DDoS Detection using Machine Learning Algorithms - Advanced Ensemble Classifier Approach

FINAL REPORT

Submitted in fulfillment for the J-Component of
CSE3502 INFORMATION SECURITY MANAGEMENT

by

Hritik Dubey 19BIT0150
Arnav Sharma 19BIT0235

Under the guidance of

Dr. JOHN SINGH K.

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering
Winter Semester 2021-2022

Abstract

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. DDos attacks are well coordinated and are costly to organizations, whether they are small or large firms in terms of bandwidth and time. They also result in the loss of user and confidential data. Therefore, it is important to develop and explore algorithms with higher accuracy that can be used for DDoS detection. The most obvious symptom of a DDoS attack is a site or service suddenly becoming slow or unavailable. But since a number of causes such as a legitimate spike in traffic can create similar performance issues, further investigation is usually required. This study aims to use ML algorithm and advanced ensemble classifier technique for DDOS detection.

Introduction

DDoS (Distributed Denial of Service) attacks are a type of flooding attack that prevents a genuine user from accessing a service. According to a recent study, it is one of the most common dangers facing the cybersecurity industry. As a result, understanding and detecting various types of DDoS threats is critical. The severity of the situation makes this one of the most serious issues in internet security, and several statistical detection approaches, such as wavelet-based, port entropy-based, and destination entropy-based, can be discovered in the literature. However, because the internet is a highly dynamic industry that is always evolving, all of these techniques are time intensive and useless. As a result, several researchers have turned to Machine Learning and Artificial Intelligence approaches to detect DDoS attacks in order to overcome these problems.

There are two types of DDoS attacks -

1. Reflection Based DDoS : They are attacks in which the attacker's identity is concealed through the use of legitimate third-party components. Attackers send packets to reflector servers with the target victim's IP address as the source IP address in order to overwhelm the victim with response packets. In this category, MSSQL and SSDP are examples of TCP-based attacks, whereas CharGen, NTP, and TFTP are examples of UDP-based attacks. DNS, LDAP, NETBIOS, and SNMP are just a few examples of attacks that can be carried out via TCP or UDP.

2. Exploitation-based attacks : These attacks can also be carried out through application layer protocols using transport layer protocols i.e., TCP and UDP. TCP based exploitation attacks include SYN flood and UDP based attacks include UDP flood and UDP- Lag. A UDP flood attack is initiated on the remote host by sending a large number of UDP packets.

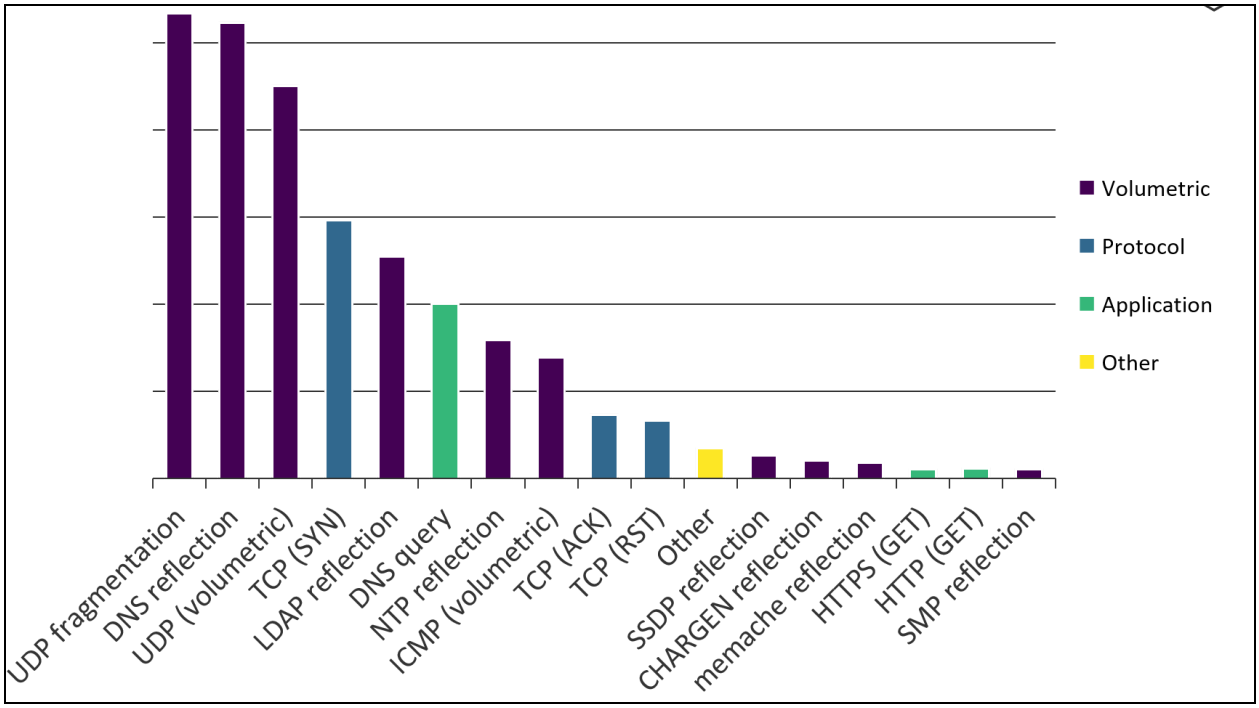


Fig.1 - Types of DDos Attack based on attack vector

Motivation

After the pandemic companies have moved their businesses online, many of them dealing with critical user & confidential company data. Apart from this several IT companies have now switched to work from home where the necessary security infrastructure is usually absent. In the given scenario, DDoS (Distributed Denial of Service) has evolved to become a major threat to businesses and government infrastructure. There is a need to deploy DDos detection & prevention systems to protect such network infrastructure.

Objectives

1. Preparing a Literature survey to identify new techniques & algorithms for DDos detection.
2. Using machine learning classifier algorithm for DDos detection.
3. Finding the best algorithm for different types of attack vectors.
4. Exploring Advanced ensemble classifier algorithm for better accuracy.

