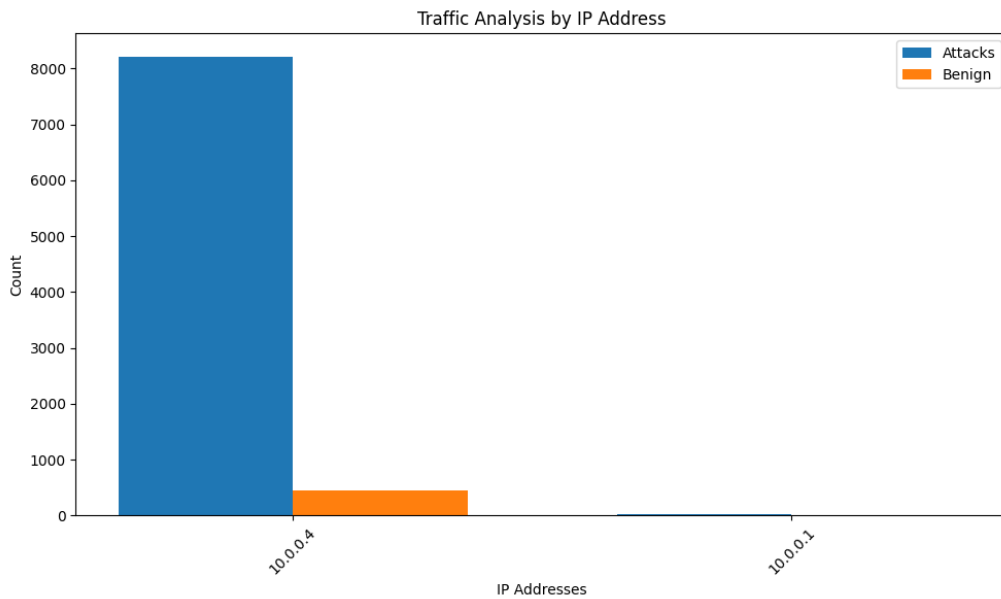


FIGURE 4.16 : LSTM Traffic Analysis

This visualization highlights that the LSTM model successfully identified the majority of the attack traffic originating from IP address **10.0.0.4**, aimed at the target IP address **10.0.0.1**.

4.4.3 Hybrid CNN-LSTM Model Results

The detection results of the hybrid CNN-LSTM model for DDoS attacks and benign traffic are presented and summarized in table 4.9.

TABLE 4.9 : Detection Results of The CNN-LSTM Model on Attack and Benign Traffic

Attack Type	Percentage of Attack (%)	Percentage of Benign (%)
Hulk	85.99	14.01
Slow Body	68.18	31.82
Slow Read	94.68	5.32
Slowloris (slow Headers)	76.03	23.97
Range	93.67	6.33
Benign Traffic	2.33	97.67

Detection of Range Attack :

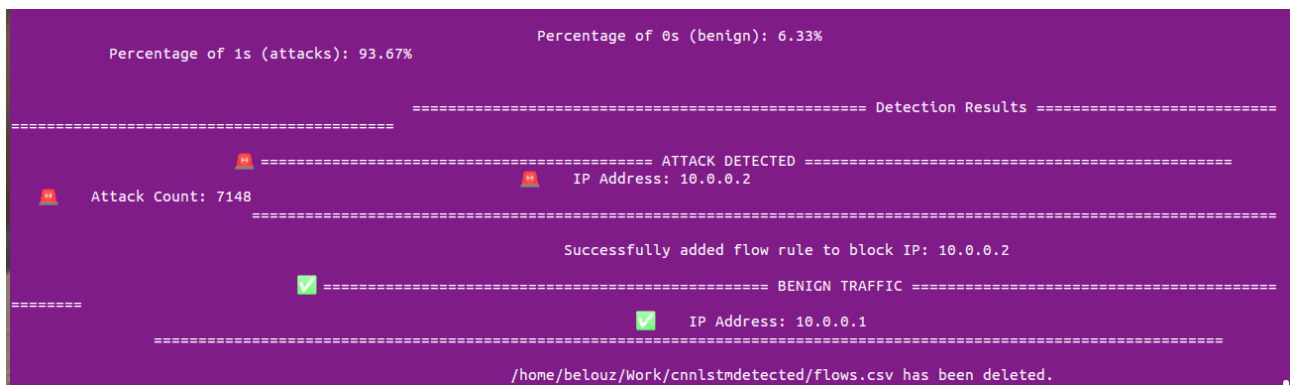


FIGURE 4.17 : Hybrid CNN-LSTM Detection Terminal Output

The detection results are displayed in the terminal (4.17), where the IDS identified and flagged traffic originating from IP address **10.0.0.2** as an attack :

- The percentage of mistaken detections, where attack traffic was incorrectly identified as benign, is **6.33%**.
- The percentage of correct detections, where attack traffic was accurately identified, is **93.67%**. It accurately detected and counted **7148** attack flows.
- The output confirms that a flow rule has been successfully added to block the malicious IP address, effectively mitigating the ongoing attack.

Interpretation :

The figure 4.18 illustrates the traffic analysis based on IP addresses, displaying the count of attack and benign traffic detected by the hybrid CNN-LSTM model.

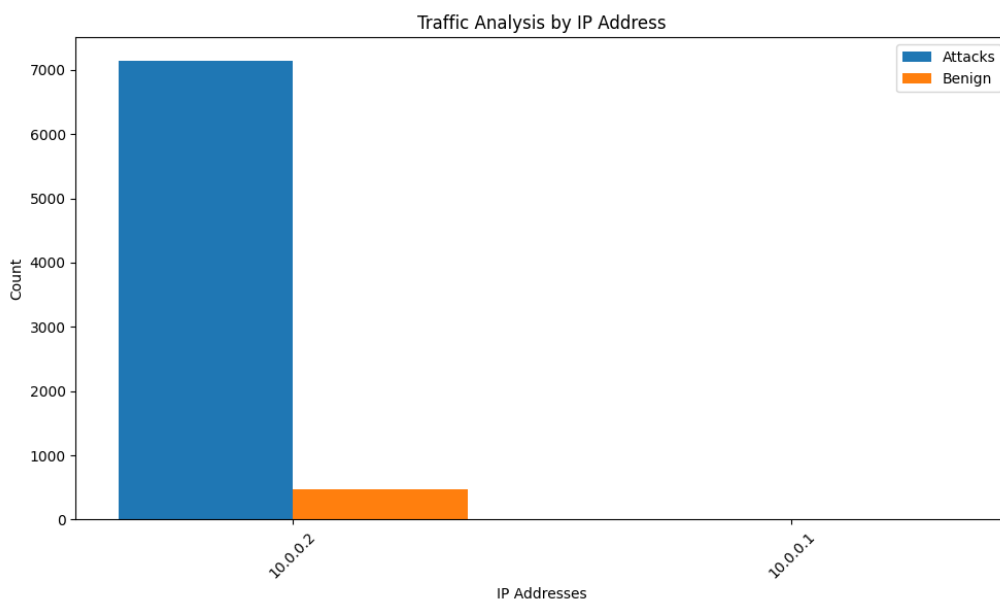


FIGURE 4.18 : Hybrid CNN-LSTM Traffic Analysis

This visualization highlights that the hybrid CNN-LSTM model successfully identified the majority of the attack traffic originating from IP address **10.0.0.2**, aimed at the target IP address **10.0.0.1**.

