Atal Bihari Vajpayee-Indian Institute of Information Technology and Management (ABV-IIITM), Gwalior, Madhya Pradesh, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Machine Learning Project On the Title of

DDoS Attack Detection In Software Defined Network

Submitted By-

ABHISHEK PATIDAR (2023ICS-01)
CHITRAKSH SINGH (2023ICS-06)

ABSTRACT

Software-Defined Networking (SDN) has transformed traditional network architectures by centralizing control and enabling dynamic management of network resources. However, the flexibility and programmability of SDN also introduce new vulnerabilities, particularly in the face of Distributed Denial of Service (DDoS) attacks. Detecting and mitigating these attacks in SDN environments require advanced techniques, including the application of machine learning algorithms.

In this study, we conduct an in-depth analysis of the "DDoS SDN Dataset," a comprehensive repository of network traffic data and attack metadata available on Kaggle. Our primary aim is to leverage machine learning techniques to enhance DDoS detection and mitigation capabilities within SDN infrastructures.

The dataset encompasses a wide range of attributes, including flow features, switch information, controller-switch communication logs, attack metadata, and network topology details. Through meticulous preprocessing and feature engineering, we explore the efficacy of various machine learning algorithms in classifying network traffic, identifying anomalous patterns, and predicting future attack trends.

Our experimental evaluation encompasses a diverse set of machine learning algorithm. We assess the performance of these algorithms using established evaluation metrics such as accuracy, precision, recall, and F1-score.

Our findings provide valuable insights into the effectiveness of different machine learning approaches for DDoS detection in SDN environments. We discuss the strengths and limitations of each algorithm, identify key factors influencing detection accuracy, and propose strategies for improving overall network security posture.

By shedding light on the complex interplay between DDoS attacks and SDN infrastructures, this study contributes to the advancement of network security research. Our results offer practical guidance for network administrators, security analysts, and researchers seeking to bolster the resilience of SDN networks against evolving cyber threats. Furthermore, our work lays the groundwork for future investigations into innovative defense mechanisms and proactive threat mitigation strategies in SDN environments.

CONTENTS

1. INTRODUCTION

1.1. INTRODUCTION

Software-Defined Networking (SDN) has emerged as a transformative paradigm in network architecture, offering centralized control, programmability, and flexibility to manage complex network infrastructures. However, alongside these advancements, SDN environments face a growing threat from Distributed Denial of Service (DDoS) attacks, which can disrupt network operations, degrade service quality, and compromise security.

DDoS attacks involve overwhelming a target network or service with a flood of malicious traffic, rendering it inaccessible to legitimate users. In SDN architectures, where control and data planes are decoupled, DDoS attacks pose unique challenges due to the dynamic nature of network management and traffic flow control.

The "DDoS SDN Dataset," available on Kaggle, presents an opportunity to delve into the complexities of DDoS attacks within SDN environments. This dataset comprises a rich collection of network traffic data and attack metadata, offering insights into attack patterns, network behavior, and potential vulnerabilities.

In this study, we aim to leverage machine learning techniques to enhance DDoS detection and mitigation capabilities in SDN environments using the "DDoS SDN Dataset" as a case study. By analyzing the dataset and applying machine learning algorithms, we seek to identify anomalous traffic patterns, classify network traffic, predict attack trends, and ultimately strengthen the security posture of SDN infrastructures.

This introduction sets the stage for our investigation into DDoS attacks in SDN environments, highlighting the significance of the problem, the challenges it poses, and the potential benefits of employing machine learning techniques for effective defense. Through our analysis, we aim to contribute to the advancement of network security research and provide actionable insights for mitigating the impact of DDoS attacks on SDN networks.

1.2. OBJECTIVES

- The primary objective of this study is to leverage machine learning techniques for the detection and mitigation of Distributed Denial of Service (DDoS) attacks within Software-Defined Networking (SDN) environments. Specifically, our goals include:

- Exploratory Analysis: Conducting an exploratory analysis of the "DDoS SDN Dataset" to understand the characteristics of network traffic and DDoS attacks within SDN infrastructures. This involves examining the distribution of features, identifying patterns, and gaining insights into the behavior of attackers and network components.

- Machine Learning Application: Applying a variety of machine learning algorithms to the dataset to classify network traffic, detect anomalies indicative of DDoS attacks, and predict attack trends. These algorithms may include supervised, unsupervised, and semi-supervised learning techniques tailored to the unique challenges of SDN environments.

- Evaluation and Performance Analysis: Evaluating the performance of machine learning models using established metrics such as accuracy, precision, recall, and F1-score. Assessing the efficacy of different algorithms in detecting and mitigating DDoS attacks in SDN environments and identifying the most effective approaches.

- Insights and Recommendations: Deriving actionable insights from the analysis to inform network administrators, security practitioners, and researchers about effective strategies for defending against DDoS attacks in SDN environments. Providing recommendations for optimizing network security policies, enhancing threat detection capabilities, and improving overall network resilience.

1.3. MOTIVATION

The motivation behind this study stems from the increasing prevalence and severity of Distributed Denial of Service (DDoS) attacks targeting Software-Defined Networking (SDN) environments. Several factors drive our interest and motivation for investigating DDoS attacks in SDN:

- Rising Threat Landscape: DDoS attacks continue to pose significant challenges to network security, with attackers leveraging increasingly sophisticated techniques to disrupt services and compromise network infrastructure. The dynamic nature of SDN introduces new attack vectors and vulnerabilities, necessitating innovative approaches for detection and mitigation.

- Critical Infrastructure Vulnerability: SDN infrastructures underpin critical services and applications across various sectors, including telecommunications, finance, healthcare, and transportation. Any disruption to SDN networks due to DDoS attacks can have far-reaching consequences, including financial losses, reputational damage, and threats to public safety.

- Complexity of SDN Environments: SDN architectures introduce complexities in network management and traffic flow control, making them susceptible to exploitation by malicious actors. The decoupling of control and data planes, centralized control, and programmability features of SDN create unique challenges for detecting and mitigating DDoS attacks effectively.

- Need for Effective Defense Mechanisms: Traditional DDoS mitigation techniques may be insufficient or ineffective in SDN environments due to their dynamic nature and scale. There is a pressing need for innovative defense mechanisms that leverage the capabilities of SDN, such as dynamic traffic routing, adaptive access control, and real-time threat intelligence integration.

