Abstract

In recent years, Software-Defined Networking (SDN) has emerged as a promising paradigm that decouples the control plane from the data plane, enabling centralized network management and enhanced programmability. However, this centralized nature also introduces new vulnerabilities, particularly to Distributed Denial of Service (DDoS) attacks that can overwhelm the control infrastructure.

This project focuses on the detection and mitigation of DDoS attacks—specifically TCP SYN flood attacks—in an SDN environment using a machine learning-based approach. A hybrid dataset was created by combining features from the CICDDoS2019 benchmark dataset and traffic generated in a simulated SDN network using Mininet and the Ryu controller. Legitimate and attack traffic were simulated using tools such as Iperf and Scapy.

After preprocessing, feature selection, and scaling, multiple classification models were trained, including Random Forest (RF), XGBoost, Convolutional Neural Networks (CNN), and Support Vector Machines (SVM). Evaluation using metrics such as accuracy, precision, recall, and F1-score revealed that the Random Forest classifier provided the best performance.

The trained RF model was then integrated into the Ryu controller for real-time detection. Bidirectional flow statistics were computed on the fly, and predicted malicious flows were promptly mitigated through flow rule modifications, such as dropping packets or blocking source IPs. This real-time detection and response mechanism effectively reduced the impact of DDoS attacks in the SDN environment.

Acknowledgement

We are most grateful to our Supervisor, Dr. Amitabha Nath, Assistant Professor in the Department of Information Technology. We are extremely grateful and indebted to him for sharing his expertise and providing sincere and valuable guidance and encouragement.

We are also indebted to the Head of the Department, Prof. Debdatta Kandar, Department of Information Technology, for permitting us to pursue this project.

We would like to take this opportunity to thank all the respected teachers of this department for being a perennial source of inspiration and showing the right path at times of necessity. We acknowledge that this project was completed entirely by our group and not by someone else.

Declaration

This is to certify that we have properly cited any material taken from other sources and have obtained permission for any copyrighted material included in this report. We take all responsibility for any report submitted as part of this project and the contents of this report.

Ibapalei Shadap - 21BTechIT19

Chadong Lowang - 21BTechIT34

Kundan Kumar Jha - 22BTechLIT02

Georgepaton Marbaniang - 22BTechLIT10

Contents

1 Introduction

1.1 Software Defined Networking (SDN)

In today's fast-evolving digital environment, traditional networking architectures often fall short in providing the flexibility, scalability, and central control needed to manage modern communication systems. Software Defined Networking (SDN) has emerged as a transformative networking paradigm that decouples the control plane from the data plane, creating a loosely coupled architecture that allows for centralized, programmable control of the entire network.

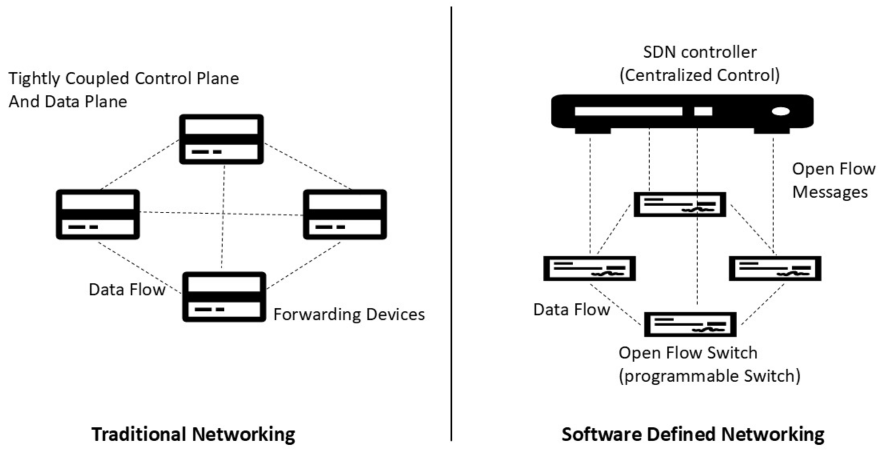


Figure 1.1.0: SDN vs Traditional Networking

The SDN controller, positioned as the network's "brain", manages and configures network behaviour through software applications, enabling dynamic response to network changes. This centralization simplifies network administration, enhances scalability, and supports automation—making SDN especially attractive for large-scale enterprise and next-generation communication. Unlike traditional networks, where configuration changes must be manually applied to individual, often vendor-specific devices, SDN enables policy upgrades and rule modifications from a central point, reducing complexity and operational overhead. In addition, SDN helps cut costs by shifting intelligence from expensive proprietary hardware to programmable software-defined systems.

1.2 Architecture of Software Defined Networking (SDN)

The architecture of SDN is centred around the separation of the control plane and data plane, which fundamentally transforms how networks are managed and operated. Traditional networks have both planes bundled inside the same physical devices, making them complex and vendor-dependent. SDN decouples them, leading to a more programmable and adaptable network. The SDN architecture typically consists of three key layers:

1. Data Plane (Infrastructure Layer)
2. Control Plane (SDN Controller Layer)
3. Application Plane (Network Applications)

1. **Data Plane (Infrastructure Layer)**
   This is the lowest layer, composed of network forwarding devices such as switches and routers (e.g., OpenFlow-enabled switches). These devices handle actual packet forwarding based on instructions from the control plane. They are simplified to perform only forwarding functions, making them faster and easier to manage.
   - Devices: Switches, routers, hardware or virtual forwarding devices.
   - Function: Packet forwarding, flow table lookup.
2. **Control Plane (SDN Controller Layer)**
   The control plane is the ”brain” of the network and plays a pivotal role in network intelligence and decision-making. It communicates with the data plane via protocols like OpenFlow, collecting network information and dictating how switches should handle traffic. The SDN controller maintains a global view of the network.
   - Devices: SDN Controller.
   - Function: Flow control, path computation, traffic engineering.
3. **Application Plane (Network Applications)**
   This is the topmost layer, where network applications reside. These applications interact with the controller via northbound APIs to request network resources, apply policies, or respond to changes in the network. Typical applications include firewalls, load balancers, DDoS detectors, intrusion detection systems, and more.

Communication Interfaces

- **Southbound API** Enables communication between the controller and the data plane. Used to install flow rules and retrieve statistics.
- **Northbound API** Enables communication between applications and the controller for policy enforcement, monitoring, and orchestration.

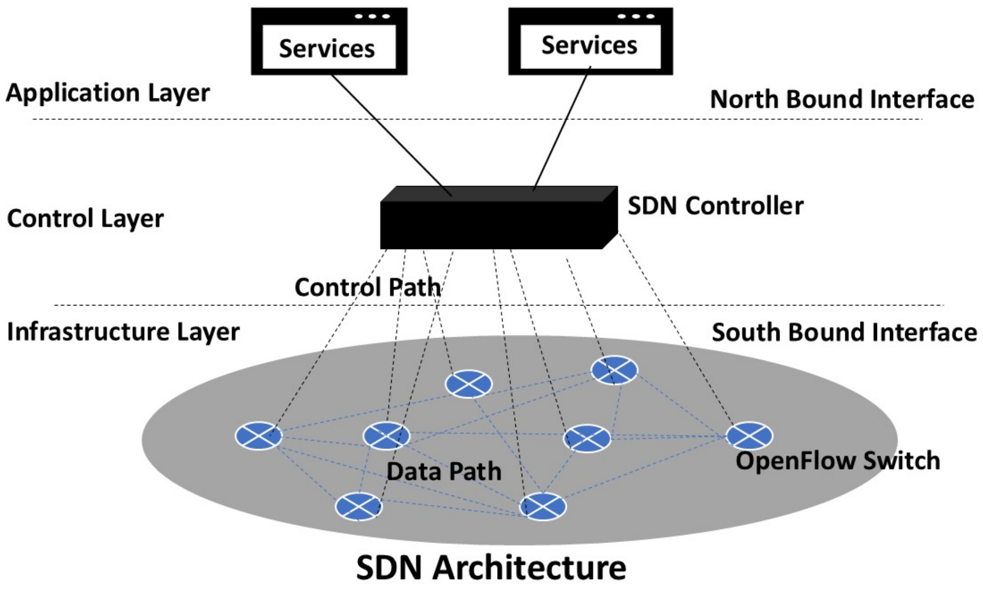


Figure 1.2.0: SDN Architecture

1.3 Security Challenges in SDN

Despite its benefits, SDN is not without its vulnerabilities. Its centralized architecture introduces a critical dependency on the controller, which can become a single point of failure if not properly secured. Common security issues in SDN include:

- Flow table saturation
- Bandwidth exhaustion
- And most critically: Distributed Denial of Service (DDoS) attacks

Understanding these threats is crucial, as SDN's flexible structure can also be exploited by attackers— particularly in the case of DDoS attacks, which can disrupt the entire network by targeting the controller or overloading flow table entries at switches.