4.4.4 Benign Traffic Detection Performance

In this test, we have sent only legitimate traffic to assess our models' ability to distinguish between real benign traffic and attacks.

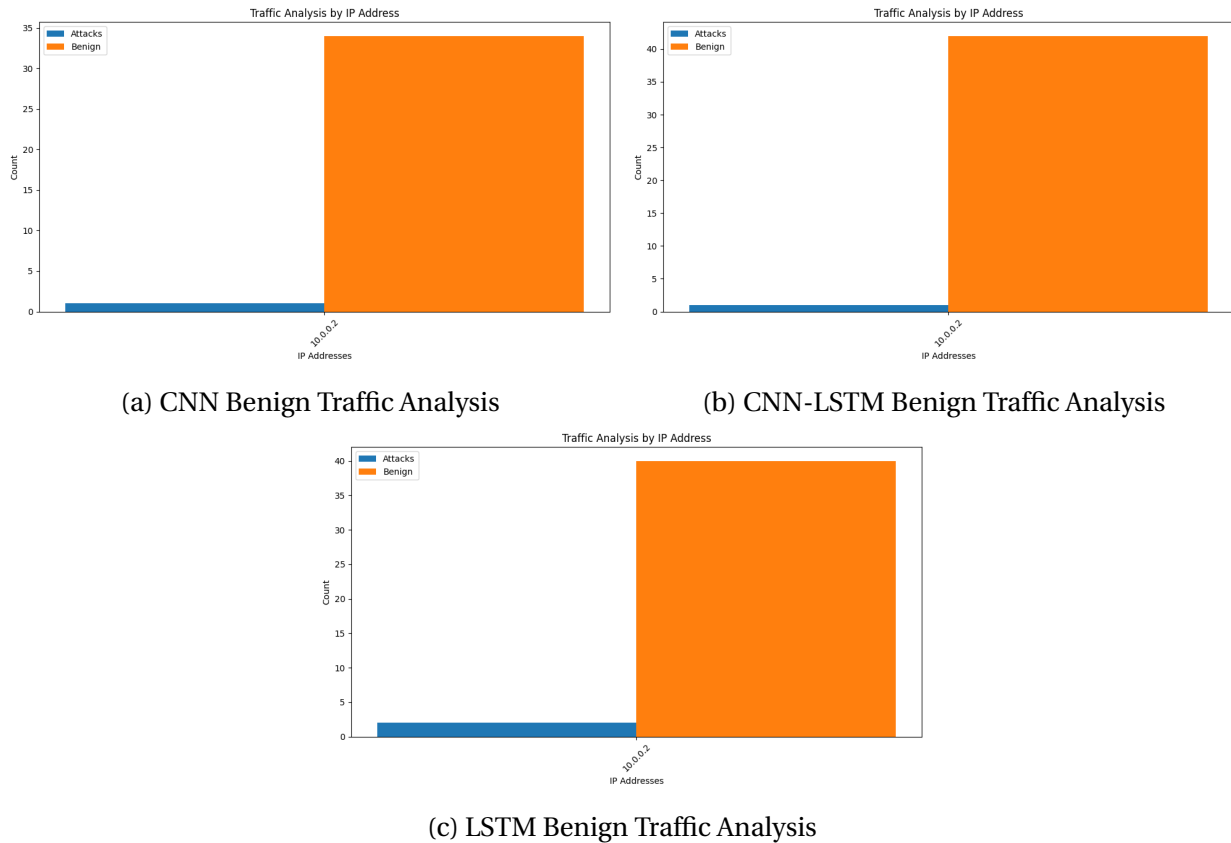


FIGURE 4.19 : Analysis of Benign Traffic Detection by Different Models

The figures 4.19a, 4.19b and 4.19c present analyses of traffic based on IP addresses, showing the frequency with which legitimate traffic was accurately classified as benign by the CNN, CNN-LSTM a LSTM models respectively. The bar charts depict the distribution of detected traffic, indicating that each model effectively identified the majority of legitimate traffic from various IP addresses. Only a small portion of the benign traffic was incorrectly flagged as an attack. This highlights the models' precision in differentiating between attack and non-attack traffic.

4.5 YOUD Model Results

In this section, we present the results of our ensemble model, YOUD, which combines the strengths of three distinct models : CNN, LSTM, and the hybrid CNN-LSTM. This ensemble model was developed to leverage the individual capabilities of each model, combining their predictions using an average voting method to enhance detection accuracy in real-time scenarios. The detection results of our ensemble model for DDoS attacks are presented and summarized (Table 4.10) as follows :

TABLE 4.10 : Performance metrics of YOUD model on different attack types

Attack Type	Percentage of Attack (%)	Percentage of Benign (%)
Hulk	88.6	11.4
Slow Body	86.26	13.74
Slow Read	99.67	0.33
Slowloris (slow Headers)	89.33	10.67
Range	98.21	1.79
Benign Traffic	2.33	97.67

This table demonstrates that the YOUD ensemble model successfully combined the strengths of the CNN, LSTM, and CNN-LSTM models in real-time detection of DDoS attacks. It achieved the highest accuracies in real-time detection, outperforming all three individual models.

4.5.1 Detection of Slow Read Attack

The YOUD model achieved a high accuracy of **99.67%** in detecting slow read attacks in real-time. Figure 4.20 illustrates the detection results, highlighting how the prevention of attacks is flagged.

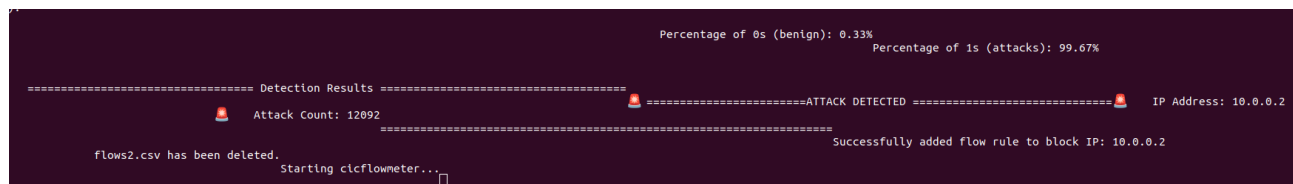


FIGURE 4.20 : YOUD Detection Terminal Output

The detection results of the YOUD model (4.20) indicate the presence of an attack from the IP address **10.0.0.4**. The attack count was high enough to trigger the flow rule modification, that effectively blocked the malicious traffic.

