TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

LALITPUR ENGINEERING COLLEGE

DDOS ATTACK DETECTION AND MITIGATION USING MACHINE LEARNING IN MULTICONTROLLER SDN ENVIRONMENT

BY

ANSHUL RAWAL[LEC077BCT002]

KAPIL PARAJULI[LEC077BCT010]

SHIVAM GUPTA[LEC077BCT022]

SAIROJ PRASAI[LEC077BCT030]

A PROJECT

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

LALITPUR, NEPAL

DECEMBER,2024

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

LALITPUR ENGINEERING COLLEGE

A Project Proposal

on

DDOS Attack detection and mitigation using Machine Learning in Multicontroller SDN environment

Submitted By

Anshul Rawal(LEC077BCT002)

Kapil Parajuli(LEC077BCT010)

Shivam Gupta(LEC077BCT022)

Sairoj Prasai(LEC077BCT030)

Submitted To:

Department of Computer Engineering

Lalitpur Engineering College

Lalitpur, Nepal

In partial fulfillment for the award of Bachelor of Engineering in Computer Engineering

Under of the supervision of

Er.Binod Sapkota, Asst.Professor of IOE, Thapathali Campus

December,2024

ACKNOWLEDGMENT

This project work would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First of all, We would like to express our sincere gratitude to our supervisor, **Er.Binod Sapkota, Asst.Professor of IOE, Thapathali Campus**, of for providing invaluable guidance, insightful comments, meticulous suggestions, and encouragement throughout the duration of this project work. Our sincere thanks also goes to the Project Coordinator, **Er. Sandesh Saran Poudel**, for coordinating the project works, providing astute criticism, and having inexhaustible patience.

We are also grateful to our classmates and friends for offering us advice and moral support. To our family, thank you for encouraging us in all of our pursuits and inspiring us to follow our dreams. We are especially grateful to our parents, who supported us emotionally, believed in us and wanted the best for us.

ANSHUL RAWAL(077BCT002)
KAPIL PARAJULI (077BCT010)
SHIVAM GUPTA(077BCT022)
SAIROJ PRASAI(077BCT030)

December,2024

ABSTRACT

This work aims to formulate an effective scheme to detect and mitigate Distributed Denial of Service (DDoS) attacks in Software Defined Networks (SDN). Distributed Denial of Service attacks are one of the most destructive attacks on the internet. A DDoS attack is aimed at disrupting the normal operation of a system by making services and resources unavailable to legitimate users by overloading the system with excessive superfluous traffic from distributed sources. Software Defined Networking, being an emerging technology, offers a solution to reduce network management complexity. It separates the control plane and the data plane. This decoupling provides centralized control of the network with programmability and flexibility. This work harnesses this programming ability and centralized control of SDN to analyze the randomness of the network flow data. The proposed technique can detect volume-based and application-based DDoS attacks like TCP SYN flood, Ping flood. The methodology is evaluated through emulation using Mininet, and detection and mitigation strategies are implemented in the RYU controller. The experimental results aim to show that the proposed method will have improved performance evaluation parameters, including attack detection time, delay to serve a legitimate request in the presence of an attacker, and overall CPU utilization. The enhanced capabilities of SDN can provide robust and scalable network management, making the system adept at handling DDoS attacks more efficiently. The system would detect incoming DDoS attacks using machine learning algorithms and mitigate the attacks in the control plane of the SDN network.

Keywords: *DDoS attacks, RYU controller, Security breaches,Software-Defined Networking (SDN),TCP SYN flood*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CSV	Comma Seperated Value
DDoS	Distributed Denial of Service
GPU	Graphical Processing Unit
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection Systems
ML	Machine Learning
NIDS	Network Intrusion Detection Systems
RAM	Random Access Memeory
RNN	Recurrent Neural Network
SDLC	Software Development Life Cycle
SDN	Software Defined Networking
SYN	Synchronize Sequence Number
TCP	Trasmission Control Protocol
UDP	User Datagram Protocol

1 INTRODUCTION

1.1 Background

Network security has become a critical concern due to the rapid expansion of networks, significantly impacting nations, businesses, and individuals. A Distributed Denial of Service (DDoS) attack is a malicious attempt to disrupt normal traffic to a targeted server, service, or network by overwhelming it with excessive internet traffic. This DDoS attack detection and mitigation system using machine learning powered by Software-Defined Networking (SDN) environment is essential due to the critical concern of network security amidst the rapid expansion of networks. A DDoS attack disrupts normal traffic to targeted servers, services, or networks by overwhelming them with excessive internet traffic. Modern Intrusion Detection Systems (IDS) must manage the vast data generated by information technology growth, detecting deviations from normal behavior to flag potential DDoS threats. Integrating SDN enhances IDS capabilities through centralized control, dynamic policy enforcement, efficient traffic management, quicker security responses, improved scalability, real-time analysis, and better overall effectiveness in counteracting cyber threats.

1.2 Motivation

With the rapid expansion of networks and information technology, the massive volume of data generated makes networks more susceptible to sophisticated Distributed Denial of Service (DDoS) attacks targeting governments, businesses, and individuals. This DDoS detection and mitigation system leverages machine learning algorithms powered by Software-Defined Networking (SDN) to identify and counteract DDoS threats. It is crucial for maintaining network integrity, confidentiality, and availability by continuously monitoring traffic, detecting abnormal patterns, and identifying potential DDoS threats in real-time. While SDN offers significant advantages over traditional networks, such as better traffic management and flexibility, it is also vulnerable to DDoS attacks. Therefore, the motivation behind developing this DDoS detection system is to address this specific weakness, ensuring that SDN environments remain secure by detecting and mitigating DDoS attacks effectively.

1.3 Problem Statement

The frequency and complexity of Distributed Denial of Service (DDoS) attacks are constantly rising, making traditional network security measures less effective at preventing such sophisticated threats. Conventional defenses can be circumvented by modern DDoS attacks, causing substantial operational, financial, and reputational harm to individuals, companies, and governments. By continuously monitoring network traffic and instantly detecting potential DDoS threats, this DDoS detection and mitigation system offers a proactive solution. However, the sheer volume and diversity of network data pose a challenge, often resulting in high false positive rates. This can overwhelm detection algorithms and delay response times. Effective mitigation techniques are also necessary to quickly and effectively eliminate identified threats, preventing further damage and ensuring the availability, confidentiality, and integrity of network resources. This initiative leverages machine learning algorithms powered by Software-Defined Networking (SDN) to address these challenges, providing a robust defense against DDoS attacks.

1.4 Project Objectives

The objectives of our project is

- To create a balanced multi-controller SDN environment, and to train and deploy an Machine Learning model for DDoS attack detection and mitigation, and test its effectiveness in the control plane.

1.5 Scope of Project

This DDoS detection and mitigation system involves continuous monitoring and analysis of network traffic to identify and address potential security threats. By detecting deviations from typical traffic patterns, the system highlights suspicious activities that may indicate DDoS attacks. Integrating advanced technologies like machine learning and Software-Defined Networking (SDN) enhances the system's detection capabilities, offering centralized management and dynamic policy enforcement.

1.6 Potential Project Applications

1.6.1 Corporate Networks

Businesses use DDoS detection and mitigation system to protect against cyber threats like malware, phishing, and unauthorized access by monitoring network traffic and swiftly addressing security breaches.

1.6.2 Protection of Critical Infrastructure

This system safeguards vital sectors such as energy, transportation, and healthcare by preventing DDoS attacks on infrastructure, ensuring uninterrupted operations and public safety.

1.6.3 Cloud Security

As cloud adoption rises, this system monitors cloud network activity, enhancing security against threats to cloud applications and data storage.

1.6.4 Government and Military Networks

This system is vital for safeguarding government and military information, defending against cyber espionage, warfare, and other threats to classified data and national security.

1.6.5 Financial Services

In the financial sector, this system combats fraud, data breaches, and unauthorized access, ensuring regulatory compliance and protecting banking and payment systems.

1.7 Originality of Project

1.7.1 Detection Algorithms

The use of advanced algorithms, such as machine learning and deep learning, to identify patterns and anomalies in network traffic. These algorithms can adapt and learn from new types of attacks, improving the system's detection capabilities over time.

1.7.2 Real-Time Monitoring

Implementing real-time monitoring techniques that allow for immediate detection and response to threats. This involves processing large volumes of data quickly and accurately to minimize the impact of potential attacks.

1.7.3 Behavioral Analysis

Analyzing the behavior of users and devices within the network to detect deviations from normal patterns. This can help identify insider threats and sophisticated attacks that may not be detectable through signature-based methods.

1.7.4 Integration with SDN

Designing NIDS that can scale to handle the growing volume of network traffic without compromising performance including optimization of the system architecture.

1.7.5 Anomaly Detection

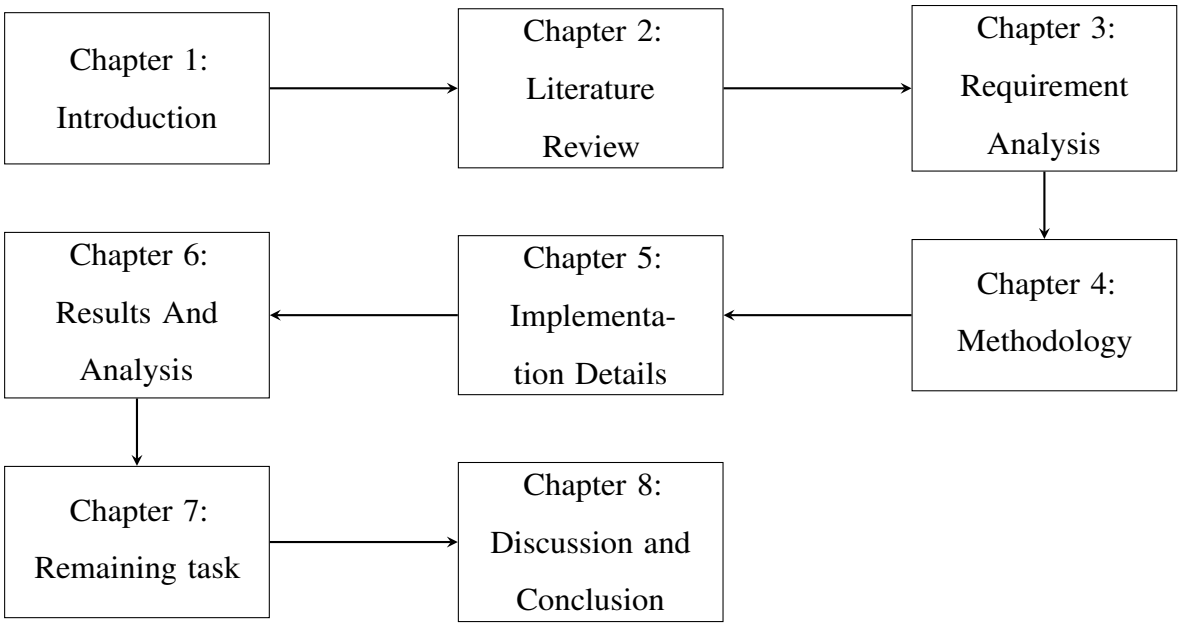
Utilizing statistical methods and anomaly detection techniques to identify unusual network activities that may indicate a potential threat. This approach can detect zero-day attacks that do not match any known signatures.

1.7.6 Context-Aware Security

Incorporating contextual information, such as the type of application, user roles, and network segments, to enhance the precision of threat detection and reduce the number of false alerts.

1.8 Organisation of Project Report

The project report is organized into eight chapters, each addressing a specific aspect of the research. Chapter 1 introduces the research topic, outlining the problem and objectives. Chapter 2 reviews relevant literature, highlighting previous work and identifying research gaps. Chapter 3 focuses on requirement analysis, detailing the necessary components and constraints for the proposed solution. Chapter 4 explains the methodology, describing the algorithms and principles employed to achieve the project objectives. Chapter 5 provides implementation details, explaining the practical execution of the proposed solution. Chapter 6 presents the results and their analysis, evaluating the success of the project in meeting its objectives. Chapter 7 identifies any remaining tasks or limitations that need further attention. Finally, Chapter 8 concludes the report by summarizing the findings, comparing them to the original objectives, and discussing broader implications.



2 LITERATURE REVIEW

Banitalebi Dehkordi, Soltanaghaei, and Boroujeni explore the detection of Distributed Denial of Service (DDoS) attacks through the application of machine learning and statistical methods in Software-Defined Networking (SDN) in their work published in the Journal of Supercomputing in 2020. Their study investigates the efficacy of utilizing machine learning algorithms and statistical techniques for identifying and mitigating DDoS attacks in SDN environments. By leveraging the programmability and centralized control of SDN, combined with the analytical power of machine learning and statistical approaches, the authors propose a method to enhance network security by detecting and mitigating DDoS attacks in real-time. Their research contributes valuable insights into the development of robust defense mechanisms against DDoS threats within SDN architectures[1].

The paper titled ”The Beacon OpenFlow controller” by D. Erickson, presented at HotSDN 2013, provides insights into the Beacon OpenFlow controller, a software component central to Software-Defined Networking (SDN). The paper delves into the functionalities and features of the Beacon controller, highlighting its role in managing OpenFlow-based networks. It likely discusses the architecture, design principles, and performance characteristics of the Beacon controller, offering a comprehensive overview of its capabilities and contributions to the SDN ecosystem. Additionally, the paper may touch upon practical applications, deployment scenarios, and potential challenges associated with the Beacon controller, contributing valuable insights to the literature on SDN controllers[2].

Fatih, Cengiz, and Enis examine the utilization of machine learning algorithms for flow-based anomaly detection systems in Software Defined Networks (SDNs) in their contribution to ”Intelligent and Fuzzy Techniques: Smart and innovative Solutions” in 2021. The authors delve into the application of machine learning techniques for anomaly detection within SDNs, aiming to enhance network security by identifying and mitigating abnormal traffic patterns. By leveraging flow-based monitoring and machine learning algorithms, their work proposes a method to detect and respond to potential threats in real-time, offering an innovative solution to the evolving challenges of network security in SDN environments[3].

The paper titled ”A Lightweight DDoS Detection and Mitigation System in Software-Defined Networks” by F. Hu, Q. Hao, and K. Bao, published in Future Generation Computer Systems in 2018, introduces a lightweight detection system designed for multi-controller architectures. The system uses statistical methods to identify anomalous traffic patterns and implements switch-level mitigation policies to prevent resource exhaustion .[4]

The paper titled ”Toward network-based DDoS detection in software-defined networks” by S. Jevtic, H. Lotfalizadeh, and D. S. Kim, presented at the 12th International Conference on Ubiquitous Information Management and Communication in 2018, addresses the growing concern of Distributed Denial of Service (DDoS) attacks in Software-Defined Networks (SDNs). The literature review likely explores existing research on DDoS detection techniques in traditional network environments and highlights the limitations of applying these methods to SDNs. It may discuss the unique characteristics of SDNs, such as centralized control and programmable data planes, and their implications for DDoS detection[5].

The paper titled ”Distributed Approach for Detecting DDoS in Software-Defined Networks Using Machine Learning Techniques” by M. N. Lima, A. M. Santos, and A. S. de Oliveira, published in Journal of Network and Computer Applications in 2020, examines the application of machine learning in distributed detection systems. It evaluates several classifiers, including Random Forest and SVM, for identifying malicious flows in multi-controller environments[6].

The paper titled ”A Multi-Level DDoS Detection and Mitigation System for Multi-Controller Software-Defined Networks” by S. Sharma and R. K. Saini, published in IEEE Transactions on Network and Service Management in 2021, proposes a multi-level detection approach combining switch-level monitoring and controller-level analysis. The study demonstrates enhanced efficiency in mitigating DDoS attacks in distributed SDN environments[7].

3 REQUIREMENT ANALYSIS

3.1 Functional Requirements

- Ensure that detection mechanisms are as latency-free as possible in order to quickly detect attacks.
- Utilize machine learning models to identify anomalies in network traffic indicative of potential attacks.
- Implement security measures to protect the detection and mitigation system from attacks and unauthorized access.
- Determine that the system integrates easily with the current security and network infrastructure, and that it can cooperate with other network defense systems to provide coordinated responses to threats.

3.2 Non Functional Requirements

- Support horizontal scaling to handle increasing traffic loads without performance degradation, with the ability to add nodes dynamically.
- Design an intuitive system with clear visualizations, easy navigation, and actionable insights.

3.3 Instrumentation Requirements

3.3.1 Hardware Requirement

For a development,we will use laptop is used for coding and development, Kaggle is utilized for model training, and Mininet is employed for defining network topologies, the hardware requirements can be divided into two main parts: the laptop specifications and the cloud resources provided by Kaggle

3.3.2 Software Requirement

Mininet

Mininet is a free software tool for emulating real computer networks in software. It is a network emulator and offers a simplified version of a real network.It is widely used in SDN to test and develop network applications, protocols, and topologies in a flexible and cost-effective manner. Mininet can simulate a complete network with hosts, switches,

and links, providing a realistic environment for experimenting with SDN controllers and applications.

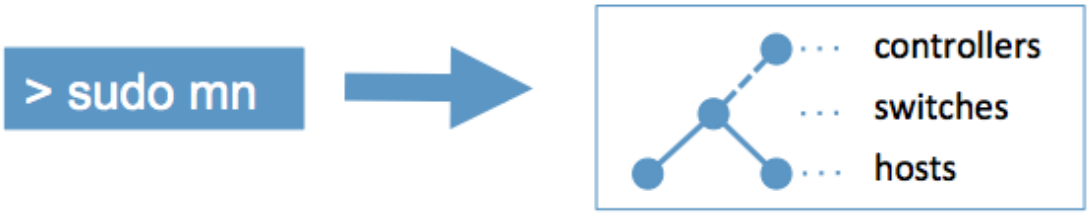


Figure 3.1: Mininet

Tensorflow

TensorFlow is an open-source software library developed by Google for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Figure 3.2: Tensorflow

Pandas and Numpy

NumPy and Pandas to efficiently handle and analyze large datasets. NumPy is essential for performing fast numerical computations and matrix operations, which are crucial for processing network traffic data. Pandas provides powerful data structures like DataFrames, allowing us to clean, manipulate, and explore the data with ease. Together,

these libraries help us preprocess the data, extract features, and prepare it for machine learning algorithms, enhancing the accuracy and efficiency of our intrusion detection system.

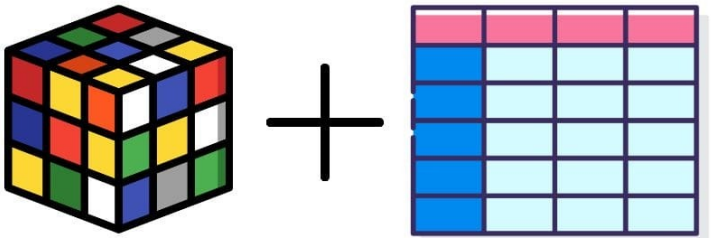


Figure 3.3: Panda and Numpy

Wireshark

Wireshark is a widely used, open source network analyzer that can capture and display real-time details of network traffic. It is particularly useful for troubleshooting network issues, analyzing network protocols and ensuring network security. Networks must be monitored to ensure smooth operations and security.

Figure 3.4: Wireshark

Git and Github

Git is a distributed version control system that can handle any size project quickly and effectively. It is free and open-source. Git facilitates collaboration by enabling the merging of changes made by several individuals into a single source. Local software is called Git. On your computer, your files and their history are kept. Another option is to store a copy of the files and their revision history on web hosts like GitHub. Working together with other developers is made easier with a central location where you can upload and download changes from each other. Git allows two people to work on separate portions of the same file and then automatically merge their changes.

3.4 Feasibility Study

3.4.1 Economic Feasibility

Integrating a DDoS attack detection and mitigation system into a 5G network involves initial investments in hardware, software, and training, which were done on our devices, but offers significant economic benefits by leveraging cost-effective cloud resources like Google Colab's GPUs for machine learning, the project minimizes upfront hardware expenses and ensures efficient resource utilization. Additionally, the system's ability to quickly detect and mitigate attacks reduces the risk of costly security incidents, downtime, and data breaches, leading to long-term savings. Automated mitigation further lowers operational costs and enhances service availability, providing a strong return on investment and ensuring continuous customer satisfaction.

3.4.2 Operational Feasibility

The operational feasibility of the DDoS attack detection and mitigation system is confirmed through Mininet simulations, which show that the system integrates smoothly with existing infrastructure and adapts to evolving network demands. Its scalable architecture and broad compatibility ensure it can handle various network environments effectively. By demonstrating its functionality in simulated topologies, we validate that the system is practical and ready for real-world deployment, providing robust network security solutions. Additionally, the ease of integration and adaptability highlight its suitability for diverse operational contexts.

3.4.3 Technical Feasibility

The technical feasibility of the DDoS attack detection and mitigation system is robust, supported by the utilization of advanced algorithms and machine learning to ensure accurate intrusion detection. The system is designed to seamlessly integrate with existing network hardware and software infrastructure, minimizing the need for major modifications. By leveraging GPUs provided by Google Colab, the system benefits from efficient and cost-effective model training, enabling rapid processing and enhanced performance. These elements combined demonstrate that the system not only meets technical requirements but also operates effectively within the current technological landscape, making it a viable solution for real-world applications.

4 METHODOLOGY

4.1 Software Development Model

4.1.1 Incremental Model

The incremental model is a software development process in which requirements are split up into several independent software development cycle modules. Every module in this paradigm goes through the phases of requirements, design, implementation, and testing. The module's functionality is increased with each new release. Until the entire system is achieved, the process is continued.

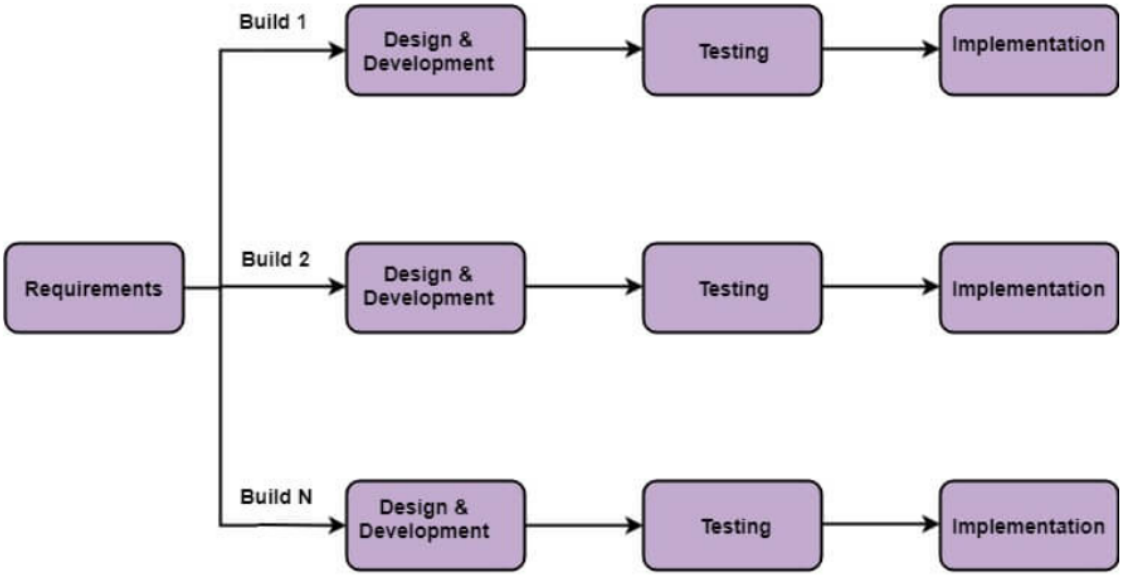


Figure 4.1: Incremental Model

- **Requirement analysis**:

  The product analysis expertise determines the requirements in the first stage of the incremental model. Additionally, the requirement analysis team is aware of the functional requirements of the system. This stage is vital to the software development process under the incremental model.