1.4 Understanding DDoS Attacks in SDN

A Distributed Denial of Service (DDoS) attack aims to render network services unavailable by flooding the target with a massive volume of traffic, overwhelming system resources and preventing legitimate users from accessing services. In the context of SDN, this threat is even more dangerous.
A typical DDoS attack involves two main steps:

1. **Compromise Phase:** Attackers hijack large numbers of unsecured devices—often IoT devices or misconfigured servers—turning them into bots or zombies.
2. **Attack Phase:** These compromised nodes are instructed to generate large volumes of malicious traffic, targe ting a specific system or network service (e.g., the SDN controller).

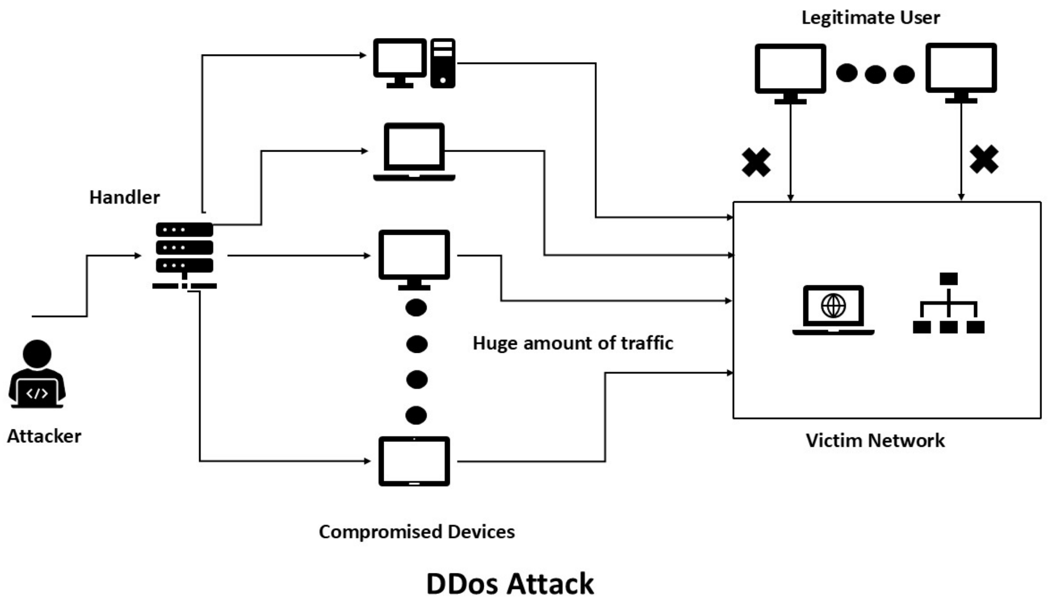


Figure 1.4.0: DDoS Attack

Among the most severe types of DDoS attacks are application-layer attacks like TCP SYN floods or UDP floods, which are difficult to detect as they often mimic legitimate traffic. In SDN, attackers can exploit how the controller processes new flows. Every unmatched flow triggers a packet_in message to the controller. A flood of such packets can overload the controller, block legitimate traffic, and even bring down the entire SDN environment.

1.5 Motivation for this Work

The transition to SDN and the increasing reliance on IoT have led to a wider attack surface and a greater demand for intelligent network security solutions. Traditional intrusion detection systems often struggle with application-layer DDoS attacks due to their low-and-slow nature. This motivates the use of Machine Learning (ML) techniques that can learn complex patterns in traffic data and detect subtle anomalies in real time.

The centralized architecture of SDN, combined with its programmability, provides an ideal foundation for building adaptive, ML-powered defence mechanisms. By analysing traffic patterns at the controller level, it's possible to detect and mitigate DDoS attacks dynamically, ensuring network resilience and uptime.

2 Project Objectives

This project aims to build a real-time DDoS detection and mitigation system within an SDN environment using machine learning techniques. The specific objectives are:

1. To design an SDN testbed using Mininet and Ryu controller.
2. To simulate Application-layer DDoS attacks (TCP SYN floods) within the SDN environment.
3. To perform feature selection using Pearson Correlation Coefficient and data preprocessing on the dataset.
4. To develop and evaluate various ML models including Random Forest (RF), XGBoost, SVM, and CNN using custom and public datasets.
5. To identify the best-performing model based on metrics like accuracy, precision, recall, and F1-score, and integrate it into the Ryu controller.
6. To implement real-time flow monitoring and feature extraction directly within the SDN controller.
7. To deploy mitigation strategies that dynamically block or limit malicious traffic in real-time.

3 Literature Review

Several studies have explored machine learning and deep learning techniques for detecting Distributed Denial of Service (DDoS) attacks in Software Defined Networks (SDNs). This section summarizes notable research contributions, highlighting the models used, datasets applied, and key outcomes.

1. **Mayadah A. Mohsin and Ali H. Hamad**
   Compared Random Forest (RF) and K-Nearest Neighbors (KNN) for DDoS detection in an SDN environment using Mininet and the Ryu controller. RF achieved superior results with an accuracy of nearly 99%.

2. **Sajid Mehmood, Rashid Amin, Jamal Mustafa, Mudassar Hussain, Faisal S. Alsubaiei, Muhammad D. Zakaria**
   Proposed a hybrid CNN-MLP model using SHAP-based feature selection and Bayesian optimization. Achieved 99.95% accuracy on the CICDDoS2019 dataset and 99.98% on the InSDN dataset.

3. **C. Srinivas, P.S. Avadhani, P. Prapoona Roja**
   Introduced an ensemble approach using RF, KNN, and XGBoost. Results showed RF with $\sim$99%, KNN $\sim$98%, and XGBoost $\sim$97% accuracy for SDN-based DDoS detection.

4. **Ashfaq Ahmad Najar and Manohar Naik S**
   Evaluated RF and MLP models on CICIDS2017 and CICDDoS2019 datasets. Random Forest showed strong performance, achieving nearly 99% accuracy in SDN environments.

5. **Venkatarri Marriboyina**
   Compared ML algorithms including RF, SVM, Decision Tree (DT), and KNN on the CICDDoS2019 dataset. RF (98.5%) outperformed SVM (97.2%), DT (95.8%), and KNN (94.3%).

6. **Wadee Alhalabi, Varsha Arya, Akshat Gaurav**
   Employed deep learning models—RNN, LSTM, GRU—for SDN traffic classification. GRU achieved 99.47% (multi-class), and RNN reached 99.99% (binary classification).

7. **Wang et al. (2020)**

Applied a deep CNN for real-time classification of DDoS traffic in SDNs. Achieved high detection accuracy ($\sim$99%) with minimal false positives.

8. **Sharma and Sahu (2021)**
   Designed a hybrid model combining XGBoost with feature selection on CICIDS2017. Reported high detection accuracy and reduced training complexity compared to traditional models.

3.0.1 Key Insights from Literature

Based on the above studies, it is evident that Random Forest (RF), Support Vector Machine (SVM), XGBoost, and Convolutional Neural Networks (CNN) are among the most widely adopted and effective algorithms for DDoS detection in Software-Defined Networks (SDNs). These models consistently demonstrate high accuracy and robustness across various datasets and attack scenarios. In particular, benchmark datasets such as CICDDoS-2019 have been instrumental in evaluating and comparing these algorithms due to their comprehensive coverage of modern DDoS attack types and real-world traffic patterns.

Therefore, our project adopts these algorithms—RF, SVM, XGBoost, and CNN—for building a reliable and efficient DDoS detection framework in an SDN environment, using the CICDDoS-2019 dataset to ensure realistic and diverse training and evaluation conditions.