- Potential of Machine Learning: Machine learning techniques offer promise for enhancing DDoS detection and mitigation capabilities in SDN environments. By leveraging the rich data available in the "DDoS SDN Dataset" and applying advanced analytics, we aim to uncover patterns, anomalies, and trends that can inform the development of robust security solutions.

2. LITERATURE SURVEY

[1] **Tonkal, Ö.; Polat, H.; Ba¸saran, E.; Cömert, Z.; Kocao ˘gluR.**
Machine Learning Approach Equipped with Neighbourhood Component Analysis for DDoS Attack Detection in Software-Defined Networking.

- The paper presents a novel machine learning approach leveraging Neighbourhood Component Analysis for DDoS attack detection in SDN, offering insights into the effectiveness of dimensionality reduction techniques in enhancing the performance of DDoS detection mechanisms.

[2] **Naziya Aslam1 · Shashank Srivastava1 · M. M. Gore.**
A Comprehensive Analysis of Machine Learning- and Deep Learning-Based Solutions for DDoS Attack Detection in SDN.
https://doi.org/10.1007/s13369-023-08075-2.

- The paper concludes by summarizing the findings of the survey and highlighting the importance of continued research and innovation in ML and DL-based solutions for DDoS attack detection in SDN. It emphasizes the need for robust, scalable, and adaptive defense mechanisms to combat the evolving threat landscape posed by DDoS attacks.

[3] **YANG Lingfeng, ZHAO Hui.**
DDoS Attack Identification and Defense using SDN based on Machine Learning Method.

- The paper concludes by summarizing the key findings and contributions of the study. It emphasizes the importance of machine learning techniques in enhancing the security and resilience of SDN infrastructures against DDoS attacks.

[4] **Jyoti Tolanur, Shilpa Chaudhari**
DDoS Attacks Analysis with Cyber Data Forensics using Weighted Logistic Regression and Random Forest.

- The paper concludes by summarizing the key findings and contributions of the study. It underscores the importance of cyber data forensics and machine learning in enhancing the detection and analysis of DDoS attacks and calls for further research to explore advanced algorithms and techniques for improving DDoS attack mitigation strategies.

3. DATA ANALYSIS

3.1. STRUCTURE OF DATA

The "DDoS SDN Dataset" available on Kaggle comprises a comprehensive collection of network traffic data and attack metadata within Software-Defined Networking (SDN) environments. This section provides an overview of the dataset, including its structure, attributes, and sources as follows.

Flow Features:

- Source IP address: The IP address of the sender or the originator of the network traffic. It identifies the device or host that initiated the communication.

- Destination IP address: The IP address of the receiver or the intended recipient of the network traffic. It identifies the device or host to which the communication is addressed.

- Source port: The port number used by the sender's application or service. It helps identify the specific application or service generating the traffic on the sender's side.

- Destination port: The port number used by the receiver's application or service. It helps identify the specific application or service intended to receive the traffic on the receiver's side.

- Protocol: The protocol used for communication within the flow, such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol), etc. Different protocols have distinct characteristics and behaviors.

- Number of packets: The total count of packets transmitted within the flow. It indicates the volume of data exchanged between the source and destination.

- Number of bytes: The total size of the data transmitted within the flow, measured in bytes. It provides additional insight into the volume of data exchanged between the source and destination.

- Duration of the flow: The time interval between the first and last packet of the flow. It represents the duration of the communication session or interaction between the source and destination.

Switch Information:

- Switch ID or Name: A unique identifier or name assigned to each switch in the SDN infrastructure. This identifier distinguishes one switch from another and is used for tracking and management purposes.

- Flow Table Entries: SDN switches maintain flow tables that determine how incoming packets should be forwarded. Attributes related to flow table entries may include:

    - Match fields: Criteria used to match incoming packets, such as source and destination IP addresses, TCP/UDP ports, VLAN tags, etc.
    - Actions: Instructions specifying what actions the switch should take when a packet matches a flow entry, such as forwarding the packet to a specific port, dropping the packet, or sending it to the controller.
    - Statistics: Information about the number of packets and bytes processed by each flow entry, as well as other statistics related to flow performance.

- Flow Modification Messages: SDN controllers communicate with switches by sending flow modification messages to dynamically adjust flow entries based on network conditions or policy changes. Attributes related to flow modification messages may include:

    - Message type: Indicates the type of message (e.g., flow add, flow modify, flow delete).
    - Flow entry parameters: Parameters specifying the characteristics of the flow entry to be added, modified, or deleted.
    - Timestamp: The timestamp indicating when the flow modification message was sent or received.

Controller-Switch Communication:

- Controller ID or Name: A unique identifier or name assigned to the SDN controller.
- Controller-Switch Messages: Messages exchanged between the controller and switches, such as flow setup requests, acknowledgments, error notifications, etc.

Attack Metadata:

- Attack Type: Specifies the type or method of the DDoS attack. Common types include UDP flood, SYN flood, ICMP flood, HTTP flood, DNS amplification, etc. Understanding the attack type helps in identifying the attack vectors and selecting appropriate mitigation strategies.

- Attack Intensity: Indicates the severity or intensity of the DDoS attack. It may be measured in terms of packet rate, bandwidth consumption, or other metrics. Higher intensity attacks can overwhelm network resources more effectively, causing greater disruption to services.

- Attack Duration: Refers to the length of time during which the DDoS attack occurred. It helps in understanding the temporal characteristics of the attack and assessing its impact on network operations over time.

- Target IP Addresses or Network Segments: Identifies the specific IP addresses or network segments that were targeted by the DDoS attack. Knowing the target(s) helps in assessing the scope and scale of the attack and identifying potential vulnerabilities in the network infrastructure.

- Characteristics of the Attacker(s): Provides information about the attackers perpetrating the DDoS attack, such as their IP addresses, geographical locations, botnet affiliations, or other identifying attributes. Understanding the characteristics of the attackers can help in attribution and law enforcement efforts.

Timestamps:

- Timestamps associated with network events and flow entries.

Network Topology Information:

- SDN topology (switches, routers, connections).
- Controller(s) information.
- Application(s) running on the network.

3.2. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is a process of analyzing and summarizing a dataset to understand its properties and characteristics. The goal of EDA in this work is to gain insights and a deeper understanding of the DDoS attack data from the Canadian Institute for Cybersecurity dataset, and to identify potential outliers that may indicate unusual or suspicious activity, trends, and patterns that may be useful in detecting and preventing DDoS attacks. EDA can involve a variety of techniques such as visualizing the data using graphs and plots, summarizing the data using statistical measures, and identifying patterns and correlations in the data using machine learning algorithms.