- **Design and Development:**

  The development process and the design of the system's functionality are successfully completed in this stage of the incremental SDLC model. The incremental model makes use of style and development phase when software develops new practicality.

- **Testing:**

  In the incremental model,the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

- **Implementation:**

  The development system's coding phase is made possible by the implementation phase. In the designing and development phase, it involves the final code, and in the testing phase, it involves testing the functionality. The number of products that are operational after this phase is finished is improved and upgraded up to the final system product.

Hardware Component	Details
CPU	Intel Xeon 2-core 2.2 GHz
GPU model	NVIDIA Tesla p100,Tesla T4
Number of GPUs(P100,T4)	1,1
CUDA Cores per GPU(P100,T4)	3584,320
Tensor Core per GPU(P100,T4)	56,320
Clock Speed	1.30 GHz(base)/1.48GHz(boost)
VRAM(P100,T4)	16GB HBM2,16GB GDDR6
Memory Bandwidth	720 GB/s, 320 GB/s
RAM	16GB

4.2 Theoretical Formulations

4.2.1 SDN

SDN is an approach to network management that describes a network architecture whose control plane is completely decoupled from the data plane.It manages networks using

software to control how data moves through the network, making it easier to adjust and optimize compared to traditional hardware-based methods. The SDN infrastructure is divided into three layers :

Data Plane Layer

It resides in the bottom layer of the SDN paradigm. It is comprised of physical switches, virtual or software-based switches routers and access point.The OpenFlow protocol is used to interconnect these devices.The functions data layer performs are dispatching,rejecting and transforming data according to the rules or policies provided by the control layer. The SDN data plane has a variety of devices that lack intelligence. They just simply carry out the controller's instructions or rules

Control Layer

This layer is considered as the brain of the entire system and contains the Controller device.The controller manages the entire traffic flow and is completely responsible for routing, sending, and dropping packets by programming.This module functions as a proxy to collect all network traffic flows from the OpenFlow protocol to serve the statistical computation.This layer controls the exchange of data between different applications and the dispatching devices.For this project we will use RYU controller.

Application Layer

This layer includes application managing the network infrastructure, traffic policies, and all these application are installed, stored, and operated independently on one or more servers within the system.They are connected and controlled by the Controller through a REST API. In this project, we only implement applications related to monitoring network infrastructure, such as monitoring the Controller's resources and monitoring the system bandwidth for throughout the experimentation process, we will monitor and observe how the system is affected in terms of resource utilization.

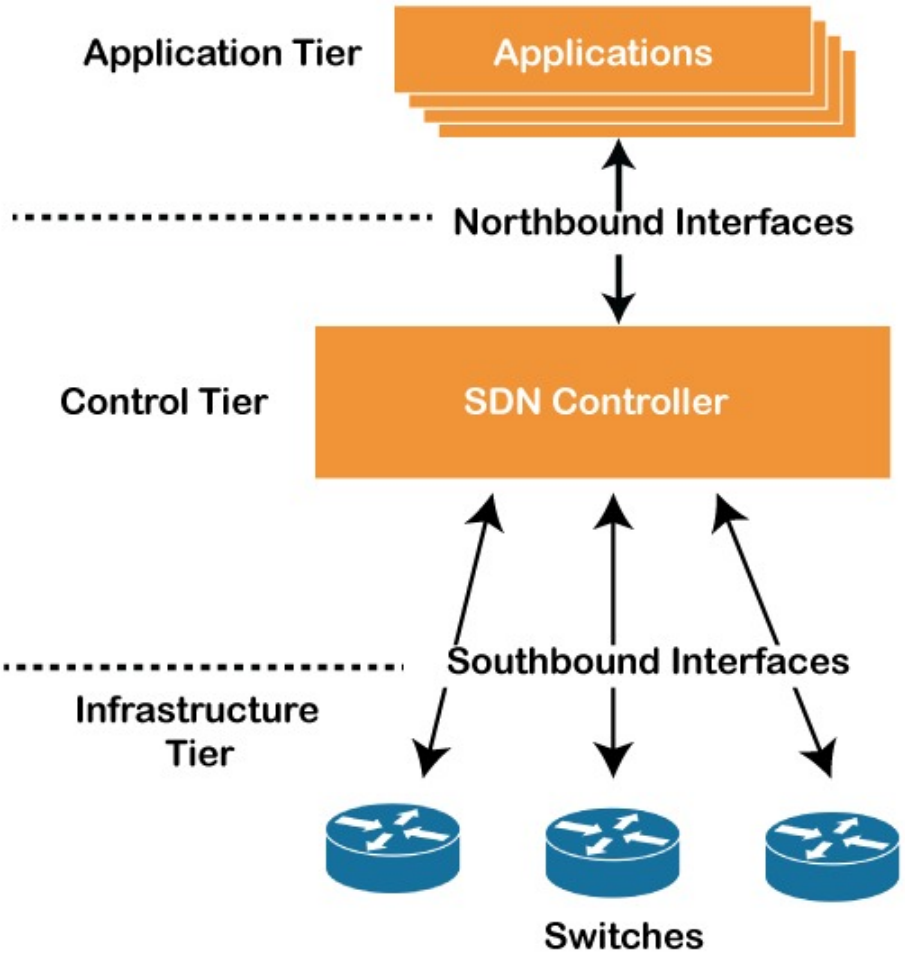


Figure 4.2: SDN architecture

4.2.2 OpenFlow

OpenFlow is a simple protocol that the SDN Controller uses over a secure channel(Transport Layer Security (TLS)) on either TCP port 6633 or 6653 to modify the flow table in supporting switch. Port 6653 is used in our project. OpenFlow has also evolved, coming under the management of the ONF founded in 2011 for the promotion and adoption of SDN through open standards development. OpenFlow has evolved to version 1.5.1 , however, hardware typically supports up to v1.3 . This is used for our project.

4.2.3 Open vSwitch

The OpenFlow protocol and an initial specification in 2008 for a virtual Switch daemon (vswitchd), produced for the GNU/Linux kernel led to the Open virtual Switch (OvS). OvS offers a softswitch solution that operates over OpenFlow and can be used in

virtualised situations where a physical switch is unnecessary.

4.2.4 OpenFlow switch internals

An OpenFlow switch contains one or more flow tables and a group table. These tables are used to check packets and decide where to send them. The switch also has a channel to connect with an external controller, as shown in figure above. The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol. The controller manages the switch by adding, updating, or removing flow entries in the flow tables. It can do this either in advance (proactively) or in response to packets arriving at the switch (reactively).

Each flow table in an OpenFlow switch contains a set of flow entries. Each flow entry consists of:

- **Cookie:** A unique value chosen by the controller. It can help filter flows. Example: `0x0`.
- **Timeouts:** Define how long the flow entry remains active:
  - **Hard timeout:** Specifies the flow entry's maximum lifetime, regardless of packet activity. Example: `50 seconds`.
  - **Idle timeout:** Removes the flow entry if no packets match within a given time. Example: `20 seconds`.
- **Priority:** Determines the order in which entries are checked. Example: `1`.
- **Match Fields:** Define packet characteristics to match, such as ingress port, source/destination addresses, or metadata. Example: `in_port=1, dl_src=00:00:00:00:00:01 dl_dst=00:00:00:00:00:02`.
- **Instructions:** Specify actions, like where to forward packets. Example: `actions=output:2`.
- **Counters:** Track statistics for matched packets, such as duration, packet count, and byte count. Example: `duration=764.169s, n_packets=242538, n_bytes=13851836796`.

The action in example specifies that the matched packets should be forwarded out from port 2 on the switch.

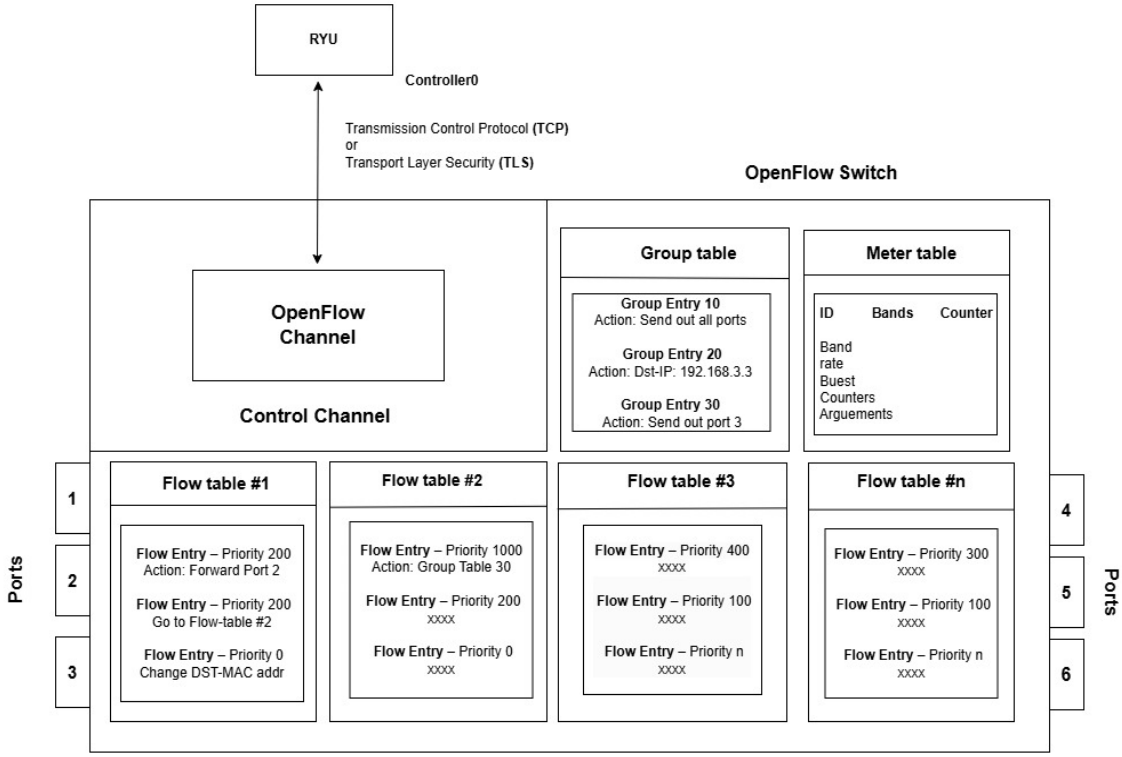


Figure 4.3: OpenFlow Switch Internals

4.2.5 Open vSwitch Architecture

Open vSwitch architecture consists of three planes: management, user space and kernel. The various parts of the architecture are as :

- **ovsdb-server:** The ovsdb-server is a key component of Open vSwitch that provides Remote Procedure Call (RPC) interfaces for managing Open vSwitch databases (OVSDB). It supports JSON-RPC communication, enabling clients to connect via active or passive TCP/IP or Unix domain sockets. This server manages the switch's database, including the switch table, and tracks external clients interacting with it. Clients use the OVSDB management protocol to read or update database tables.

  The server can work with database files specified on the command line, defaulting to /usr/local/etc/openvswitch/conf.db if none are provided. These database files must already be created and initialized using tools like ovsdb-tool create. By centralizing database management and communication, ovsdb-server plays a crucial role in configuring and controlling Open vSwitch deployments.

- **ovs-vswitchd:** It is the main userspace program of Open vSwitch, responsible for managing and controlling multiple OvS switches on the local machine. It operates as a daemon, reading the desired configuration from the ovsdb-server over an Inter-Process Communication (IPC) channel. Additionally, it updates the database with status and statistical information, enabling effective monitoring and management of the switches.

- **ovs-vsctl:** It is a command-line tool used to query and update the configuration of ovs-vswitchd, working in conjunction with the ovsdb-server. It allows users to perform various tasks such as configuring ports, adding or removing bridges, setting up bonding, and managing VLAN tagging. This makes it a versatile tool for managing Open vSwitch setups.

- **ovs-dpctl:**It is a command line tool to create, modify, and delete OvS datapaths. This program works only with datapaths that are implemented outside of 'ovs-vswitchd' itself, such as Operating System kernel-based datapaths.

- **ovsdb-tool:** It is a command-line program for managing OVSDB files. It does not interact directly with running OvS database servers

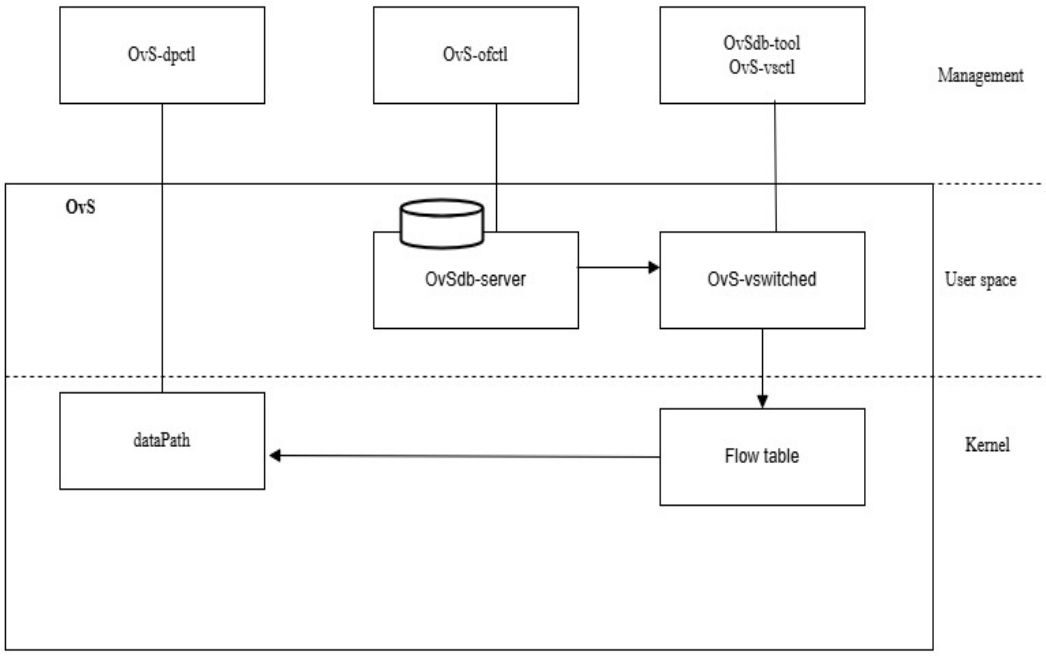


Figure 4.4: Open vSwitch

4.2.6 Distributed Denial of Service(DDoS)

A Distributed Denial of Service(DDoS) attack is malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of internet traffic.

- **Working of DDoS attack**

  Attackers use botnets, networks of compromised computers infected with malware, to generate large volumes of traffic for DDoS attacks. These computers are controlled remotely without the owners' knowledge. The attack floods the target with excessive traffic, causing it to become slow, unavailable, or even crash.

- **Types of DDoS Attacks**

  Volume-based Attacks: These include ICMP floods, UDP floods, and other spoofed-packet floods. The goal is to saturate the bandwidth of the attacked site.

  Protocol Attacks: These include SYN floods, fragmented packet attacks, Ping of Death, and more. They consume actual server resources or intermediate communication equipment, such as firewalls and load balancers.

  Application Layer Attacks: These include low-and-slow attacks, GET/POST floods, and more, targeting specific applications and services.

- **Identifying DDoS attack**

  The most obvious sign of a DDoS attack is a sudden slowdown or unavailability of a site or service. However, similar performance issues can also arise from legitimate traffic spikes, so it's important to investigate further. Traffic analytics tools are crucial for identifying potential DDoS attacks. They help spot signs such as unusually high traffic from a single IP address or range, indicating that the traffic might be orchestrated rather than organic. Additionally, a flood of traffic from users sharing the same device type, location, or browser version suggests coordinated behavior. Unexplained surges in requests to specific pages or endpoints can signal targeted attacks, while unusual traffic patterns, such as spikes at odd hours or repetitive trends, often indicate malicious activity. By analyzing

these signs, organizations can differentiate between legitimate traffic increases and DDoS attacks.

4.3 System Block Diagram

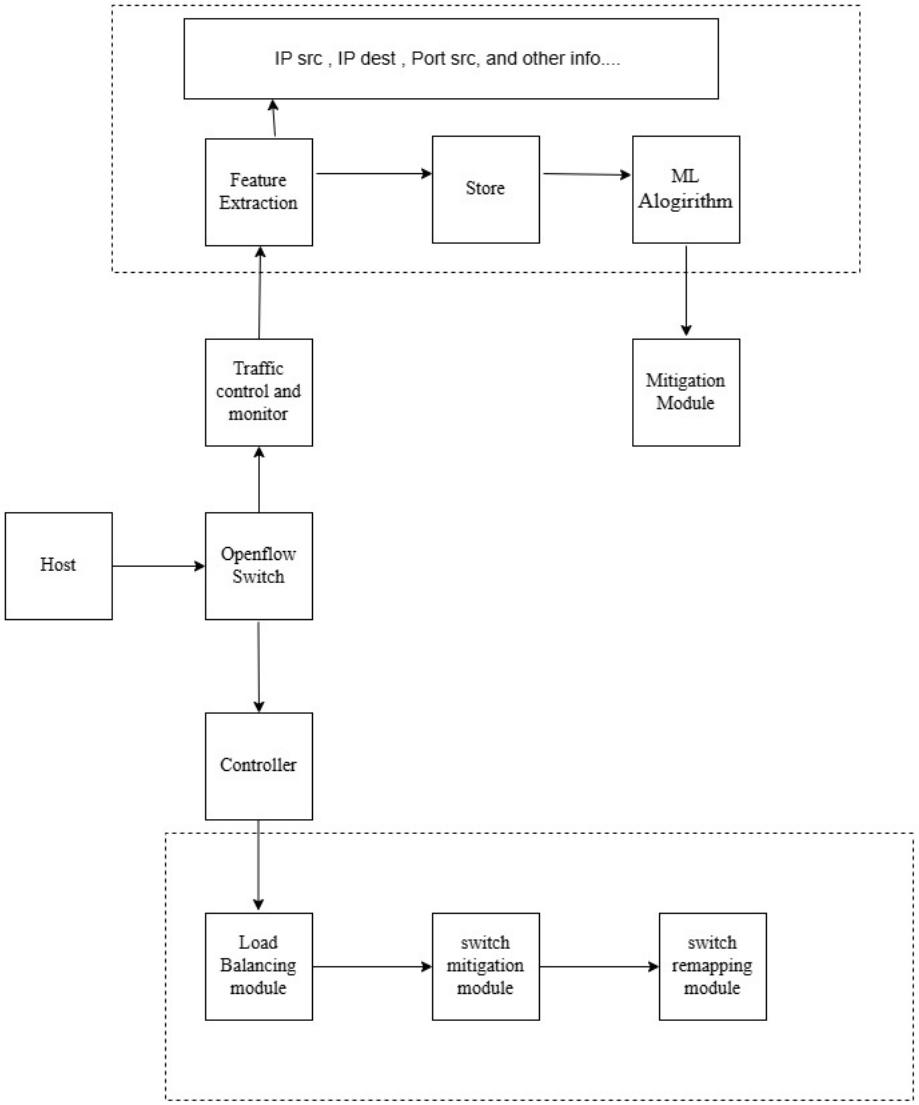


Figure 4.5: System Design

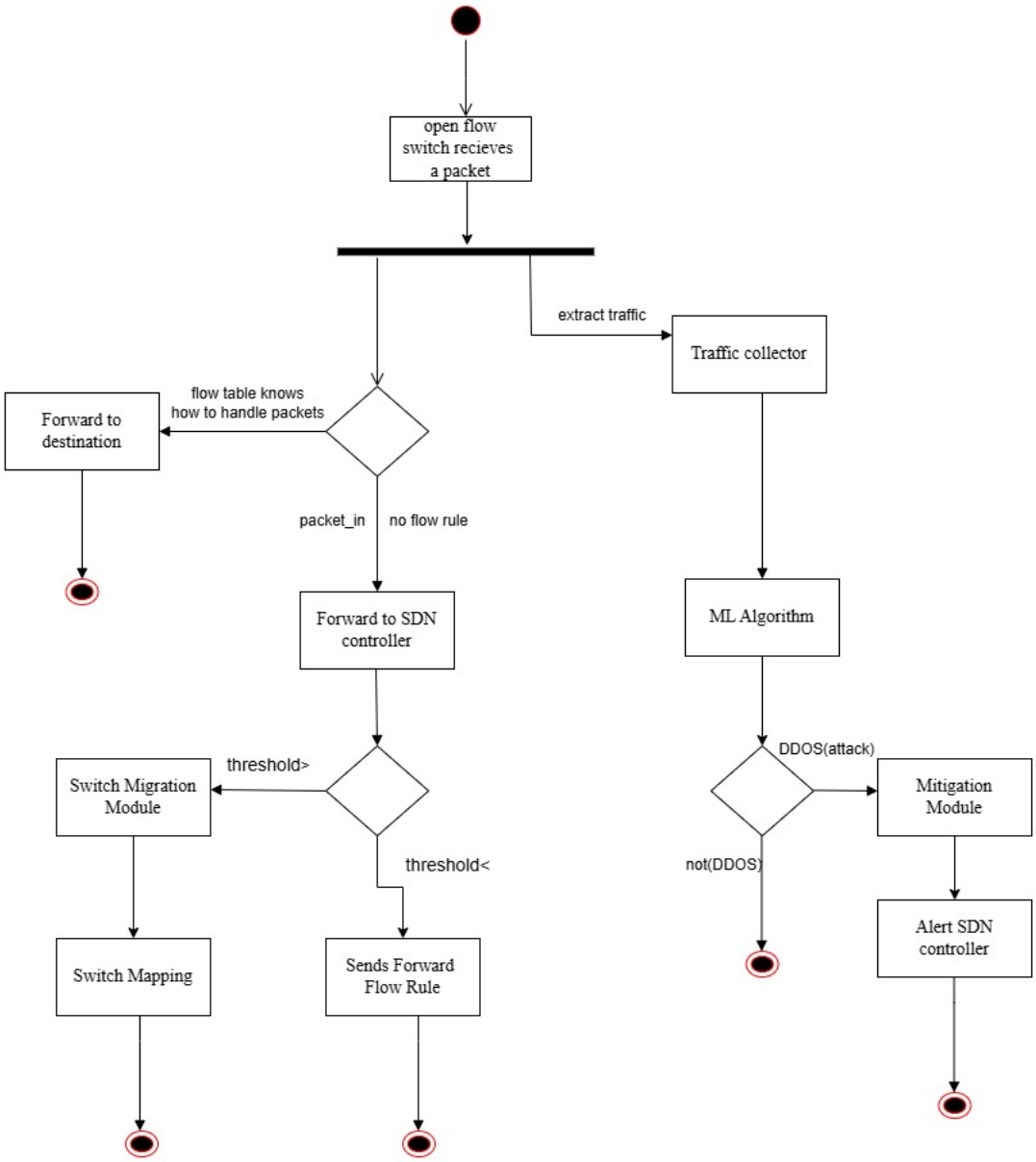


Figure 4.6: System Flow

Figure **??** shows the intended system block diagram of the proposed system.