4 Dataset

The dataset used for this project is the CICDDoS2019 dataset, provided by the Canadian Institute for Cybersecurity (CIC). This dataset contains labelled network traffic data collected from a realistic testbed, simulating various Distributed Denial of Service (DDoS) attack scenarios, along with normal traffic. It consists of 88 extracted features per network flow, representing packet statistics, durations, flow directions, and protocol-specific characteristics. The dataset offers a diverse range of attack types and is suitable for evaluating intrusion detection and classification models.
https://www.unb.ca/cic/datasets/ddos-2019.html

4.1 Feature Selection

The CICDDoS2019 dataset contains 88 features, including numeric flow statistics, protocol-specific flags, and categorical attributes. We applied a structured feature selection strategy which involved removing irrelevant, redundant, or correlated features, and retaining only those with high predictive value for DDoS detection.

4.1.1 Pearson Correlation Coefficient

To identify dependencies among numeric features, the Pearson Correlation Coefficient (PCC) was used. PCC quantifies the linear correlation between two continuous variables, returning a value between -1 and +1:

- Values close to +1 or -1 indicate strong correlation.
- Values close to 0 suggest independence.

This measure is critical for detecting multicollinearity, which can lead to overfitting and biased feature importance in machine learning models.

4.1.2 Removal of Non-Numeric and Categorical Features

Several features were non-numeric (e.g., strings or categorical identifiers) and not directly usable in machine learning models without encoding. These were removed based on the following reasons:

Table 4.1: **Features Removed Due to Categorical or Non-Generalizable Nature**

Feature Name	**Type**	**Reason for Removal**
Source IP	Categorical (string)	Identifies a device; causes overfitting and poor generalization
Destination IP	Categorical (string)	Same issue as above
Source Port	Nominal (numeric-like)	Arbitrary; no behavioral meaning
Destination Port	Nominal	Too specific to services; not generalizable
Protocol	Categorical	Already reflected in flow behavior metrics

These fields do not contribute meaningful statistical behaviour and risk biasing the model toward memorizing specific values.

4.1.3 Pearson Correlation Analysis

The correlation matrix was calculated using the pandas library, and a heatmap was visualized using seaborn for easier interpretation:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