Dataset contains 104345 rows and 23 columns where the label column contains the labeled data.
In label attribute 1 specifies Attack happened and 0 specifies benign.

```
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

Figure 1.

```
data.label.unique()
```

```
array([0, 1], dtype=int64)
```

```
data.label.value_counts()
```

```
0    63561
1    40784
Name: label, dtype: int64
```

Figure 2.

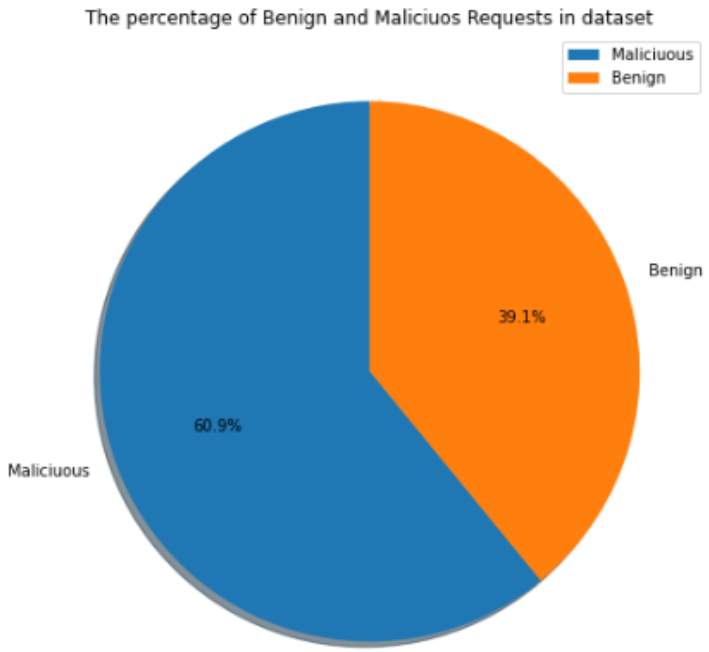

Figure 3.

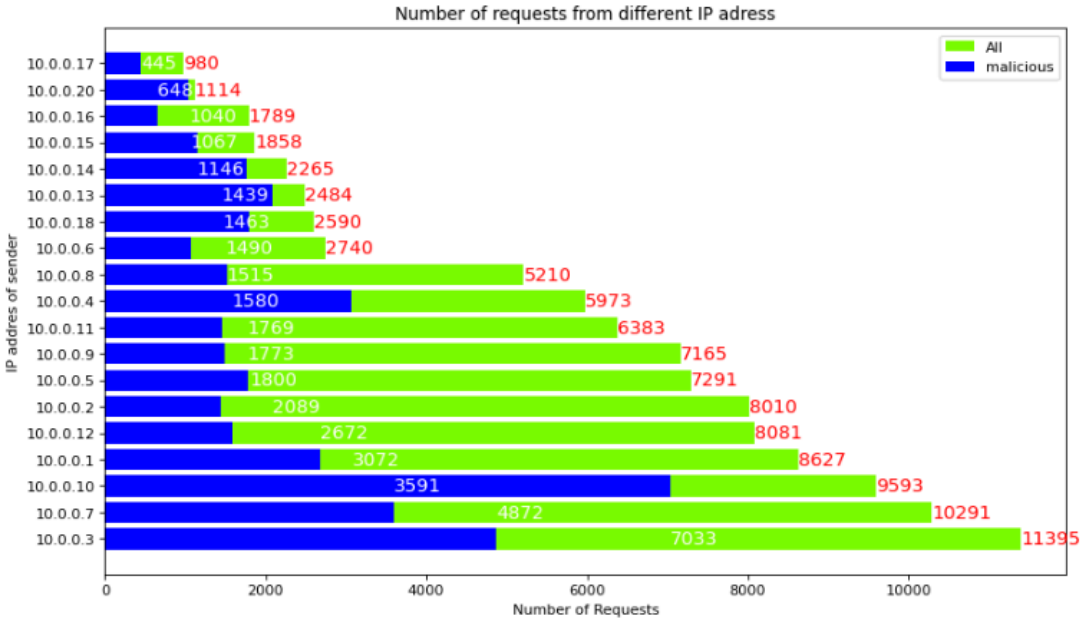

Figure 4.

10.0.0.20 IP address participating in sending more malicious packets that can cause DDos attack. Source IP addresses that generate a disproportionately large volume of traffic compared to other sources in the network may be indicative of DDoS attack activity. Such IP addresses might be responsible for flooding the target with a high volume of packets, overwhelming its resources and causing service disruption.

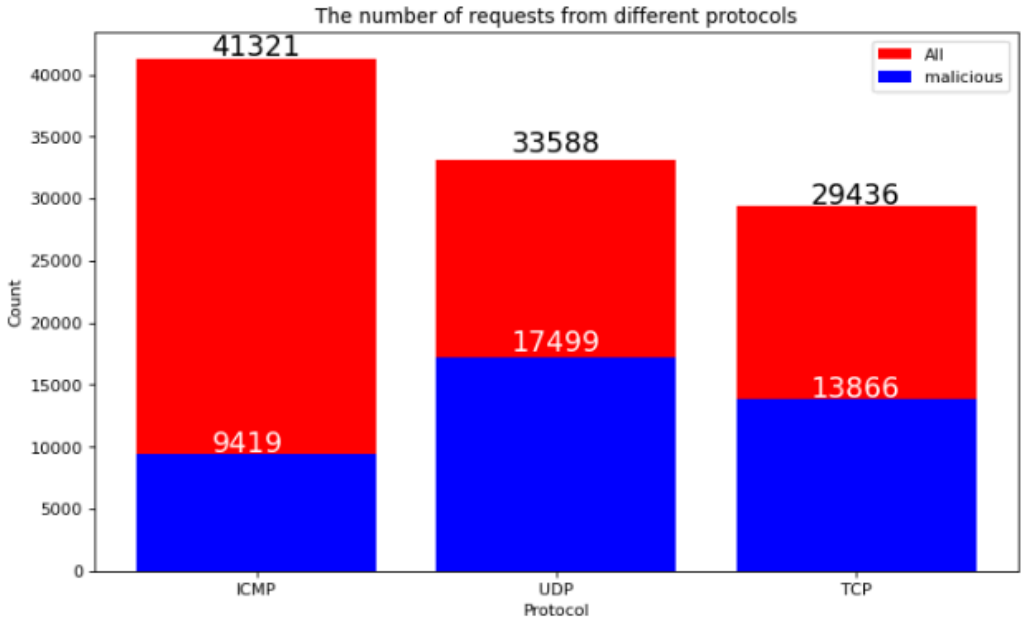

Figure 5.

UDP (User Datagram Protocol):

- UDP is connectionless and does not require a handshake before data transmission, making it faster and more efficient for sending data packets.
- UDP-based DDoS attacks often involve flooding the target with large volumes of UDP packets, exploiting vulnerabilities in services that rely on UDP for communication.

TCP (Transmission Control Protocol):

- TCP is connection-oriented and requires a three-way handshake (SYN, SYN-ACK, ACK) before data transmission, providing reliability and error checking.
- TCP-based DDoS attacks typically involve overwhelming the target with excessive connection requests or exploiting vulnerabilities in the TCP protocol stack.

ICMP (Internet Control Message Protocol):

- ICMP is primarily used for diagnostic and error messaging within IP networks, including functions such as ping (echo request/reply) and traceroute.
- ICMP-based DDoS attacks involve flooding the target with ICMP packets, such as ICMP echo request packets (ping flood attacks), to consume network bandwidth and overwhelm the target's resources.
- ICMP-based attacks can also be used for reconnaissance purposes, including ICMP scanning and ICMP tunneling attacks.

In terms of which protocol is more commonly used in DDoS attacks, it can vary depending on the specific attack vectors, the target's vulnerabilities, and the attacker's objectives. UDP flood attacks are often favored for their simplicity and efficiency in generating high volumes of traffic.

3.3. PREPROCESSING OF DATA

Preprocessing of a dataset involves a series of steps performed to clean, transform, and prepare the data for analysis or model training.

3.3.1. DEALING WITH NULL VALUES

Tot_kbps and rx_kbps contain 506 null values. The number of rows with null values is relatively small compared to the total size of the dataset and removing them does not significantly impact the analysis Therefore deleting rows which contain null values.

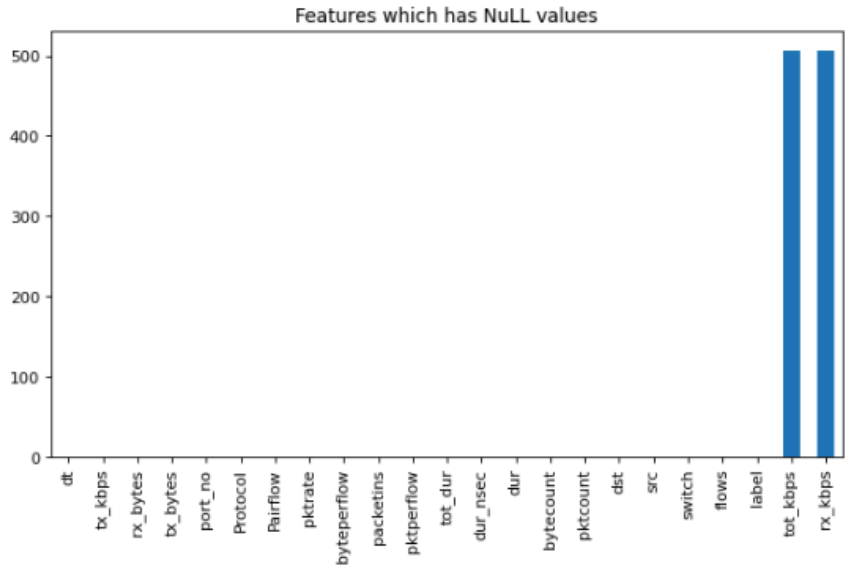

Figure 6.