Problem Statement

The problem addressed by this study is to find the best ML algorithm for different types of DDos Attacks. As Discussed earlier DDos Attacks are of two types reflection & exploitation based. This study Aims to use a dataset for all these attacks and find the most suitable algorithm.

Literature Survey to identify Existing Systems

Title of the paper	Algorithm/Tools used	Advantages	Issues/Drawbacks
Supervised Learning to Detect DDoS Attacks [1]	Classification and Regression Trees(CART) and Naive Bayes Machine learning classifiers to detect DDoS attacks.	Obtained high performance without using IP addresses and ports. accuracy score of 0.99 with a recall value of 1.00 along with an F1-Score of 0.97 on the training dataset using Decision Tree and an accuracy of 0.98, recall value of 0.93, and an F1-Score of 0.82 on the training using Naive Bayes.	The algorithms devised were highly focused on detecting backscatter traffic and detection of whether there was a DDoS attack or not without Focusing on the type of attack.
DDoS Attack Detection Algorithms Based on Entropy Computing [2]	Uses the Multilayer Perceptron network to successfully classify normal and attack state (binary) or DDos source.DDos victim or normal(3 types) of DDoS type attack with high true positive and low false positive rate	DDoS detection rate with an accuracy of 99.9971%.	Majorly focusing on UDP attacks
Smart Detection: An Online Approach for DoS/DDoS Attack Detection Using Machine Learning [3]	Normal traffic and DDoS signatures were extracted, labeled, and stored in a database. SDS was then created using feature selection techniques. Finally, the most accurate MLA was selected, trained, and loaded into the traffic classification system. software uses the Random Forest Tree algorithm to classify network traffic based on samples taken by the sFlow protocol directly from network devices.	Was able to classify various types of DoS/DDoS attacks, such as TCP flood, UDP flood, HTTP flood, and HTTP slow. Furthermore, the performance of the proposed method was compared against recent and related approaches. the proposed system acquired DR and PREC higher than 93% with FAR less than 1%.	Should have a better hit rate among attack classes and an automatic parameter calibration mechanism that maximizes the detection rate of attacks.
HTTP flood attack detection in application layer using machine	Bio-Inspired Anomaly based HTTP-Flood Attack Detection (BIFAD) is used. In this, we adopted the Bat algorithm, which is a bioinspired approach with magnified speed in search. First we defined feature	Better precision, recall and accuracy wrt to ATP FCAAIA	Very complex to implement and not scalable

learning metrics and bio inspired bat algorithm [4]	metrics to identify the request stream behavior is of attack or normal. The Second contribution is to customize the Bat algorithm to train and test.		
IoT Ddos Attack Detection Using Machine Learning [5]	We utilize the sources of pandemic modeling to IoT networks consisting of WSNs. We build a proposed framework to detect abnormal defense activities. We also used many machine learning and data mining algorithms such as LSVM, Neural Network, and Decision tree.	The merge between random forest and decision tree achieved high accuracy to detect attacks in comparison to neural BP	Not mentioned
DoS Attack Detection Using Machine Learning and Neural Network [6]	The detection is specifically focused on application layer DoS attack detection rather than at transport and network DoS attack detection . The latest DoS attack dataset CIC IDS 2017 dataset is used in the experiment..	Random forest provided with better precision and accuracy	This detection system classifies CIC IDS 2017 dataset into Benign and DoS attack.The proposed system does not multi classified into attacks such as Hearbleed, slowhttptest, slowloris and http flood
DDOS Detection Using Machine Learning Technique [7]	Command-based ping of death technique was used to perform DDoS attacks. Random forest algorithm was used to train the model which resulted in 99.76% of correctly classified instances.	The authors explained the working of dos attack using ping of death , also the during the attack ,high cpu utilization is shown for better understanding .The algorithm chosen for prevention from such attacks results in high accuracy .	Implementation of deep learning technique for the classification of the instances could be performed for better clarity
DDoS Attacks Detection Using Machine Learning Algorithms [8]	PCA (Principal component analysis)-based feature reduction and RNN-based prediction method to detect DDoS attacks	The PCA-RNN detection method with several existing DDoS attacks such as ack-flood,DRDos,UDP flood,Icmp flood and SYN flood was performed and this method provided us with better detection accuracy, higher detection efficiency, and wider applicability	Comparing the PCARNN and RNN algorithms, PCA-RNN prediction time significantly decreased, but the accuracy rate and F1 value only slightly decreased,

DDoS attack detection using machine learning techniques in cloud computing environments [9]	A DDoS detection system based on the C.4.5 (decision tree) algorithm was introduced to mitigate the DDoS threat.. Machine learning techniques such as k means,naive bayes were used so as to compare with the C.4.5 algorithm and also to validate the system.	A high detection and efficiency rate, the detection rate of it could reach more than 98% was found also with the increase in the duration of DDoS attacks, the higher is the attack detection rate of this system	The model proposed did not comply to realtime attack traffic detection and mitigation aimed to security challenges
DDoS attack detection approach (called DeepDefense). [10]	In this paper, the author proposed a deep learning based DDoS attack detection approach (called DeepDefense). A recurrent deep neural network is made in order to learn patterns from sequences of network traffic and trace network attack activities.	As RNN gives is the independence from input window size which is used in previous machine learning methods usually is task-dependent. This brings a limitation for these methods to detect different types of attacks,also it becomes hard to train a long-term sequence for CNN method. However, RNN has shown the capability to solve these problem.	Slow and complex training procedure
Proactive Detection of DDoS Attacks Utilizing k-NN Classifier in an Anti-DDos Framework [11]	The author uses k-Nearest Neighbour (K-NN) in his study to classify the status of the network into Normal, preattack and attack DDoS class status in-order to pre-identify the DDoS attack.	Other algorithms such as Naive Bayesian, C4.5 and K-Means have been explored to achieve an accuracy of 91:4%; 98:8% and 85:9% respectively	No issues found
Maximizes the Linear Quadratic Gaussian (LQG) control cost function under energy constraint. [12]	They consider the optimal jamming attack that maximizes the Linear Quadratic Gaussian (LQG) control cost function under energy constraint. After analyzing the properties of the cost function under an arbitrary attack schedule, the authors derive the optimal jamming attack schedule and the corresponding cost function.	System Stability under the optimal attack schedule is considered. They Further investigate the optimal attack schedule in a WNCS with multiple subsystems.	No issues found