4.3.1 Packet Incoming

This is the entry point for network packets. The packets can be from any source IP (IP src) and destined for any destination IP (IP det) using specific source and destination ports (Port src, Port det).

4.3.2 Traffic Collector

It collects the incoming network packets and prepares them for further processing. The collector may filter packets based on specific criteria, such as IP addresses, ports, or protocols.

4.3.3 Monitor

This component monitors the incoming traffic flow and can detect anomalies in traffic patterns, such as sudden spikes in traffic volume or unusual protocol usage.

4.3.4 Feature Extractor

It extracts relevant features from the packets that can be used to train and evaluate the machine learning model. The features can include packet length, packet header information, payload content, or other relevant attributes.

4.3.5 Store In Dataset

The extracted features are stored in a dataset for training and testing the ML model. The dataset is continuously updated with new features to ensure the model remains effective in detecting new threats.

4.3.6 ML Algorithm

This is the core of the intrusion detection system. The ML algorithm is trained on the collected dataset to learn patterns and anomalies associated with different types of attacks. The algorithm can be based on various machine learning techniques, such as decision trees, neural networks, or support vector machines.

4.3.7 Predict Result

The trained ML algorithm receives the extracted features of incoming packets and predicts the likelihood of an attack. The prediction is based on the learned patterns and anomalies from the training dataset.

4.3.8 If Attack

This is a decision point where the system determines if the predicted attack likelihood is high enough to warrant action. The threshold for this decision can be set based on the desired level of security and the cost of false positives.

4.3.9 Forward Packet

If the predicted attack likelihood is low, the packet is forwarded to its intended destination without any intervention.

4.3.10 Attack Mitigation

If the system detects a high probability of an attack, it takes steps to mitigate the threat. These steps can include blocking the attacker, dropping the malicious packet, or taking other security actions. The mitigation process can also involve alerting security personnel or triggering automated responses to contain the attack.

4.4 Description of Algorithms

4.4.1 XGBoost

XGBoost (eXtreme Gradient Boosting) is an advanced machine learning algorithm designed for supervised learning tasks such as classification and regression. It is based on gradient tree boosting, a method that builds an ensemble of decision trees in an iterative manner to optimize an objective function. XGBoost is particularly popular for its scalability, efficiency, and ability to handle large-scale datasets, making it the preferred choice for a variety of predictive modeling problems.

At the core of XGBoost is a regularized learning objective, which balances model accuracy and complexity to prevent overfitting. The objective function consists of two components: a differentiable convex loss function to measure prediction error and a regularization term to penalize model complexity. This is expressed as:

$$L(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \quad \text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2. \tag{4.1}$$

Here, $l(y_i, \hat{y}_i)$ measures the error between the predicted value $\hat{y}_i$ and the true value y_i, and $\Omega(f)$ is a regularization term that penalizes model complexity. T represents the number of leaves in a tree, and $\|w\|^2$ is the L2 norm of the leaf weights. The parameters γ and λ control the regularization strength.

XGBoost uses additive training, where new decision trees are sequentially added to the model to minimize the objective function. The predictions are updated by incorporating a new tree f_t that optimizes the following objective at each step:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t). \tag{4.2}$$

Each tree improves predictions by learning from the residual errors of the previous trees. The optimal weights for the tree leaves are calculated using:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{4.3}$$

where g_i and h_i are the first and second-order gradient statistics of the loss function, respectively, and I_j represents the set of instances in leaf j.

To find the best splits in decision trees, XGBoost employs a scoring formula to calculate the reduction in loss achieved by a potential split:

$$L_{\text{split}} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma, \tag{4.4}$$

where G_L and G_R are the sums of gradients for the left and right child nodes, H_L and H_R are the sums of second-order gradients, and γ is the regularization term for tree splits. This ensures that each split contributes effectively to the model's performance.

Additionally, XGBoost incorporates a weighted quantile sketch to propose candidate split points. This is particularly useful for handling large datasets with skewed distributions or missing values. The weighted rank for a candidate split z is calculated as:

$$r_k(z) = \frac{\sum_{(x,h)\in D_k, x<z} h}{\sum_{(x,h)\in D_k} h}, \tag{4.5}$$

where D_k is the dataset for feature k, and h represents the second-order gradient statistics.

XGBoost also includes features like shrinkage (learning rate adjustment) and column subsampling, which further enhance its ability to prevent overfitting and improve computation speed. These innovations, along with its ability to leverage parallel and distributed computing, make XGBoost a powerful tool for solving complex machine learning problems.

4.4.2 Model Evaluation

The primary metrics used to evaluate the performance of object detection models are the mean Average Precision (mAP) values calculated across different Intersection over Union (IOU)thresholds. In machine learning, the testing data is employed to conduct cross-validation, a process that validates the model's efficacy against unseen data. Various cross-validation techniques exist, which are broadly categorized as exhaustive and non-exhaustive approaches.Exhaustive cross-validation involves testing all possible

combinations and iterations of training and testing datasets. On the other hand, non-exhaustive cross-validation techniques generate randomized partitions of training and testing subsets. While the exhaustive approach offers more comprehensive insights into the dataset, it requires significantly more time and resources compared to non-exhaustive approache

4.4.3 Hyperparameter

A hyperparameter is a parameter whose value can be used to control the learning process.Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.The following are the attributes of hyperparameters

4.5 Performance Metrices

The terms used for correct, incorrect and missed detections for calculating different metrices are:

- TP (True Positive): The number of correct detections in each frame.
- FP (False Positive): The number of incorrect detections in each frame.
- FN (False Negative): The number of missed detections in each frame.
- TN (True Negative): The number of instances where the model correctly classifies a non-object as a non-object.

4.5.1 Precision

It measures measures the accuracy of the positive predictions made by a model.In other words, It measures the proportion of true positive predictions out of all positive predictions made.

$$Accuracy = \frac{Numberofcorrectpredictions(TP)}{Totalnumberofpredictions(TP+FP)} \tag{4.6}$$

4.5.2 Recall

Recall (sensitivity or true positive rate) measures the ability of a model to correctly detect all relevant objects of interest. It is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Unlike precision that only comments on the correct positive predictions out of all positive predictions, recall provides an indication of missed positive predictions.

$$Recall = \frac{TruePositives(TP)}{TruePositives + FalseNegatives(TP + FN)} \tag{4.7}$$

4.5.3 F1 Score

the F1 score is as a combined metric that balances both precision and recall, providing more comprehensive evaluation of the model's performance. The F1 score ranges between 0 and 1, where a score of 1 indicates perfect precision and recall, and a score of 0 indicates poor performance.

$$F1 - score = \frac{2 * Precision * Recall}{Precision + recall} \tag{4.8}$$

4.5.4 Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.It helps in understanding the model's performance by breaking down the predictions into categories such as true positives, false positives, true negatives, and false negatives.

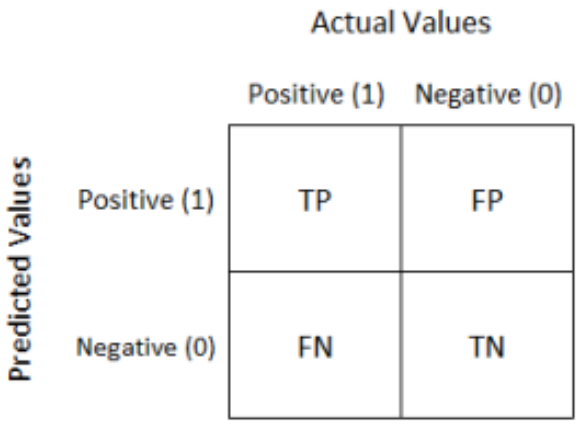


Figure 4.7: Confusion Matrix

4.5.5 Tuning

Model Tuning or hyperparameter tuning is the process of optimizing the hyperparameters of a machine learning model to improve its performance. Hyperparameters are the parameters that govern the behavior of learning algorithm and structure of the model.They can also be said as the configuration settings for the model training.In order to perform model tuning various hyperparameter optimization techniques can be utilized such as gradient descent , random search etc.

5 IMPLEMENTATION DETAIL

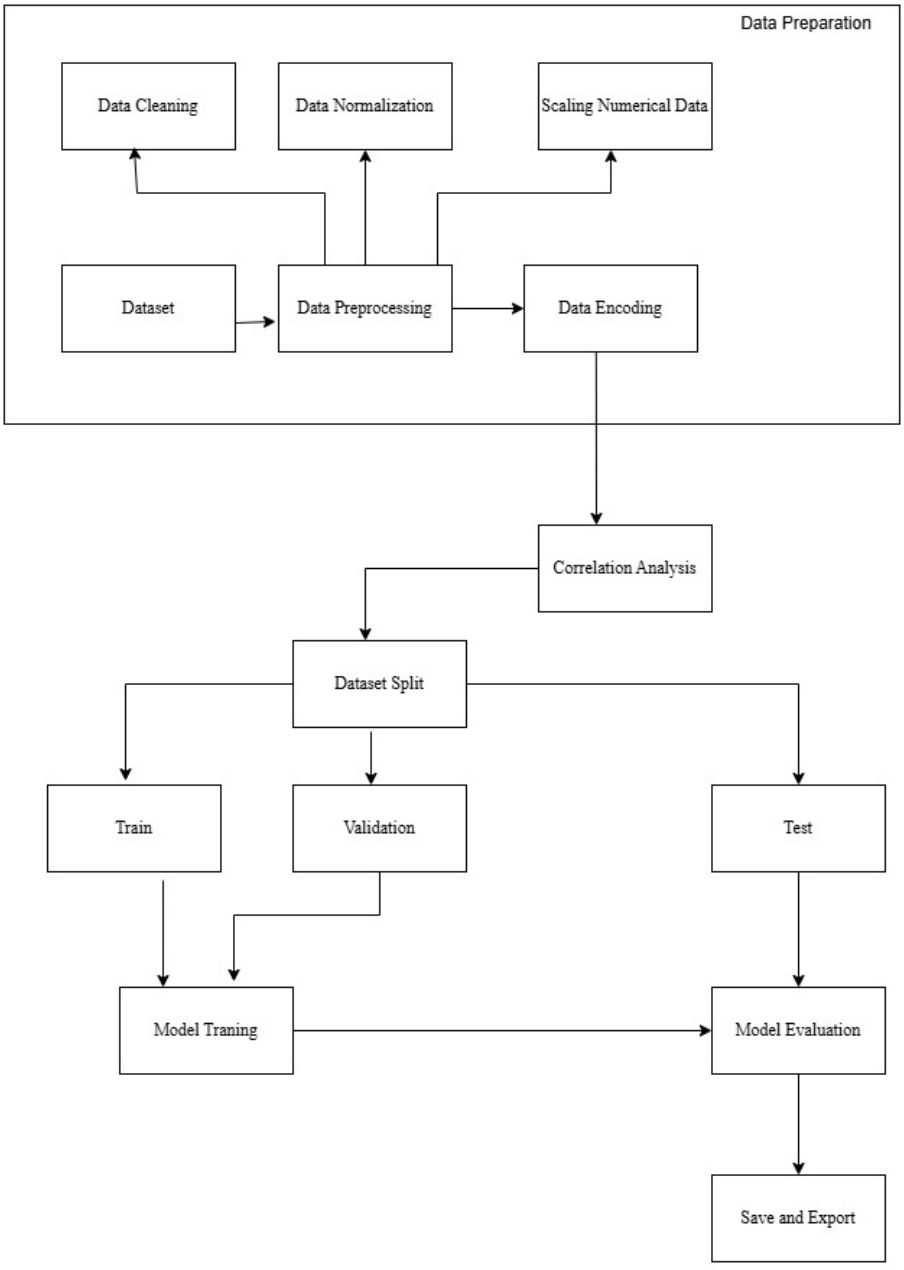


Figure 5.1: Implementation details

5.1 Dataset Creation

Dataset Creation is a critical aspect of the project. The dataset was created in our own environment using locally available resources.This section explains the detailed methodology for generating normal and malicious (DDoS) traffic using a Mininet network configured with multiple controllers and custom topologies. The dataset was generated

by simulating both regular network activities and various DDoS attack scenarios. This subsection explores the implementation and theoretical considerations underpinning the dataset generation process.

5.1.1 Network Topology Design

To simulate a realistic SDN environment, a custom multi-controller network topology was developed. The topology consisted of six switches (s1 through s6), each connected to two or more hosts, resulting in a total of ten hosts. Hosts were assigned specific IP addresses within the range 10.0.0.x/24, ensuring proper network segmentation and traffic control.

The topology was configured to include direct links between adjacent switches, forming a linear chain. This design emulates real-world hierarchical network structures, where traffic flows vertically across multiple tiers.

Three remote controllers were implemented, each assigned to manage specific switches within the network. Controllers c1, c2, and c3 operated on ports 6653, 6654, and 6655, respectively. These controllers ensured load balancing and efficient traffic distribution across the network. Each switch was explicitly assigned to one of the controllers as

- **Controller 1:** Manages Switch 1 and Switch 2.
- **Controller 2:** Manages Switch 3 and Switch 4.
- **Controller 3:** Manages Switch 5 and Switch 6.

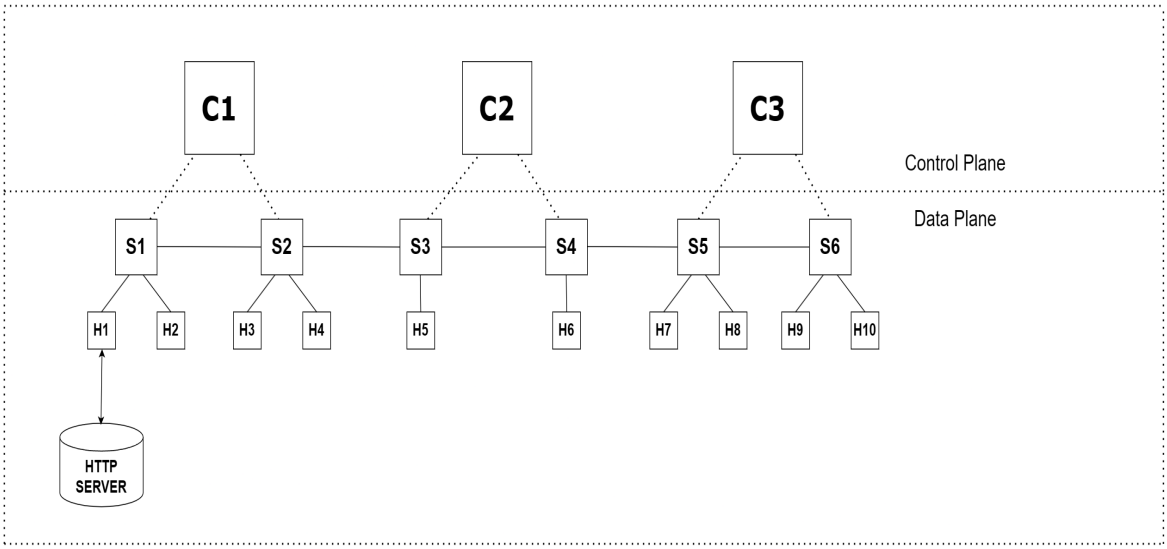


Figure 5.2: Network Topology Design

```
network@network-VirtualBox: ~/project/mininet
network@network-VirtualBox:~/project/mininet$ sudo python3 multi_controller_topo.py
*** Creating controllers
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(s1, h1) (s1, h2) (s1, s2) (s2, h3) (s2, h4) (s2, s3) (s3, h5) (s3, s4) (s4, h6) (s4, s5) (s5, h7) (s5, h8)
 (s5, s6) (s6, h9) (s6, h10)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Adding controllers
*** Starting network
*** Starting controller
c1 c2 c3
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Waiting for switches to connect
s1 s2 s3 s4 s5 s6
*** Testing network
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Results: 0% dropped (90/90 received)
*** Testing network again
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Results: 0% dropped (90/90 received)
*** Running CLI
```

Figure 5.3: Network Topology in Mininet

5.1.2 Simulating Normal Traffic

Normal network conditions are emulated for the dataset. To emulate normal network conditions, a host (h1) was configured as a central server, hosting an HTTP server and TCP/UDP servers using SimpleHTTPServer and iperf. Other hosts in the network acted as clients, simulating a variety of interactions with the server, such as:

Web Traffic

- **Host `h1`:** It acts as a web server hosting files and responds to HTTP requests.This is the server where other hosts can send their various types of normal web traffics.
- **Other hosts:** All the other hosts can send requests to the server to Download files (e.g., `index.html` and `test.zip`) from `h1` using `wget`.

ICMP Traffic

- This is the traffic related to ICMP requets. Hosts exchange ping requests to simulate ICMP traffic.

TCP and UDP Traffic

- **Iperf servers:** These servers are run on `h1` for both TCP (port 5050) and UDP (port 5051) traffic where other hosts can send requests to.
- **Other hosts:** They can initiate TCP and UDP connections to the server present`h1` on respective ports.

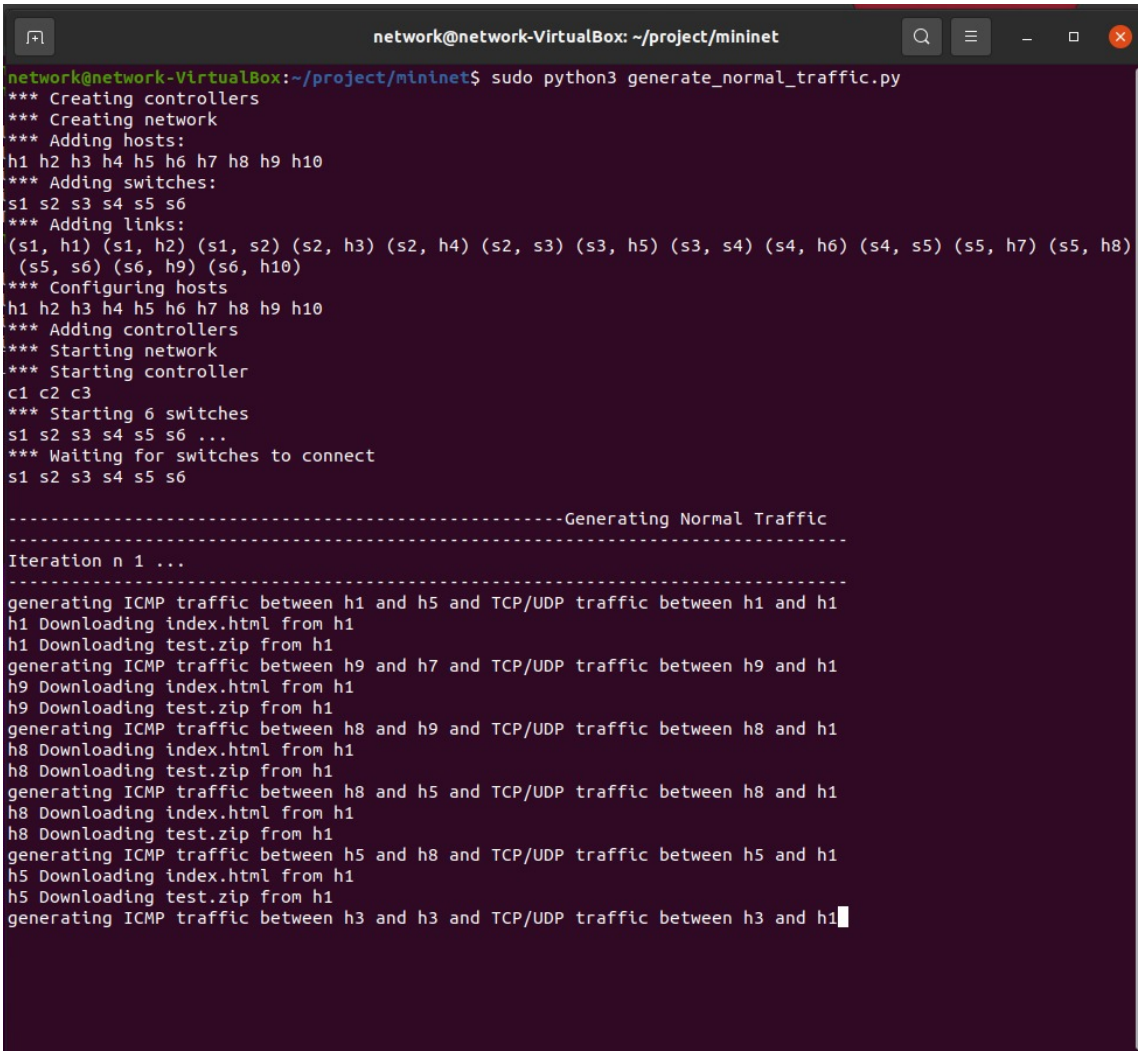


Figure 5.4: Simulating Normal Traffic

5.1.3 Traffic Scheduling

Traffic is generated iteratively, with randomized source hosts and destinations, ensuring varied traffic flows.This ensures that the traffic created is more diverse and randomized. The scheduling process includes the following steps:

- Randomly select a source host.
- Generate ICMP, TCP, and UDP traffic between the source host and a randomly generated destination IP.
- Simulate file downloads from the web server (‘h1’).
- Perform many iterations to create sufficient traffic for baseline analysis.

5.1.4 Simulating DDOS Traffic

The DDoS traffic simulation involved generating malicious activities using common attack patterns. Hosts within the network were leveraged to execute these attacks against target IP addresses, including h1, which acted as the victim. The following DDoS scenarios were simulated:

1. **ICMP (Ping) Flood:** An overwhelming number of ICMP packets were sent to a target, exhausting its resources.
2. **UDP Flood:** Continuous streams of UDP packets were directed to the victim, causing network congestion.
3. **TCP-SYN Flood:** Repeated SYN packets were sent to initiate connections without completing the handshake, overloading the victim's TCP stack.
4. **LAND Attack:** Spoofed packets with the same source and destination IP addresses were used to confuse the target, leading to resource depletion.