```
data.isnull().sum()
```

```
dt              0
switch          0
src             0
dst             0
pktcount        0
bytecount       0
dur             0
dur_nsec        0
tot_dur         0
flows           0
packetins       0
pktperflow      0
byteperflow     0
pktrate         0
Pairflow        0
Protocol        0
port_no         0
tx_bytes        0
rx_bytes        0
tx_kbps         0
rx_kbps       506
tot_kbps      506
label           0
dtype: int64
```

Figure 7.

```
data = data.dropna()
```

```
data.isnull().sum()
```

```
dt            0
switch        0
src           0
dst           0
pktcount      0
bytecount     0
dur           0
dur_nsec      0
tot_dur       0
flows         0
packetins     0
pktperflow    0
byteperflow   0
pktrate       0
Pairflow      0
Protocol      0
port_no       0
tx_bytes      0
rx_bytes      0
tx_kbps       0
rx_kbps       0
tot_kbps      0
label         0
dtype: int64
```

Figure 8.

3.3.2. FEATURE SELECTION

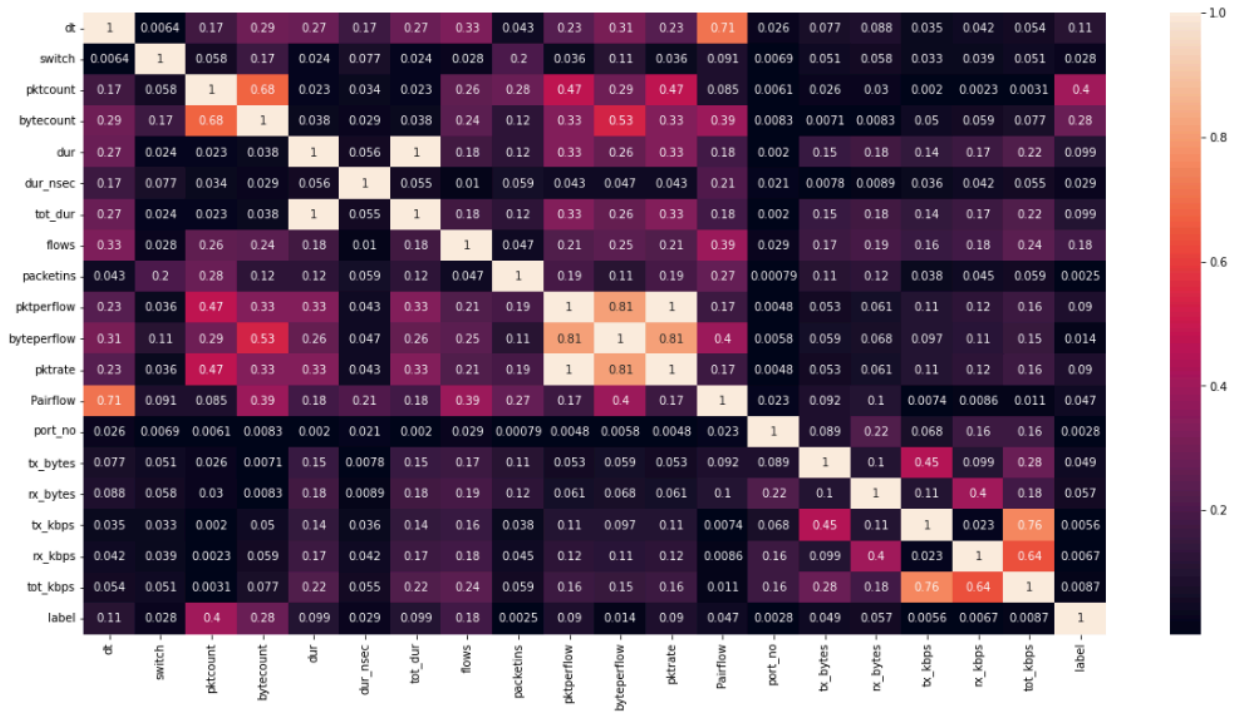

Figure 9.

important_features = ['src', 'pktcount', 'dst', 'byteperflow', 'pktperflow', 'pktrate', 'tot_kbps', 'rx_kbps', 'flows', 'bytecount', 'dt', 'Protocol', 'dur', 'tot_dur']

'src': This likely refers to the source IP address of the network traffic. The source IP address indicates the origin of the data packets within the network.

'pktcount': This could represent the count or number of packets associated with a particular flow or network connection.

'dst': Similar to 'src', this likely refers to the destination IP address of the network traffic. The destination IP address indicates the intended recipient of the data packets within the network.

'byteperflow': This could represent the average number of bytes transmitted per flow or network connection.

'pktperflow': Similar to 'byteperflow', this could represent the average number of packets transmitted per flow or network connection.

'pktrate': This could represent the rate or frequency of packet transmission, typically measured in packets per second (pps) or packets per minute (ppm).

'tot_kbps': This could represent the total throughput or bandwidth usage in kilobits per second (kbps) for the network traffic.

'rx_kbps': This could represent the received bandwidth usage in kilobits per second (kbps) for the network traffic.