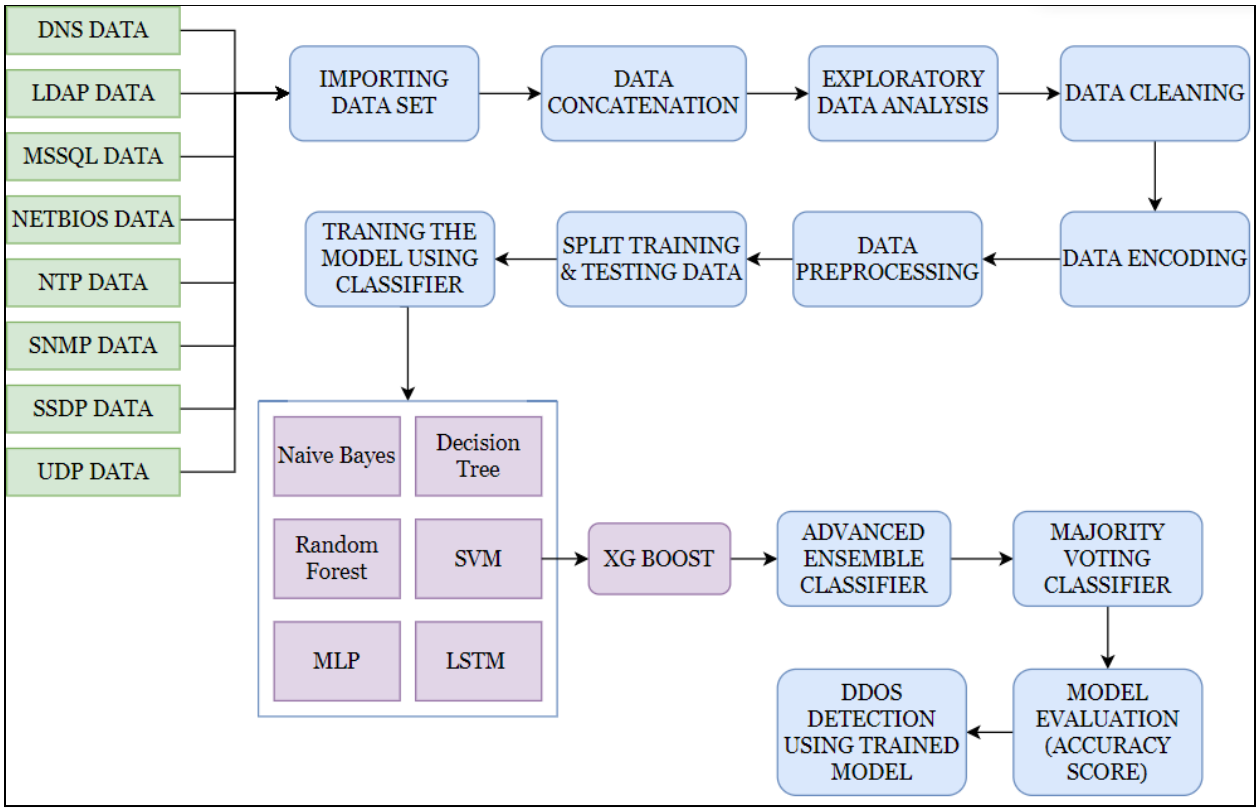
Dynamic entropy based DoS attack detection method [13]	The paper presents us with an intelligent DOS detection framework comprising modules for data generation, feature ranking, generation, and training and testing.	The proposed framework is experimentally tested under actual IoT attack scenarios, and the accuracy of the results is greater than that of traditional classification	Complex procedures may be involved. Better and faster implementation might be possible to improve the accuracy
"Preventing DoS attack in sensor networks: a game theoretic approach," [14]	Authors have used grasshopper optimization algorithm (GOA) with a machine learning algorithm called GOIDS. This approach is based on creating an intrusion detection system (IDS) to fulfill the requirements of the monitored environment.	GOIDS selects the most relevant features from the dataset and then selected features are passed to the classifiers like SVM(Support Vector Machine), MLP(Multi layer Perceptron),etc. GOIDS with decision trees acquire high detection and accuracy with a low false– positive rate.	Various classifiers are used and tested which results in a high amount of training time.
"DoS Attack Prevention on IPS SDN Networks," [15]	A combination of two technologies deployed on traditional networks that have been adapted to the SDN architecture is used.	Implementing an intrusion prevention system (IPS) in an OpenFlow environment that emerged to enhance network security by analyzing data that travels through its structure executed through the command of an SDN controller.	No issues found.
. A Detection Algorithm for DoS Attack in the Cloud Environment. [16]	Proposed a detection algorithm for the attack of DOS in cloud computing. The authors focus on the security perspective of cloud computing against DOS	A new concept has been analysed and tested to deal with DOS attacks in cloud computing.	Cloud computing is a vast and new field and hence testing and training is required to validate the result.
Authentication Flooding DOS Attack Detection and Prevention in 802.11 [17]	Here an algorithm for the detection and prevention authentication request flood attacks by using MAC filter buffer for the maintenance and filtering of MAC as well as buffer monitoring has been proposed	The proposed algorithm has improved the detection performance by 98.4% compared to the related work.	No particular issues found
"An Algorithm for Moderating DoS Attack in Web Based Application," [18]	Here authors have proposed an aegis algorithm which can be used to moderate DoS attack in Web application Vulnerability.	A new algorithm has been proposed which monitors the possible DoS attacks in web applications.	As of now only vulnerability scanner is the tool which is used to detect DoS attacks in web applications.