4.5.2 Interpretation

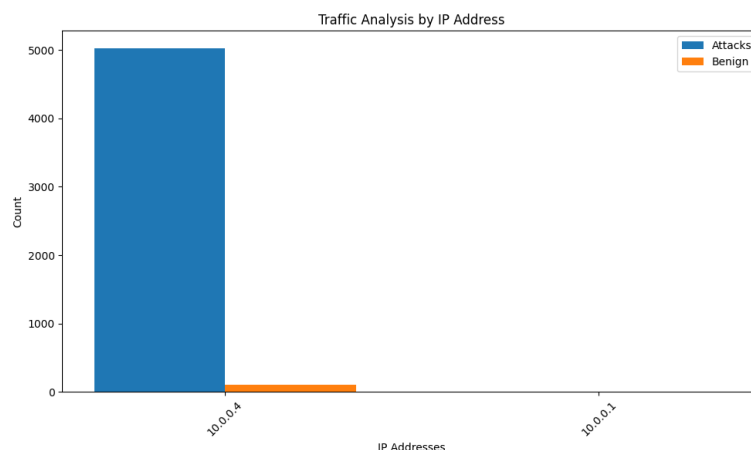


FIGURE 4.21 : Attack Traffic Analysis by IP Address

The bar chart presented in figure 4.21 illustrates the distribution of detected traffic from the IP addresses. It shows that the majority of Attack traffic was identified as an attack originated from the host with the IP address **10.0.0.4**, targeting the host with the IP address **10.0.0.1**.

The YOUD ensemble model exhibited high accuracy in detecting slowhttptest slow read attack in real time, effectively differentiating between malicious and benign traffic. The successful application of flow rules to block attacking IP addresses in real-time further confirms the system's ability to mitigate such attacks.

4.5.3 Discussion

The YOUD ensemble model outperformed individual CNN, LSTM, and hybrid CNN-LSTM models in detecting different type of DDoS attacks. By leveraging an average voting method to combine predictions from these models, YOUD achieved the highest accuracies, significantly enhancing the robustness of our IDS in real-time scenarios. Additionally, the mitigation module effectively blocked malicious traffic, as demonstrated by CICflowmeter's inability to capture further traffic from the attacking IP addresses post-detection and mitigation.

By combining the strengths of LSTM, CNN, and CNN-LSTM models, the ensemble approach enhanced the robustness and reliability of the IDS, providing comprehensive protection against network threats.

4.6 Results of The YOUD Model with Flask Application

The Flask application showcases real-time detection results for each monitored network interface within our custom topology on a user-friendly web page.

4.6.1 Display Explanation

Our Flask based application presents real-time detection results in a structured and readable format. Each monitored interface is shown in a separate block with the following details :

The results provided by our Flask application demonstrate the effectiveness of our Intrusion Detection System (IDS) in real-time network traffic analysis and attack detection.

Display Explanation

The interface of our Flask application shows critical information about network traffic across different interfaces. It includes :

- **Interface** : The specific network interface being monitored (e.g., s1-eth2).
- **IP Addresses** : The IP addresses associated with the monitored traffic.
- **Benign Traffic** : The percentage of traffic identified as normal or non-malicious.
- **Attack Traffic** : The percentage of traffic detected as malicious.

For each interface, the detection results provide :

- **Detection Status** : Indicates whether an attack has been detected.
- **IP Address** : The IP address from which the attack originated.
- **Attack Count** : The number of attack packets detected.

4.6.2 Real-time Detection with Flask

The figure 4.22 provides a clear visualization of the attack detection results for the network interfaces s1-eth1 and s1-eth2 within our Flask based graphical interface.

For s1-eth1, the system effectively monitored the network traffic and reported that there is an attack on this interface. The detection results indicate an attack originating from the IP address 10.0.0.2, with a concerning count of 24,726 attack flows. This demonstrates the accuracy and reliability of our YOUD model in distinguishing between normal and malicious traffic, ensuring that significant security threats are promptly identified and reported.

In contrast, the results for s1-eth2 further emphasize the system's robust attack detection capabilities. An attack was detected from the same IP address 10.0.0.2 originating from this interface. This highlights the efficacy of our detection system in identifying multiple points of attack within the network, ensuring comprehensive security monitoring and response.

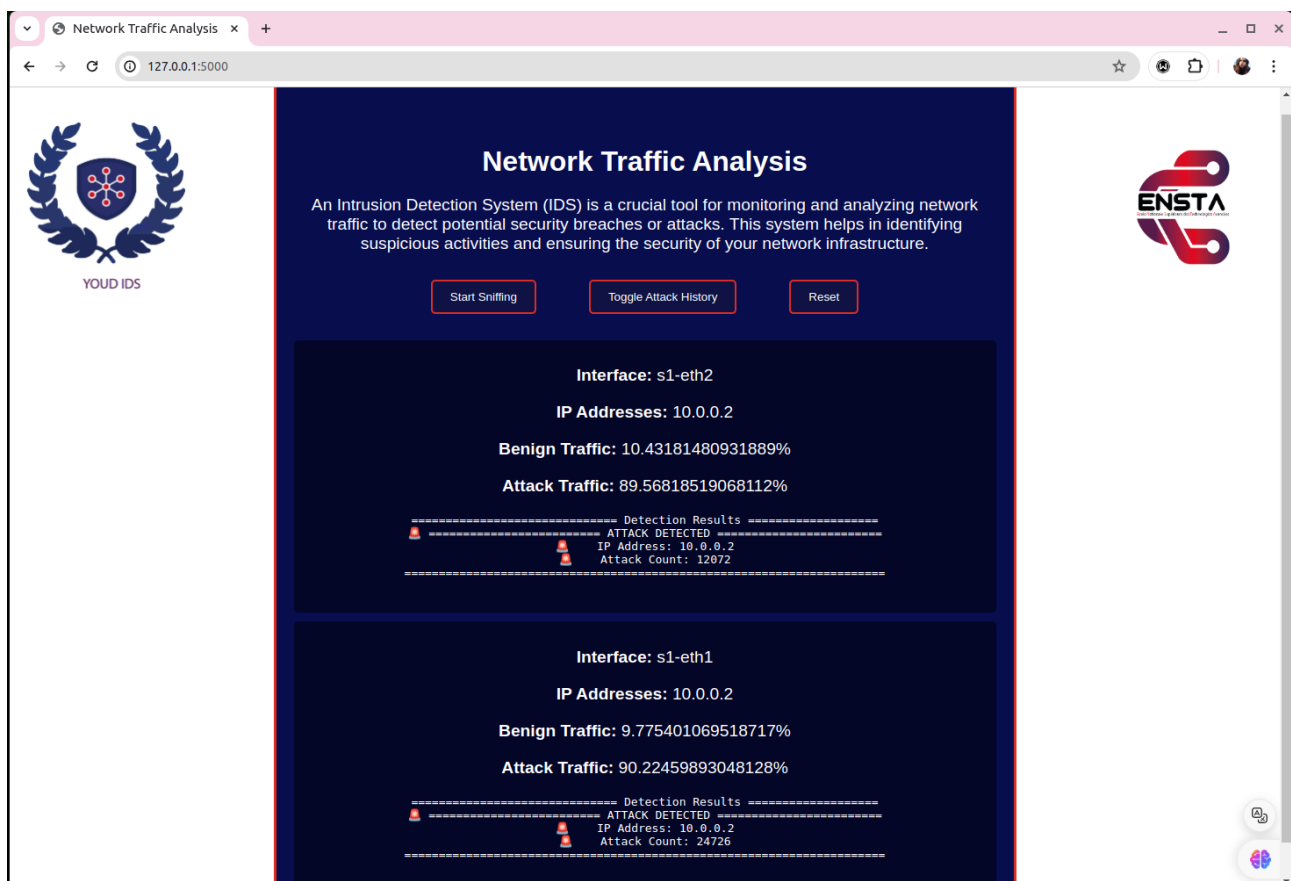


FIGURE 4.22 : Real-time detection results for interfaces s1-eth1 and s1-eth2

Notably, the use of multithreading in our implementation ensures that multiple network interfaces are monitored simultaneously, without compromising performance. Each interface operates within its own thread, allowing for real-time analysis and immediate reporting of suspicious activities. This approach enhances the overall efficiency and responsiveness of the detection system, ensuring that potential threats are promptly identified and mitigated.