'flows': This could represent the number of distinct flows or network connections within the dataset.

'bytecount': This likely represents the total number of bytes transmitted in the network traffic.

'dur': This could represent the duration of each flow or network connection, typically measured in seconds or milliseconds.

'tot_dur': This could represent the total duration of all flows or network connections within the dataset.

These features provide valuable information about network traffic patterns, characteristics of flows, and potential indicators of DDoS attacks. Analyzing these attributes can help identify anomalies, detect malicious activity, and improve network security measures.

Removing src, dst, and dt because source ip (src) ,destination ip (dst) and date timestamp(dt) are irrelevant to the objective. In some cases, these features might indeed be relevant or could be transformed into more meaningful representations through feature engineering techniques.

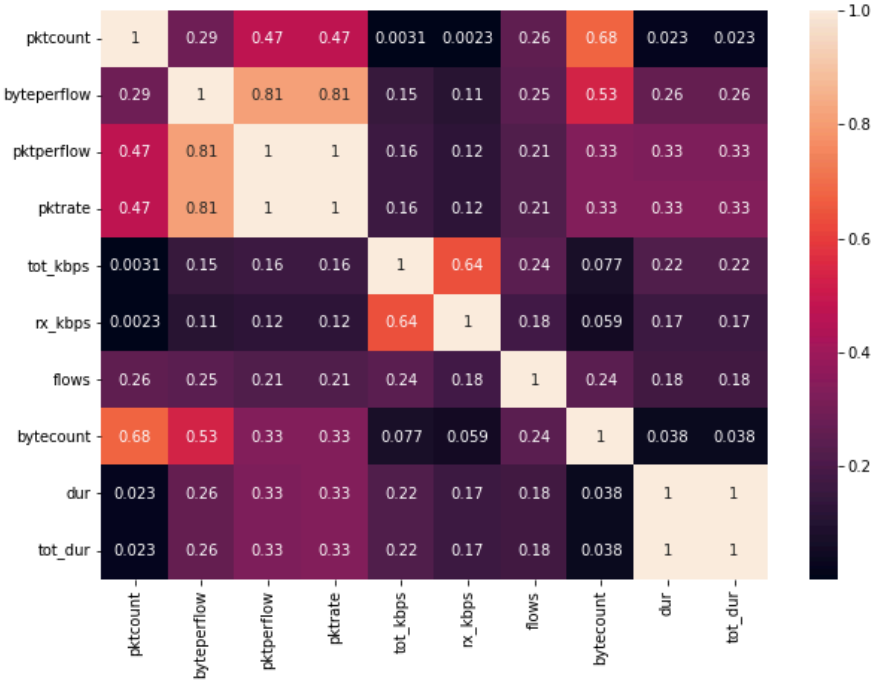

Figure 10.

Pkperflow and pktrate correlation value is 1.
Dur and total_dur correlation value is 1.
Therefore removing dur, pkrate, pktperflow.

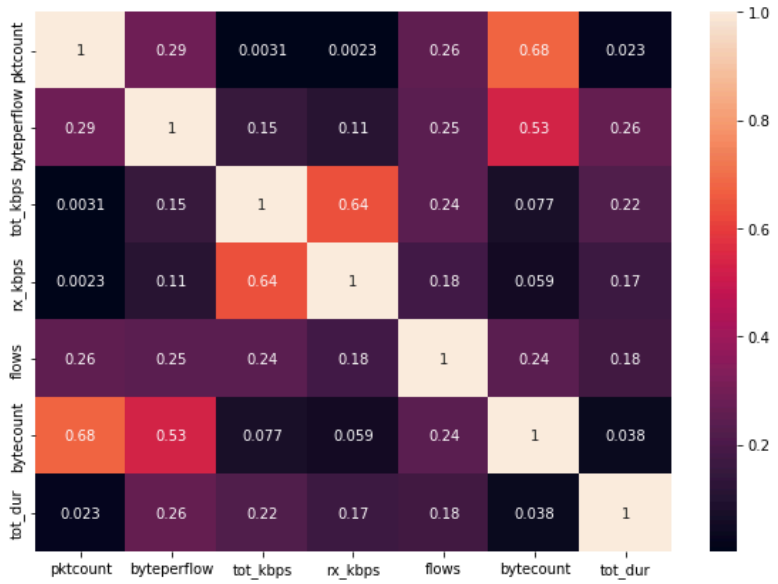

Figure 11.

3.3.3. ENCODING

One-hot encoding is a technique used to convert categorical variables into numerical format, which can be used as input for machine learning algorithms.

	pktcount	byteperflow	tot_kbps	rx_kbps	flows	bytecount	tot_dur	Protocol_ICMP	Protocol_TCP	Protocol_UDP
0	45304	14428310	0.0	0.0	3	48294064	1.010000e+11	0	0	1
1	126395	14424046	0.0	0.0	2	134737070	2.810000e+11	0	0	1
2	90333	14427244	0.0	0.0	3	96294978	2.010000e+11	0	0	1
3	90333	14427244	0.0	0.0	3	96294978	2.010000e+11	0	0	1
4	90333	14427244	0.0	0.0	3	96294978	2.010000e+11	0	0	1

Figure 12.

3.3.4. FEATURE SCALING

StandardScaler is a popular technique used for feature scaling in machine learning and data analysis. It is used to transform the features such that they have a mean of 0 and a standard deviation of 1.

It helps improve the performance and stability of machine learning algorithms by standardizing the scale of features and is a standard preprocessing step in many machine learning workflows.

4. IMPLEMENTATION

4.1. SPLITTING OF DATASET