"Model-based adaptive DoS attack mitigation," [19]	A model-based adaptive architecture and algorithm for detecting DoS attacks at the web application level is proposed. Using a performance model and a decision engine traffic is filtered and suspicious traffic is monitored.	a scalable implementation demonstrate effective mitigation of attacks launched using a real-world DoS attack tool.	No particular issues found
Slowcomm: Design, development and performance evaluation of a new slow DoS attack, [20]	In this paper, we present the novel threat called SlowComm and we show how it can successfully lead a DoS on a targeted system using a small amount of attack bandwidth	Since the proposed attack is not bound to a specific protocol, it can be considered a protocol independent attack, proving the ability it has to affect different Internet services	The attack discovered is new and hence validation and testing is required to get firm results.
A dynamic MLP-based DDoS attack detection method using feature selection and feedback, Computers & Security [21]	In this paper, we chose the multilayer perceptrons (MLP) to demonstrate and solve the proposed problem. In our solution, we combined sequential feature selection with MLP to select the optimal features during the training phase and designed a feedback mechanism to reconstruct the detector when perceiving considerable detection errors dynamically.	Our method was effective in perceiving the detection errors when their saliency accumulated to a certain degree and then reconstruct the detector according to updated data	Not mentioned
Detection of HTTP Flooding Attacks in Cloud Using Dynamic Entropy Method. [22]	An active student portal was built on a cloud computing environment and a new dataset is generated so as to carry out this study. This work assimilates several supervised algorithmsDecision Tree, Support Vector Machine,Logistic Regression and KNN for the purpose of classification	SVM algorithm showed greater results from rest of other algorithms. Based on the study SVM is the best fit algorithm and thereby used in several intrusion detection purposes	Decision tree and KNN performed very poorly

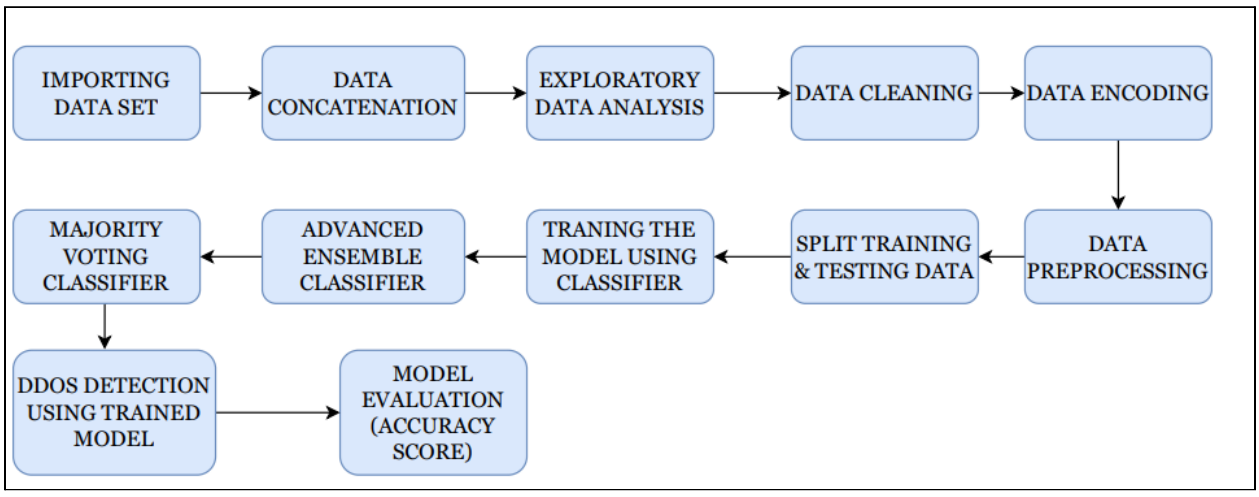
Detection and Defense Algorithms of Different Types of DDoS Attacks Using Machine Learning. [23]	This study uses the Multilayer Perceptron network to successfully classify normal and attack state (binary) or DDos source, DDos victim or normal(3 types) of DDoS type attack with high true positive and low false positive rate, majorly focusing on UDP attacks.	A hybrid methodology to select features using the wrapper Genetic Algorithm(GA) and MLP to classify the attack was implemented in the study which improved the accuracy to 99.9971%	Dataset used is obsolete and hence not reliable.
DDoS attack detection using machine learning techniques in cloud computing environments. [24]	This research is conducted to design a technique called Packet Threshold Algorithm (PTA) coupled with SVM in order to detect four types of DDoS attacks such as TCP SYN flood, UDP flood, Ping of Death and Smurf	The results of this research indicate that the PTA-SVM technique is able to ensure that DDoS attacks are better than the three selected techniques in terms of detection accuracy 99.1% with false positive rates 1.11.	Not mentioned
"An impact analysis: Real time DDoS attack detection and mitigation using machine learning," [25]	C4.5 algorithm was devised to detect DDoS attacks by forming a decision tree to effectively detect the signatures of these attacks.	A comparison of C4.5 with a machine learning algorithm showed that it has an accuracy of 98.8% for correctly classifying a threat as an attack or normal.	The model only predicted whether there was an attack or not. They do not detect the types of attacks
Automated DDOS attack detection in software defined networking [26]	we use Software Defined Networking (SDN), a promising network architecture, for dropping malicious traffic in propagation path to avoid avalanche effect on the victim server in the traditional network	This achieves a distributed anomaly detection to detect and respond to IoT-based DDoS attacks in real time, and avoids the overload of the controller. Machine learning is used in the OF switches with around 99% precision	Did not include more complex ddos attack and didn't use sdn to assess situation
"An anomaly based VoIP DoS attack detection and prevention method using fuzzy logic	This paper discussed a new proposed model called Hop count inspection algorithm (HCF)- Support Vector Machine (SVM) to binary classify DDoS attacks as Normal and attack	The accuracy turned out to be f 98.99% which is higher than many conventional methods.	Random Forest and Decision Tree performed relatively poor with an accuracy of 93% and 61.76% respectively