Overall, the screenshot exemplifies the effectiveness of integrating the YOUD model with a multithreaded Flask application, offering a user-friendly platform for real-time intrusion detection and network security management.

4.7 Conclusion

In this chapter, we focused on the implementation and evaluation of our proposed solutions for detecting DDoS attacks. Through rigorous testing and real-time analysis, we demonstrated the effectiveness of our models in identifying various types of DDoS attacks. The implementation involved integrating individual deep learning models, such as CNN, LSTM, and CNN-LSTM, into our system and testing their performance.

Our final YOUD model, which combines the strengths of these individual models, proved to be the most effective. It accurately detected studied types of DDoS attacks with superior accuracy, outperforming each individual deep learning model. The results validate the robustness and efficiency of the YOUD model in real-time DDoS attack detection, confirming its potential as a reliable solution for enhancing network security.

Conclusion

In conclusion, SDN emerged to address the limitations of traditional networks by providing separation of control and data planes, programmability, and enabling network monitoring through software applications. This paradigm shift has significantly mitigated network vulnerabilities, including susceptibility to DDoS attacks. We enhanced the CICDOS-2017 dataset by incorporating updated generated traffic using online tools, allowing for more accurate training and evaluation of our approaches. Our project successfully implemented and evaluated CNN, LSTM, and hybrid CNN-LSTM algorithms for detecting various application layer DDoS attacks within an SDN environment. The models exhibited exceptional performance in real-time detection scenarios. Specifically, they achieved accuracies exceeding 95% for Hulk attacks, demonstrating their capability to detect high-volume traffic surges. For Slow Read attacks, the models maintained accuracies above 93%, highlighting their proficiency in identifying attacks that exploit slow data transfer rates. Additionally, the models consistently detected Slow Range attacks with accuracies surpassing 93%, and effectively managed to identify both Slow Body and Slowloris attacks with accuracies exceeding 70%. These results underscore the robustness and versatility of our models in addressing various DDoS attack strategies. These results highlight the models' effectiveness in accurately identifying different types of DDoS attacks.

Integrating these models into the YOUD ensemble approach and leveraging the Ryu controller, our solution, powered by a multithreaded Flask application, demonstrated high effectiveness in identifying application layer DDoS attacks. The YOUD ensemble model combined the strengths of CNN, LSTM, and CNN-LSTM models, resulting in improved detection accuracy and reduced false positives. Notably, the YOUD model successfully detected all the previously mentioned types of attacks, achieving accuracies surpassing 80% across the board, further underscoring its robustness and reliability in real-time DDoS attack detection. Our implementation leveraged the `simple_switch_13.py` application from the Ryu controller's southbound API to manage packet forwarding in an SDN environment. We developed a mitigation module (IPS) that deployed flow rules to the Ryu controller based on IDS detections, effectively blocking attackers and mitigating DDoS attacks. This approach demonstrated the comprehensive efficacy of integrating IDS and IPS within an SDN framework, providing robust network protection and ensuring stability.

Prespectives

Looking forward, future work will focus on further enhancing our DL models by evaluating them with more efficient datasets that encompass a broader spectrum of attack types. This will ensure our system's readiness to combat evolving threats in network security effectively. By continuing to refine our approach and expanding the scope of evaluation, we aim to contribute significantly to the field of cybersecurity in SDN environments. Additionally, a crucial next step involves implementing our solution in a realistic datacenter topology to assess its efficacy in a real-world network environment. This practical deployment will provide invaluable insights into the performance and scalability of our approach under real operational conditions, reinforcing its relevance and applicability in practical cybersecurity scenarios.

Appendix A: Materials and Software Tools

A.1 Ubuntu

Ubuntu is a popular open-source operating system based on Debian Linux, known for its user-friendliness, extensive support community, and robust performance. Its stability and reliability are crucial for running deep learning models and ensuring consistent performance during real-time DDoS attack detection. Ubuntu supports a wide range of software and libraries essential for deep learning and cybersecurity, including frameworks like TensorFlow and PyTorch. The extensive support community and comprehensive documentation available for Ubuntu are invaluable for troubleshooting and optimizing our project, ensuring quick solutions to any issues. Additionally, Ubuntu allows for fine-tuning and optimization of system performance, which is vital for processing large amounts of network traffic data in real time and ensuring our deep learning models run efficiently. Its strong security features further enhance the suitability of Ubuntu for this project.

A.2 Python

Python version 3.9 served as the cornerstone of our project development, offering versatility and robustness in handling complex tasks efficiently. Within this development environment, Matplotlib and Seaborn were instrumental in visualizing data distributions and model performance, providing insights for iterative refinement. NumPy and Pandas facilitated seamless data manipulation and preprocessing, ensuring compatibility with our chosen dataset, CICIDS2017. Additionally, Flask enabled us to develop a responsive web interface for real-time monitoring and control of the IDS system. Scikit-learn rounded out our toolkit, providing essential evaluation metrics to validate the effectiveness of our models. Python's ecosystem and these libraries collectively underpinned the development of a comprehensive and adaptable security solution.

A.3 Emulation Tool

Mininet :

Mininet is a virtual network emulator that allows the creation of a network environment including switches, routers, end-hosts, connections, and SDN controllers [43]. It utilizes operating system virtualization features to scale up to hundreds of nodes. Mininet employs OpenFlow switches and routers, enabling research, development, learning, prototyping, and testing. This tool allows users to deploy the same code and testing standards into a major production network after adding new features or designs. Network packets are directed through the SDN controller for processing [12].

We chose this software for its simplicity and speed. For rapid progress and an enhanced development experience, it was essential for us to quickly launch, modify, and finalize topologies for prototyping. In our initial plan, we anticipated the need to modify our topology during development.

Features of Mininet :

The Mininet network emulator offers several key advantages for network simulation and testing, including :

- Lightweight virtualization
- Flexible and extensible network topology creation
- Realistic network performance
- Supports custom topologies and complex network configurations
- Easily integrates with SDN controllers like Ryu and OpenDaylight
- Open source and actively maintained

Usage in Research :

Mininet was utilized in this project to create custom network topologies, allowing us to simulate DDoS attack scenarios. This setup enabled us to test the efficacy of our deep learning models in detecting and mitigating such attacks, providing a controlled environment for rigorous experimentation and validation.

A.4 Network Controllers

A.4.1 Ryu

Ryu is an open-source software-defined networking framework that facilitates the development and management of network control applications. It provides a rich set of software components and a high-level API, allowing developers to create custom network management solutions without dealing with low-level protocol details [44]. Designed to enhance network flexibility by simplifying traffic management. By Supporting various versions of the OpenFlow protocol for communication between the controller and network devices.