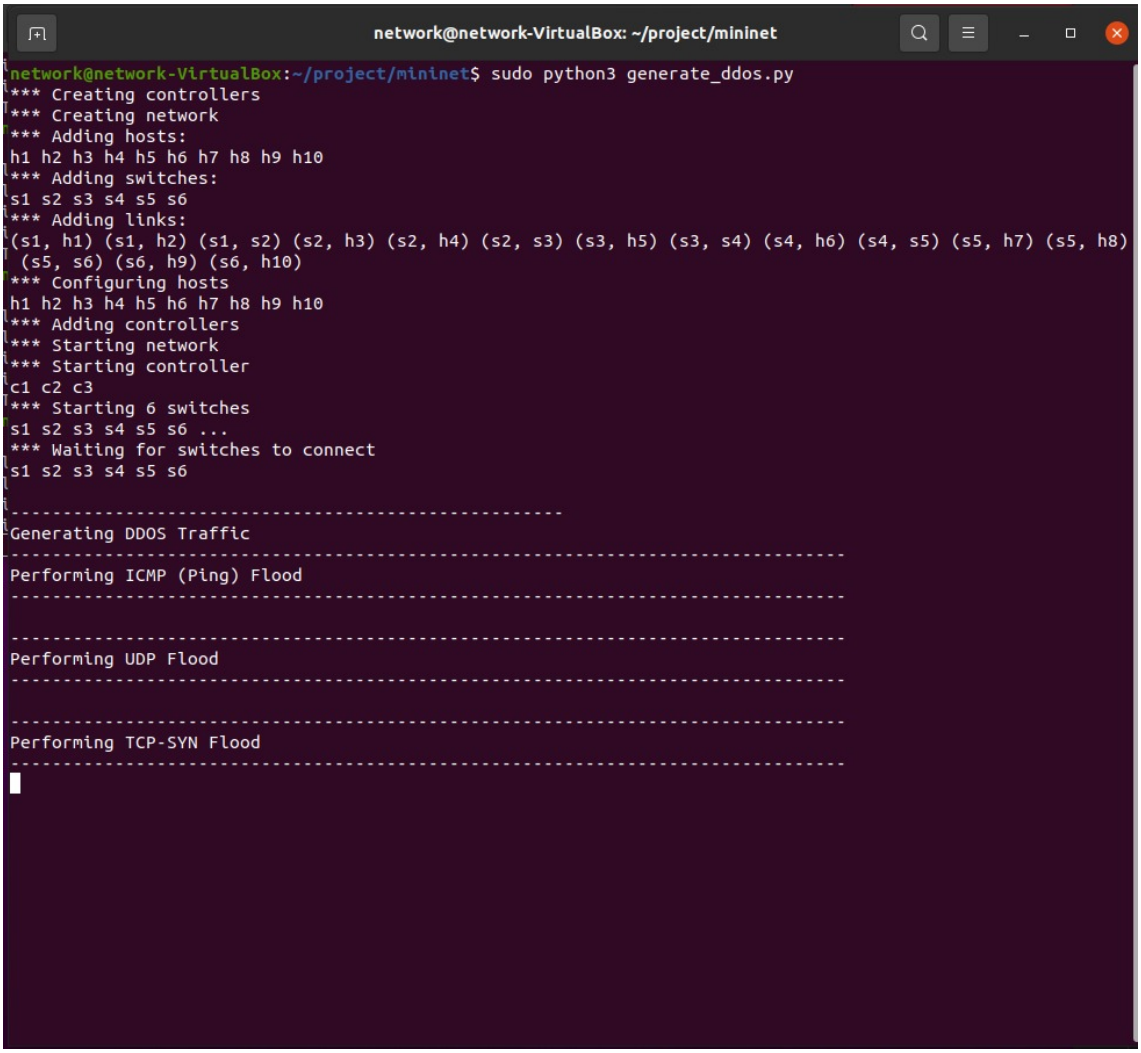


Figure 5.5: Simulating DDoS Traffic

5.1.5 DDoS Traffic Scheduling

The scheduling process for DDoS traffic generation mirrors the methodology used for normal traffic, with adaptations to simulate attack scenarios. The steps involved are as follows:

- **Randomly select a source host (attacker):** A host within the network is chosen to execute the attack.
- **Launch ICMP (Ping) flood attacks:** ICMP packets are sent in overwhelming numbers to a specified random victim (e.g., `h1`).
- **Initiate UDP flood attacks:** Continuous streams of UDP packets with randomized port numbers are directed at the randomized victim.

- **Execute TCP-SYN flood attacks:** SYN packets are repeatedly sent to initiate connections without completing the handshake, overloading the random victim's TCP stack.
- **Perform LAND attacks:** Spoofed packets with identical source and destination IPs are used to confuse the victim, leading to resource depletion.

5.1.6 Dataset Collection

The dataset collection process for this project involves the development of a custom Ryu controller application that monitors network traffic and extracts flow-level statistics from SDN switches. The captured data is systematically recorded in a CSV file to facilitate further analysis and training of machine learning models for DDoS attack detection.

Flow Monitoring

The controller application actively tracks connected switches (datapaths) and periodically sends requests to retrieve flow statistics. This ensures continuous monitoring of network traffic across all switches in the topology.These all are used for extraction of packet statistics.

Statistics Extraction

For each flow entry, the controller collects a comprehensive set of attributes, including:

- **Network-layer attributes:** Source and destination IP addresses.
- **Transport-layer attributes:** Source and destination ports for TCP and UDP traffic, and ICMP-specific fields (e.g., ICMP code and type).
- **Flow-level attributes:** Flow duration, packet count, byte count, and associated timeouts (idle and hard timeouts).
- **Derived metrics:** Packet count per second and byte count per second, calculated to enhance the dataset's utility for machine learning purposes.

Data Logging

The collected flow statistics are stored in a CSV file in a structured format, with each record containing several key fields. These fields include the Timestamp, which indicates the exact time when the flow statistics were captured. The Datapath ID identifies the switch from which the statistics originated, ensuring proper tracking of network activity across multiple devices. Each flow is assigned a unique Flow ID, based on the combination of source and destination IP addresses, ports, and protocol type, enabling easy identification and differentiation of flows. Additionally, the Traffic Metrics field contains critical data, such as packet count, byte count, and the derived rates for these metrics, providing insights into the traffic volume and intensity. Finally, Additional Protocol Information captures protocol-specific details, including ICMP-specific fields like the ICMP code and type, further enhancing the depth of the data collected for analysis.

Structured Dataset

The logged data is saved in csv file. The CSV file's structured format ensures compatibility with data preprocessing pipelines and machine learning models.

5.1.7 Dataset Attributes

All the data attribute which has been recorded for the dataset are as follows:-

- **Timestamp:** The exact time when the flow statistics were collected from the switch. It helps track when the data was recorded.
- **Datapath ID:** A unique identifier for the switch (datapath) in the network. It tells which switch the flow data belongs to.
- **Flow ID:** A unique identifier for each flow, formed by combining the source IP, destination IP, source port, destination port, and protocol. It helps differentiate between different network flows.
- **IP Src:** The source IP address from which the data is sent in the flow. This is the

origin of the network traffic.

- **TP Src:** The source transport layer port number (either TCP or UDP) used by the sender in the flow. It helps identify the application or service initiating the connection.
- **IP Dst:** The destination IP address where the data is being sent in the flow. This is the target of the network traffic.
- **TP Dst:** The destination transport layer port number (either TCP or UDP) used by the receiver in the flow. It indicates the service or application receiving the data.
- **IP Proto:** The protocol used in the flow, such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol), or ICMP (Internet Control Message Protocol). It helps understand how the data is being transmitted.
- **ICMP Code:** For ICMP traffic, this is the code field that provides additional information about the type of ICMP message. It helps further classify ICMP packets, such as whether it's a request or reply.
- **ICMP Type:** For ICMP traffic, this is the type field indicating the kind of ICMP message (e.g., echo request, echo reply). It helps categorize the message.
- **Flow Duration (Sec):** The duration of the flow in seconds. It shows how long the flow has been active.
- **Flow Duration (nSec):** The duration of the flow in nanoseconds. It provides more precision in measuring the flow's activity.
- **Idle Timeout:** The time duration (in seconds) a flow remains idle before it is removed from the flow table. It indicates how long a flow can stay inactive before being timed out.
- **Hard Timeout:** The maximum time duration (in seconds) for which a flow is allowed to exist in the switch's flow table, after which it is removed regardless of activity.

- **Flags:** Flags indicating special conditions of the flow, such as whether it is active, idle, or part of a specific flow rule set. This helps in identifying the status or characteristics of the flow.
- **Packet Count:** The total number of packets that have been transmitted in the flow. It helps measure the volume of traffic for that particular flow.
- **Byte Count:** The total number of bytes transmitted in the flow. It provides information about the overall data size transferred in the flow.
- **Packet Count per Second:** The rate at which packets are transmitted per second in the flow. It helps measure the flow's packet transmission rate.
- **Packet Count per nSecond:** The rate at which packets are transmitted per nanosecond in the flow. It gives a very fine-grained measurement of packet flow speed.
- **Byte Count per Second:** The rate at which bytes are transmitted per second in the flow. It helps measure the flow's data transmission rate.
- **Byte Count per nSecond:** The rate at which bytes are transmitted per nanosecond in the flow. This gives a highly detailed measurement of data transfer speed.
- **Label:** A static value used to classify the flow, often for labeling purposes in analysis or machine learning models. In this case, it's set to 0 but could represent different categories in other scenarios.

5.2 Exploratory Data Analysis

5.2.1 Frequency of Ip Source

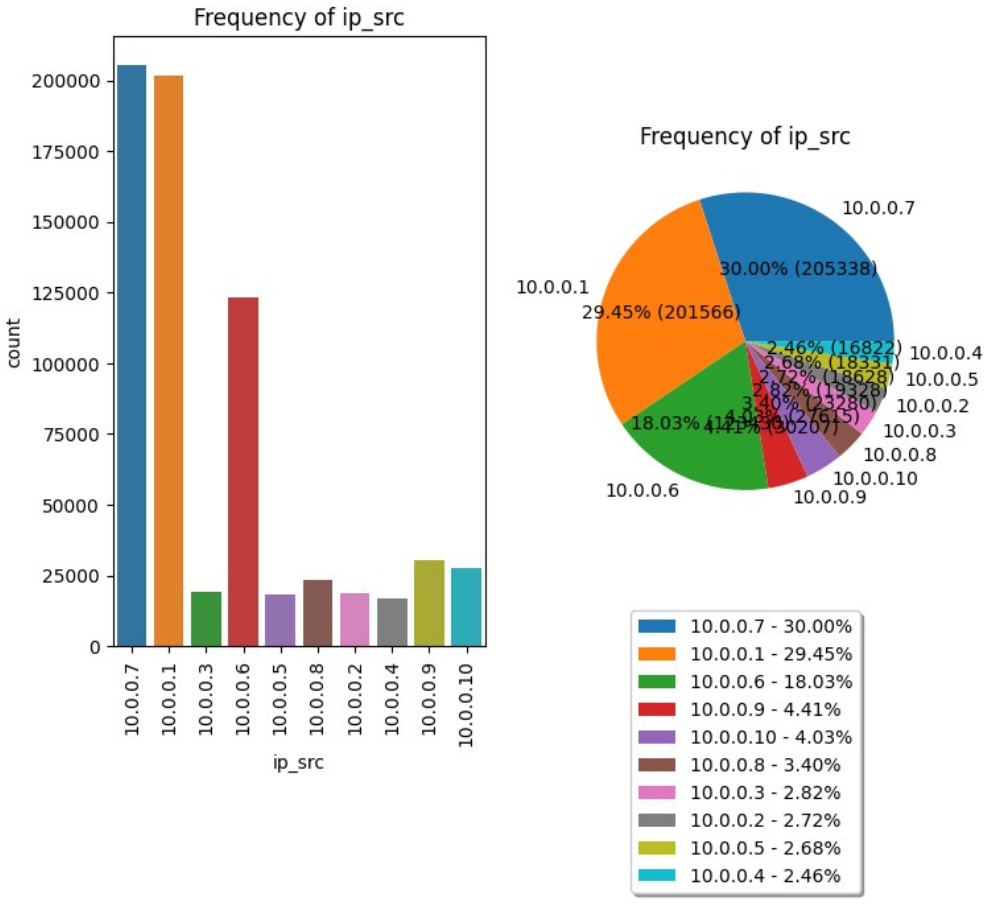


Figure 5.6: Frequency of Ip Source

The above figure highlights the frequency distribution of source IP addresses (ip-src) in a dataset through a bar chart and a pie chart. The bar chart shows the count of occurrences for each IP address, with 10.0.0.7, 10.0.0.1, and 10.0.0.6 clearly dominating, having significantly higher counts compared to the other IPs. The pie chart provides a proportional view, indicating that 10.0.0.7 represents 30.00 percent (205,338 occurrences), 10.0.0.1 accounts for 29.45 percent (201,566 occurrences), and 10.0.0.6 contributes 18.03 percent (123,412 occurrences). Together, these three IPs constitute the majority of the traffic. In contrast, the remaining IP addresses, such as 10.0.0.8 and 10.0.0.10, have relatively small shares, each contributing less than 5 percent. This distribution reflects a significant concentration of traffic among a few IPs, indicating potential network patterns or anomalies. Such insights are crucial for understanding traffic dynamics, detecting irregularities, and optimizing or securing the network.

5.2.2 Frequency of Ip Destination

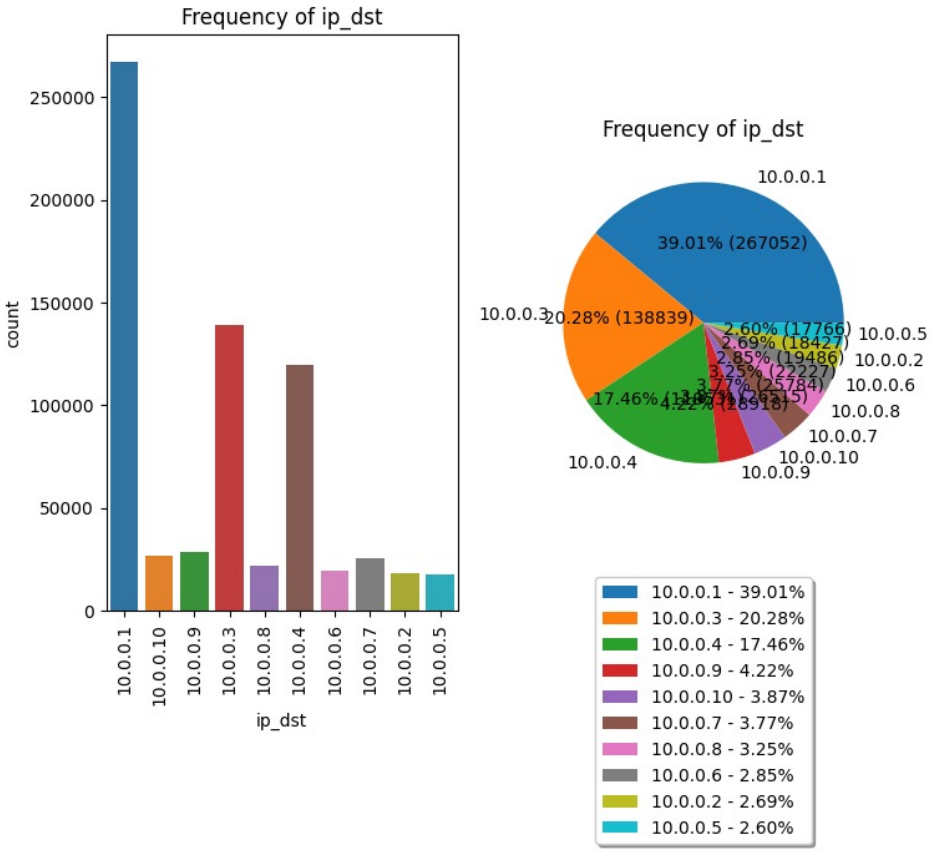


Figure 5.7: Frequency of Ip Destination

This figure above displays the frequency distribution of destination IP addresses (ip-dst) in a dataset using a bar chart and a pie chart. The bar chart reveals that 10.0.0.1 dominates with the highest count, followed by 10.0.0.3 and 10.0.0.4, while the remaining IPs, such as 10.0.0.9, 10.0.0.7, and 10.0.0.8, have much lower frequencies. The pie chart shows that 10.0.0.1 accounts for 39.01 percent (267,052 occurrences), 10.0.0.3 for 20.28 percent (138,839 occurrences), and 10.0.0.4 for 17.46 percent (119,487 occurrences), collectively handling the majority of the traffic. The other IPs, including 10.0.0.9 (4.22 percent) and 10.0.0.7 (3.77 percent), contribute less significantly, with the smallest shares being 10.0.0.2 (2.69 percent) and 10.0.0.5 (2.60 percent). This distribution highlights a concentration of traffic among a few destination IPs, suggesting specific routing patterns or potential anomalies, making it valuable for network traffic analysis and optimization.

5.2.3 Frequency of Datapath ID(Switch)

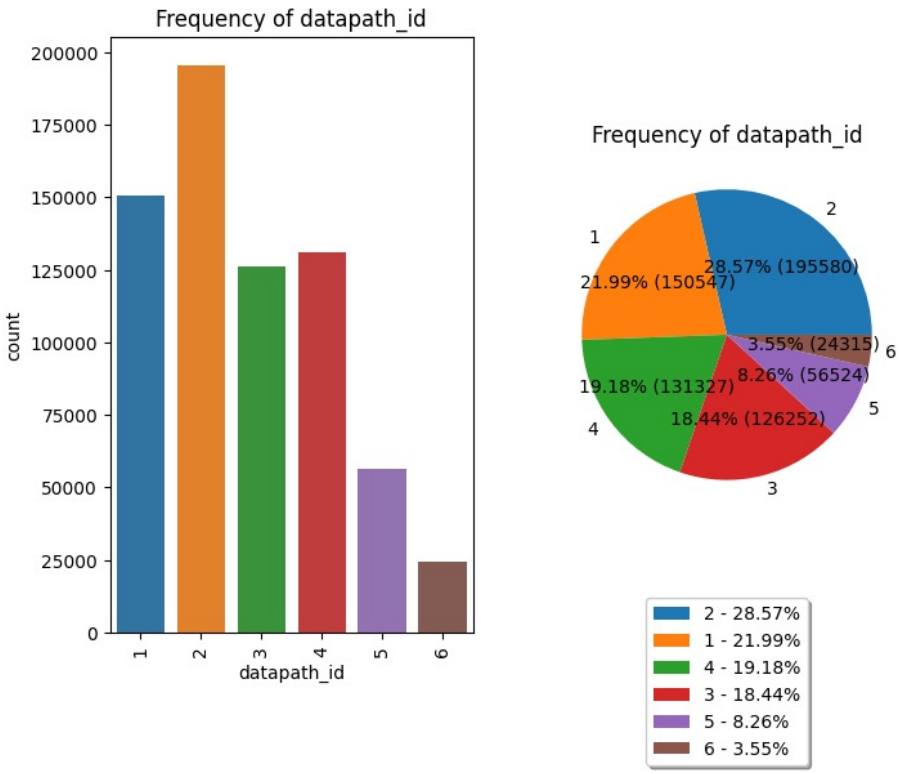


Figure 5.8: Frequency of datapath id

This figure illustrates the frequency distribution of the datapath id in a data set using a bar chart and a pie chart. The bar chart on the left shows the count of occurrences for each datapath ID, with 2 having the highest frequency, followed by 1, 4, and 3, while 5 and 6 have relatively lower counts. The pie chart on the right provides a proportional representation, where datapath id 2 accounts for 28.57 percent (195,580 occurrences), 1 contributes 21.99 percent (150,547 occurrences), 4 has 19.18 percent (131,327 occurrences) and 3 has 18.44 percent (126,252 occurrences). Meanwhile, 5 and 6 contribute smaller shares, 8.26 percent (56.724 occurrences) and 3.55 percent (26.515 occurrences), respectively. This distribution indicates that datapath-id 2 and 1 are the most active or significant datapaths, collectively handling over 50 percent of the traffic, while the others play relatively smaller roles. This analysis is useful for understanding the distribution of traffic across data paths, identifying critical pathways, and optimizing network performance.

5.2.4 Frequency of Ip Protocol

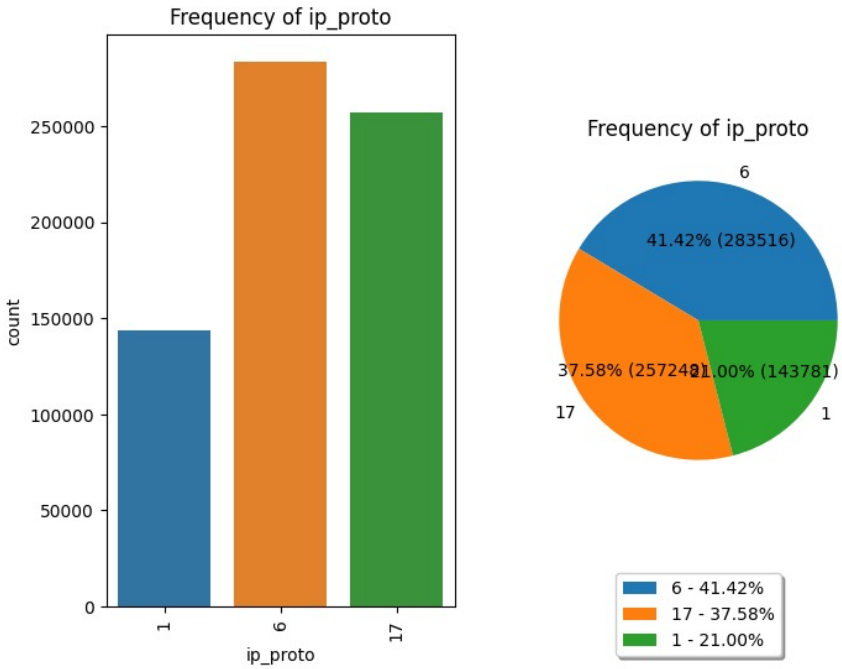


Figure 5.9: Frequency of Ip Protocol

The figure displays the distribution of the ip-protocol field, which represents different protocol types, using both a bar chart and a pie chart. The x-axis of the bar chart shows the protocol values (1, 6, and 17), while the y-axis represents their respective frequencies in the dataset. The data reveal that protocol type 6 has the highest occurrence, reaching approximately 283,516, followed by protocol 17 with 257,242 occurrences, and protocol 1 with the lowest count of 143,781.The pie chart complements this by showing the proportional distribution of these protocols: 6 makes up 41.42 percent of the total, 17 accounts for 37.58 percent, and 1 represents 21.00 percent. These visualizations suggest that protocol types 6 and 17 dominate the dataset, with protocol 1 being less common. This variability in the usage of the protocol could reflect the nature of the traffic or the specific network environment analyzed. The consistent representation across both charts provides a clear understanding of the frequency and proportional breakdown of protocol usage in the network data.