[27]			
A Study of Machine Learning Algorithms for DDoS [28]	This paper highlights the performance of various algorithms like Back Propagation(BP), PCA-BP, Long Short Term Memory(LSTM), PCALSTM, PCA(SVM), (RNN) and (PCA-RNN) on basis of accuracy, sensitivity, precision, F1 and prediction time	PCA-RNN method has higher detection efficiency and accuracy using KDD99 [18] dataset. Time complexity was reduced by using dimensionality reduction	The dataset used is mostly suitable to detect the wider range of attacks and not particularly DDoS attack because of which the proposed method is only suitable in case of binary classification of DDoS attack.
Automated DDOS attack detection in software defined networking [29]	HPSOGA(Hybrid of Particle Swarm Optimization into genetic algorithm), RBFNN(Radial Basis Function Neural Network)	Individuals are categorised into divisions such as elites(best performing individuals), swarm(worst performing individuals). Hybrid optimization strategy showed promising results as an alternative to forecasting tools.	No real proof as to the hybrid optimization strategy will act as an alternative to the forecasting tools because only certain amount of data (upperhalf of elites) is used as input in RBF-NN
"An anomaly based VoIP DoS attack detection and prevention method using fuzzy logic," [30]	In this paper, by the help of normal SIP traffic, an anomaly based method for detecting this kind of attacks due to different types of SIP signaling packets, is presented. A Finite State Machine (FSM) is used for extracting SIP traffic parameters and specifications in normal conditions.	Fuzzy logic is used for detecting attacks using extracted parameters. The proposed method is fully implemented and tested with the help of Spirent test device.	Results showed that this method could detect and prevent DoS attacks with high probability and without causing overhead on the SIP server. No disadvantages as such mentioned

Architecture Diagram - High level (Flowchart Maker & Online Diagram Software)



Architecture Diagram - Low level (Flowchart Maker & Online Diagram Software)



Implementation

DDoS Detection using Advanced Ensemble Classifier Approach

```
import pandas as pd
import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1) Importing Dataset

```
DrDoS_Portmap_data_2_0_per = pd.read_csv('/content/drive/My Drive/DDos_Dataset/Portmap.csv')
#UDPLag_data_2_0_per = pd.read_csv('/content/drive/My Drive/DDos_Dataset/UDPLag.csv')
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (85) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

2) Concatenating Dataset - UDPLag,Portmap data

```
#data = pd.concat([DrDoS_Portmap_data_2_0_per,UDPLag_data_2_0_per])
data = DrDoS_Portmap_data_2_0_per
```

```
data.shape
```

```
(191694, 88)
```

```
data[' Label'].value_counts()
```

```
Portmap    186960
BENIGN       4734
Name:  Label, dtype: int64
```

3) Dropping unnamed columns

```
# Drop Unnamed:0, Unnamed:0.1 columns
data = data.drop('Unnamed: 0', axis = 1)
```

```
data.columns
```

```
Index(['Flow ID', ' Source IP', ' Source Port', ' Destination IP',
       ' Destination Port', ' Protocol', ' Timestamp', ' Flow Duration',
       ' Total Fwd Packets', ' Total Backward Packets',
       'Total Length of Fwd Packets', ' Total Length of Bwd Packets',
       ' Fwd Packet Length Max', ' Fwd Packet Length Min',
       ' Fwd Packet Length Mean', ' Fwd Packet Length Std',
       'Bwd Packet Length Max', ' Bwd Packet Length Min',
       ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s',
       ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max',
       ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std',
       ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean',
       ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags',
       ' Bwd PSH Flags', ' Fwd URG Flags', ' Bwd URG Flags',
       ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s',
       ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length',
       ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance',
       'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count',
       ' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count',
       ' CWE Flag Count', ' ECE Flag Count', ' Down/Up Ratio',
       ' Average Packet Size', ' Avg Fwd Segment Size',
       ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Fwd Avg Bytes/Bulk',
       ' Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk',
       ' Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate', 'Subflow Fwd Packets',
       ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes',
       'Init_Win_bytes_forward', ' Init_Win_bytes_backward',
       ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean',
       ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std',
       ' Idle Max', ' Idle Min', 'SimillarHTTP', ' Inbound', ' Label'],
      dtype='object')
```

4) Removing Null Data

```
data_real = data.replace(np.inf, np.nan)
```

```
data_real.isnull().sum().sum()
```

```
19600
```

```
data_df = data_real.dropna(axis=0)
```

```
data_df.isnull().sum().sum()
```

```
0
```

```
data_df.isnull().sum().sum()
```

```
0
```

```
data_df
```