Features of Ryu :

The Ryu SDN framework offers several notable benefits for developing and managing SDN applications, including :

- Support for OpenFlow and other protocols
- Provides a high-level API that simplifies the development of SDN applications
- Easily extensible and flexible
- Actively maintained by a community of developers, ensuring that the framework stays up-to-date with the latest SDN technologies and practices.

Usage in Research :

Choosing Ryu as controller for our SDN Based framework, enabled us to focus on the core logic of our applications while relying on the framework to handle the underlying protocol interactions and network management tasks. This approach accelerates the development process and facilitates the creation of robust and scalable SDN applications.

A.4.2 OpenDayLight (ODL)

OpenDaylight (ODL) is a modular open-source SDN controller platform hosted by the Linux Foundation. It aims to accelerate the adoption of SDN and create a solid foundation for network functions virtualization (NFV). It provides centralized, programmatic control and monitoring of network devices through open protocols such as OpenFlow. Similar to other SDN controllers like Ryu and Pox, OpenDaylight supports OpenFlow and offers ready-to-install network solutions [45]. ODL acts as an interface for quickly and intelligently connecting network devices to optimize network performance, similar to how an operating system provides an interface for computer hardware.

Features of ODL :

The OpenDaylight SDN controller provides a range of features that make it a powerful and flexible platform for managing network infrastructures, including :

- Open-Source modular architecture
- Support for multiple southbound and northbound interfaces
- High scalability and robustness
- Supports multiple network protocols and standards

Usage in Research :

OpenDaylight was chosen as an alternative SDN controller to validate our approach across different control platforms. Its selection was influenced by its ease of installation compared to Ryu, which often presents challenges in finding the appropriate source with the right requirements for its launch.

A.5 Packet Analyzers**Wireshark :**

Wireshark is the most widely used network protocol analyzer in the world, celebrated for its extensive and powerful features. Compatible with Windows, OS X, and Linux, this free packet sniffer application serves various purposes, including network troubleshooting, protocol analysis, software development, and educational use [46].

Wireshark excels in capturing, viewing, and analyzing data packets, offering sophisticated tools for these tasks. It includes advanced wireless protocol analysis capabilities, which are invaluable for network administrators dealing with wireless network issues. With the right driver support, Wireshark can capture wireless

traffic and decode it into a readable format, enabling administrators to diagnose problems like poor performance, intermittent connectivity, and other typical network challenges. is a widely-used network protocol analyzer. It allows users to capture and interactively browse the traffic running on a computer network [47].

Wireshark, a widely-used network protocol analyzer, offers several key features that enhance its functionality and usability, including :

- Supports real-time traffic capture and analysis of saved capture files.
- Compatible with standard packet formats like pcap and pcapng.
- Applies powerful filters to focus on specific traffic patterns and protocols.
- Dissects and analyzes numerous network protocols across different OSI layers.
- Uses color coding to highlight different types of traffic for easier identification.
- Available on Windows, macOS, and various Unix-like systems.

Wireshark was used to capture and analyze network traffic during DDoS attack simulations to understand the attack patterns and validate our detection models.

A.6 Flow Managers

A.6.1 OpenFlow Manager

OpenFlow Manager (OFM) is a specialized application that operates on top of the OpenDaylight platform. It is used to visualize network topologies created using the OpenFlow protocol, enabling the programming of network paths and the collection of network statistics. This application facilitates the management and optimization of OpenFlow-enabled networks by providing intuitive visualization and control tools [48]. So it is a tool for managing and configuring OpenFlow switches. It provides a user-friendly interface for monitoring and managing flow entries in the switches.

The capabilities of our OpenFlow Manager are extensive and include the following features :

- Create, modify, and delete flow entries in switches.
- Real-time monitoring of flow tables.
- Automate the deployment and configuration of network devices.
- Automatically discover network topology and visualize it.
- Support for multiple OpenFlow versions.

A.6.2 CICFlowMeter

CICFlowMeter is a network traffic flow generator developed by the Canadian Institute for Cybersecurity (CIC). It is used for extracting flow features as CSV files from packet capture (PCAP) files after sniffing the network packets. This tool analyzes network traffic data and generates detailed flow records, which are essential for various cybersecurity tasks such as intrusion detection, network monitoring, and anomaly detection [49]. CICFlowMeter supports a wide range of flow features, including basic ones like packet counts

and byte counts, as well as more advanced features like flow duration and inter-arrival times. These features provide comprehensive insights into network behavior and help enhance the accuracy and effectiveness of cybersecurity measures.

Features of CICFlowMeter :

CICFlowMeter offers a comprehensive set of features for network traffic analysis, which includes :

- Generation of flow-based features from PCAP files
- Captures bidirectional flows, meaning it records traffic in both directions (source to destination and destination to source) as a single flow.
- Supports various network protocols, including TCP, UDP, and ICMP, for a comprehensive analysis of network traffic.
- Extracts over 80 features from network traffic, including basic flow features (duration, number of packets, bytes) and advanced statistical features (flow inter-arrival times, packet lengths).
- Exports extracted features to CSV files, which can be easily imported into various data analysis and visualization tools.

Usage in Research :

CICFlowMeter was utilized to extract detailed features from captured network traffic. These features were subsequently used as input for our deep learning models to detect anomalies. By converting raw packet data into structured flow records, CICFlowMeter enabled the deep learning models to effectively analyze patterns and identify irregularities in network behavior, enhancing the accuracy of anomaly detection.

A.7 DDoS Attack and Legitimate Traffic Generation Tools

DDoS generation tools are software applications designed to simulate DDoS attacks against systems or servers. These tools are specifically tailored to test the resilience and effectiveness of trained models under high-stress conditions, replicating real-world DDoS attack scenarios. By generating controlled and scalable traffic patterns, they enable cybersecurity professionals to evaluate their models' performance in detecting and mitigating DDoS threats in real-time environments. Additionally, there're tools create synthetic traffic patterns that resemble genuine user interactions with network services and applications. Legitimate traffic generation tools are essential for network testing, performance monitoring, and application validation. Numerous tools are available that can produce both legitimate-looking traffic and attack traffic, capable of bypassing existing DDoS defense solutions. Many tools are available that can generate traffic patterns similar to real users and mimic harmful DDoS attacks, bypassing current DDoS defense systems [50].

A.7.1 Hping3

Hping is a versatile network tool capable of handling random packet sizes and fragmentation. It serves various purposes such as firewall rule testing, port scanning, and protocol-based network performance assessment. Implemented in TCL, it offers a command-line interface for easy operation. Hping3, a scriptable program utilizing the Tcl language, enables the sending and receiving of custom ICMP/UDP/TCP packets and displaying target replies similar to ICMP responses [51]. Hping3 is a command-line oriented TCP/IP packet assembler/analyzer. It is one of the most popular tools for security auditing and testing of firewalls

and networks.

Hping3 offers a versatile set of features for network testing and analysis, including :

- Reply Display : It can display target replies, similar to how ping shows ICMP replies.
- Arbitrary Packet Body and Size : It supports sending packets with arbitrary body content and size.
- Network Performance Testing : It allows for testing network performance using different protocols.
- Scriptability : Hping3 is scriptable using the Tcl language, allowing for automation and customization of tasks.