correlation_matrix = df.corr()
plt.figure(figsize=(25, 25))
sns.heatmap(correlation_matrix, cmap='coolwarm', center=0, square=True)
plt.title('Pearson Correlation Heatmap - Numeric Features Only')
plt.show()
```

This correlation heatmap (Fig.2.1.3) highlights feature pairs with strong correlation. Clusters of red squares ($|r| \geq 0.9$) indicated groups of highly related features.

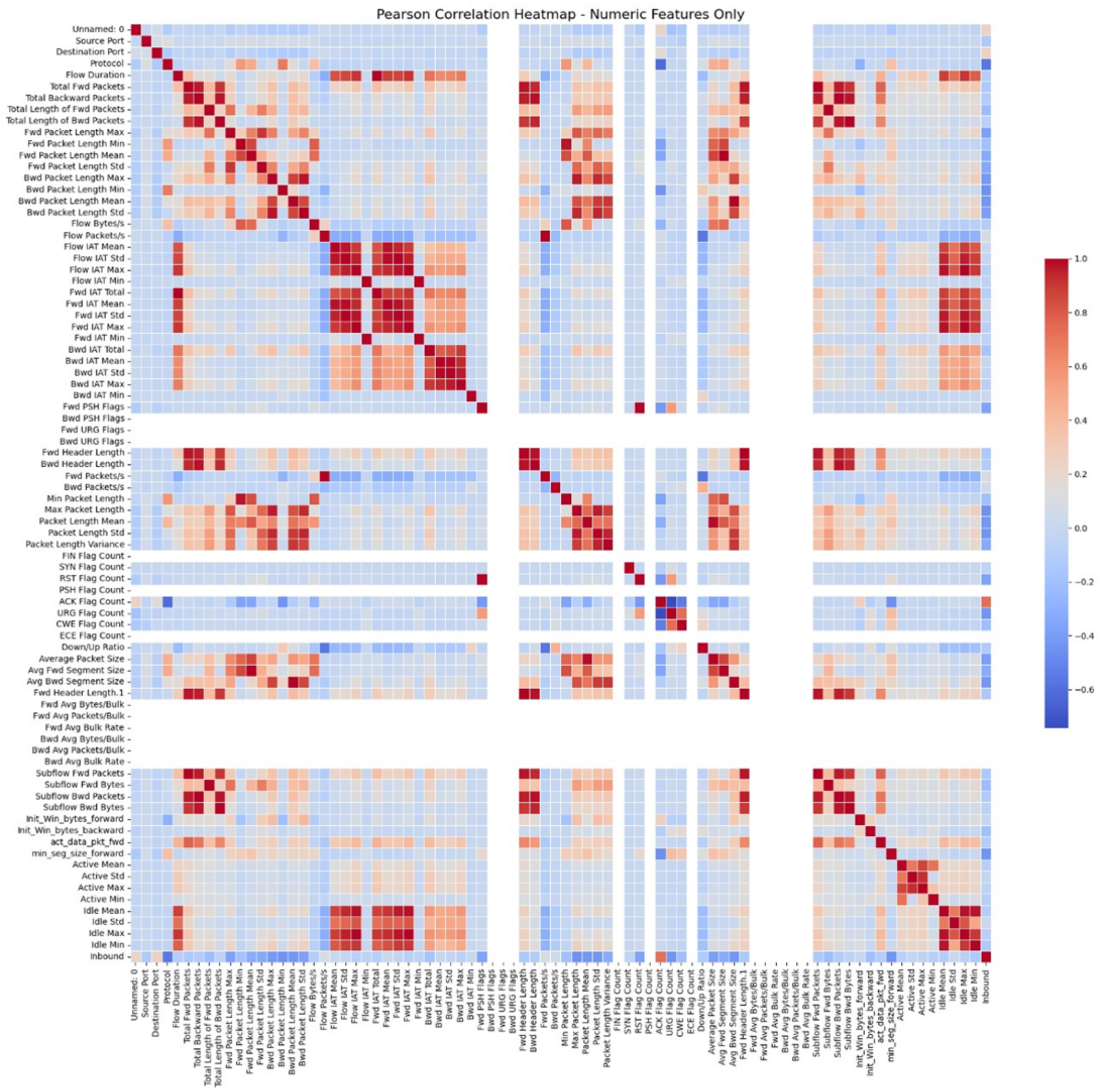


Figure 4.1.3: Pearson Correlation Heatmap

We manually inspected the correlation heatmap, where dark red or dark blue squares indicate features that are strongly correlated (close to +1 or -1). These show that two features are giving almost the same information.

To avoid redundancy, we selected only one feature from each highly correlated group — the one that was more general, stable, or meaningful — and removed the others. This helped reduce duplication, simplified the model, and improved performance.

Here's a summary of retained features and those removed due to high correlation:

Selected Feature	**Correlated Features Removed**	**Rationale**
Flow Duration	Fwd IAT Total, Bwd IAT Total, Flow IAT Mean, Active Mean, Idle Mean	Total session time; simpler and more stable
Total Fwd Packets	Subflow Fwd Packets, Fwd Packets/s, Fwd Header Length	Avoids duplication from subflows
Total Backward Packets	Subflow Bwd Packets, Bwd Packets/s, Bwd Header Length	Complements forward count; others are partial views
Fwd Packet Length Mean	Fwd Packet Length Max/Min/Std, Fwd Segment Size Avg, Fwd URG/PSH Flags	Mean is more general and stable
Bwd Packet Length Mean	Bwd Packet Length Max/Min/Std, Bwd Segment Size Avg, Bwd URG/PSH Flags	Mean is more general and stable
Flow Bytes/s	Total Length of Fwd/Bwd Packets, Fwd/Bwd Bulk Bytes, Avg Bulk Size	Combined byte volume with time produces a compact signal.
Flow Packets/s	Fwd/Bwd Packets/s, Subflow Packets/s	Flow-wide frequency is sufficient
Average Packet Size	Packet Length Max/Min/Mean	Single general size metric
Packet Length Std	Fwd/Bwd Packet Length Std, IAT Std	Measures size variability without directional redundancy

Table 4.2: Feature selection based on correlation and redundancy

4.1.4 Removal of Sparse, Redundant, and Low-Value Features

Several remaining features were excluded even though they were not highly correlated. These were dropped based on domain knowledge, statistical analysis, and interpretability.

1. **Protocol Flags and TCP Internals**
   Examples: ACK Flag Count, FIN Flag Count, URG Flag Count, Fwd/Bwd URG/PSH Flags
   Reason: They were mostly constant or zero even in attack traffic.

2. **TCP Window and Header Sizes**
   Examples: Init_Win_bytes_forward, Init_Win_bytes_backward, Fwd/Bwd Header

Length
Reason: Low variance, less informative about attack behavior

3. **Bulk Transfer Features**
   Examples: Fwd/Bwd Bulk Bytes, Avg Bytes/Bulk, Bulk Packets
   Reason: Mostly zero

4. **Subflow Features**
   Examples: Subflow Fwd Bytes, Subflow Bwd Bytes, Subflow Packets
   Reason: Duplicates of total flow metrics already included (Total Fwd/Back Packets)

5. **IAT and Activity Time Features**
   Examples: Flow IAT Mean/Max/Min, Fwd/Bwd IAT Mean/Std, Active/Idle Max/Min/Mean
   Reason: Often unstable, harder to interpret or generalize

4.1.5 Final Feature Set

After all filtering steps, only 9 numeric, independent, and relevant features were retained. These cover a broad range of behavioral signals while avoiding redundancy and noise:

Feature	**Behavior Captured**
Flow Duration	Total duration of the communication
Total Fwd Packets	Count of packets sent by source
Total Backward Packets	Count of response packets
Fwd Packet Length Mean	Average packet size in the forward direction
Bwd Packet Length Mean	Average packet size in the backward direction
Flow Bytes/s	Byte rate over the entire flow
Flow Packets/s	Packet rate over the entire flow
Average Packet Size	General measure of flow payload size
Packet Length Std	Variability of packet sizes — low variance may imply attack patterns

These features were also visualized in a final heatmap (Figure 4.1.5) to confirm the absence of strong internal correlation ($|r| < 0.6$), ensuring they provide unique and non-redundant information.

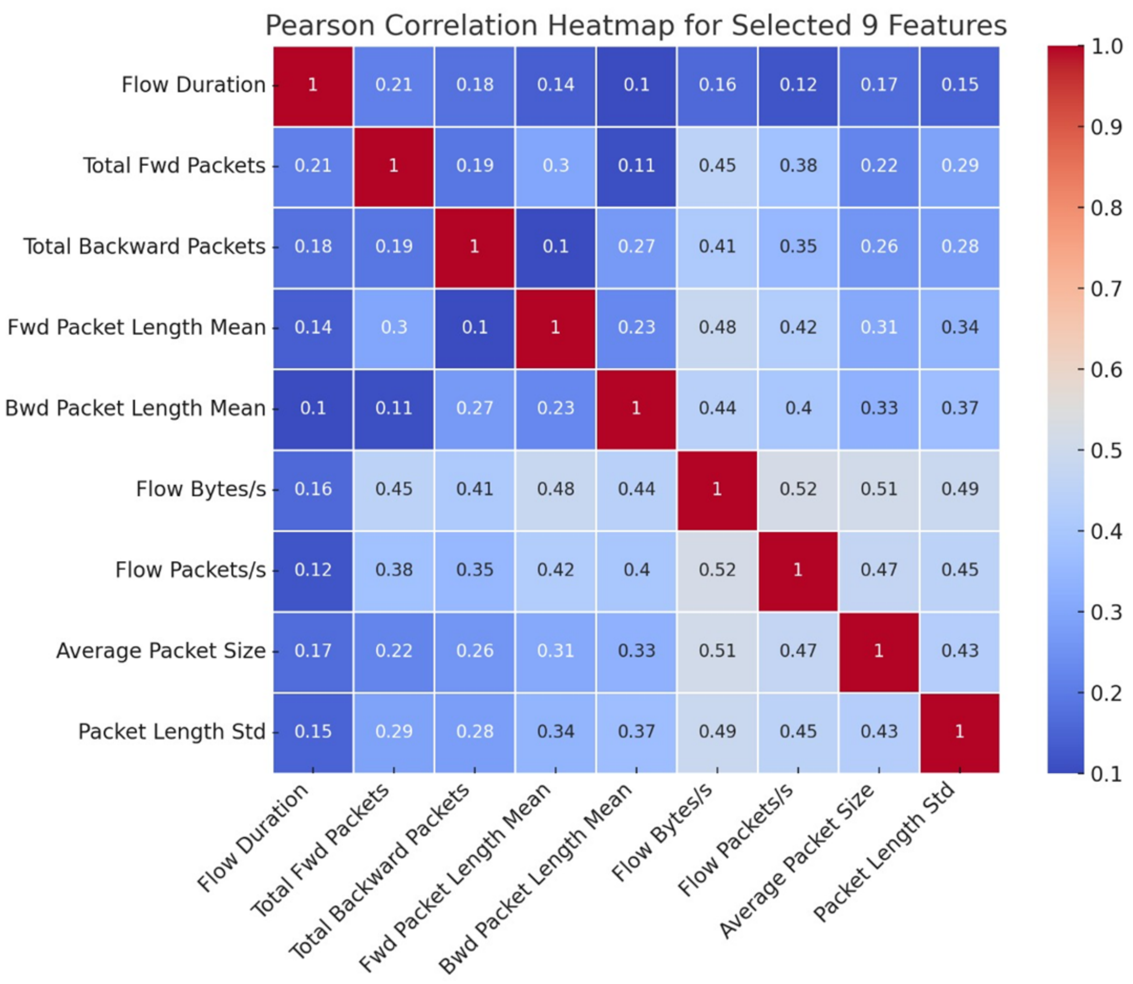


Figure 4.1.5: Pearson Correlation Heatmap for Selected 9 Features

This multi-step feature selection process reduced the original 88 features to a concise and effective subset of 9, improving model interpretability, generalization, and computational efficiency for DDoS attack detection.

5 Machine Learning Models

To detect Distributed Denial of Service (DDoS) attacks effectively, we experimented with both traditional machine learning models and deep learning techniques. The selected models cover a range of classification strategies, including tree-based ensembles, hyperplane-based classifiers, boosted learners, and neural networks, ensuring both diversity and robustness in detection performance.

5.1 Random Forest (RF)

Random Forest is an ensemble learning method based on decision trees. It constructs multiple trees during training and outputs the class that is the mode of the predictions of the individual trees.

- Performs well on imbalanced datasets and noisy features
- Naturally handles non-linear decision boundaries
- Provides feature importance, aiding interpretability
- Robust against overfitting due to the bagging (bootstrap aggregation) technique

Given the high dimensionality and variability of network traffic features, RF is ideal for general-purpose DDoS classification and acts as a strong baseline.

5.2 Support Vector Machine (SVM)

SVM is a margin-based classifier that constructs an optimal hyperplane to separate data into classes. It can use kernel functions (e.g., linear, RBF) to handle non-linear relationships.

- Effective in high-dimensional spaces
- Good at distinguishing fine-grained boundaries
- Works well when the number of features is larger than the number of samples
- Handles binary and multi-class classification reliably

SVM is suitable for scenarios where subtle variations in network flow behavior distinguish normal traffic from attacks. Its ability to generalize well on small feature sets (e.g., our 9 selected features) makes it particularly relevant.

5.3 Extreme Gradient Boosting (XGBoost)

XGBoost is a powerful boosted tree-based algorithm that builds trees sequentially, where each tree corrects the errors of the previous ones. It includes advanced regularization and optimization features.

- High accuracy due to boosting
- Efficient in handling imbalanced data and sparse features
- Built-in handling of missing values
- Fast training with optimized memory usage

XGBoost is known for its top-tier performance on structured/tabular data — like flow-level network features. It's well-suited for extracting patterns in DDoS behavior across large datasets.

5.4 Convolutional Neural Network (CNN)

CNNs are a type of deep learning model originally developed for image data, but adaptable to sequential or structured data. They apply convolutional filters to capture local patterns and hierarchical features.

- Can learn complex feature interactions without manual engineering
- Effective at capturing local dependencies in time-series or structured flow data
- Robust to input variations (noise, scale, order)
- Potential to outperform shallow models with enough data

By reshaping flow feature vectors into 2D grids or treating them as temporal signals, CNNs can learn discriminative representations of attack vs. normal behavior. They serve as a deep learning counterpart to traditional classifiers.

6 SDN Emulation

This section describes the emulated Software-Defined Networking (SDN) environment used to simulate and capture packets, specifically focusing on TCP SYN flood attacks. The setup includes tools and components such as Mininet, Ryu controller, Scapy, tcpdump, and Iperf, integrated into a Linux-based system for real-time traffic generation and monitoring. Here, we have used the default Ryu Controller script provided with the package, called simple_switch_13.py, which is an.

6.1 Environment Setup

The SDN environment was emulated on a Linux (Ubuntu 20.04) system with the following key tools:

- Mininet: An SDN network emulator used to simulate the network topology and run host-switch-controller environments.
- Ryu Controller: A default Ryu controller(simple_switch_13.py) was used for monitoring and feature extraction and later a Python-based custom SDN controller was used to monitor flow-level events and apply custom logic for packet processing, feature extraction, and mitigation actions.
- Scapy: A packet manipulation tool used to generate custom attack traffic such as TCP SYN floods with spoofed IPs.
- tcpdump: It is a powerful command-line packet analyzer used for capturing and inspecting network traffic in real time.
- Iperf: A network testing tool used to generate legitimate traffic for baseline comparisons and flow behavior analysis.

All components were installed and configured to work seamlessly in a virtual machine environment to facilitate isolated and repeatable experimentation.

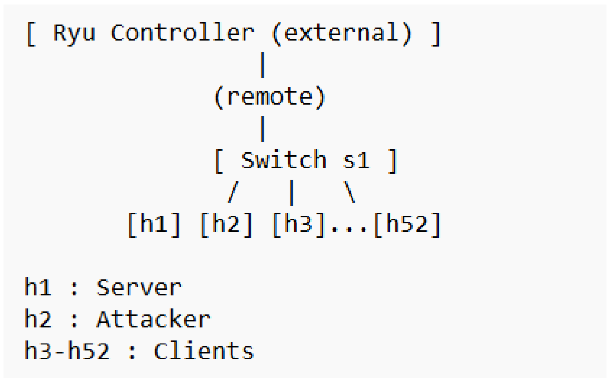
6.2 Network Topology

We designed a custom topology using Mininet to simulate real-world scenarios of both legitimate users and attackers. The topology included one switch connected to multiple hosts representing clients, a victim server, and an attacker.

- 1 switch (s1)
- 1 victim server (h1)
- 1 attacker (h2)
- 50 legitimate clients (h3 to h52)

All nodes were connected to the same switch to reflect a flat network structure typically seen in internal networks.

The switch is further connected to the Ryu controller which runs externally(outside mininet).