5.2.5 Frequency of ICMP Code

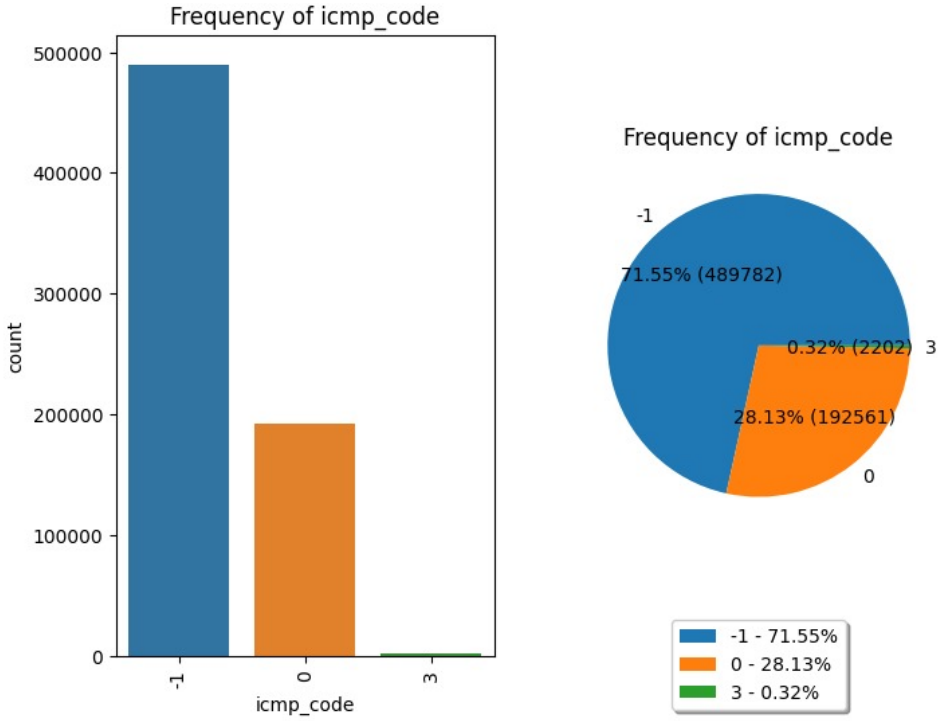


Figure 5.10: Frequency of ICMP Code

The figure shows the distribution of the icmp code values using a bar chart and a pie chart. The x-axis of the bar chart represents the icmp-code values (-1, 0, and 3), while the y-axis shows their respective frequencies. The data reveal that the icmp-code value -1 has the highest frequency, with 489,782 occurrences (71.55 percent). This is followed by the value 0, which has 192,561 occurrences (28.13 percent). The value 3 has the lowest frequency, appearing only 2,202 times (0.32 percent).The pie chart provides a visual representation of these proportions, highlighting the dominance of the icmp code value -1. The substantial difference in the occurrence of these values suggests that -1 is the most significant or commonly observed code in this dataset, while 0 is moderately frequent, and 3 is rare. These charts together illustrate the distribution of icmp-code values, providing insights into the network data's characteristics.

5.2.6 Frequency of ICMP Type

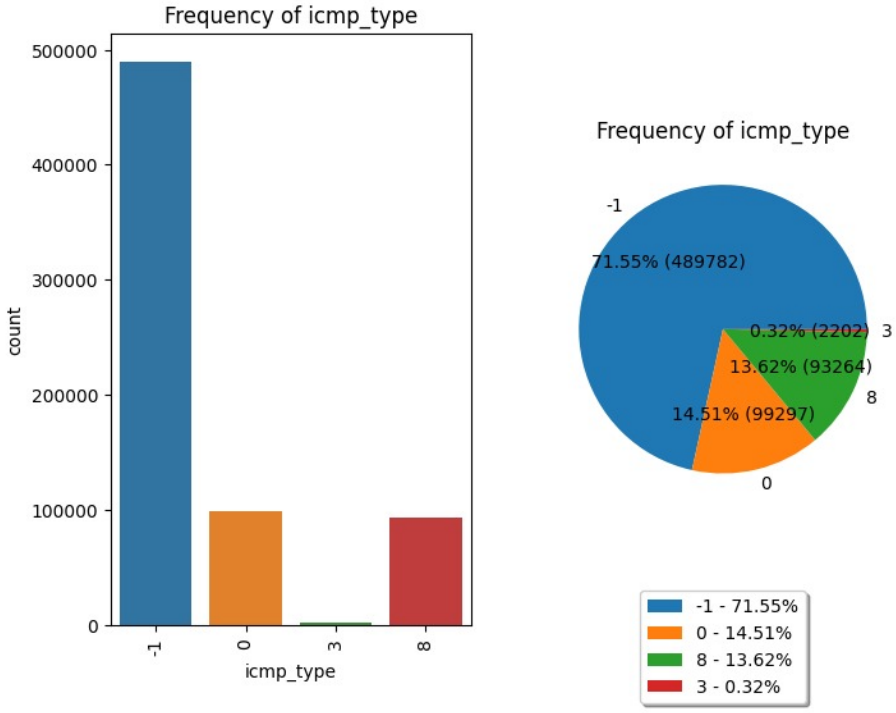


Figure 5.11: Frequency of ICMP Type

The figure illustrates the distribution of icmp-type values using a bar chart and a pie chart. The x-axis in the bar chart represents the icmp-type values (-1, 0, 3, and 8), while the y-axis indicates their respective frequencies. The data shows that the icmp-type value -1 has the highest frequency, appearing 489,782 times (71.55 percent). This is followed by the value 0 with 99,297 occurrences (14.51 percent) and value 8 with 93,264 occurrences (13.62 percent). The value 3 has the lowest frequency, appearing only 2,202 times (0.32 percent).The pie chart visually reinforces these proportions, emphasizing the dominance of -1 as the most frequent icmp-type value, with 0 and 8 being moderately frequent and 3 being rare. These visualizations suggest that -1 is a key icmp-type in the dataset, while values 0 and 8 also play a significant role, albeit to a lesser extent. The analysis highlights the variability in the distribution of icmp-type values, potentially reflecting patterns or characteristics of the underlying network data.

5.2.7 Frequency of Label

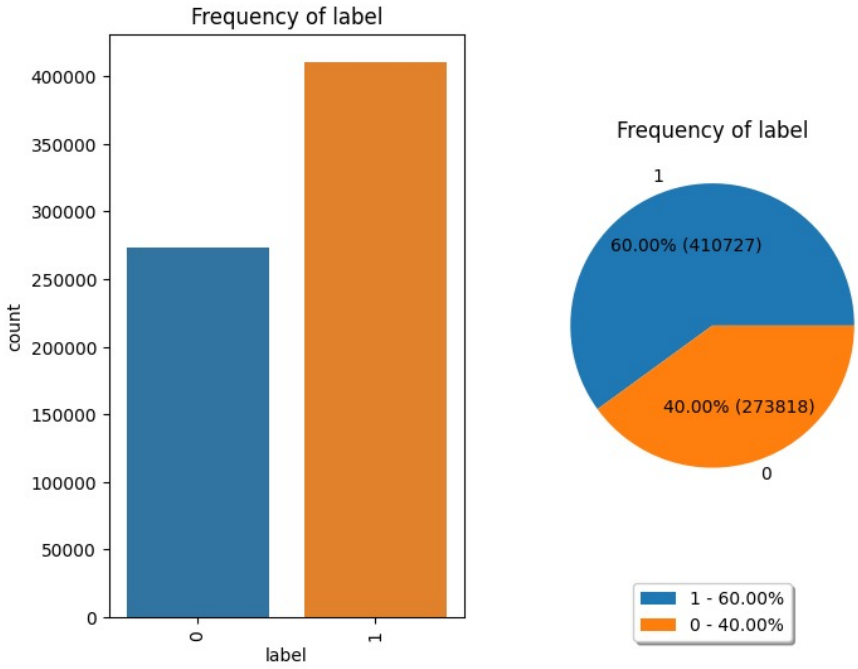


Figure 5.12: Frequency of Label

The figure provides a detailed visualization of the distribution of labels in the dataset, where 0 corresponds to normal data and 1 corresponds to DDoS data. The bar chart reveals that normal data (0) occurs 273,818 times, accounting for 40.00 percent of the total dataset, while DDoS data (1) appears significantly more frequently, with 410,727 occurrences, representing 60.00 percent. The pie chart reinforces this disparity, visually highlighting the dominance of DDoS data over normal data. This distribution suggests that the dataset contains a substantial imbalance, with DDoS data being more prevalent than normal observations. The total number of data points in the dataset is 684,545, reflecting the combined occurrences of normal and DDoS data. This imbalance could play a critical role in subsequent analyses, such as network traffic modeling or anomaly detection, emphasizing the need for strategies to address the over representation of DDoS data in the dataset.

5.2.8 Distribution Plot of Timestamp

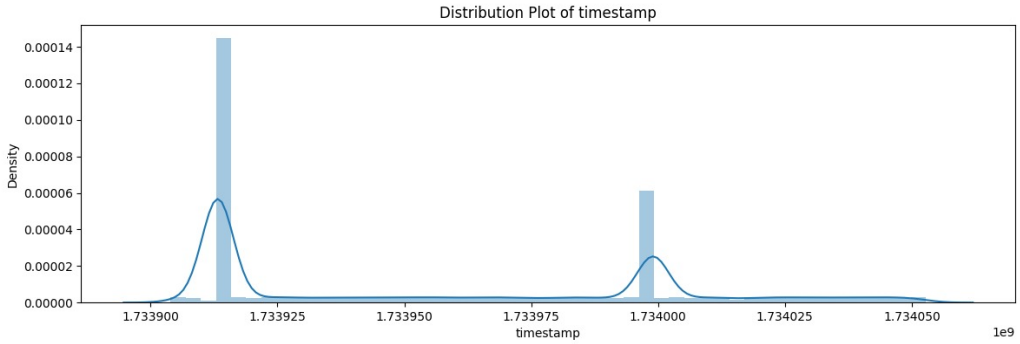


Figure 5.13: Distribution Plot of timestamp

The distribution plot illustrates the timestamp density of forward packets received over a period, and the observed peaks suggest the timing of potential Distributed Denial of Service (DDoS) attacks. The x-axis represents the timestamps, while the y-axis indicates the density of forward packets. The prominent peaks in the graph indicate instances of unusually high network traffic, which could correspond to times when a DDoS attack was executed. These attacks flood the network with a large number of packets in a short time, overwhelming the system and causing a sudden spike in activity. Between these peaks, the density significantly decreases, reflecting normal or reduced traffic levels when no such attacks were observed. Overall, the figure provides a visual representation of abnormal traffic patterns, with the peaks acting as indicators of DDoS attack events. This insight is crucial for analyzing network vulnerabilities, identifying attack timings, and implementing appropriate countermeasures to mitigate their impact.

5.2.9 Distribution Plot of Tp Source

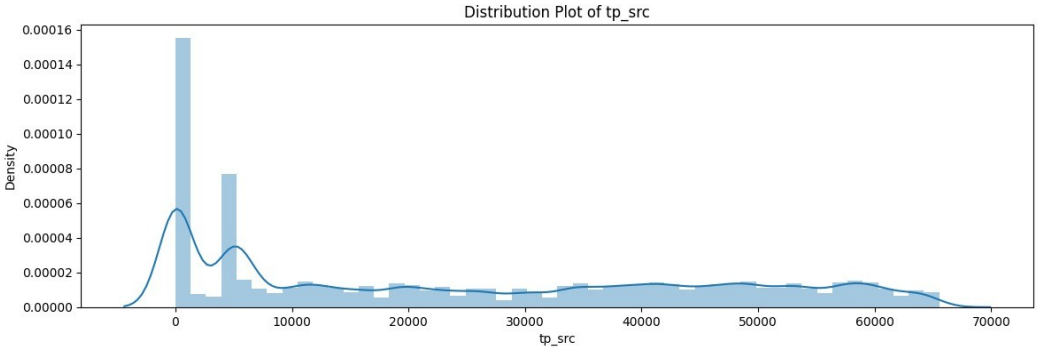


Figure 5.14: Distribution plot of tp source

The distribution plot of tp-src (source ports) illustrates the density of source port usage in the observed network traffic. The x-axis represents the source port numbers, while the y-axis shows the density of occurrences for each port. The plot reveals prominent peaks at lower port numbers, indicating that these ports were frequently used, likely due to standard communication protocols or targeted network activities. As the source port numbers increase, the density gradually declines, suggesting that higher port numbers were utilized less frequently, which aligns with typical network behavior where lower ports are reserved for well-known services. The distinct peaks in specific port ranges may also point to unusual activity, such as DDoS attacks, where attackers repeatedly target certain ports to flood the network. Beyond the peaks, the distribution spreads across a broader range of higher ports, indicating more random or diverse usage, possibly from ephemeral connections. These patterns provide valuable insights into the nature of network traffic, helping to identify potential vulnerabilities or targets of malicious activities, such as specific ports being exploited during an attack. Understanding these trends can inform strategies to secure the network, such as monitoring high-density ports or implementing access controls.

5.2.10 Distribution of Tp Destination

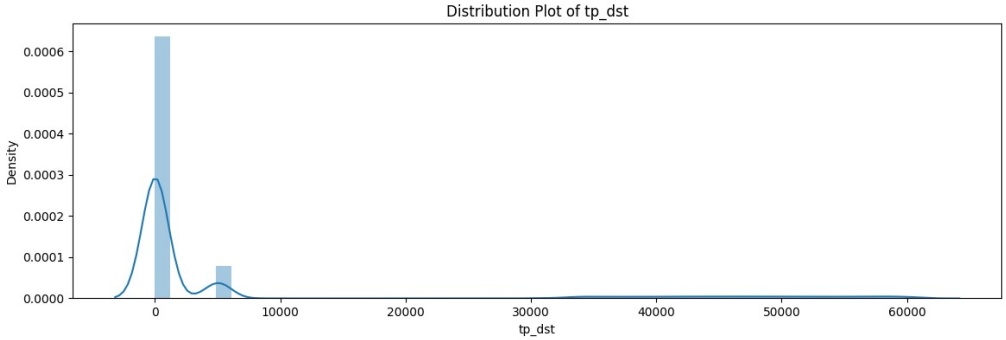


Figure 5.15: Distribution Plot of tp destination

The distribution plot of tp-dst (destination ports) provides insights into the density of destination port usage in network traffic. The x-axis represents destination port numbers, while the y-axis shows their density. The plot reveals a prominent peak at lower port numbers, indicating that these ports are frequently used or targeted in network communications. This is expected, as lower ports are typically associated with well-known services and protocols. Beyond this peak, the density sharply decreases, suggesting that higher port numbers are used less frequently. This aligns with typical network behavior, where most traffic concentrates on specific service-related ports. The sharp concentration at lower ports may also suggest potential anomalies, such as targeted activities during DDoS attacks or exploitation attempts focusing on critical services. While the density at higher ports is minimal, there is a subtle spread, possibly representing background traffic, ephemeral connections, or less common activities. Overall, the plot highlights the importance of monitoring traffic to frequently targeted destination ports, enabling network administrators to implement defenses such as firewalls or access controls to mitigate potential threats.

5.2.11 Distribution Plot of Flow Duration In Seconds

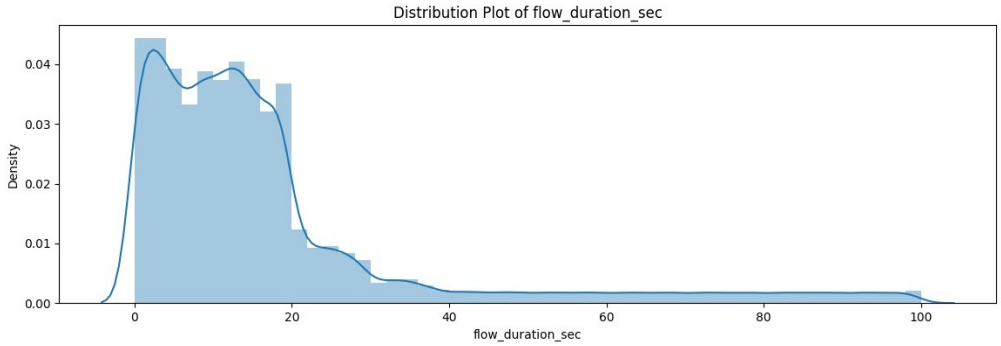


Figure 5.16: Distribution plot of flow duration in sec

The distribution plot of flow duration-sec provides insights into the density of flow durations in seconds. The x-axis represents flow durations (in seconds), while the y-axis shows their density. The plot reveals a prominent peak at shorter durations, indicating that most network flows are brief. This is expected, as many network communications involve short-lived transactions, such as web requests or small data transfers.Beyond this peak, the density gradually decreases, forming a long tail that extends toward higher durations. This suggests that while longer flows are less common, they do occur and may represent specific use cases such as large file transfers, streaming, or prolonged connections. The long tail may also hint at potential anomalies, such as inefficient or stuck connections.The sharp concentration at shorter durations may suggest routine network behavior, but monitoring these flows for unexpected spikes or patterns is critical to identify potential threats, such as floods of short-lived connections during DDoS attacks. Similarly, the long-duration flows warrant examination, as they could signal unusual activities or inefficient resource usage. Overall, this plot underscores the importance of analyzing both common and uncommon flow durations to enhance network efficiency and security. Network administrators can use this data to fine-tune monitoring tools, optimize resource allocation, and address potential vulnerabilities effectively.

5.2.12 Distribution Plot of Flow Duration In Nano-Seconds

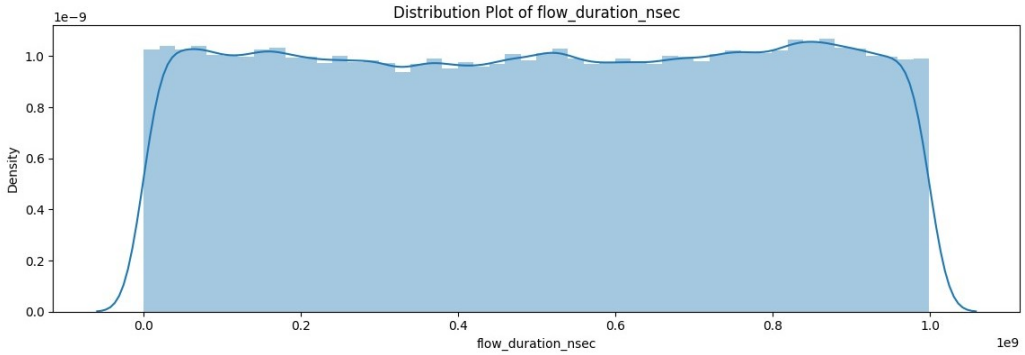


Figure 5.17: Distribution plot of flow duration in n-sec

The distribution plot of flow duration-nsec provides insights into flow durations in the context of potential DDoS activity. The x-axis represents flow durations in nanoseconds, while the y-axis shows their density. The plot reveals a nearly uniform distribution, with consistent density across most of the range, suggesting a high volume of flows with evenly spread durations. This behavior aligns with characteristics of DDoS attacks, where malicious traffic often generates flows with randomized or uniform patterns to overwhelm the target system. At the extremes (near 0 and 1×10^9nanoseconds), the density sharply decreases, indicating fewer flows at these boundary durations. However, the flat density in the main range could reflect the deliberate strategy of attackers to evade detection by mimicking normal traffic patterns. The uniformity of flow durations might also signify automated attack tools generating traffic at consistent intervals. Overall, this plot underscores the importance of analyzing flow duration patterns to detect potential DDoS attacks. The even spread of durations, combined with the sharp drop-offs at the boundaries, may point to abnormal traffic patterns requiring further investigation. Network administrators should closely monitor these flows and implement mitigation strategies, such as rate-limiting or anomaly detection systems, to counteract possible DDoS threats.

5.3 Data Preprocessing

5.3.1 Data Cleaning

Data cleaning is the process of identifying and removing errors and inconsistencies in data to improve its quality. It is a critical step in data preprocessing that ensures the accuracy, consistency, and reliability of data before it is used for analysis, modeling.In this step, the missing data values in our dataset were identified and the corresponding rows were removed from the dataset. Similarly, the NAN values and infinity values which are the noises in our dataset were also removed respectively. The dataset also consists of various data which were constant throughout.These data can be considered redundant as they would not contribute any meaningful information to the learning process. These redundant values can lead to inefficiencies, overfitting, and reduced model performance.Hence,they were removed from the dataset.

5.3.2 Data Transformation

Dataset transformation is a critical step in data preprocessing that involves converting raw data into a format suitable for analysis and modeling without changing the content of the dataset. In this step, the timestamp of the dataset is normalized to convert a timestamp column in a DataFrame to a datetime object and sort the DataFrame by this timestamp. The dataset is also scaled to standardized the data. It is done by removing the mean and scaling to unit variance. This process is essential to ensure that each feature contributes equally to the analysis and improves convergence.

5.3.3 Data Encoding

Data encoding is an essential preprocessing step in machine learning, especially when working with categorical data. Many algorithms are designed to work with numerical inputs, so categorical data—like names, categories, or labels—needs to be converted into a numerical format to be used effectively. The goal of encoding is to transform these non-numeric values into numbers in a way that maintains their relationships and ensures they make sense to the algorithm. For example, categories like ”red,” ”blue,” and ”green” can be assigned numerical codes, or rankings might be used for ordered categories. This process helps the algorithm interpret and process the data correctly, ensuring the model learns from it effectively.

Label Encoding

Label encoding is a simple method that converts each category in a feature into a unique integer. For instance, categories could be assigned like values of 0 and 1. This approach works well for binary categories, such as ”yes” and ”no,” which are typically labeled as 0 and 1 in our data.

IP Encoding

In this step,categorical data such as IP addresses are transformed into a format suitable for models, which typically require numerical input. In this step, source and destination ip addresses are encoded respectively.

5.3.4 Correlation and Feature Extraction

Correlation analysis is a statistical method used to measure and describe the strength and direction of the relationship between two or more variables, with correlation coefficients ranging from -1 to 1 indicating the degree and direction of association. It helps to understand how variables are related to each other. Feature selection using correlation analysis involves identifying and selecting the most relevant variables for building a predictive model by examining the relationships between features and the target variable, as well as the relationships among the features themselves. In this step, highly correlated columns were dropped to address multicollinearity issues and reduce redundancy in the dataset. This is done iteratively by analyzing step by step.