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	...	Active Std	Active Max	Active Min	Idle
0	192.168.50.254-224.0.0.5-0-0-0	192.168.50.254	0	224.0.0.5	0	0	2018-11-03 09:18:16.964447	114456999	45	0	...	28337.112288	98168.0	3.0	9529
1	192.168.50.253-224.0.0.5-0-0-0	192.168.50.253	0	224.0.0.5	0	0	2018-11-03 09:18:18.506537	114347504	56	0	...	121314.911865	420255.0	4.0	9493
2	172.217.10.98-192.168.50.6-443-54799-6	192.168.50.6	54799	172.217.10.98	443	6	2018-11-03 09:18:18.610576	36435473	6	2	...	0.000000	62416.0	62416.0	3637
3	172.217.7.2-192.168.50.6-443-54800-6	192.168.50.6	54800	172.217.7.2	443	6	2018-11-03 09:18:18.610579	36434705	6	2	...	0.000000	62413.0	62413.0	3637
4	172.217.10.98-192.168.50.6-443-54801-6	192.168.50.6	54801	172.217.10.98	443	6	2018-11-03 09:18:18.610581	36434626	6	2	...	0.000000	62409.0	62409.0	3637
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
191689	172.16.0.5-192.168.50.4-855-47131-17	172.16.0.5	855	192.168.50.4	47131	17	2018-11-03 10:01:48.919833	1	2	0	...	0.000000	0.0	0.0	
191690	172.16.0.5-192.168.50.4-856-53617-17	172.16.0.5	856	192.168.50.4	53617	17	2018-11-03 10:01:48.920175	1	2	0	...	0.000000	0.0	0.0	
191691	172.16.0.5-192.168.50.4-857-9612-17	172.16.0.5	857	192.168.50.4	9612	17	2018-11-03 10:01:48.920464	1	2	0	...	0.000000	0.0	0.0	
191692	172.16.0.5-192.168.50.4-858-23408-17	172.16.0.5	858	192.168.50.4	23408	17	2018-11-03 10:01:48.920466	49	2	0	...	0.000000	0.0	0.0	
191693	172.16.0.5-192.168.50.4-859-50418-17	172.16.0.5	859	192.168.50.4	50418	17	2018-11-03 10:01:48.920515	1	2	0	...	0.000000	0.0	0.0	

181894 rows × 87 columns

5) Cleaning & Refining Data

```
data_X = data_df.drop([' Label', 'SimillarHTTP'], axis = 1)
```

```
data_X.columns
```

```
Index([' Flow ID', ' Source IP', ' Source Port', ' Destination IP',
       ' Destination Port', ' Protocol', ' Timestamp', ' Flow Duration',
       ' Total Fwd Packets', ' Total Backward Packets',
       'Total Length of Fwd Packets', ' Total Length of Bwd Packets',
       ' Fwd Packet Length Max', ' Fwd Packet Length Min',
       ' Fwd Packet Length Mean', ' Fwd Packet Length Std',
       'Bwd Packet Length Max', ' Bwd Packet Length Min',
       ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s',
       ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max',
       ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std',
       ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean',
       ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags',
       ' Bwd PSH Flags', ' Fwd URG Flags', ' Bwd URG Flags',
       ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s',
       ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length',
       ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance',
       'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count',
       ' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count',
       ' CWE Flag Count', ' ECE Flag Count', ' Down/Up Ratio',
       ' Average Packet Size', ' Avg Fwd Segment Size',
       ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Fwd Avg Bytes/Bulk',
       ' Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk',
       ' Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate', 'Subflow Fwd Packets',
       ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes',
       'Init_Win_bytes_forward', ' Init_Win_bytes_backward',
       ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean',
       ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std',
       ' Idle Max', ' Idle Min', ' Inbound'],
      dtype='object')
```

```
data_X.shape
```

```
(181894, 85)
```

```
data_y = data_df[' Label']
```

```
data_y.shape
```

```
(181894,)
```

```
data_df.isnull().sum().sum()
```

```
0
```

```
data_y.unique()
```

```
array(['BENIGN', 'Portmap'], dtype=object)
```

```
data_X
```

```
data_X
```

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets
0	192.168.50.254-224.0.0.5-0-0-0	192.168.50.254	0	224.0.0.5	0	0	2018-11-03 09:18:16.964447	114456999	45	0
1	192.168.50.253-224.0.0.5-0-0-0	192.168.50.253	0	224.0.0.5	0	0	2018-11-03 09:18:18.506537	114347504	56	0
2	172.217.10.98-192.168.50.6-443-54799-6	192.168.50.6	54799	172.217.10.98	443	6	2018-11-03 09:18:18.610576	36435473	6	2
3	172.217.7.2-192.168.50.6-443-54800-6	192.168.50.6	54800	172.217.7.2	443	6	2018-11-03 09:18:18.610579	36434705	6	2
4	172.217.10.98-192.168.50.6-443-54801-6	192.168.50.6	54801	172.217.10.98	443	6	2018-11-03 09:18:18.610581	36434626	6	2
...	...	...	...	...	...	...	...	...	...	...
191689	172.16.0.5-192.168.50.4-855-47131-17	172.16.0.5	855	192.168.50.4	47131	17	2018-11-03 10:01:48.919833	1	2	0
191690	172.16.0.5-192.168.50.4-856-53617-17	172.16.0.5	856	192.168.50.4	53617	17	2018-11-03 10:01:48.920175	1	2	0
191691	172.16.0.5-192.168.50.4-857-9612-17	172.16.0.5	857	192.168.50.4	9612	17	2018-11-03 10:01:48.920464	1	2	0
191692	172.16.0.5-192.168.50.4-858-23408-17	172.16.0.5	858	192.168.50.4	23408	17	2018-11-03 10:01:48.920466	49	2	0
191693	172.16.0.5-192.168.50.4-859-50418-17	172.16.0.5	859	192.168.50.4	50418	17	2018-11-03 10:01:48.920515	1	2	0

181894 rows × 85 columns

6) Data Pre-processing : Label Encoding for the Dataset

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
data_y_trans = le.fit_transform(data_y)
```

```
data_y_trans
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

```
le_fid = LabelEncoder()
```

```
le_fid.fit(data_X['Flow ID'])
data_X['Flow ID'] = le_fid.fit_transform(data_X['Flow ID'])
```

```
le_SIP = LabelEncoder()
```

```
le_SIP.fit(data_X[' Source IP'])
data_X[' Source IP'] = le_SIP.fit_transform(data_X[' Source IP'])
```

```
le_DIP = LabelEncoder()
```

```
le_DIP.fit(data_X[' Destination IP'])
data_X[' Destination IP'] = le_DIP.fit_transform(data_X[' Destination IP'])
```

```
le_timestamp = LabelEncoder()
le_timestamp.fit(data_X[' Timestamp'])
data_X[' Timestamp'] = le_timestamp.fit_transform(data_X[' Timestamp'])
```

```
data_X
```