```
[ Ryu Controller (external) ]
                |
             (remote)
                |
             [ Switch s1 ]
              /   |   \
       [h1] [h2] [h3]...[h52]

h1 : Server
h2 : Attacker
h3-h52 : Clients
```

Figure 6.2.0: Topology

6.3 Traffic Simulation

To simulate realistic network conditions:

- Iperf was used to generate legitimate TCP traffic from clients (h3–h52) to the server (h1).

```
h3 iperf -c 10.0.0.1 -t 10 (sends packets for 10 seconds)
```

- Scapy was used on the attacker node (h2) to launch TCP SYN flood targeting h1, using spoofed IPs and high packet rates to emulate real-world attack behaviors.

```
from scapy.all import *
from random import randint

target_ip = "10.0.0.1"
target_mac = "00:00:00:00:00:01"
iface = "h2-eth0"

for _ in range(100):
    src_ip = f"10.0.0.{randint(100, 254)}"
    ether = Ether(dst=target_mac)
    ip = IP(src=src_ip, dst=target_ip)
    tcp = TCP(sport=RandShort(), dport=5001, flags="S")
    pkt = ether / ip / tcp
    sendp(pkt, iface=iface, verbose=0)
```

- Traffic was then captured using tcpdump.

```
h1 tcpdump -i h1-eth0 -w packets.pcap & (writes into packets.pcap)
```

6.4 Feature Extraction and Custom Dataset Creation

Captured packets from the network interface were saved in .pcap format. To prepare data for model training and real-time inference, the following steps were followed:

1. **Flow-Level Aggregation**
   Using a custom Python script, the .pcap files were processed to extract bidirectional flow-level statistics. Flows were identified using the 6-tuple (src_ip, dst_ip, src_port, dst_port, Ip version, protocol).

2. **Selected Features**
   Based on analysis and feature selection (from CICDDoS-2019 dataset), the following 9 features were extracted:

   - Flow Duration
   - Total Forward Packets
   - Total Backward Packets
   - Forward Packet Length Mean
   - Backward Packet Length Mean
   - Flow Bytes/s
   - Flow Packets/s
   - Average Packet Size
   - Packet Length Std

3. **Labelling**
   Flows were labeled as either benign or attack based on the source IP and behavior patterns manually. Legitimate traffic was labeled as 0, while SYN flood traffic was labeled as 1.

4. **Dataset Compilation**
   The newly generated custom dataset was combined with the preprocessed CICDDoS-2019 dataset to form a comprehensive training set. This allowed the model to learn from both benchmark and real-time simulated traffic.

7 Training and Testing

7.1 Data Preprocessing for Model Training

To ensure reliable and fair model training, the dataset underwent multiple preprocessing steps. These included cleaning, normalization, balancing, and dataset splitting, as outlined below.

7.1.1 Data Cleaning

Before model training, the dataset underwent several cleaning steps to ensure consistency, remove noise, and prepare it for numerical processing:

1. **Column Name Cleaning:**
   Leading and trailing whitespaces were removed from all column names to standardize feature access and avoid naming conflicts during processing.

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

2. **Replacing Infinite Values:**
   All occurrences of positive and negative infinity (`inf`, `-inf`) were converted to `NaN`. These values often result from undefined operations (e.g., division by zero) and can disrupt model training.

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

3. **Removing Missing Data:**
   Rows containing `NaN` values were dropped to ensure the dataset contained only complete and valid entries.

   ```
   df.dropna(inplace=True)
   ```

4. **Label Encoding:**
   Categorical labels in the target column (e.g., 0 for BENIGN, 1 for DDoS SYN) were encoded into numeric form using label encoding, enabling compatibility with machine learning algorithms.

```
label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])
```

7.1.2 Feature Normalization

To prevent scale bias among features with different value ranges, Min-Max Normalization was applied:

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

This technique scales all feature values to the range $[0, 1]$ without distorting their relative differences.

```
    from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
```

This normalization ensures that no single feature dominates the learning process due to its scale, which is especially important for algorithms sensitive to feature magnitudes.

7.1.3 Class Balancing via Undersampling

The dataset exhibited significant class imbalance, with a much higher number of attack flows (10,171,913 cases) compared to normal flows (30,662 cases). Such imbalance can bias the model toward predicting the majority class.

To address this, random undersampling was applied to the majority class (attack traffic) to match the size of the minority class (normal traffic). This ensured a balanced dataset, allowing the model to learn equally from both classes.

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X_scaled, y)
```

This step helped prevent overfitting to the dominant class and improved the model’s ability to detect minority-class instances (i.e., normal traffic).

7.1.4 Train-Test Split

The balanced and normalized dataset was then split into training and testing subsets using an 80-20 ratio:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X_resampled, y_resampled, test_size=0.2, random_state=42, stratify=y_resampled)
```

- 80% of the data was used for training
- 20% was reserved for evaluation
- Stratification ensured equal class distribution in both subsets

This split allowed for fair evaluation of model performance on unseen data while preserving the balanced class distribution achieved during preprocessing.

8 Results

8.1 Random Forest Classifier

The Random Forest (RF) algorithm was selected for its robustness, ability to handle high-dimensional data, and built-in feature importance evaluation. It is a popular ensemble method that constructs multiple decision trees and outputs the majority vote for classification, making it resistant to overfitting and noise.

8.1.1 Initial Training Observations

Initially, the model was trained using all available features, excluding only the Timestamp column (which was non-numeric and caused processing errors). This led to an accuracy of 100%, which was considered highly suspicious. This prompted further investigation, as perfect accuracy often indicates:

- Severe class imbalance, allowing the model to simply learn the dominant class
- Feature leakage or inclusion of low-quality/redundant features

8.1.2 Addressing Class Imbalance

To mitigate the above, Random Undersampling was applied (see Section 2.1.3), reducing the number of attack samples to match the number of benign ones. Despite this correction, the RF model still reported 100% accuracy, suggesting the need for better feature selection.

8.1.3 Feature Selection Improvement

A correlation-based feature selection approach was implemented (Section 1.2.4) using Pearson correlation heatmaps. From the original 88 features, only 9 statistically independent and semantically relevant features were retained:

- Flow Duration, Total Fwd Packets, Total Backward Packets
- Fwd Packet Length Mean, Bwd Packet Length Mean
- Flow Bytes/s, Flow Packets/s
- Average Packet Size, Packet Length Std

This selection reduced redundancy, eliminated low-variance or protocol-specific fields, and significantly improved model generalization.

8.1.4 Final Training Procedure

The model was retrained using the cleaned, normalized, undersampled, and feature-selected dataset. Hyperparameter tuning was performed using GridSearchCV with 3-fold cross-validation and `f1_macro` as the scoring metric:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf = RandomForestClassifier(random_state=42, class_weight='balanced')

param_grid = {
    'n_estimators': [200, 300],
    'max_depth': [None, 20],
    'min_samples_split': [2],
    'min_samples_leaf': [1],
}

grid = GridSearchCV(rf, param_grid, cv=3, n_jobs=-1, scoring='f1_macro', verbose=2)
grid.fit(X_train, y_train)

best_rf = grid.best_estimator_
```