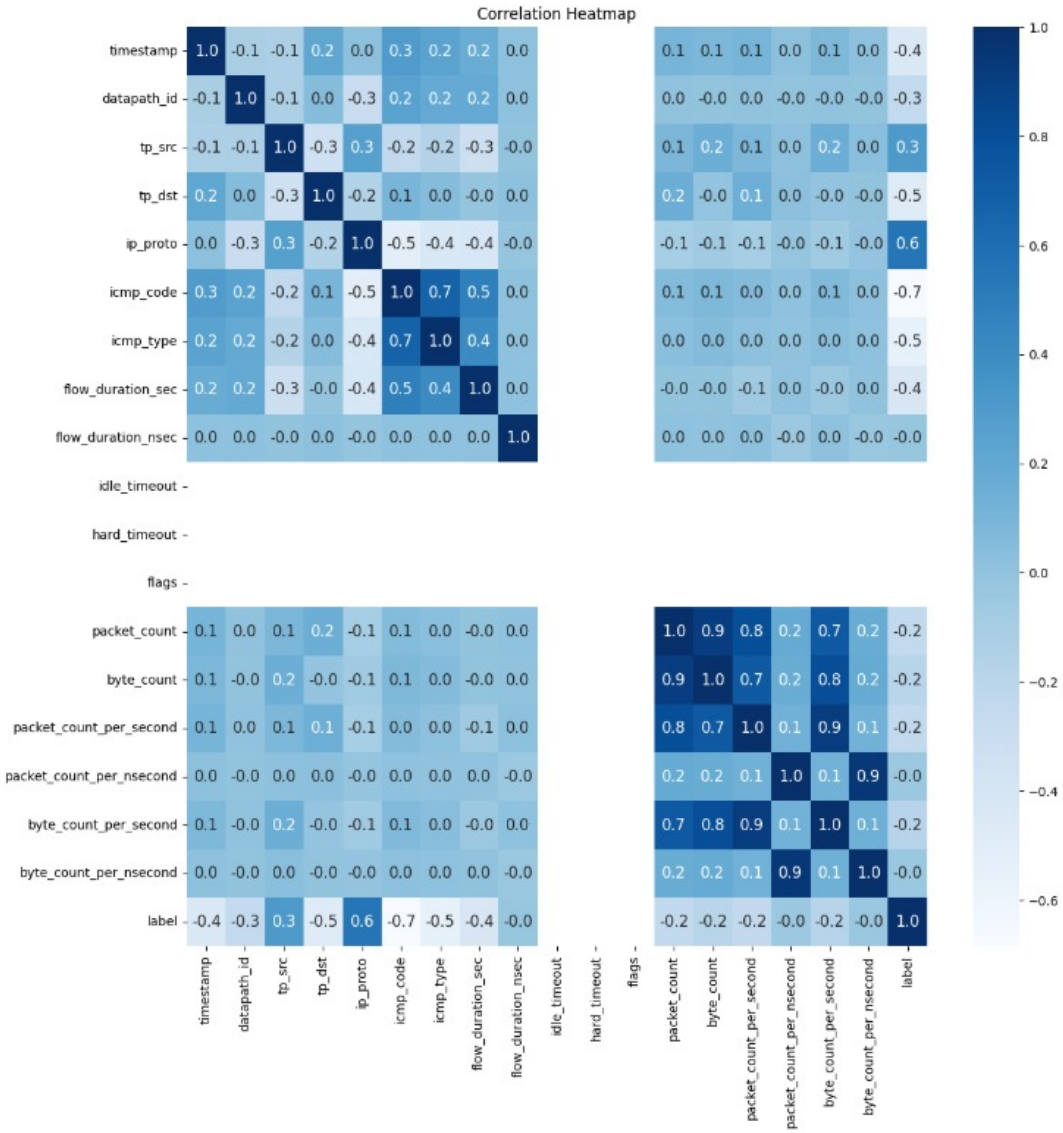


Figure 5.18: Correlation

Here, We can see the flags, idle_timeout, hard_timeout has score of 0 as the value in the whole dataset has same value and has no importance for the training of our model.Thus after removing these data attributes there is new correlation map which seems satisfactory for our model.

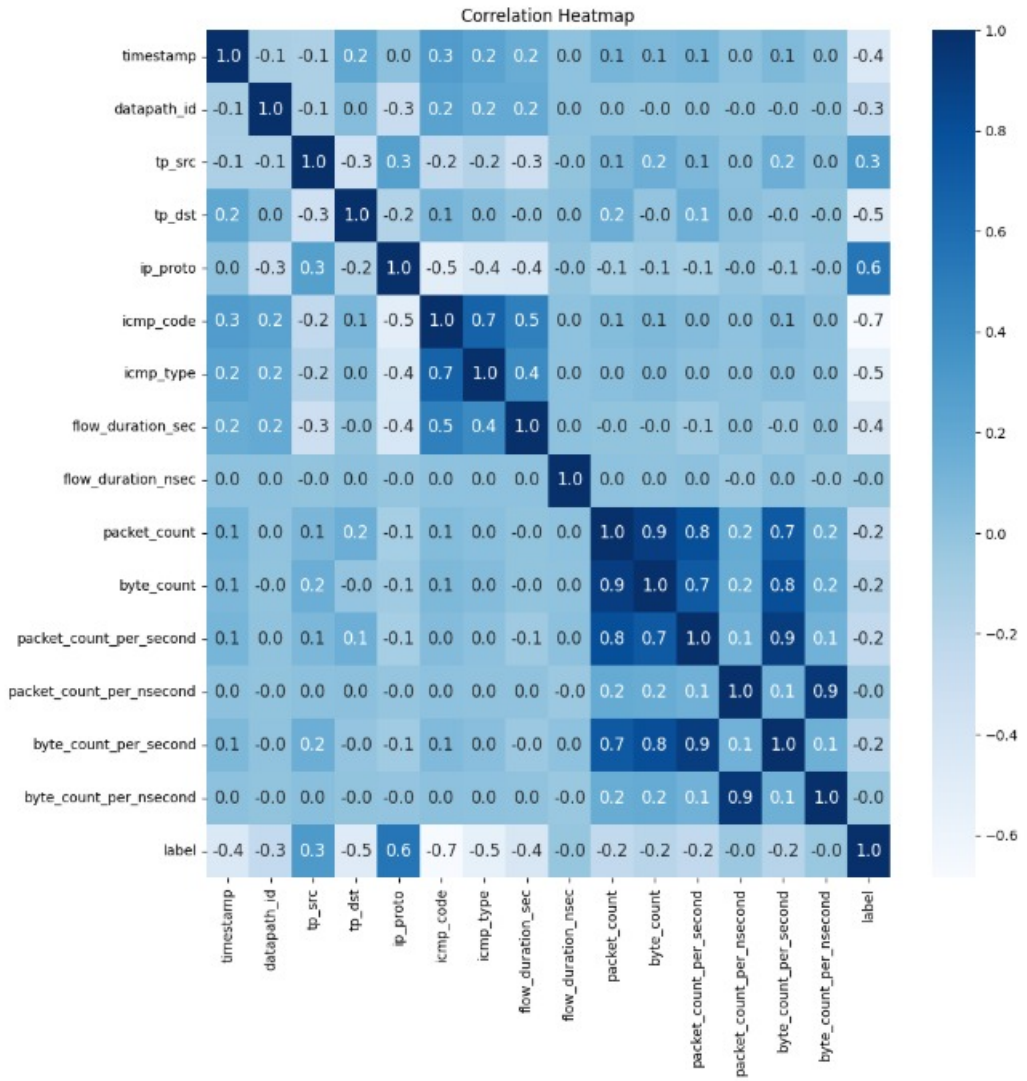


Figure 5.19: Correlation

5.3.5 Model Training

Hyper parameters	Value
n_estimators	[100,200,300]
max_depth	[None,5,10,20,30]
min_child_weight	[1]
gamma	[0,0.5,1,1.5,2,5]
subsample	[0.6,0.8,1.0]
colsample_bytree	[0.6,0.8,1.0]
colsample_bylevel	[0.6,0.8,1.0]
reg_alpha	[1]
reg_lambda	[0]

Network Topology

In order to implement the system we require a robust network topology. This can be done either with physical infrastructure with all the necessary hardwares to implement switches and host to configure a network. However, it is very expensive and a easier alternative is to emulate the same infrastructure in our existing hardware by the use of Mininet. The implementation is no different than using exact hardware. Thus, the network topology for this project is designed and implemented using Mininet, a network emulation tool that enables the creation and testing of virtual network environments. The topology consists of three key components: controllers, switches, and hosts. These components are interconnected to simulate a structured and efficient network.

Controllers

The topology employs three remote controllers implemented using the Ryu SDN framework. These controllers are responsible for managing the network's operation and ensuring smooth communication between devices. Each controller is linked to two

switches, dividing the network into three manageable segments. This segmentation allows for better load balancing and fault isolation.

Switches

Six switches are utilized in this topology, all of which support OpenFlow version 1.3. OpenFlow serves as the communication protocol between the switches and the controllers. The switches are distributed among the controllers as follows:

- **Controller 1:** Manages Switch 1 and Switch 2.
- **Controller 2:** Manages Switch 3 and Switch 4.
- **Controller 3:** Manages Switch 5 and Switch 6.

This hierarchical arrangement ensures that each controller oversees a specific part of the network, reducing complexity and improving overall performance.

Hosts

The topology includes ten hosts, which are end devices that connect to the switches. These hosts are distributed across the switches as follows:

- **Switch 1:** 2 hosts
- **Switch 2:** 2 hosts
- **Switch 3:** 1 host
- **Switch 4:** 1 host
- **Switch 5:** 2 hosts
- **Switch 6:** 2 hosts

The allocation of hosts ensures balanced network utilization and prevents congestion. Hosts connected to the same switch can communicate directly, while inter-switch communication is managed by the controllers.

5.4 Openflow Communication

This section gives a detail explanation of the communications within the established network topology for the working of the network.

5.4.1 Initial Openflow Handshake

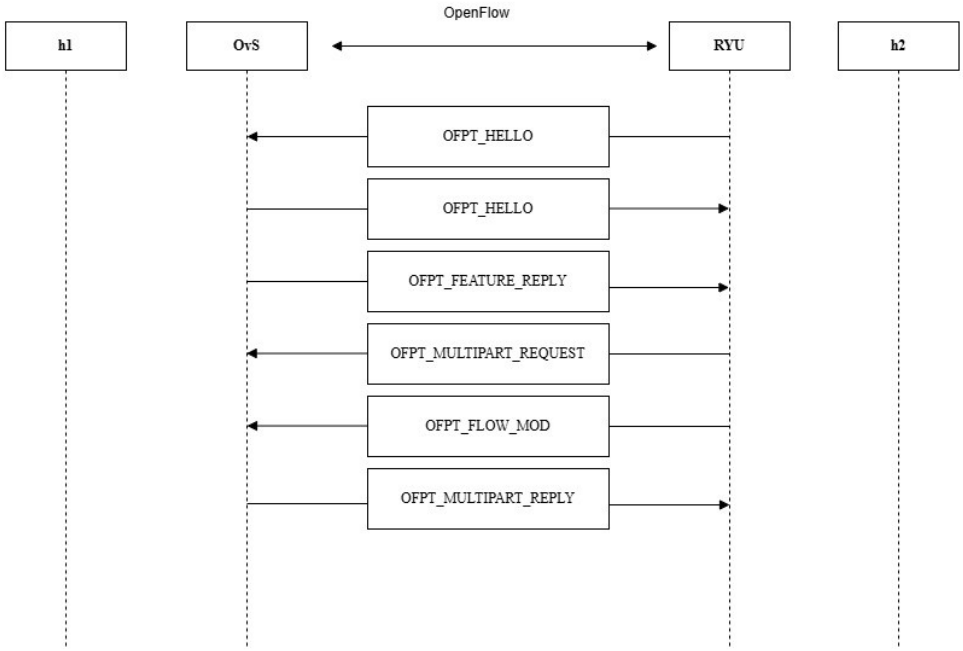


Figure 5.20: Openflow Handshake

After the connection has been established between controller and switches, the Ryu controller and the Switch exchange OFPT_HELLO messages, each can determine the OpenFlow version of the other and work on lowest common version. After this initial exchange each device can exchange further messages using OpenFlow v1.3 in our system. Next the Ryu controller sends an OFPT_FEATURES_REQUEST message to get the Datapath ID (DPID) and capabilities of the switch. The DPID is a 64 bit number that uniquely identifies a datapath. The lower order 48 bits are used for the switch MAC address, while the higher order 16 bits are defined by the implementer, for example for a VLAN ID. The switch responds with a OFPT_FEATURES_REPLY which includes the DPID and the capabilities supported by the datapath. The Ryu controller then sends a multipart request for the switch port descriptions to which it receives a reply with a description of all the ports on the switch that support OpenFlow.

The Ryu controller sends the switch an OpenFlow Flow Modification message. This message specifies that for any match, the output should be sent to the controller. The switch then sends PFPT_PORT_STATUS messages to the Ryu controller for each of its ports as shown in figure.

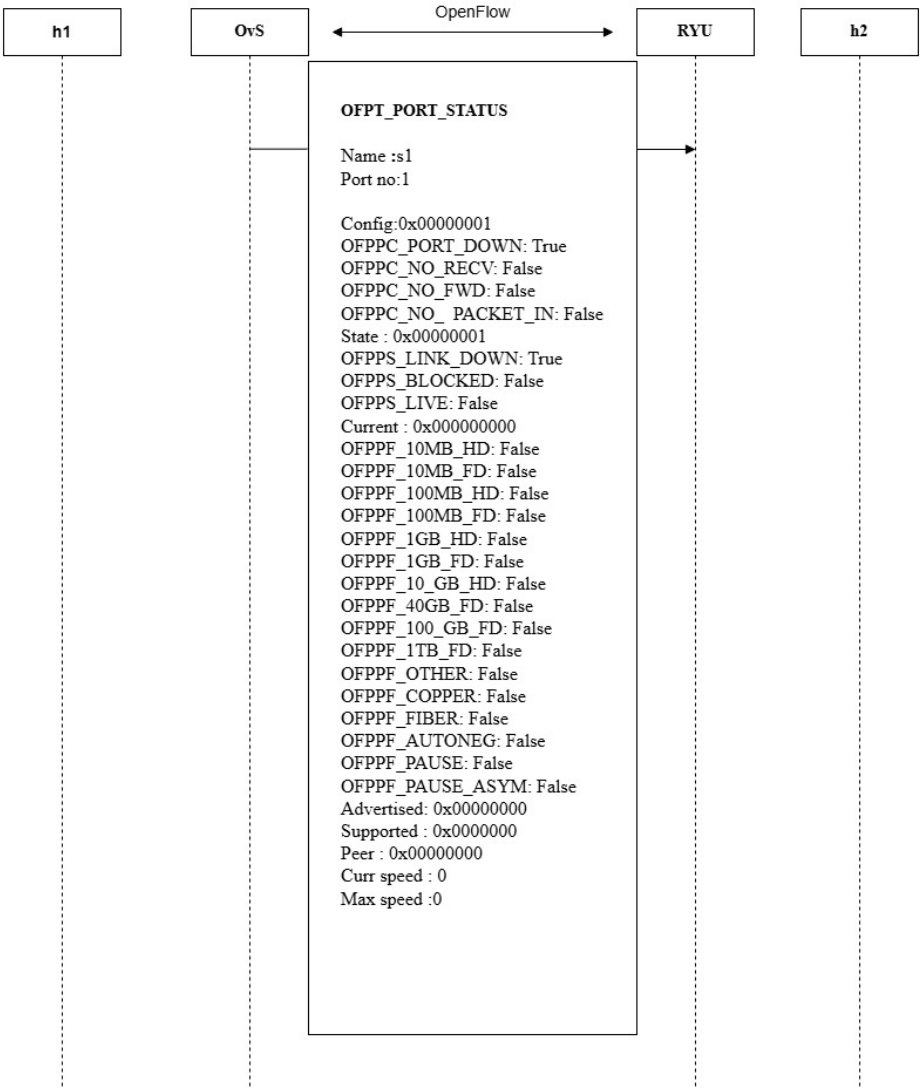


Figure 5.21: Openflow Port Status

5.4.2 Echo Request-Reply

The OpenFlow switch periodically sends an OFPT_ECHO_REQUEST message to the Ryu controller to verify connectivity and ensure the controller is responsive. In return, it expects an OFPT_ECHO_REPLY message from the controller as confirmation.

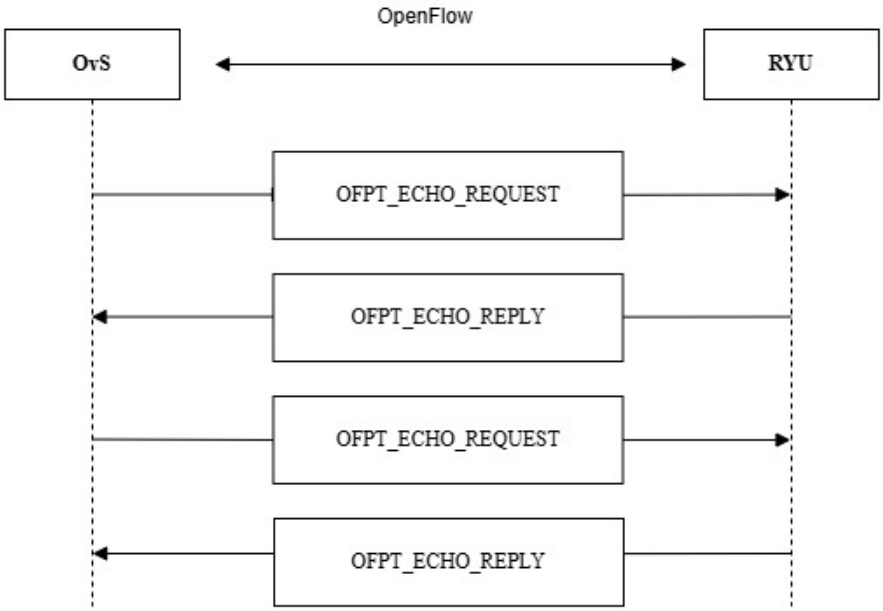


Figure 5.22: Echo Request-Reply

5.4.3 Table-miss Flow Entry

As switch receives packets, ICMP packets is received at the switch.It carries out a table entry check to see if there is existing flow rules on the table and finds that it has a 'table-miss' no flow entry. To handle table miss condition , the Ryu controller proactively installs a wildcard flow entry in the switch's flow table. This entry, with the lowest priority of 0, acts as a default rule that matches all packets.This ensures that no packet is dropped simply because there isn't a matching rule in the flow table.When a packet hits this entry, it is forwarded to the controller using the Controller port action (action = output:Controller port). This allows the controller to analyze the packet and decide how to process it dynamically, ensuring unmatched packets are not dropped.

ICMP Packets

This is an example of table-miss flow entry in switch for ICMP packets. When table-miss happens at switch then The OvS sends an OFPT_PACKET_IN message to the Ryu controller with a unique Buffer Identifier 256 for a decision. The Ryu controller responds to the OvS switch using the Buffer IDentifier 256 with the decision to flood to all switch all ports since this is a ICMP packet.

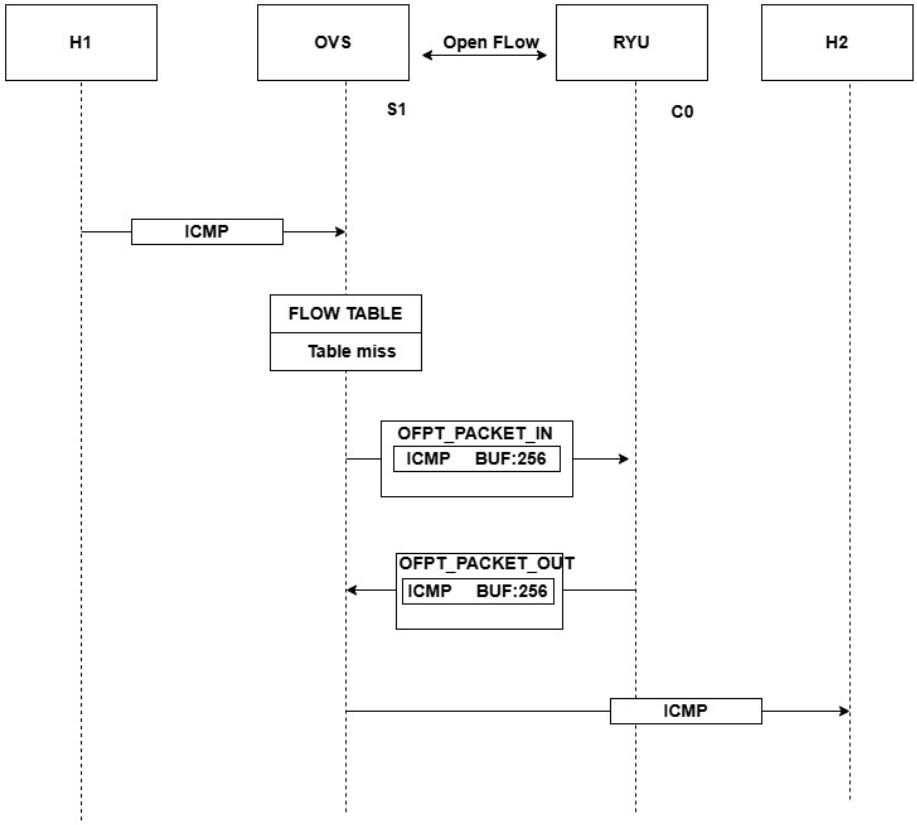


Figure 5.23: ICMP Packets

Normal Traffic

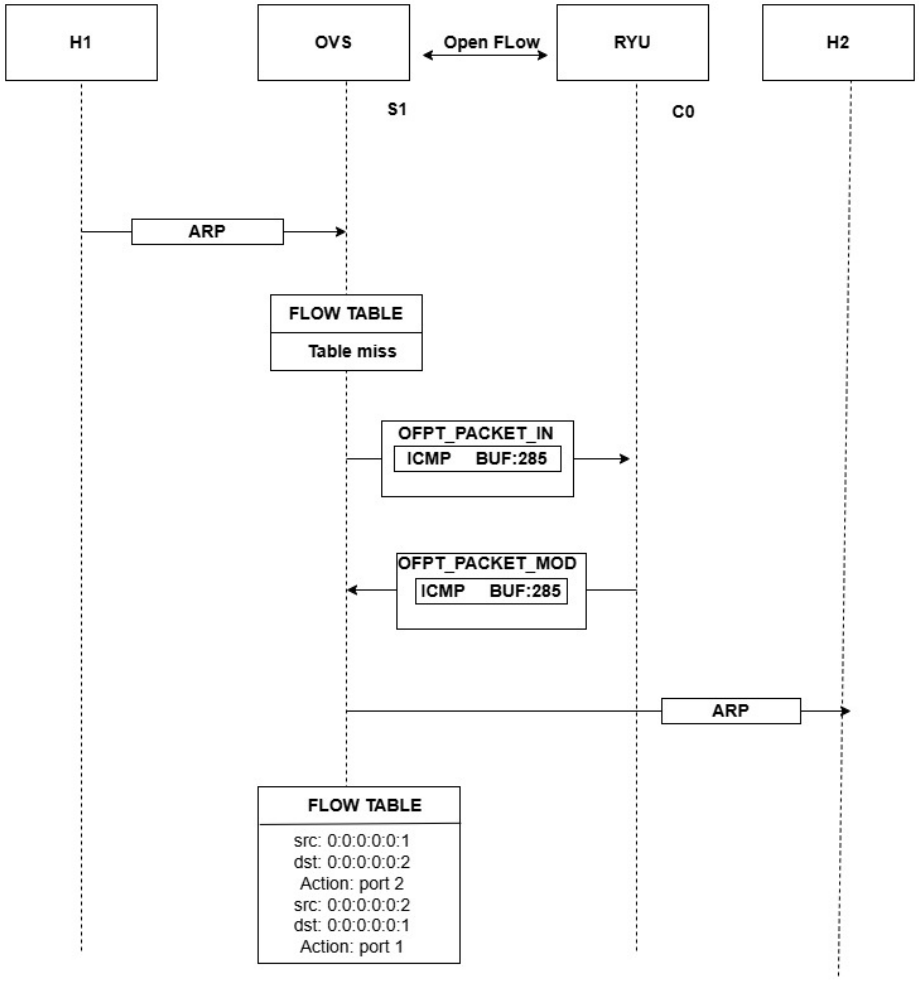


Figure 5.24: Normal Traffic

Consider different type of packet. This is generated by generating traffic from one host to another namely h1 to h2. The packet sent to the Ryu controller via the OFPT_PACKET_IN packet is Address Resolution Protocol (ARP) and therefore the switch must learn the source and destination instead of flooding as in the example for ICMP.