```
def __init__(self, data):
    self.data = data
    X = preprocessing.StandardScaler().fit(self.data).transform(self.data)
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, random_state=42, test_size=0.3)
```

Figure 13.

Splitting the dataset is a fundamental step in machine learning that involves dividing the available data into separate subsets for training, validation, and testing purposes. The purpose of splitting the dataset is to ensure that the model is trained on one set of data, tuned on another set, and evaluated on yet another set. This helps assess the model's performance and generalization ability accurately. Here's an explanation of each subset:

Training Set:

The training set comprises a majority portion of the dataset (typically 70-80%) and is used to train the machine learning model.
The model learns from the patterns and relationships in the training data, adjusting its parameters to minimize the error between predicted and actual outcomes.

Validation Set:

The validation set is a smaller portion of the dataset (typically 10-20%) that is used to tune hyperparameters and evaluate the model's performance during training.
Hyperparameters are parameters that control the learning process of the algorithm and are not learned from the data. Examples include the number of trees in a random forest or the learning rate in a neural network.
The model's performance is monitored on the validation set, and hyperparameters are adjusted iteratively to optimize performance.

Test Set:

The test set is a separate portion of the dataset (typically 10-20%) that is used to assess the final performance of the trained model.
It serves as an unbiased evaluation of the model's ability to generalize to new, unseen data.
The model's predictions on the test set are compared to the actual outcomes to calculate evaluation metrics such as accuracy, precision, recall, F1-score, etc.

4.2. MACHINE LEARNING ALGORITHM

4.2.1. LOGISTIC REGRESSION

```
def LogisticRegression(self):
    solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

    start_time = time.time()
    results_lr = []
    accuracy_list = []
    for solver in solvers:
        LR = LogisticRegression(C=0.03, solver=solver).fit(self.X_train, self.y_train)
        predicted_lr = LR.predict(self.X_test)
        accuracy_lr = accuracy_score(self.y_test, predicted_lr)
        #print("Accuracy: %.2f%%" % (accuracy_lr * 100.0))
        #print('################################################################')
        results_lr.append({'solver' : solver, 'accuracy': str(round(accuracy_lr * 100, 2)) + "%",
                           'Coefficients': {'W' : LR.coef_, 'b': LR.intercept_}})

        accuracy_list.append(accuracy_lr)

    solver_name = solvers[accuracy_list.index(max(accuracy_list))]
    LR = LogisticRegression(C=0.03, solver=solver_name).fit(self.X_train,self.y_train)
    predicted_lr = LR.predict(self.X_test)
    accuracy_lr = accuracy_score(self.y_test, predicted_lr)
    print("Accuracy: %.2f%%" % (accuracy_lr * 100.0), '\n')
    print("########################################################################")
    print('Best solver is : ', solver_name)
    print("########################################################################")
    print(classification_report(predicted_lr, self.y_test), '\n')
    print("########################################################################")
    confusion_matrix = metrics.confusion_matrix(self.y_test, predicted_lr)
    cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.grid(False)
    cm_display.plot(ax=ax)
    print("--- %s seconds --- time for LogisticRegression" % (time.time() - start_time))
```

Figure 14.

4.2.2. GAUSSIAN NAIVE BAYES

```
nb = GaussianNB()

# Define hyperparameters grid
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]  # Example values for var_smoothing
}

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=nb, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print best hyperparameters and best accuracy score
print("Best hyperparameters:", grid_search.best_params_)
print("Best accuracy score:", grid_search.best_score_)

# Evaluate model performance on test set
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test set accuracy:", test_accuracy)

predicted_nb = grid_search.predict(np.array(X_test))
confusion_matrix = metrics.confusion_matrix(y_test, predicted_nb)
cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)
cm_display.plot(ax=ax)
```

Figure 15.

4.2.3. K-NEAREST NEIGHBORS

```
knn = KNeighborsClassifier()

# Define hyperparameters grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # Example values for n_neighbors
    'weights': ['uniform', 'distance'],  # Example values for weights
    'p': [1, 2]  # Example values for p (1: Manhattan distance, 2: Euclidean distance)
}

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print best hyperparameters and best accuracy score
print("Best hyperparameters:", grid_search.best_params_)
print("Best accuracy score:", grid_search.best_score_)

# Evaluate model performance on test set
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test set accuracy:", test_accuracy)

predicted_knn = grid_search.predict(np.array(X_test))
confusion_matrix = metrics.confusion_matrix(y_test, predicted_knn)
cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)
cm_display.plot(ax=ax)
```

Figure 16.

4.2.4. SUPPORT VECTOR MACHINE