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	...	min_seg_size_forward	Active Mean
0	177419	90	0	124	0	0	0	114456999	45	0	...	0	8185.583333
1	177418	89	0	124	0	0	1	114347504	56	0	...	0	35028.416667
2	177151	92	54799	33	443	6	2	36435473	6	2	...	20	62416.000000
3	177340	92	54800	76	443	6	3	36434705	6	2	...	20	62413.000000
4	177152	92	54801	33	443	6	4	36434626	6	2	...	20	62409.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
191689	143559	17	855	97	47131	17	181886	1	2	0	...	20	0.000000
191690	143805	17	856	97	53617	17	181887	1	2	0	...	20	0.000000
191691	144101	17	857	97	9612	17	181888	1	2	0	...	20	0.000000
191692	144152	17	858	97	23408	17	181889	49	2	0	...	20	0.000000
191693	144492	17	859	97	50418	17	181890	1	2	0	...	20	0.000000

181894 rows × 85 columns

7) Feature Selection

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import ExtraTreesClassifier

#selecting 20 best features
# select_best= SelectKBest(chi2, k=20)
# X_feat_20 = select_best.fit_transform(data_X, data_y_trans)
# X_feat_20.shape

model = ExtraTreesClassifier(random_state=42)
model.fit(data_X, data_y_trans)
```

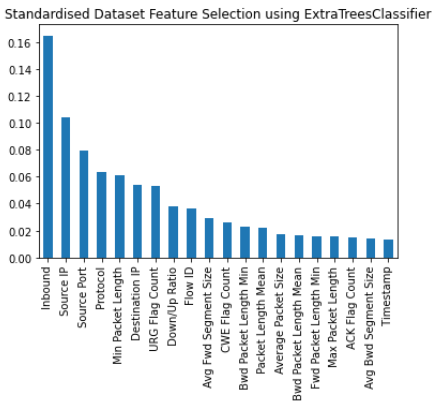
```
ExtraTreesClassifier(random_state=42)
```

```
model.feature_importances_
```

```
array([3.60737053e-02, 1.04065890e-01, 7.93933971e-02, 5.36406091e-02,
       8.95169438e-03, 6.36769526e-02, 1.33697902e-02, 3.57436439e-03,
       1.58925632e-03, 5.16725257e-04, 9.37966408e-04, 7.17894462e-05,
       6.34416746e-03, 1.57486187e-02, 5.31238967e-03, 4.01214218e-03,
       7.95025935e-03, 2.31847642e-02, 1.62275390e-02, 3.15709768e-03,
       7.05495113e-03, 4.89361362e-03, 4.67012483e-04, 5.65595996e-03,
       2.68109273e-03, 4.12374926e-04, 5.45248578e-03, 5.17978236e-04,
       4.14389502e-04, 1.82586069e-03, 3.13739527e-04, 2.20346261e-03,
       7.24387483e-04, 2.37415340e-04, 3.28031195e-04, 1.13306813e-02,
       1.13280875e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.97897621e-04, 4.21226568e-04, 5.99891153e-03, 6.77187930e-04,
       6.07867414e-02, 1.56362702e-02, 2.17667404e-02, 7.36298859e-03,
       2.69233670e-04, 0.00000000e+00, 6.32956411e-05, 8.21630379e-03,
       0.00000000e+00, 1.50308355e-02, 5.31526892e-02, 2.59537357e-02,
       0.00000000e+00, 3.76219987e-02, 1.75971274e-02, 2.89965744e-02,
       1.41223463e-02, 4.63525208e-04, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       3.71072965e-04, 4.57236076e-03, 1.05089171e-03, 9.55876167e-04,
       2.21275764e-03, 1.56218089e-03, 6.72020418e-04, 6.92684765e-04,
       1.50717516e-04, 7.92430907e-05, 2.14648002e-04, 3.00372690e-04,
       3.15689309e-04, 1.75722349e-04, 2.22685778e-03, 1.49727759e-03,
       1.64973354e-01])
```

In [ ]:
```
feature_importance_std = pd.Series(model.feature_importances_, index=data_X.columns)
feature_importance_std.nlargest(20).plot(kind='bar', title='Standardised Dataset Feature Selection using ExtraTreesClassifier')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0286f34a50>
```



In [ ]:
```
data_X.shape
```

Out[ ]:
```
(181894, 85)
```

In [ ]:
```
data_new_20features_X = data_X[[' Timestamp', ' Source Port', ' Min Packet Length', ' Fwd Packet Length Min', 'Flow ID', ' Packet Length Mean', ' Fw
```

8) Train Test Split for Normal dataset 84 Features

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_X, data_y_trans, test_size = 0.30, random_state = 42)
```

```
X_train.shape
```

```
(127325, 85)
```

```
X_test.shape
```

```
(54569, 85)
```

9) Standardization of the 84 Feature Dataset

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_std = ss.fit_transform(X_train)
X_test_std = ss.fit_transform(X_test)
```

10) Train Test Split for 20 Feature Dataset

```
from sklearn.model_selection import train_test_split
X_train_20, X_test_20, y_train_20, y_test_20 = train_test_split(data_new_20features_X, data_y_trans, test_size = 0.30, random_state = 42)
```

11) Standardization of the 20 Feature Dataset

```
from sklearn.preprocessing import StandardScaler
ss_20 = StandardScaler()
X_train_std_20 = ss_20.fit_transform(X_train_20)
X_test_std_20 = ss_20.fit_transform(X_test_20)
```