- **Initial Packet Handling**: When an ARP packet arrives at the OpenFlow switch without a matching flow entry, the switch sends an OFPT_PACKET_IN message to the Ryu controller, including details about the unmatched packet.

- **Controller's Response**: The Ryu controller processes the received ARP packet and responds by sending an OFPT_FLOW_MOD message. This message installs flow rules in the OpenFlow switch to handle similar packets in the future.
- **Installed Flow Rules**: The following specific flow rules are added to the switch:
  - Packets with a source MAC address of 00:00:00:00:00:02 and a destination MAC address of 00:00:00:00:00:01 are forwarded to port 1.
  - Packets with a source MAC address of 00:00:00:00:00:01 and a destination MAC address of 00:00:00:00:00:02 are forwarded to port 2.
- **Default Rule**: Packets that do not match these specific rules are handled by the default rule, which sends another OFPT_PACKET_IN message to the Ryu controller for further processing.
- **Automatic Forwarding**: Once the flow rules are installed, the OpenFlow switch automatically forwards matching packets without involving the controller, reducing latency and load.
- **Timeout and Rule Removal**: Each flow rule has an idle timeout. If no matching packet is received within this period, the flow rule is removed from the switch's flow table. When this happens, unmatched packets trigger a new OFPT_PACKET_IN message, and the process repeats.

In this way, packets are handled in SDN environment with the use of RYU controllers. The controller is responsible for installing new flow rules to accomodate the packets and network is sustained. Once the flow rules are set for new packets for same traffic. Thus, there is no need for the OpenFlow switch to consult the Ryu controller because it already has manual flows for these events. Hence there is no communication with the Ryu controller yet the traffic sustains.

5.5 Load Balancing System

Load balancing helps to create a balanced SDN network. This is done to balance the load among each of the multiple controllers present in the topology. Each controller is connected to their own switches and load respectively. In some scenarios, some part of

network can experience particularly more load than other network. Thus one controller might experience large amount of load traffic than other controllers at the same time. This creates unbalanced network and can hamper the performance and overall experience of end user. Load balancing is done to solve this problem. The general process is to alleviate the load by migrating some of the switches in the overloaded controller to its neighbouring controller temporarily. After the situation has been resolved the switch would be migrated back to its original position. This section outlines the components and processes involved in monitoring traffic, detecting load imbalances, and initiating load balancing actions.

5.5.1 Monitoring Network Traffic

The first step in load balancing is to continuously monitor the network traffic and collect statistics about active flows in each switch. This is achieved using OpenFlow statistics requests.The Ryu controller sends flow statistics requests to the connected switches using the OFPFlowStatsRequest. The flow statistics include data like the number of active flows. This indicates the load on the switches from the hosts.A monitoring thread is initialized in the controller that requests flow statistics from each connected switch at regular intervals (every 5-10 seconds)or less for more realtime monitoring. Thus the controller always has up-to-date information about the network's traffic distribution.

5.5.2 Flow Threshold and Load Balancing Trigger

Once the controller has collected the flow statistics, it compares the number of active flows with a predefined threshold. If a switch exceeds this threshold, the controller determines that load balancing is necessary to prevent congestion and ensure smooth traffic flow in the network. The flow threshold is the number of active flows a switch can handle before triggering load balancing actions. In this implementation, the threshold is set to 1000 active flows per switch.The threshold is experimental and can change as per the network load and other requirements accordingly. If the flow count exceeds the threshold, the controller logs the event and triggers the load balancing method, which initiates the load balancing actions which is to migrate the switches.

5.5.3 Inter-Controller Communication

In a multicontroller SDN setup, it is essential to ensure that all controllers are aware of the current state of the network and can coordinate their actions to handle traffic load. This is achieved using a message broker like Kafka.

Kafka Setup

The Kafka setup is crucial for inter-controller communication. The controller establishes a connection to the Kafka server and sends flow statistics to a specific topic. This allows other controllers to consume the data in real-time.

- **Producer Setup**: The controller acts as a Kafka producer, sending messages containing flow statistics to the Kafka topic `controller_states`. These statistics include details such as the datapath ID and flow count, allowing other components in the network to process and analyze the flow data.
- **Message Format**The Kafka producer sends flow statistics as messages that are structured in a *JSON-like* format.
  - Each message includes the following data:
    - **Controller ID**: The unique indentifier for the Controller type.
    - **Datapath ID**:The unique identifier for the OpenFlow switch.
    - **Flow Count**: The number of active flows in the switch.
    - **Timestamp**: The time at which the statistics were recorded.
- **Kafka Consumer**: Other controllers in the SDN network can act as Kafka consumers. These controllers subscribe to the `controller_states` topic, where they receive the flow statistics messages and messages for load balancing action for switch migrations.

5.5.4 Load Balancing Logic

Load balancing happens when active flows exceeds the threshold. All the controllers are periodically sending their flow statistics via Kafka. When load on one of the controllers

exceeds the threshold for active flows then it consumes the data from other controller via Kafka. The controller calculates the total load for each controller and decides which other controller would be the best fit to migrate its switch to based on the statistics. Once the decision has been made for switch migration, the information is passed to other controllers via Kafka to perform necessary changes to accommodate the incoming switch. The switch migration is then initiated and migrated. This migration is done temporarily which would be reverted back after the load has been normal. In this way load balancing is done to balance the load within the multi controller SDN network.

5.6 DDOS Attack Detection and Mitigation

Not all high traffic is legit, many of them can be result of a DDOS attack which is aimed to shut down the normal operation of controller. So, it needs to be detected and mitigated in order to secure the network. The module deals with monitoring, obtaining the traffic data from the network and utilize them to detect the DDOS attack.

5.6.1 Traffic Control Module

The module deals with the monitoring and obtaining the traffic data from the network. The Flow Collector initiates the process by contacting the controller to request traffic information. Upon receiving the request, the controller utilizes the OpenFlow protocol to gather data from the flow tables of all switches connected to it. Each switch connected to the controller receives a flow-stats request from the controller. This request asks the switch for flow statistics. Each switch responds to the flow-stats request by sending a flow-stats reply message to the controller. This message contains all flow entries from all flow tables within the switch.The controller collects these flow-stats reply messages from all switches and compiles the traffic data.Once the controller has compiled the traffic data from all switches, it responds to the Flow Collector with the requested information.

5.6.2 Detection Module

After receiving statistical information from the flow tables, the Attack Detection Module use the ML model trained for the detection of network anomalies. If the traffic flow is deemed normal, the packets continue to enter the system. However, if the traffic flow is predicted as dangerous, the information about that flow is forwarded to the Mitigation Module.

5.6.3 Mitigation Module

When a hazardous traffic flow is identified by the detection module,the mitigation module will request the controller to block the traffic flows from the corrupted switch and the port. Suppose the switch with switch-id-1 and port 1 is currently receiving this hazardous traffic flow. The Mitigation Module will send a request to the Controller to instruct switch-id-1 to remove the traffic flows coming from port 1. Simultaneously, the Mitigation Module sends continuous alerts to the administrator for system monitoring. After the attack flow concludes, after some time, the system will automatically re-establish the connection to port 1 to maintain system operation.

6 RESULT AND ANALYSIS

6.1 Performance Metrics

6.1.1 XGB Model Training Scores

The result of the training are shown below:

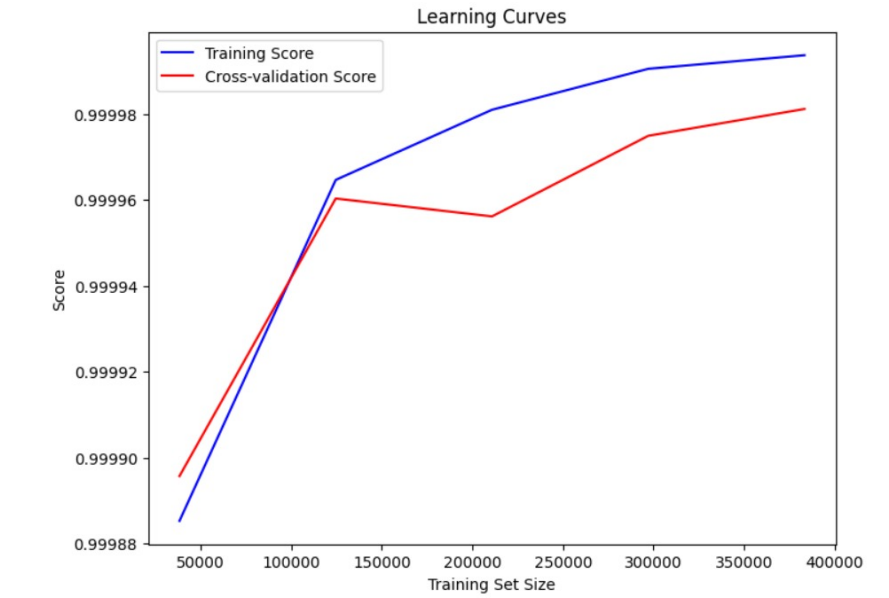


Figure 6.1: Learning Curve

The figure shows the learning curve for an XGB (Extreme Gradient Boosting) model, illustrating how the training score (blue line) and cross-validation score (red line) change with increasing training dataset size. The training score starts high and continues to improve as more data is added, approaching near-perfect accuracy, indicating that the model is effectively capturing patterns in the training data.The cross-validation score begins lower, reflecting the model's performance on unseen data. As the training set grows, the cross-validation score improves steadily and eventually stabilizes, suggesting the model's generalization capability increases with more data. The small gap between the training and cross-validation scores at larger dataset sizes indicates limited overfitting and strong generalization. This learning curve highlights that the XGB model learns efficiently and generalizes well as the dataset size increases. The stabilization of both scores at larger training sizes suggests that the model has reached its optimal performance, and adding more data may have minimal impact on further improvement.

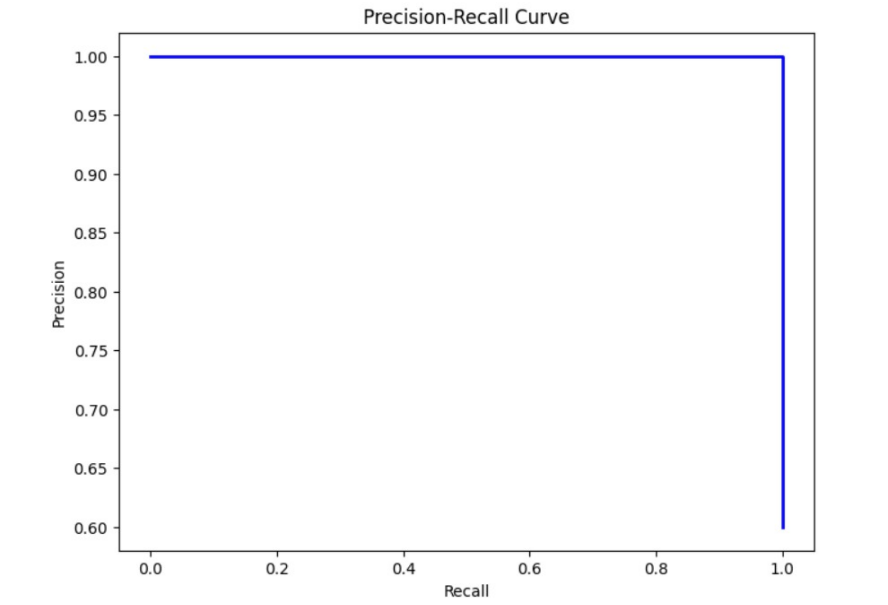


Figure 6.2: recall Scores v/s precision

The graph illustrates the relationship between recall and precision scores for a classification model. Recall is the proportion of actual positive cases that the model correctly identifies, while precision measures the proportion of predicted positive cases that are actually positive.In the graph, as precision increases, recall also increases, indicating a strong and positive correlation between these two metrics. This contrasts with the typical inverse relationship often observed between precision and recall. The sharp increase in recall when precision is lower suggests that the model becomes significantly better at identifying actual positive cases as its confidence in predictions improves.This pattern reflects the model's ability to strike a balance between minimizing false negatives and false positives. A high recall score signifies that the model is effective at capturing most of the true positive cases, while a high precision score demonstrates its ability to avoid false positives. Thus, the graph is a valuable tool for evaluating the model's overall performance.

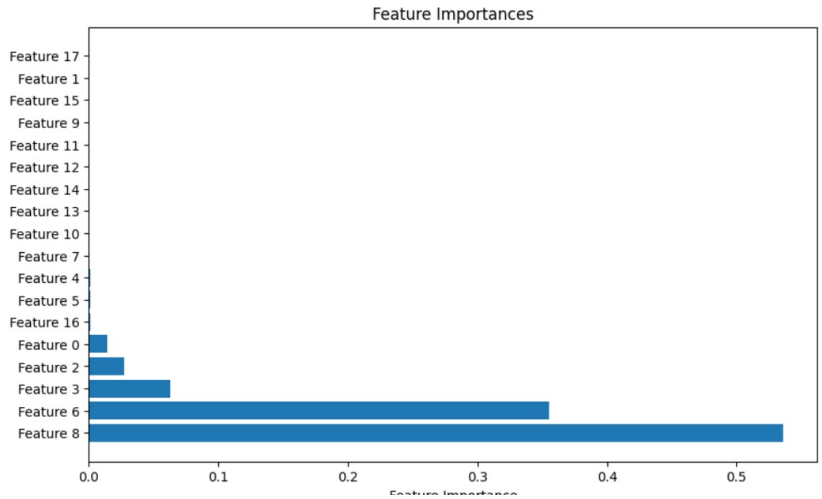


Figure 6.3: Feature Importance

The figure illustrates the Feature Importance for the machine learning model employed in the DDoS attack detection system, highlighting the relative significance of each feature in the model's predictive performance. Feature 8 emerges as the most influential, contributing over 50 percent to the model's decision-making process, emphasizing its critical role in distinguishing between legitimate and malicious traffic. Feature 6 and Feature 3 follow as significant contributors, further enhancing the model's ability to detect anomalies effectively. Other features, such as Feature 0 and Feature 2, also play a role, though their impact is comparatively smaller. Features like Feature 17 and others towards the bottom of the chart show minimal influence, suggesting they add little value to improving the model's predictions. This analysis underscores the importance of prioritizing key traffic attributes, as identified by the feature importance scores, to ensure the system's robustness in detecting and mitigating DDoS attacks efficiently. The results highlight the model's reliance on a few critical features for accurate anomaly detection, showcasing the efficiency and focus of the implemented approach.

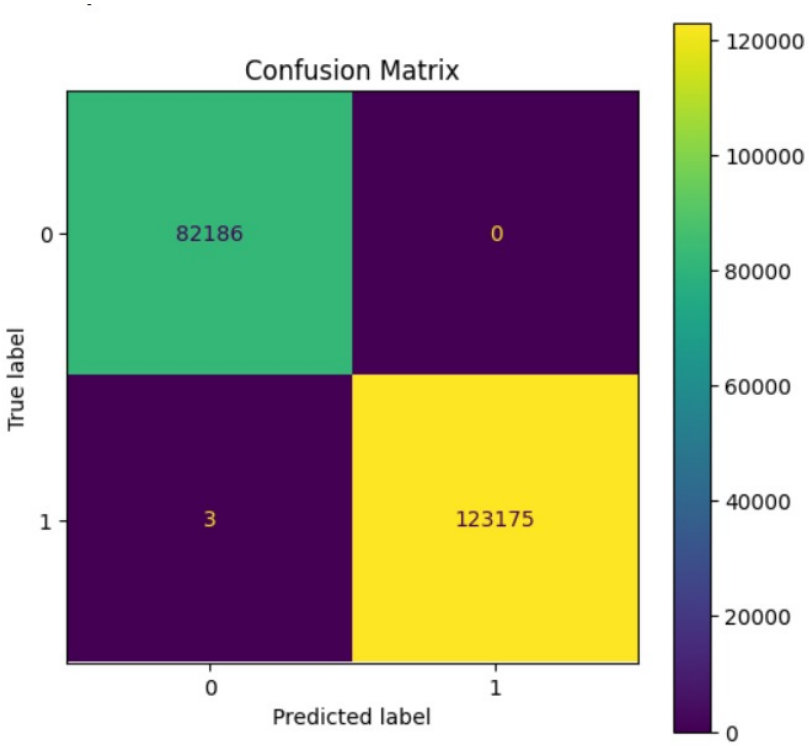


Figure 6.4: Confusion Matrix

The confusion matrix shows the performance of a classification model. It tells us how many times the model correctly predicted each class (DDOS and Benign) and how many times it made mistakes.

True Positives (TP): 123175 - The model correctly predicted 123175 instances as DDoS attacks (Class 1). True Negatives (TN): 82186 - The model correctly predicted 82186 instances as benign traffic (Class 0). False Positives (FP): 0 - The model incorrectly predicted 0 instances as DDoS attacks when they were actually benign traffic. False Negatives (FN): 3 - The model incorrectly predicted 3 instances as benign traffic when they were actually DDoS attacks.