```
def SupportVectorMachine(self):
    start_time = time.time()
    accuracy_list = []
    result_svm = []
    kernels = ['linear', 'poly','rbf', 'sigmoid']
    #kernels = ['rbf']
    for kernel in kernels:
        SVM = svm.SVC(kernel=kernel).fit(self.X_train, self.y_train)
        predicted_svm = SVM.predict(self.X_test)
        accuracy_svm = accuracy_score(self.y_test, predicted_svm)
        result_svm.append({"kernel" : kernel, "accuracy": f"{round(accuracy_svm*100,2)}%"})
        print("Accuracy: %.2f%%" % round((accuracy_svm * 100.0),2))
        print('################################################################################')
        accuracy_list.append(accuracy_svm)

    kernel_name = kernels[accuracy_list.index(max(accuracy_list))]
    SVM = svm.SVC(kernel=kernel_name).fit(self.X_train, self.y_train)
    predicted_svm = SVM.predict(self.X_test)
    accuracy_svm = accuracy_score(self.y_test, predicted_svm)
    print(f"Accuracy of SVM model {round(accuracy_svm,2)*100}%", '\n')
    print("################################################################################")
    print('best kernel is : ', kernel_name)
    print("################################################################################")
    print(classification_report(predicted_svm, self.y_test))
    print("################################################################################")
    confusion_matrix = metrics.confusion_matrix(self.y_test, predicted_svm)
    cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.grid(False)
    cm_display.plot(ax=ax)
    print("--- %s seconds ---" % (time.time() - start_time))
```

Figure 17.

4.2.5. DECISION TREE

```
def DecisionTree(self):
    start_time = time.time()
    tree = DecisionTreeClassifier()
    dt_search = GridSearchCV(tree, param_grid={'criterion' : ['gini', 'entropy'],
                                    'max_depth' : [2,3,4,5,6,7,8, 9, 10],
                                    'max_leaf_nodes' : [2,3,4,5,6,7,8,9,10, 11]},
                        n_jobs=-1, cv=5, scoring='accuracy', verbose=2)

    dt_search.fit(self.X_train, self.y_train)

    criterion = dt_search.best_params_['criterion']
    max_depth = dt_search.best_params_['max_depth']
    max_leaf_nodes = dt_search.best_params_['max_leaf_nodes']

    dtree = DecisionTreeClassifier(criterion=criterion,
                                   max_depth=max_depth,
                                   max_leaf_nodes=max_leaf_nodes).fit(self.X_train, self.y_train)
    predicted_dt = dtree.predict(self.X_test)
    accuracy_dt = metrics.accuracy_score(self.y_test, predicted_dt)
    print(f"criterion: {criterion}, max depth: {max_depth}, max_leaf: {max_leaf_nodes}")
    print(f"The Accuracy is : {round(accuracy_dt * 100,2)}%")
    print("##############################################################################")
    print(classification_report(predicted_dt, self.y_test))
    print("##############################################################################")
    confusion_matrix = metrics.confusion_matrix(self.y_test, predicted_dt)
    cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.grid(False)
    cm_display.plot(ax=ax)

    print("--- %s seconds ---" % (time.time() - start_time))
```

Figure 18.

4.2.6. RANDOM FOREST

```
def RandomForest(self):
    start_time = time.time()
    RF = RandomForestClassifier(criterion='gini',
                                n_estimators=500,
                                min_samples_split=10,
                                #min_samples_leaf=1,
                                max_features='auto',
                                oob_score=True,
                                random_state=1,
                                n_jobs=-1).fit(self.X_train, self.y_train)

    predicted_rf = RF.predict(self.X_test)
    rf_accuracy = accuracy_score(self.y_test, predicted_rf)
    print(f"Accuracy of RF is : {round(rf_accuracy*100,2)}%", '\n')
    print("##############################################################################")
    print(classification_report(predicted_rf, self.y_test))
    print("##############################################################################")
    confusion_matrix = metrics.confusion_matrix(self.y_test, predicted_rf)
    cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['normal','attack'])

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.grid(False)
    cm_display.plot(ax=ax)

    print("--- %s seconds ---" % (time.time() - start_time))
```

Figure 19.

5. RESULTS

5.1. CONFUSION MATRIX

5.1.1. LOGISTIC REGRESSION

```
Accuracy: 75.21%

########################################################################
Best solver is :  sag
########################################################################
              precision    recall  f1-score   support

           0       0.85      0.77      0.81     20968
           1       0.60      0.72      0.65     10184

    accuracy                           0.75     31152
   macro avg       0.73      0.74      0.73     31152
weighted avg       0.77      0.75      0.76     31152


########################################################################
--- 3.6410019397735596 seconds --- time for LogisticRegression
```

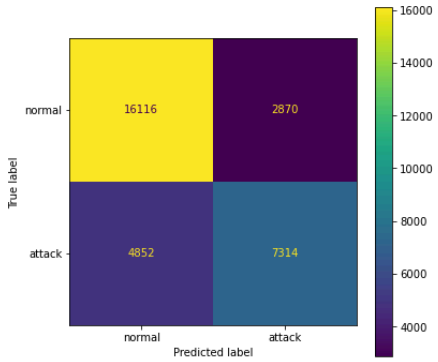

Figure 20.

5.1.2. GUASSIAN NAIVE BAYES

```
Best hyperparameters: {'var_smoothing': 1e-09}
Best accuracy score: 0.6362210552940025
Test set accuracy: 0.6326720595788392

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x131e7f7f910>
```

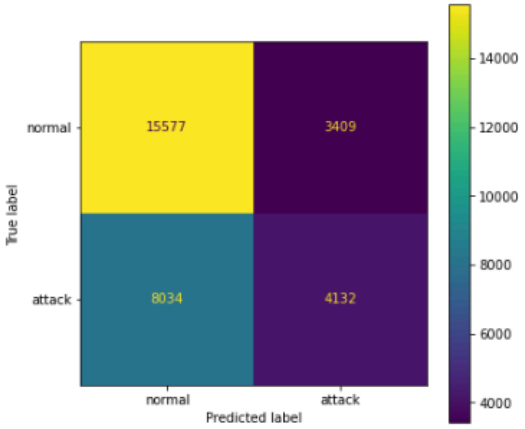

Figure 21.

5.1.3. K-NEAREST NEIGHBORS

```
Best hyperparameters: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Best accuracy score: 0.992749731561495
Test set accuracy: 0.996822033898305

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x131e7f3dfd0>
```

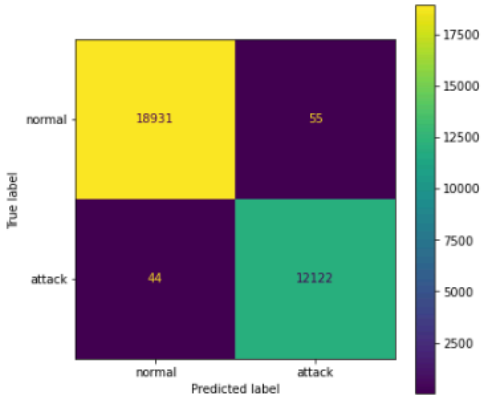

Figure 22.

5.1.4. SUPPORT VECTOR MACHINE