Hping3 was used to generate various types of DDoS attacks to test the performance of our deep learning models in detecting and mitigating these attacks.

A.7.2 iPerf

iPerf is a multi-threaded client-server network testing tool and traffic generator used for active measurements of the maximum achievable bandwidth on IP networks. It supports multiple connections and can generate both network layer and application layer traffic, making it useful for simulating network conditions, including DDoS attacks for testing purposes. For each test, iPerf3 reports bandwidth, loss, and other parameters, providing detailed insights into network performance [52]. Thus, iPerf is a comprehensive tool for network performance testing and analysis, suitable for a wide range of applications and platforms.

iPerf provides a range of features designed to facilitate network performance testing, including :

- Cross-Platform Compatibility : Available on Windows, Linux, Android, MacOS X, FreeBSD and others.
- Multiple Simultaneous Connections : The client and server can handle multiple simultaneous connections using the -P option.
- Flexible Testing Duration : Tests can be run for a specified time using the -t option, or for a set amount of data to transfer using the -n or -k options.
- Traffic Generation and DDoS Simulation : Capable of generating network layer and application layer traffic, making it useful for simulating network conditions, including DDoS attacks, for testing purposes.

A.7.3 SlowHTTPTest

SlowHTTPTest is a flexible tool designed to replicate various application layer Denial of Service (DoS) attacks. It offers extensive configuration options to simulate attacks like Slowloris, Slow HTTP POST, and Slow Read. These attacks aim to exhaust server resources by initiating and maintaining multiple low-bandwidth connections, effectively causing service degradation or unavailability [53]. By mimicking these attacks, SlowHTTPTest assists in assessing a system's resilience and readiness to mitigate such threats.

SlowHTTPTest provides several features for simulating and analyzing low-bandwidth application layer DoS attacks :

- Simulation of Common DoS Attacks : It implements various low-bandwidth application layer DoS attacks, including Slowloris, Slow HTTP POST, and Slow Read.

- **Slowloris Simulation** : It can simulate the Slowloris attack, which aims to exhaust server resources by keeping multiple connections open with incomplete HTTP requests.
- **Slow Read Simulation** : Capable of simulating the Slow Read attack, exploiting the TCP persist timer to drain the server's concurrent connections pool by slowly reading response data.
- **Slow HTTP POST Attack Simulation** : It can replicate the Slow HTTP POST attack, where attackers send HTTP POST requests but transmit the data very slowly, tying up server resources.
- By simulating these DoS attacks, SlowHTTPTest helps assess a server's resilience and effectiveness in mitigating such threats.

A.8 TensorFlow and Keras

TensorFlow is an open-source machine learning framework developed by Google. It offers a comprehensive ecosystem for building, training, and deploying machine learning models. As one of the most widely used machine learning frameworks, TensorFlow supports a broad range of tasks, from neural network training to reinforcement learning, making it a adaptable tool in the field of artificial intelligence. TensorFlow's ecosystem includes tools and libraries that cater to various aspects of machine and deep learning workflows. These tools help streamline data preprocessing, model building, training, evaluation, and deployment, thus enabling developers and researchers to focus more on innovation and less on the complexities of implementation. TensorFlow's flexibility and scalability allow it to be used in various applications, from small-scale experiments to large-scale industrial projects.

TensorFlow offers a wide range of features to support machine learning and deep learning applications :

- **Open-Source** : TensorFlow is an open-source framework, allowing for community contributions and widespread use.
- **Scalability** : It supports distributed computing, enabling large-scale machine learning projects.
- **Flexibility** : TensorFlow can be used for various machine learning tasks, including neural network training, reinforcement learning, and more.
- **Compatibility** : TensorFlow is compatible with multiple platforms, including CPUs, GPUs, and TPUs, for optimized performance.
- **High-Level APIs** : Provides high-level APIs like Keras for easy model building and training.

Keras is a high-level neural networks API, written in Python, that provides a user-friendly and modular approach to building and training deep learning models. Keras allows researchers and developers to quickly prototype, test, and iterate on their neural network architectures. Keras is designed to run on top of popular deep learning frameworks, including TensorFlow, which allows it to leverage the powerful computation capabilities of TensorFlow while maintaining a simple and intuitive interface.

Keras abstracts much of the complexity involved in deep learning, offering a suite of pre-built layers, activation functions, and optimization algorithms that can be easily combined to construct complex models. This makes it an ideal choice for both beginners and experienced practitioners who need to implement state-of-the-art models efficiently. Additionally, Keras supports a variety of neural network architectures, including fully connected networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs), making it versatile for a wide range of applications such as image recognition, natural language processing, and time series forecasting.

By integrating seamlessly with TensorFlow, Keras benefits from TensorFlow's extensive ecosystem, which includes tools for model deployment, data preprocessing, and distributed training. This integration ensures

that models built with Keras can be easily scaled and deployed in production environments, providing a robust and scalable solution for deep learning tasks.

Keras offers several features to streamline the development of deep learning models :

- **User-Friendly** : Keras is designed for fast experimentation, with a user-friendly API that reduces the complexity of building neural networks.
- **Modularity** : It allows for easy customization and combination of various neural network components like layers, models, and optimizers.
- **Extensibility** : Keras supports custom layers and models, enabling advanced research and experimentation.
- **Integration** : Seamlessly integrates with TensorFlow, leveraging its powerful features for efficient computation and deployment.
- **Comprehensive Documentation** : Keras provides extensive documentation and community support, facilitating learning and development.

Appendix B: Traffic Generation Commands

B.1 Attack Generation Commands

Command	Arguments	Explanation
<code>sudo slowhttptest -u http://url</code>	- H : slow headers (Slowloris) - B : slow body (R-U-Dead-Yet) - X : slow read (Slow Read) - R : range attack	slowhttptest tool allows to generate various http based attacks, including slow headers (Slowloris), slow body (R-U-Dead-Yet), slow read, and range attacks.
<code>python3 hulk.py</code>	- Target IP : the target IP address - Port NO : the port number	hping3 performs packet crafting, and troubleshoot network connectivity, it allows users to send custom packets with various protocols such as TCP and UDP.

TABLE 2.1 : Attacks generation commands

B.2 Legitimate Traffic Generation Commands

Command	Arguments
<code>sudo hping3 -S -p 80 -i u1000 <target_ip></code>	- -S : sends SYN packets - -p 80 : targets port 80 - -i u1000 : interval of 1000 microseconds (1 packet per millisecond). - <target_ip> : the targeted IP address
<code>sudo hping3 -udp -p 80 -i u1000 <target_ip></code>	- -udp : sends UDP packets - -p 80 : targets port 80 - -i u1000 : interval of limits of the traffic to 1000 microseconds (1 packet per millisecond). - <target_ip> : the targeted IP address

TABLE 2.2 : Legitimate traffic generation commands

Appendix C: Dataset Preprocessing

C.1 Dataset Cleaning

C.1.1 Stripping Whitespace from Column Names

The initial step involved standardizing the column names by removing any leading or trailing whitespace. This measure ensures consistency and prevents issues that may arise from improperly formatted column names.

```
1 df.columns = df.columns.str.strip()
```