8.1.5 Best Model Evaluation

Best Parameters:

- `n_estimators`: 300
- `max_depth`: None
- `min_samples_split`: 2
- `min_samples_leaf`: 1

Test Set Metrics:

- Accuracy: ∼99.36%
- Precision: ∼99.55%
- Recall: ∼99.69%
- F1 Score: ∼99.62%

8.1.6 Evaluation Results and Confusion Matrix

The confusion matrix below summarizes the final RF model's performance on the test set:

Class	**Predicted Correctly**	**Misclassified**
Benign (0)	5853 (True Negatives)	212 (False Positives)
DDoS (1)	6045 (True Positives)	19 (False Negatives)

Table 8.1: Confusion Matrix for Random Forest Classifier on Test Data

- The model achieves high classification accuracy while maintaining low error rates.
- The low false negative rate (FN = 19) indicates that DDoS attacks were reliably detected.
- • While some false positives (212) were observed, they remain acceptable given the priority to avoid undetected attacks.

This refined training pipeline, along with careful feature selection, resulted in a balanced, high-performing Random Forest model suitable for DDoS detection.

8.2 Support Vector Machine (SVM) Classifier

Support Vector Machines (SVM) are powerful binary classifiers that aim to find the optimal hyperplane that maximizes the margin between classes. SVMs are particularly effective in high-dimensional spaces and are robust to overfitting in low-noise environments.

8.2.1 Model Setup and Training

An SVM classifier was trained using the Radial Basis Function (RBF) kernel, which allows for non-linear decision boundaries and is well-suited for complex classification problems like DDoS detection.

```
from sklearn.svm import SVC

svm_model = SVC(kernel='rbf', probability=True, random_state=42)
svm_model.fit(X_train, y_train)
```

The same 9 selected features, normalized dataset, and undersampled class balance were used to ensure fair comparison.

8.2.2 Performance Evaluation

After training, predictions were generated for test sets:

Test Set Metrics:

- Accuracy: 85.32%
- Precision: 87.15%
- Recall: 85.32%
- F1 Score: 85.12%

8.2.3 Evaluation Results and Confusion Matrix

The confusion matrix below summarizes the SVM model's performance on the test set:

	Predicted: 0	**Predicted: 1**
Actual: 0	4445 (TN)	1567 (FP)
Actual: 1	213 (FN)	5904 (TP)

Table 8.2: Confusion Matrix for SVM on Test Data

- The classifier demonstrates high recall for the attack class (DDoS), correctly identifying 5904 out of 6117 attack flows.
- However, false positives (1567) are higher compared to RF, suggesting that SVM is more prone to misclassifying benign flows as attacks.
- Despite this, overall performance remains strong and generalizes well, as training and testing metrics are closely aligned.

8.3 XGBoost Classifier

Extreme Gradient Boosting (XGBoost) is a scalable and efficient ensemble method based on gradient-boosted decision trees. It is known for its high accuracy, built-in regularization, and ability to handle imbalanced data effectively.

8.3.1 Model Setup and Training

The XGBoost classifier was trained using the selected 9 features, the normalized and undersampled dataset (see Sections 1.2.4 and 2.1.3), and an evaluation set to monitor learning performance.

```
from xgboost import XGBClassifier

eval_set = [(X_train, y_train), (X_test, y_test)]
xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    n_estimators=100
)
xgb_model.fit(X_train, y_train, eval_set=eval_set, verbose=True)
```

8.3.2 Performance Evaluation

Test Set Metrics:

- Accuracy: 97.95%
- Precision: 99.05%
- Recall: 97.95%
- F1 Score: 97.94%

The model performs consistently well, indicating strong generalization and low bias.

8.3.3 Evaluation Results and Confusion Matrix

The confusion matrix below summarizes the XGBoost model's performance on the test set:

	Predicted: 0	**Predicted: 1**
Actual: 0	5955 (TN)	57 (FP)
Actual: 1	192 (FN)	5925 (TP)

Table 8.3: Confusion Matrix for XGBoost on Test Data

- The model achieves very high precision (99%), meaning very few benign flows are misclassified as attacks.
- False negatives (192) are higher than in RF but lower than SVM, showing a strong balance between detecting attacks and minimizing false alarms.
- • Overall, XGBoost provides a good trade-off between accuracy and interpretability, while also being efficient and scalable.

8.4 Convolutional Neural Network (CNN)

Convolutional Neural Networks, while traditionally used in image processing, can also be adapted for structured data classification. When used on reshaped tabular data, 1D CNNs can capture local feature patterns and relationships effectively, especially when sequences of features carry meaning.

8.4.1 Model Setup and Training

The CNN was trained using the selected 9 features, the normalized and undersampled dataset (see Sections 1.2.4 and 2.1.3)The dataset was reshaped to a 3D input format ((samples, features, 1)) to suit the requirements of Conv1D layers. The network architecture includes two convolutional layers followed by dense layers with dropout to reduce overfitting.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