```
Accuracy: 75.65%
######################################################################
Accuracy: 91.69%
######################################################################
Accuracy: 91.77%
######################################################################
Accuracy: 56.07%
######################################################################
Accuracy of SVM model 92.0%

######################################################################
best kernel is :  rbf
######################################################################
              precision    recall  f1-score   support

           0       0.90      0.96      0.93     17757
           1       0.95      0.86      0.90     13395

    accuracy                           0.92     31152
   macro avg       0.92      0.91      0.91     31152
weighted avg       0.92      0.92      0.92     31152

######################################################################
--- 1065.645780324936 seconds ---
```

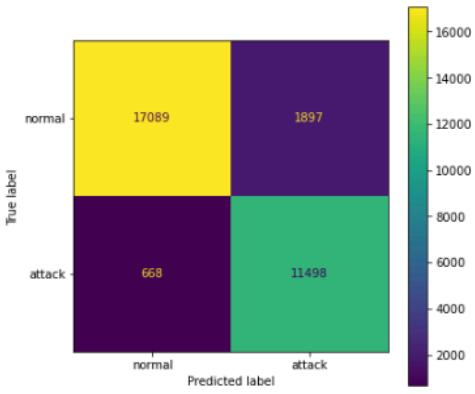

Figure 23.

5.1.5. DECISION TREE

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
criterion: gini, max depth: 6, max_leaf: 11
The Accuracy is : 94.19%
################################################################################
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     17287
           1       1.00      0.87      0.93     13865

    accuracy                           0.94     31152
   macro avg       0.95      0.94      0.94     31152
weighted avg       0.95      0.94      0.94     31152

################################################################################
--- 49.034101486206055 seconds ---
```

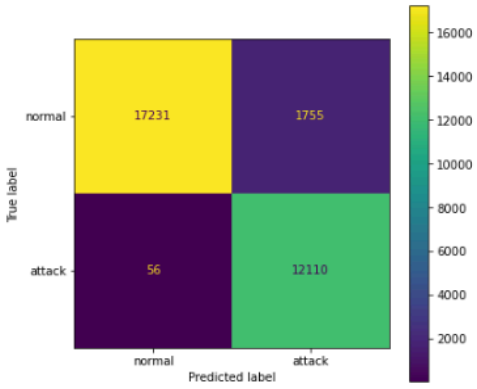

Figure 24.

5.1.6. RANDOM FOREST

```
Accuracy of RF is : 99.42%

################################################################################
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     18922
           1       1.00      0.99      0.99     12230

    accuracy                           0.99     31152
   macro avg       0.99      0.99      0.99     31152
weighted avg       0.99      0.99      0.99     31152

################################################################################
--- 14.339549779891968 seconds ---
```

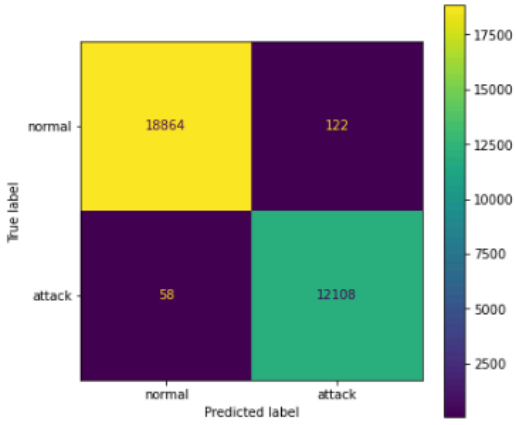

Figure 25.

5.2. COMPARATIVE ANALYSIS

Algorithm	Accuracy	Precision	Recall
LR	75.21	76.85	84.88
KNN	99.68	99.76	99.71
DT	94.18	99.67	90.75
SVM	91.76	96.23	90.00
RF	99.24	99.69	99.35
NB	63.26	65.97	82.02

6. CONCLUSION

In conclusion, effective management of DDoS attacks is crucial for maintaining the integrity and availability of network resources. The dataset analyzed provides valuable insights into network traffic patterns, attack metadata, and network topology information, enabling the development of robust defense mechanisms against DDoS attacks.

Moving forward, machine learning algorithms can be leveraged to classify network traffic and detect anomalous behavior indicative of DDoS attacks. By training models on labeled data and evaluating their performance using metrics such as accuracy, precision, recall, and F1-score, we can develop effective DDoS detection systems capable of mitigating the impact of attacks in real-time.

In conclusion, the analysis of the DDoS SDN dataset provides valuable insights into the dynamics of DDoS attacks in software-defined networks. By leveraging machine learning techniques and network security best practices, organizations can enhance their ability to detect, mitigate, and respond to DDoS attacks, safeguarding the integrity and availability of critical network infrastructure.

7. REFERENCE

[1]Tonkal, Ö.; Polat, H.; Ba¸saran, E.; Cömert, Z.; Kocao ˘glu, R. Machine Learning Approach Equipped with Neighbourhood Component Analysis for DDoS Attack Detection in Software-Defined Networking. Electronics 2021, 10, 1227. https://doi.org/10.3390/electronics10111227.

[2] Naziya Aslam1 · Shashank Srivastava1 · M. M. Gore, A Comprehensive Analysis of Machine Learning- and Deep Learning-Based Solutions for DDoS Attack Detection in SDN, Arabian Journal for Science and Engineering (2024) 49:3533–3573 https://doi.org/10.1007/s13369-023-08075-2.

[3] YANG Lingfeng, ZHAO Hui, DDoS Attack Identification and Defense using SDN based on Machine Learning Method, 2375-527X/18/$31.00 ©2018 IEEE DOI 10.1109/I-SPAN.2018.00036

[4] Jyoti Tolanur, Shilpa Chaudhari, DDoS Attacks Analysis with Cyber Data Forensics using Weighted Logistic Regression and Random Forest, 2023 International Conference on Device Intelligence, Computing and Communication Technologies, (DICCT) | 978-1-6654-7491-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/DICCT56244.2023.10110133.d

[5] SDN DDos dataset: https://data.mendeley.com/datasets/jxpfjc64kr/1