C.1.2 Handling Negative Values in Numeric Columns

We identified all the numeric columns and set all the negative values to zero to address potential data anomalies.

```
1 num = df._get_numeric_data()
2 num[num < 0] = 0
```

C.1.3 Removing Zero Variance Columns

columns with zero variance, indicating that all the values within those columns were identical. Subsequently, we removed these columns to enhance the dataset's efficiency and clarity.

```
1 zero_variance_cols = [col for col in df.columns if len(df[col
    ↪ ].unique()) == 1]
2 df.drop(columns=zero_variance_cols, axis=1, inplace=True)
```

C.1.4 Replacing Infinite Values and Removing Not a Number Numeric Data Type (NaN)

To ensure that the dataset was free from undefined or non-numeric values, we replaced any infinite values with Not a Number numeric data type (NaN) and subsequently removed rows containing NaN values.

```
1 df.replace([np.inf, -np.inf], np.nan, inplace=True)
2 df.dropna(inplace=True)
```

C.1.5 Removing Columns with Identical Values

Pairs of columns with identical values were identified and the redundant columns were removed to ensure unique and valuable information :

```

1      column_pairs = [(i, j) for i, j in combinations(df.columns, 2)
2                      ↪ if df[i].equals(df[j])]
3      ide_cols = [col_pair[1] for col_pair in column_pairs]
      df.drop(columns=ide_cols, axis=1, inplace=True)

```

C.1.6 Removing Duplicate Rows

Duplicate rows were removed to maintain the integrity of the dataset and avoid redundancy.

```

1      df.drop_duplicates(inplace=True)

```

C.1.7 Standardizing Column Names

We standardized the column names by converting them to lowercase, replacing spaces with underscores, and removing parentheses to facilitate seamless data processing.

```

1      df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
2                      ↪ _').str.replace('(', '').str.replace(')', '')

```

C.1.8 Visualizing Label Distribution

Understanding the distribution of different attacks types in our dataset is crucial for effective model training. We created a count plot of the attacks labels, excluding the benign traffic label as showed in figure 3.1, to visualize the prevalence of various attacks types.

```

1      filtered_df = df[df['label'] != 'BENIGN']
2      plot = sns.countplot(data=filtered_df, y='label')
3      plt.xscale('log')
4      fig = plot.get_figure()
5      plt.show()

```

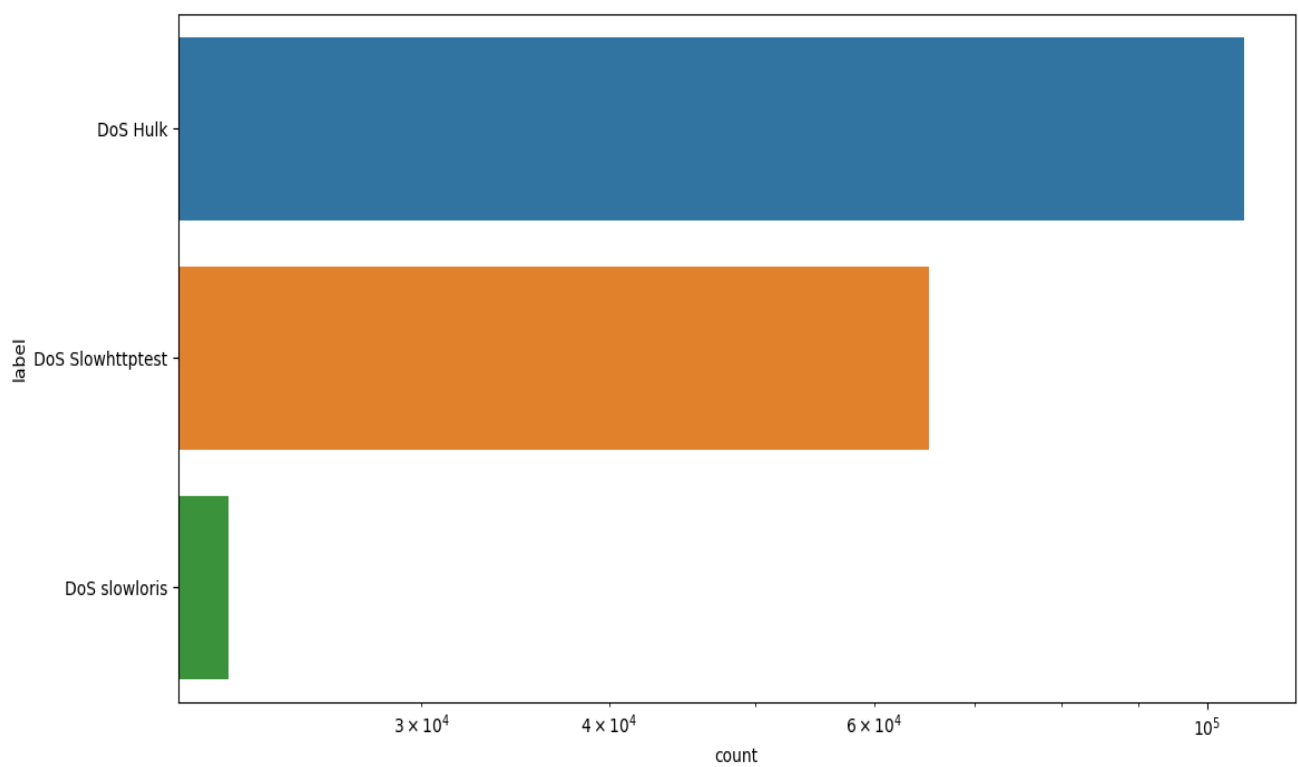


FIGURE 3.1 : Attacks Label Counts

Bibliography

- [1] A. Hashemi, G. Orzechowski, and A. Mikkola. Multibody dynamics and control using machine learning. *Multibody System Dynamics*, 58:397–431, 2023.
- [2] Ottavio Calzone. An intuitive explanation of lstm. <https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c>, April 10, 2022.
- [3] Open Networking Foundation. Software-defined networking: The new norm for networks. <https://opennetworking.org/wp-content/uploads/2011/09/wp-sdn-newnorm.pdf>, April 13, 2012.
- [4] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.
- [5] Zohaib Latif, Kashif Sharif, Fan Li, Md Monjurul Karim, Sujit Biswas, and Yu Wang. A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, 156:102563, 2020.
- [6] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and Communication Networks*, 9(18):5803–5833, 2016.
- [7] Md. Rayhan Ahmed, Salekul Islam, Swakkhar Shatabda, et al. Intrusion detection system in software-defined networks using machine learning and deep learning techniques –a comprehensive survey. *TechRxiv*, November 21 2022.
- [8] Anish Ghosh and T. Manoranjitham. A study on load balancing techniques in sdn. *International Journal of Engineering & Technology*, 7:174–177, 2018.
- [9] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.
- [10] Abdullah Ahmed Bahashwan, Mohammed Anbar, Selvakumar Manickam, Taief Alaa Al-Amiedy, Mohammad Adnan Aladaileh, and Iznan H Hasbullah. A systematic literature review on machine learning and deep learning approaches for detecting ddos attacks in software-defined networking. *Sensors*, 23(9):4441, 2023.
- [11] Amran Mansoor, Mohammed Anbar, Abdullah Ahmed Bahashwan, Basim Ahmad Alabsi, and Shaza Dawood Ahmed Rihan. Deep learning-based approach for detecting ddos attack on software-defined networking controller. *Systems*, 11(6):296, 2023.
- [12] Bushra Alhijawi, Sufyan Almajali, Hany Elgala, Haythem Bany Salameh, and Moussa Ayyash. A survey on dos/ddos mitigation techniques in sdns: Classification, comparison, solutions, testing tools and datasets. *Computers and Electrical Engineering*, 99:107706, 2022.
- [13] Noe Marcelo Yungaicela-Naula, Cesar Vargas-Rosales, and Jesus Arturo Perez-Diaz. Sdn-based architecture for transport and application layer ddos attack detection by using machine and deep learning. *IEEE Access*, 9:108495–108512, 2021.