input_shape = (9, 1)
model = Sequential([
    Conv1D(filters=32, kernel_size=2, activation='relu', input_shape=input_shape),
    MaxPooling1D(pool_size=1),
    Conv1D(filters=64, kernel_size=2, activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(y)), activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(X_train, y_train,epochs=100,batch_size=256, validation_split=0.2)
```

8.4.2 Performance Evaluation

Test Set Metrics:

- Accuracy: ∼97.24%
- Precision: ∼97.52%
- Recall: ∼97.03%
- F1 Score: ∼97.23%

These metrics are derived from the confusion matrix, which confirms that CNN performs competitively with tree-based models in this setting.

8.4.3 Evaluation Results and Confusion Matrix

The confusion matrix below summarizes the CNN model's performance on the test set:

	Predicted: 0	**Predicted: 1**
Actual: 0	5970 (TN)	144 (FP)
Actual: 1	182 (FN)	5782 (TP)

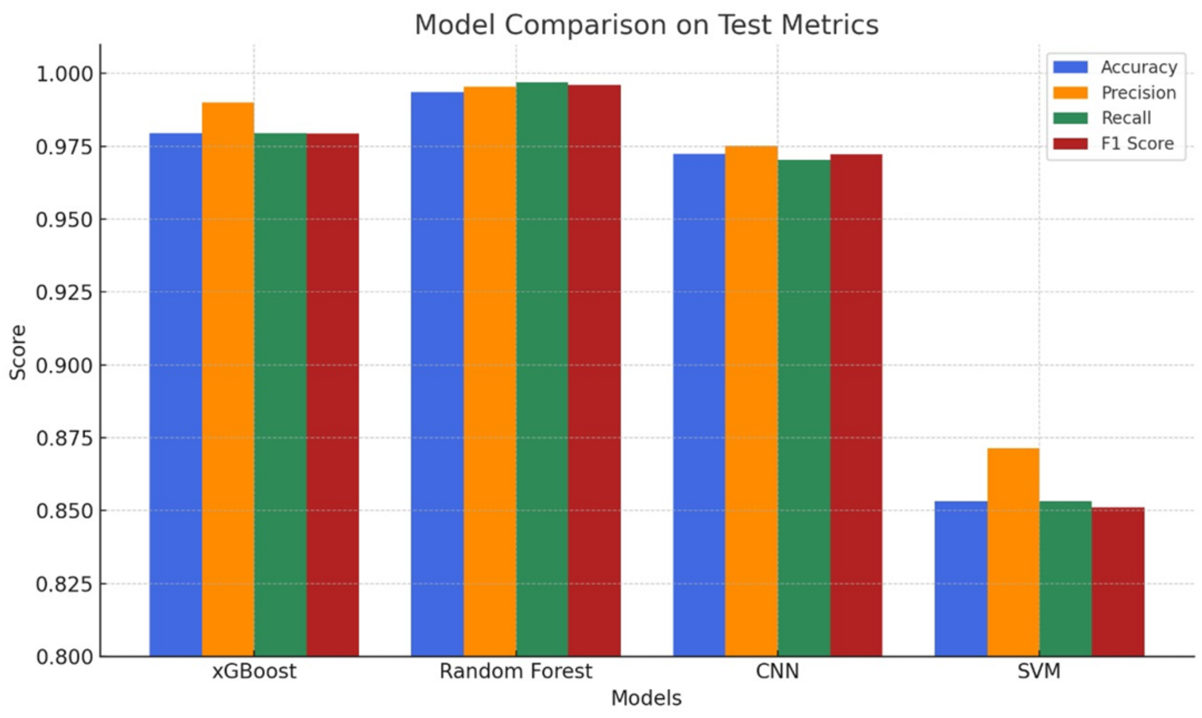
Table 8.4: Confusion Matrix for CNN on Test Data

- The model shows strong recall and precision, with moderate false positives and false negatives.
- Compared to SVM, CNN provides significantly better accuracy and generalization.
- Though slightly below RF and XGBoost in raw performance, CNN is beneficial in capturing complex non-linear relationships in data.

8.5 Model Comparison and Final Selection

After training and evaluating all four models — Random Forest, XGBoost, Convolutional Neural Network (CNN), and Support Vector Machine (SVM) — a comprehensive comparison was conducted based on the key evaluation metrics: Accuracy, Precision, Recall, and F1 Score.

To visualize and compare their test set performance, the following chart was created:



Model Comparison on Test Metrics

This bar graph illustrates the overall performance across all four metrics. The values were calculated using the respective confusion matrices and classification_report outputs from scikit-learn or TensorFlow/Keras.

8.5.1 Final Observations

- **Random Forest** achieved the highest scores across all metrics, with an F1-score of approximately 99.62%. It also maintained excellent balance between precision and recall, making it ideal for minimizing both false positives and false negatives.
- **XGBoost** followed closely behind, with strong performance and slightly lower recall and F1 compared to Random Forest.
- **CNN** also performed well, validating the feasibility of applying deep learning to tabular network traffic data. However, it did not outperform the ensemble-based models.
- **SVM** while conceptually simple and powerful in small-scale problems, underperformed in this case, likely due to the high dimensionality and complexity of the dataset. Its recall was particularly low, making it unsuitable for reliable DDoS detection.

8.5.2 Final Decision

Given its superior and consistent performance, Random Forest was selected as the final model for further analysis and deployment. Its advantages include:

- High accuracy and balanced performance
- Resistance to overfitting due to averaging across multiple decision trees
- Ease of interpretation and feature importance extraction
- Robustness to noise and small feature dependencies

Therefore, all subsequent analysis and visualizations will be based on the Random Forest classifier trained on the selected 9 features from the CICDDoS2019 dataset.

9 Real-Time Implementation

In this section, the integration of the trained Random Forest model into the custom Ryu controller for real-time DDoS attack detection and mitigation is presented. The goal was to implement a closed-loop detection and response mechanism within the SDN controller using flow-level statistics computed in real time.

9.1 Integration Architecture

The real-time system operates as follows:

1. **Traffic Simulation**: Normal and attack traffic are continuously generated using Iperf (for legitimate flows) and Scapy (for TCP SYN flood attacks).
2. **Packet Monitoring**: The Ryu controller listens for packet_in events triggered by unknown flows reaching the switch.
3. **Feature Extraction**: When a new flow is detected, relevant features are computed based on bidirectional traffic statistics.
4. **Model Prediction**: The pre-trained Random Forest model is loaded in the controller and used to classify each flow as either benign or malicious.
5. **Mitigation**: If a flow is predicted as an attack, the controller installs a flow rule to drop packets, preventing further damage.

This setup allows the controller to not only detect but also respond to attacks dynamically, ensuring continuous protection of the network.

9.2 Flow Tracking and Feature Collection

Each active traffic flow is identified using a combination of:

- IP version (IPv4 or IPv6)
- Source and destination IP addresses
- Source and destination ports
- Transport protocol (TCP or UDP)

For every such flow, the controller tracks the following nine statistical features in real time:

1. **Flow Duration** – Total time the flow has been active
2. **Total Forward Packets** – Number of packets sent from source to destination
3. **Total Backward Packets** – Number of packets sent from destination to source
4. **Forward Packet Length Mean** – Average size of forward-direction packets
5. **Backward Packet Length Mean** – Average size of backward-direction packets
6. **Flow Bytes per Second** – Total bytes divided by duration
7. **Flow Packets per Second** – Total packets divided by duration
8. **Average Packet Size** – Overall average size across all packets
9. **Packet Length Standard Deviation** – Statistical deviation in packet sizes

These features are computed using `NumPy` in the controller logic, providing a lightweight but informative representation of flow behavior.

9.3 Real-Time Traffic Classification

Once sufficient data is collected for a flow, its features are passed to a pre-trained Random Forest (RF) model:

```
rf_trained2_20250617_0928.joblib
```

The model outputs a probability score representing the likelihood of the flow being malicious. To enhance detection reliability and reduce false positives, a dynamic thresholding mechanism is implemented:

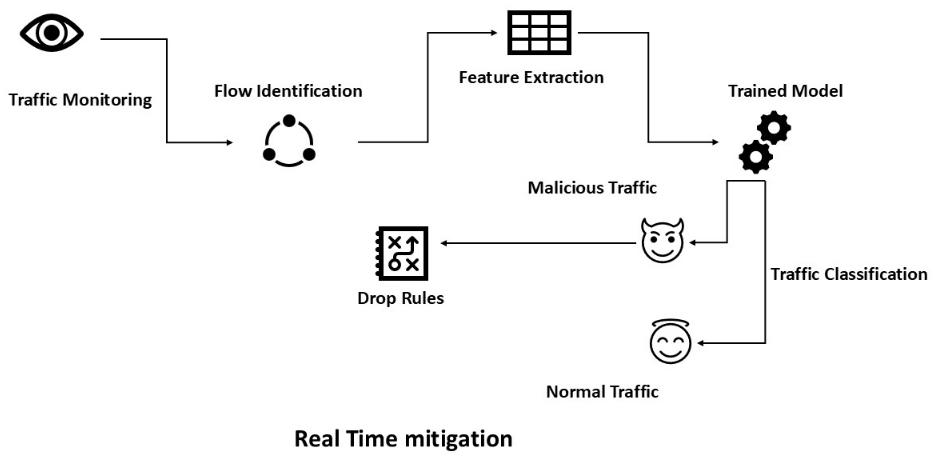
- If fewer than 20 flows have been analyzed, a default threshold of 0.5 is used.
- Once 20 or more flows are seen, the threshold is adjusted to the 75th percentile of recent malicious probability scores.

This adaptive thresholding allows the system to adjust to varying traffic conditions and better distinguish between benign traffic spikes and actual DDoS attacks.

9.4 Mitigation of Application-Layer DDoS Attacks in SDN

In this section, we describe the mitigation component of our SDN-based DDoS defence system. We implemented a real-time DDoS mitigation mechanism in a Software Defined Network (SDN) environment using the Ryu controller. The mitigation logic operates by inspecting incoming packets, extracting flow-level features, classifying them using a trained machine learning model (Random Forest), and deploying flow rules to block malicious traffic dynamically specifically focusing on SYN flood attacks.

The mitigation system works in conjunction with the previously trained Random Forest (RF) model, which classifies traffic flows as either benign or malicious based on statistical flow-level features.



Mitigation

9.4.1 Overview of Mitigation Logic

The mitigation process follows a pipeline that includes:

1. **Traffic Monitoring** – Capturing packets via Ryu's PacketIn events.
2. **Flow Identification** – Using IP headers and port information to identify unique traffic flows.
3. **Feature Extraction** – Generating a set of flow-level statistics for each active flow.

4. **Traffic Classification** – Feeding these features to a pre-trained Random Forest classifier to estimate the probability of a DDoS attack.
5. **Dynamic Flow Blocking** – Installing OpenFlow drop rules for malicious flows, effectively mitigating attack traffic at the switch level.

9.4.2 Mitigation via Flow Rule Installation

If a flow is classified as malicious (i.e., its probability exceeds the current threshold):

- A high-priority drop rule is installed on the switch using OpenFlow (`OFPFlowMod`) with no actions, immediately blocking future packets from that flow.
- The flow ID is added to the `blocked_flows` dictionary to avoid reprocessing and redundant classification.

This approach ensures that malicious traffic is dropped directly at the data plane level, reducing load on the controller and preserving bandwidth for legitimate users.

9.4.3 Flow Timeout and Cleanup

To maintain controller efficiency:

- Blocked flows are removed from memory after 300 seconds (5 minutes).
- Inactive flows are cleared from statistics tracking after 60 seconds.

This prevents memory overflow and stale state accumulation.

The mitigation mechanism efficiently leverages the centralized control and programmability of SDN to identify and block SYN flood-based DDoS attacks in real time. By extracting flow-level features and using a trained Random Forest model for classification, the system dynamically installs flow rules to prevent malicious traffic from impacting the network. This approach ensures quick response, reduces false positives through adaptive thresholding, and maintains overall network performance with timely flow cleanup and memory management.

9.5 Summary

This real-time setup demonstrates the practical application of a machine learning-based DDoS detection and mitigation system in SDN. By integrating the trained Random Forest model into the Ryu controller, the system is capable of making instant classification decisions. After classification, the flows being predicted as attacks are then dropped using drop rules.

10 Real-Time Implementation Results

In the real-time SDN environment using the trained Random Forest model, the system achieved high performance in detecting and mitigating DDoS attacks based on flow-level features. The results from live traffic classification are as follows:

	Predicted: Normal (0)	**Predicted: Attack (1)**
Actual: Normal (0)	2447 (TN)	31 (FP)
Actual: Attack (1)	82 (FN)	2440 (TP)

Table 10.1: Confusion Matrix of Real-Time DDoS Detection

- The low false positive rate (only 31 out of 4925 benign flows) ensures legitimate users are almost never mistakenly blocked.
- The high true positive rate confirms that most attacks are caught early, enabling effective mitigation in real-time.
- The 82 false negatives suggest room for improvement, possibly through online learning or model tuning, but the miss rate is low enough not to compromise overall network security.

These results demonstrate that the real-time integration of a machine learning model into the SDN controller can effectively and efficiently detect and mitigate DDoS attacks with high precision and low overhead. The system balances detection accuracy with reliability, making it suitable for deployment in production-grade networks.

11 Limitations

Although the model was trained and evaluated using the CICDDoS2019 dataset, which closely resembles real-world attack scenarios with up-to-date DDoS traffic patterns, the deployment and testing environment used for real-time detection — namely Mininet and the Ryu controller — remains a simulation.

While Mininet provides a practical platform for prototyping and emulating SDN networks, it does not fully capture the complexity, scale, and variability of actual production environments. Real-world SDN infrastructures typically involve:

- Physical switches and controllers distributed across geographic locations,
- More heterogeneous and noisy traffic patterns,
- Unpredictable latencies, hardware limitations, and failure scenarios.

This difference introduces a potential limitation: the excellent performance observed in the simulated testbed may not fully reflect how the system would perform under real-world constraints, such as:

- Traffic from hundreds or thousands of simultaneous users,
- Interactions with other services running on the network,
- The processing and memory limitations of physical SDN hardware.

Therefore, while the results from this project are promising, further validation on real or large-scale SDN deployments is necessary to fully assess the robustness and scalability of the proposed DDoS detection and mitigation system.

12 Conclusion and Future Work

This project successfully demonstrates a machine learning-based approach to detecting and mitigating DDoS attacks in a Software-Defined Networking (SDN) environment. By leveraging the centralized control and programmability offered by SDN, a real-time intrusion detection system was developed using a Random Forest (RF) classifier trained on a hybrid dataset consisting of both benchmark data (CICDDoS-2019) and custom traffic generated in Mininet. The environment was simulated using tools such as Mininet for network emulation, Ryu as the SDN controller, Scapy/Iperf for traffic generation, and tcpdump for capturing packets. Flow-based features were carefully selected and extracted to accurately differentiate between benign and malicious traffic. Among the models tested (RF, XGBoost, CNN, and SVM), the Random Forest model provided the highest performance in terms of accuracy, precision, recall, and F1 score.

Real-time implementation was achieved by integrating the trained model into the Ryu controller, enabling on-the-fly feature extraction, traffic classification, and dynamic mitigation strategies. This real-time, closed-loop system effectively detected and mitigated SYN flood attacks with minimal overhead, highlighting the viability of intelligent, adaptive security in SDN architectures.

Looking ahead, future work could focus on deploying the system in a real-world SDN environment to evaluate scalability, latency, and robustness under actual network conditions. Additionally, extending the model from binary to multi-class classification would enable finer-grained detection of various attack types (e.g., TCP, UDP, HTTP-based floods). Incorporating online or adaptive learning methods would also allow the system to evolve with changing traffic patterns and novel threats, eliminating the need for frequent offline retraining. These enhancements would further strengthen the practicality and resilience of machine learning-based DDoS defense mechanisms in dynamic network environments.

13 Bibliography

1. CIC-DDoS2019 Dataset.
   Canadian Institute for Cybersecurity - CICDDoS2019 Dataset for DDoS Attack Detection. https://www.unb.ca/cic/datasets/ddos-2019.html

2. Mayadah A. Mohsin, Ali H. Hamad.
   Performance Evaluation of SDN DDoS Attack Detection and Mitigation Based on Random Forest and K-Nearest Neighbors Machine Learning Algorithms.
   https://doi.org/10.1109/ICASET49439.2020.9141278IEEE Conference Paper, DOI: 10.1109/ICASET49439.2020.9141278

3. Sajid Mehmood, Rashid Amin, Jamal Mustafa, Mudassar Hussain, Faisal S. Alsubaiei, Muhammad D. Zakaria.
   Distributed Denial of Services (DDoS) Attack Detection in SDN Using Optimizer-Equipped CNN-MLP.
   https://doi.org/10.3390/app11115119MDPI Applied Sciences, DOI: 10.3390/app11115119

4. C. Srinivas, P.S. Avadhani, P. Prapoona Roja.
   Enhanced DDoS Detection Using Advanced Machine Learning Techniques in SDN.
   https://doi.org/10.1109/ICACDS53775.2022.9744602IEEE Conference Paper, DOI: 10.1109/ICACDS53775.2022.9744602

5. Ashfaq Ahmad Najar, Manohar Naik S.
   DDoS Attack Detection Using MLP and Random Forest.
   https://doi.org/10.1109/SMARTVISION53546.2021.9520544IEEE Conference Paper, DOI: 10.1109/SMARTVISION53546.2021.9520544

6. Venkatarri Marriboyina.
   DDoS Detection in SDN using Machine Learning Techniques.
   https://www.semanticscholar.org/paper/DDoS-Detection-in-SDN-using-Machine-Learning-Marriboyina/29e5375d804fc86bcdbf93229ff7e0536c0d5ea7

7. Xiang Wang, Jingyi Zhang, Jie Ma, Qiming Wang.
   DDoS Attack Detection Based on CNN in Software Defined Network.
   https://doi.org/10.1109/CCDC49329.2020.9098401

8. Amandeep Kaur, C. Rama Krishna, Nilesh Vishwasrao Patil *A comprehensive review on Software-Defined Networking (SDN) and DDoS attacks: Ecosystem, taxonomy, traffic engineering, challenges and research directions* https://www.sciencedirect.com/science/article/pii/S1574013724000753
9. Batchu R.K. and Seetha H
A generalized machine learning model for DDoS attacks detection using hybrid feature selection and hyperparameter tuning
https://www.sciencedirect.com/science/article/abs/pii/S1389128621004394?via
10. Yungaicela-Naula N.M., Vargas-Rosales C., and Perez-Diaz J.A.
SDN-based architecture for transport and application layer DDoS attack detection by using machine and deep learning.
https://ieeexplore.ieee.org/document/9502698/
11. Ravi N. and Shalinie S.M.
Learning-driven detection and mitigation of DDoS attack in IoT via SDN-cloud architecture.
https://ieeexplore.ieee.org/document/8993716/