6.2 Network Topology Implementation

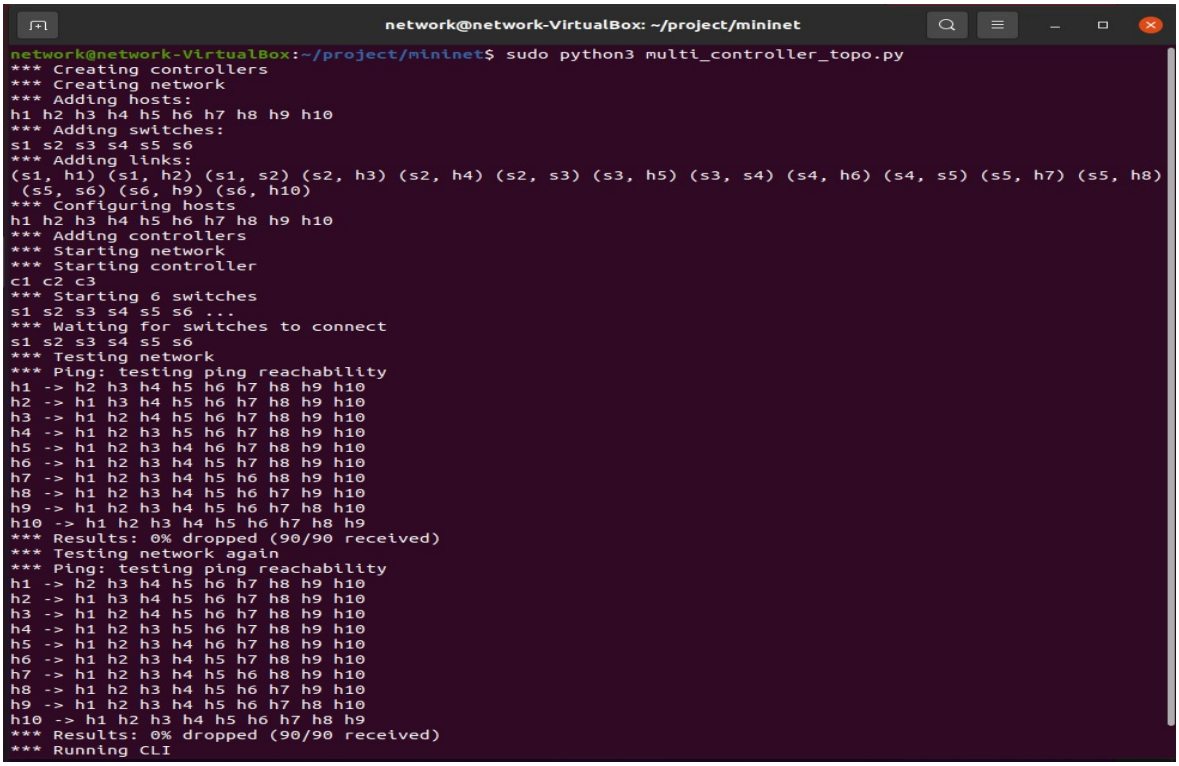


Figure 6.5: Network Topology in Mininet

Figure 6.6: Controller 1

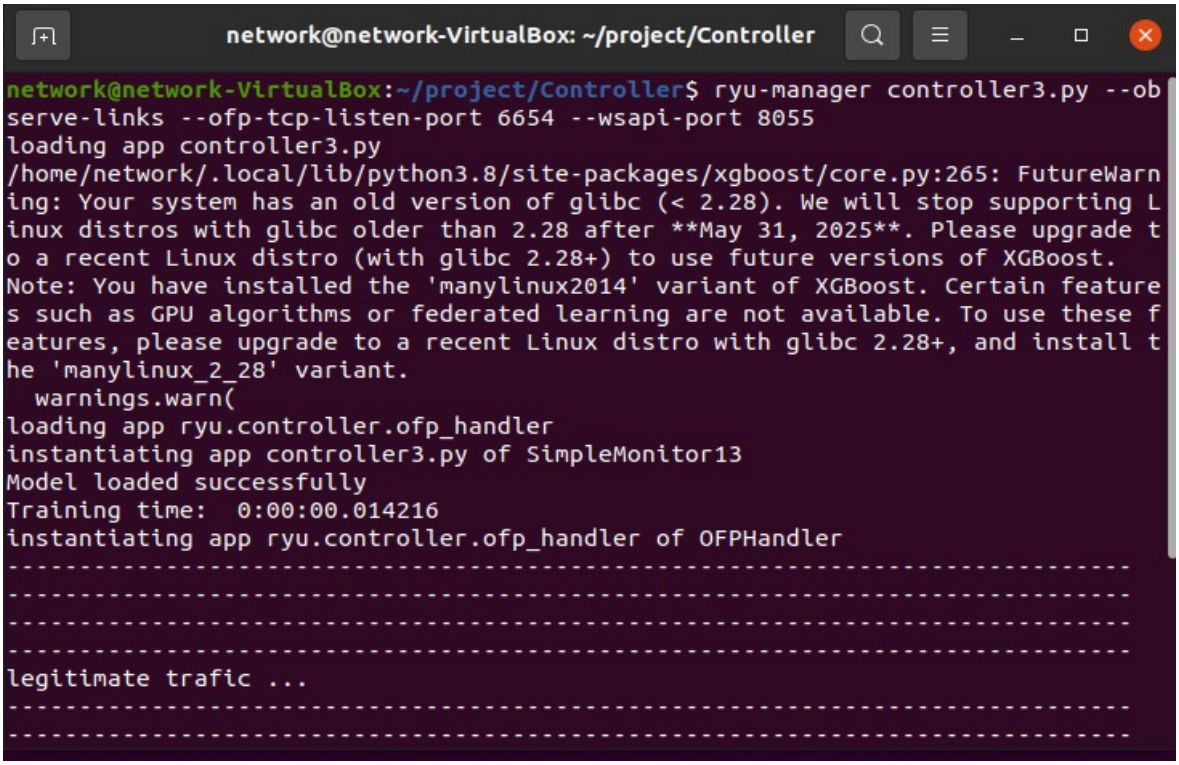


Figure 6.7: Controller 2

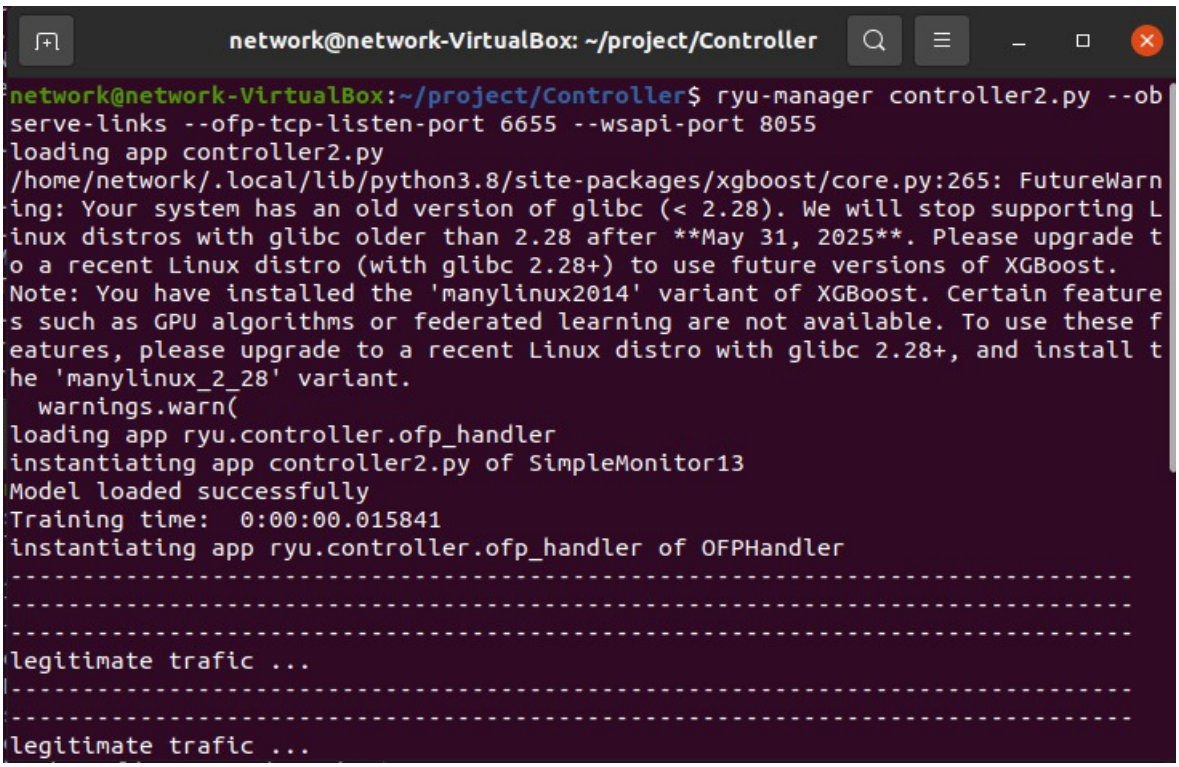


Figure 6.8: Controller 3

The terminal shows the output of a network simulation environment.The commands being run in the terminal suggest that the simulation involves 3-controllers, six switches (s1-s6) and fifteen hosts (h1-h10). The command 'pingall' is being used to test connectivity between all the hosts in the simulated network. The output of this command shows that all hosts are reachable, with no packet loss. The simulation environment appears to be functioning correctly.

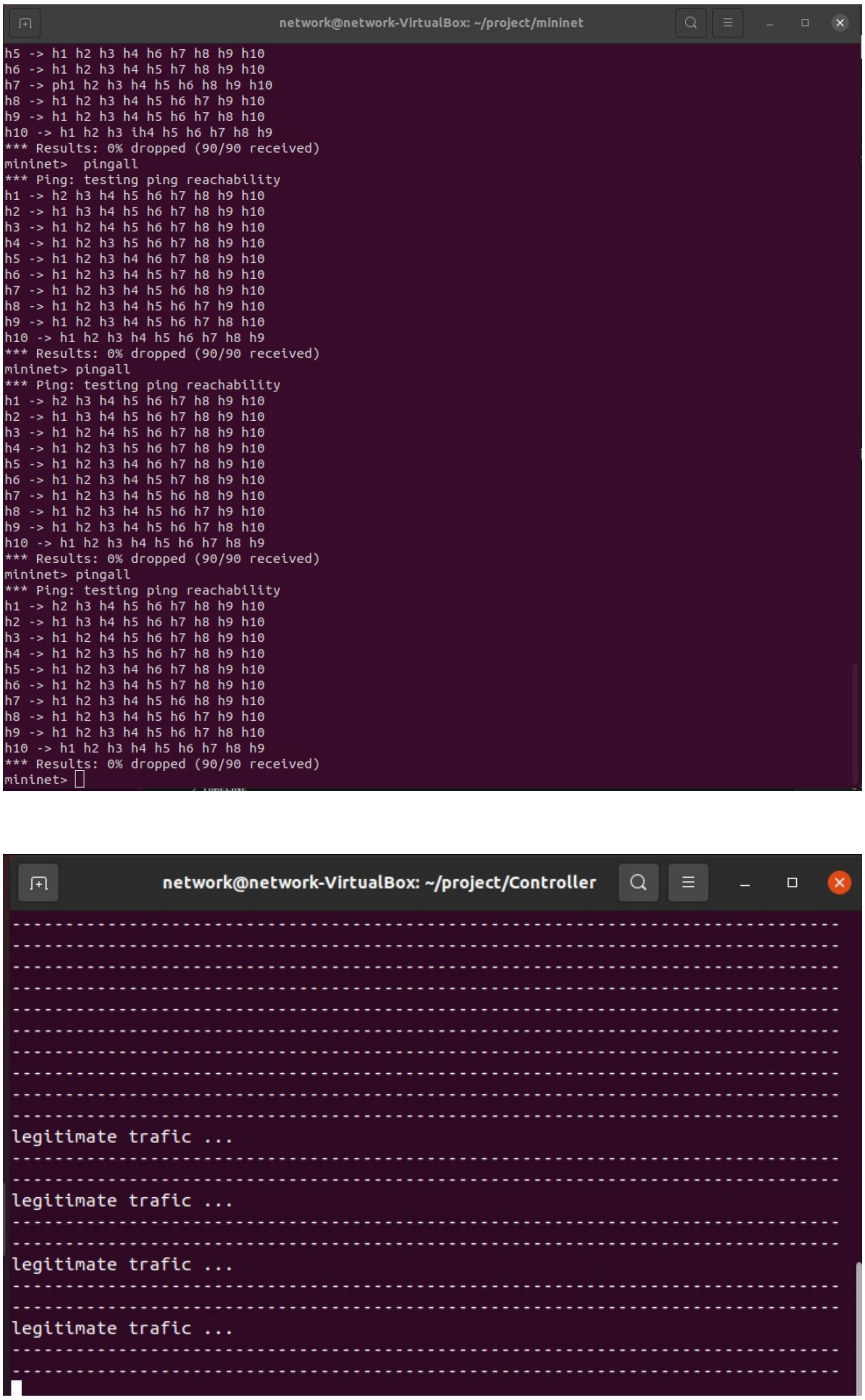


Figure 6.9: Crediting Traffic as Legitimate

The terminal shows the loading process of a program called ”ryu-manager” that handles network traffic.The terminal shows a network simulator called ”mininet” running a ping test between different nodes. The test checks the reachability between nodes in a network. All nodes are connected and there were no dropped packets.

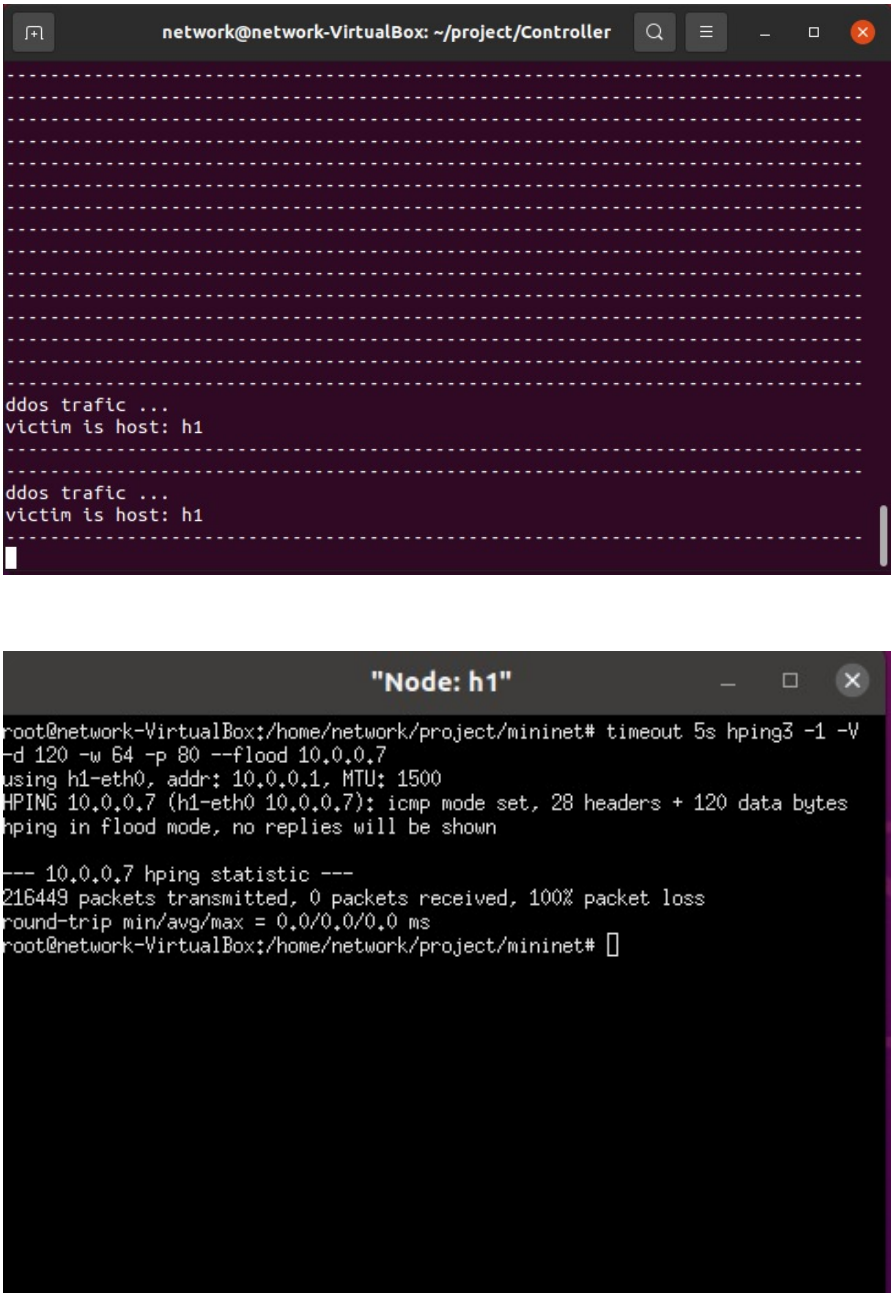


Figure 6.10: DDoS Traffic

The terminal shows that model correctly predicting DDos traffic generated from hping and alerting successfully.

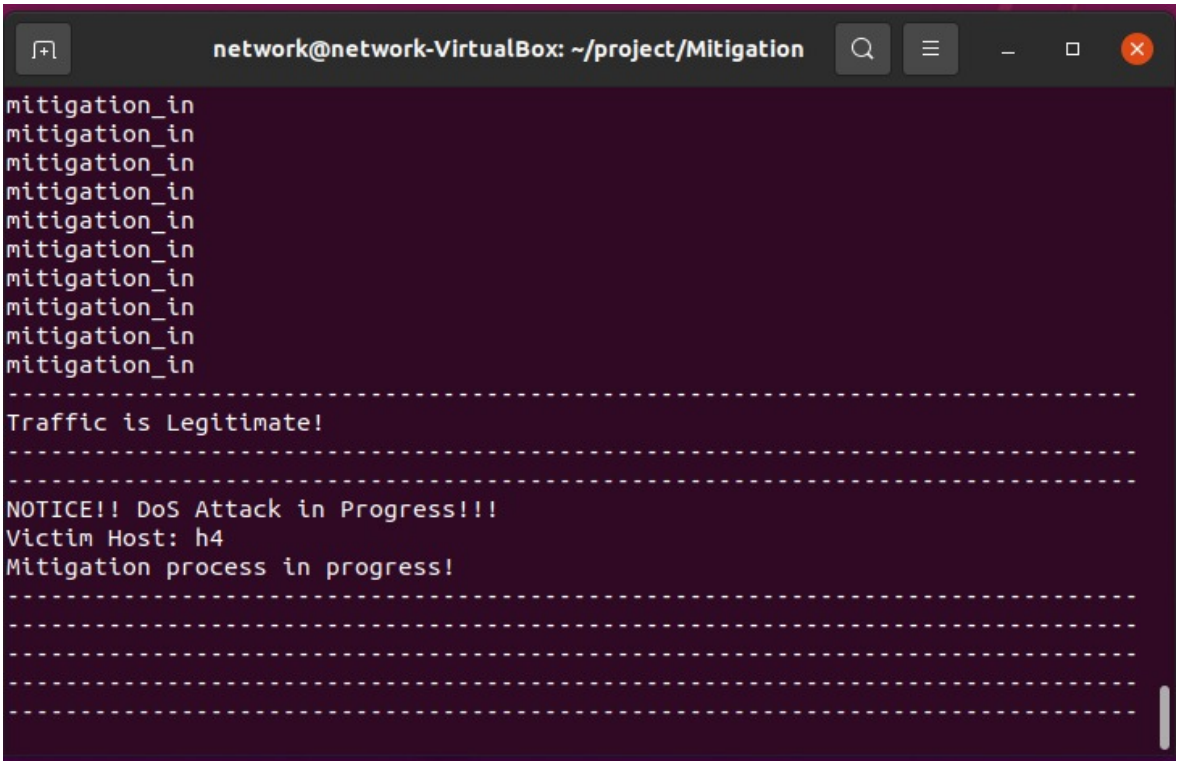


Figure 6.11: Mitigation

The terminal shows the working of mitigation module blocking the traffic from victim host.The module blocks the incoming traffic temporarily for blocking the DDOS traffic into the system

The terminal below shows the output of the terminal alerting and monitoring active flows for load balancing. The load balancing occurs when the active flows exceeds the thresholds which can be seen happening in the output of the terminal.The output also shows the alert for performing load balancing.

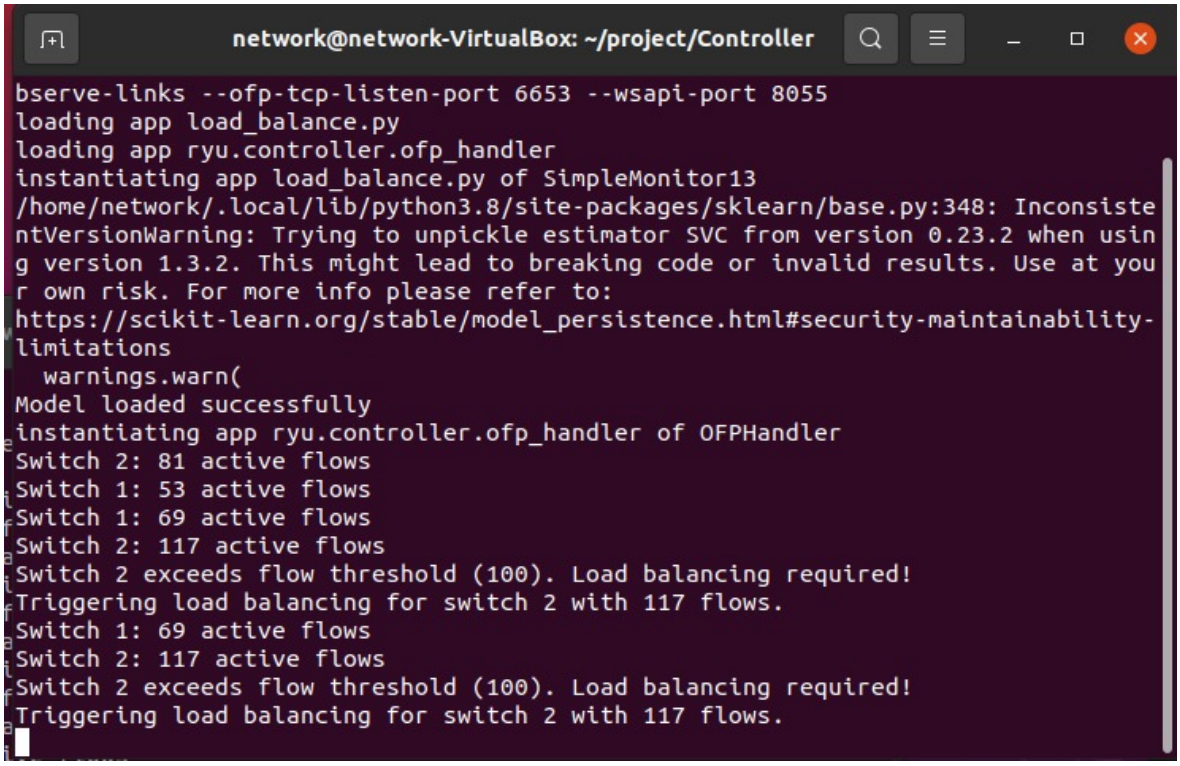


Figure 6.12: Load Balancing

7 REMAINING TASK

With the completion of data collecting, pre-processing, visualization, modeling, and network topology implementation, the project has been moving forward steadily.
For timely completion of the project,these are the remaning tasks:

- Model training of LGBM and Comparision.
- Integrating Load Balancing sub system into main system

8 DISCUSSION AND CONCLUSION

This project proposes a DDoS attack detection and mitigation system SDN, utilizing machine learning algorithms to identify and counteract DDoS threats. The system involves continuous monitoring and analysis of network traffic, detecting deviations from typical traffic patterns, and providing real-time alerts and detailed logs. The implementation includes data collection, preprocessing, network topology, traffic control module, detection module, and mitigation module. The network topology includes hosts, switches, and links, with the RYU controller managing the SDN-based. The traffic control module obtains traffic data from the network, while the detection module uses an ML model to predict potential threats. The mitigation module requests the controller to block hazardous traffic flows and sends alerts to the administrator for system monitoring. The project's modular design allows for easy integration and testing of each component, providing a robust defense mechanism against DDoS threats within SDN architectures.

APPENDIX A

A.1 Project Schedule

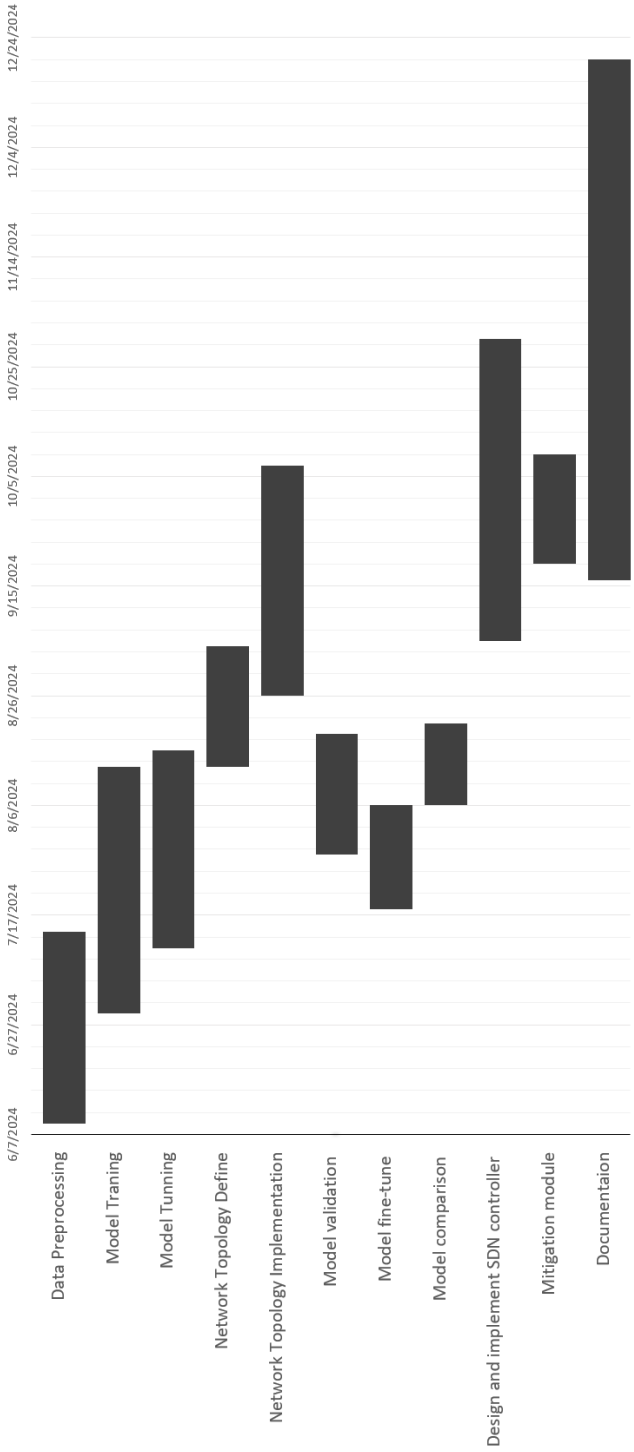


Figure A.1: Gantt Chart

REFERENCES

[1] A. Banitalebi Dehkordi, M. R. Soltanaghaei, and F. Z. Boroujeni. The ddos attacks detection through machine learning and statistical methods in sdn. *J. Supercomput.*, 77(3):2383–2415, 2020.

[2] D. Erickson. The beacon openflow controller. In *HotSDN 2013 - Proc. 2013 ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw.*, pages 13–18, 2013.

[3] A. M. Fatih, G. Cengiz, and K. Enis. Usage of machine learning algorithms for flow based anomaly detection system in software defined networks. In *Intelligent and Fuzzy Techniques: Smart and Ennovative Solutions*, pages 1156–1163, 2021.

[4] F. Hu, Q. Hao, and K. Bao. A lightweight ddos detection and mitigation system in software-defined networks. *Future Generation Computer Systems*, 79:205–214, 2018.

[5] S. Jevtic, H. Lotfalizadeh, and D. S. Kim. Toward network-based ddos detection in software-defined networks. In *12th International Conference on Ubiquitous Information Management and Communication*, 2018.

[6] M. N. Lima, A. M. Santos, and A. S. de Oliveira. Distributed approach for detecting ddos in software-defined networks using machine learning techniques. *Journal of Network and Computer Applications*, 167:102748, 2020.

[7] S. Sharma and R. K. Saini. A multi-level ddos detection and mitigation system for multi-controller software-defined networks. *IEEE Transactions on Network and Service Management*, 2021.