- [14] Fatmah Alanazi, Kamal Jambi, Fathy Eassa, Maher Khemakhem, Abdullah Basuhail, and Khalid Al-subhi. Ensemble deep learning models for mitigating ddos attack in software-defined network. *Intelligent Automation & Soft Computing*, 33(2), 2022.
- [15] Hossein Hadian Jazi, Hugo Gonzalez, Natalia Stakhanova, and Ali A Ghorbani. Detecting http-based application layer dos attacks on web servers in the presence of sampling. *Computer Networks*, 121:25–36, 2017.
- [16] Bindu Bala and Sunny Behal. Ai techniques for iot-based ddos attack detection: Taxonomies, comprehensive review and research challenges. *Computer Science Review*, 52:100631, 2024.
- [17] Z Zhigang. What is a ddos attack? ddos attack types and defense. huawei. <https://info.support.huawei.com/info-finder/encyclopedia/en/DDoS+Attack.html>, July 26, 2023.
- [18] Krushang Sonar and Hardik Upadhyay. A survey: Ddos attack on internet of things. *International Journal of Engineering Research and Development*, 10(11):58–63, 2014.
- [19] R Sridaran. An overview of ddos attacks in cloud environment. *Int. J. Adv. Networking Appl.(IJANA)*, pages 124–127, 2015.
- [20] Irfan Sofi, Amit Mahajan, and Vibhakar Mansotra. Machine learning techniques used for the detection and analysis of modern types of ddos attacks. *International Research Journal of Engineering and Technology*, 4(6):1085–1092, 2017.
- [21] Dan Tang, Yudong Yan, Siqi Zhang, Jingwen Chen, and Zheng Qin. Performance and features: Mitigating the low-rate tcp-targeted dos attack via sdn. *IEEE Journal on Selected Areas in Communications*, 40(1):428–444, 2021.
- [22] Noe M Yungaicela-Naula, Cesar Vargas-Rosales, Jesus Arturo Perez-Diaz, Eduardo Jacob, and Carlos Martinez-Cagnazzo. Physical assessment of an sdn-based security framework for ddos attack mitigation: Introducing the sdn-slowrate-ddos dataset. *IEEE Access*, 11:46820–46831, 2023.
- [23] A Dhanapal and P Nithyanandam. The slow http ddos attacks: Detection, mitigation and prevention in the cloud environment. *Scalable Computing: Practice and Experience*, 20(4):669–685, 2019.
- [24] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, and et al. Billinge, Simon JL. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8(1):59, 2022.
- [25] Ki Hwan Kim, Seung Hong Choi, and Sung-Hong Park. Improving arterial spin labeling by using deep learning. *Radiology*, 287(2):658–666, 2018.
- [26] Kanchan M Tarwani and Swathi Edem. Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, 48(6):301–304, 2017.
- [27] Mohammad Mustafa Taye. Understanding of machine learning with deep learning: Architectures, workflow, applications and future directions. *Computers*, 12(5):91, 2023.
- [28] Koichiro Yasaka, Hiroyuki Akai, Akira Kunimatsu, Shigeru Kiryu, and Osamu Abe. Deep learning with convolutional neural network in radiology. *Japanese journal of radiology*, 36:257–272, 2018.
- [29] Koichiro Yasaka, Hiroyuki Akai, Osamu Abe, and Shigeru Kiryu. Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced ct: a preliminary study. *Radiology*, 286(3):887–896, 2018.
- [30] Mayank Mishra. Convolutional neural networks, explained. <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>, Aug 26, 2020.

-
- [31] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):1–74, 2021.
 - [32] Sakshi Indolia, Anil Kumar Goswami, Surya Prakesh Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia computer science*, 132:679–688, 2018.
 - [33] Dishashree26 Gupta. Fundamentals of deep learning – activation functions and when to use them? <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>, 25 Mar, 2024.
 - [34] Sushmita Poudel. Recurrent neural network (rnn) architecture explained. <https://medium.com/@poudelsushmita878/recurrent-neural-network-rnn-architecture-explained-1d69560541ef>, August 28, 2023.
 - [35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
 - [36] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.
 - [37] Long short term memory (lstm) : de quoi s'agit-il ? <https://datascientest.com/long-short-term-memory-tout-savoir>, October 02, 2023.
 - [38] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.
 - [39] Furqan Rustam, Ali Raza, Muhammad Qasim, Sarath Kumar Posa, and Anca Delia Jurcut. A novel approach for real-time server-based attack detection using meta-learning. *IEEE Access*, 12:39614–39627, 2024.
 - [40] Bo Dong and Xue Wang. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pages 581–585, 2016.
 - [41] Xiaoyu Liang and Taieb Znati. A long short-term memory enabled framework for ddos detection. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.
 - [42] Chih-Hsiang Hsieh, Wei-Kuan Wang, Cheng-Xun Wang, Shi-Chun Tsai, and Yi-Bing Lin. Efficient detection of link-flooding attacks with deep learning. *Sustainability*, 13(22):12514, 2021.
 - [43] Mininet. <https://mininet.org/>, 2024.
 - [44] VR Sudarsana Raju. Sdn controllers comparison. In *Proceedings of science globe international conference*, pages 1–5, 2018.
 - [45] Sumit Badotra and Japinder Singh. Open daylight as a controller for software defined networking. *International Journal of Advanced Research in Computer Science*, 8(5), 2017.
 - [46] Wireshark · download. <https://www.wireshark.org/download.html>.
 - [47] Usha Banerjee, Ashutosh Vashishtha, and Mukul Saxena. Evaluation of the capabilities of wireshark as a tool for intrusion detection. *International Journal of computer applications*, 6(7):1–5, 2010.
 - [48] CiscoDevNet/opendaylight-openflow-app. <https://github.com/CiscoDevNet/OpenDaylight-Openflow-App>.
 - [49] cicflowmeter. <https://pypi.org/project/cicflowmeter/>.
-

- [50] Sunny Behal and Krishan Kumar. Characterization and comparison of ddos attack tools and traffic generators: A review. *International Journal of Network Security*, 19(3):383–393, 2017.
- [51] hping3: Kali linux tools. kali linux. <https://www.kali.org/tools/hping3/>.
- [52] iperf - the ultimate speed test tool for tcp, udp and sctp test the limits of your network + internet neutrality test. iperf.fr. <https://iperf.fr/>.
- [53] slowhttptest: Kali linux tools. <https://www.kali.org/tools/slowhttptest/>.