```
X_train_std_20.shape
```

```
(127325, 20)
```

```
y_train_20.shape
```

```
(127325,)
```

```
X_test_std_20.shape
```

```
(54569, 20)
```

```
y_test_20.shape
```

```
(54569,)
```

I) Random Forest Classification

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train_std_20, y_train_20)
```

```
RandomForestClassifier()
```

```
rf_y_pred = rf.predict(X_test_std_20)
```

```
rf_y_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
print("Classification Report for Random Forest: \n", classification_report(le.inverse_transfor
```

```
Classification Report for Random Forest:
               precision    recall  f1-score   support

      BENIGN       1.00      1.00      1.00      1375
     Portmap       1.00      1.00      1.00     53194

    accuracy                           1.00     54569
   macro avg       1.00      1.00      1.00     54569
weighted avg       1.00      1.00      1.00     54569
```

```
rf_conf_mat = confusion_matrix(y_test_20, rf_y_pred)
print("Random Forest Confusion: \n", rf_conf_mat)
```

```
Random Forest Confusion:
 [[ 1375     0]
 [    5 53189]]
```

```
acc_score = accuracy_score(y_test_20, rf_y_pred)
print("Accuracy Score for Random_Forest: \n", acc_score*100)
```

```
Accuracy Score for Random_Forest:
 99.99083728857043
```

II) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train_std_20, y_train_20)
```

```
DecisionTreeClassifier()
```

```
dt_y_pred = dt.predict(X_test_std_20)
```

```
print("Classification Report for Decision Tree: \n", classification_report(le.inverse_t
```

```
Classification Report for Decision Tree:
               precision    recall  f1-score   support

      BENIGN       0.92      0.75      0.83      1375
     Portmap       0.99      1.00      1.00     53194

    accuracy                           0.99     54569
   macro avg       0.96      0.88      0.91     54569
weighted avg       0.99      0.99      0.99     54569
```

```
dt_conf_mat = confusion_matrix(y_test_20, dt_y_pred)
print("Decision Tree Confusion: \n", dt_conf_mat)
```

```
Decision Tree Confusion:
 [[ 1037   338]
 [   89 53105]]
```

```
acc_score_dt = accuracy_score(y_test_20, dt_y_pred)
print("Accuracy Score for Decision Tree: \n", acc_score_dt*100)
```

```
Accuracy Score for Decision Tree:
 99.21750444391503
```

III) SVM

```
from sklearn.svm import LinearSVC
```

```
svm = LinearSVC(multi_class = 'ovr')
svm.fit(X_train_std_20, y_train_20)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning: Liblinear failed to co
s.
  ConvergenceWarning,
LinearSVC()
```

```
y_pred_svm = svm.predict(X_test_std_20)
```

```
svm.score(X_test_std_20, y_test_20)
```

```
0.999450237314226
```

```
print("Classification Report for Random Forest: \n", classification_report(le.inverse_transform(y_test_20),
```

```
Classification Report for Random Forest:
              precision    recall  f1-score   support

      BENIGN       0.98      0.99      0.99      1375
     Portmap       1.00      1.00      1.00     53194

    accuracy                           1.00     54569
   macro avg       0.99      1.00      0.99     54569
weighted avg       1.00      1.00      1.00     54569
```

```
svm_conf_mat = confusion_matrix(y_test_20, y_pred_svm)
print("SVM Confusion Matrix: \n", svm_conf_mat)
```

```
SVM Confusion Matrix:
 [[ 1366     9]
 [   21 53173]]
```

```
acc_score_svm = accuracy_score(y_test_20, y_pred_svm)
print("Accuracy Score for SVM: \n", acc_score_svm*100)
```

```
Accuracy Score for SVM:
 99.9450237314226
```

IV) Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB()
gnb.fit(X_train_std_20, y_train_20)
gnb_y_pred = gnb.predict(X_test_std_20)
```

```
print("Classification Report for Naive Bayes: \n", classification_report(le.inverse_transform
```

```
Classification Report for Naive Bayes:
               precision    recall  f1-score   support

      BENIGN       0.82      1.00      0.90      1375
     Portmap       1.00      0.99      1.00     53194

    accuracy                           0.99     54569
   macro avg       0.91      1.00      0.95     54569
weighted avg       1.00      0.99      0.99     54569
```

```
gnb_conf_mat = confusion_matrix(y_test_20, gnb_y_pred)
print("Naive Bayes Confusion Matrix: \n", gnb_conf_mat)
```

```
Naive Bayes Confusion Matrix:
 [[ 1375     0]
 [  302 52892]]
```

```
acc_score_gnb = accuracy_score(y_test_20, gnb_y_pred)
print("Accuracy Score for Naive: \n", acc_score_gnb*100)
```

```
Accuracy Score for Naive:
 99.4465722296542
```

V) MLP

```
from __future__ import print_function
import pandas as pd
import numpy as np
np.random.seed(1337)  # for reproducibility
from keras.preprocessing import sequence
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding
from keras.layers import LSTM, SimpleRNN, GRU
from keras.datasets import imdb
from keras.utils.np_utils import to_categorical
from sklearn.metrics import (precision_score, recall_score,f1_score, accuracy_score,mean_squared_error,mean_absolute_error
from sklearn import metrics
from sklearn.preprocessing import Normalizer
import h5py
from keras import callbacks
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
```

```
y_train_MLP_20 = np.array(y_train_20)
y_test_MLP_20 = np.array(y_test_20)

y_train_MLP_onehot_20 = to_categorical(y_train_MLP_20)
y_test_MLP_onehot_20 = to_categorical(y_test_MLP_20)

X_train_20_MLP = np.array(X_train_std_20)
X_test_20_MLP = np.array(X_test_std_20)
```

```
batch_size = 1000

# 1. define the network
model = Sequential()
model.add(Dense(1024,input_dim=20,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(2024,activation='relu'))
model.add(Dropout(0.01))
# model.add(Dense(3024,activation='relu'))
# model.add(Dropout(0.01))
# model.add(Dense(2500,activation='relu'))
# model.add(Dropout(0.01))
model.add(Dense(2000,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(1000,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(500,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(200,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(2))
model.add(Activation('softmax'))
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
```

```
Epoch 1/3
128/128 [==============================] - 126s 980ms/step - loss: 0.6719 - accuracy: 0.9613 - val_loss: 0.6543 -
Epoch 2/3
128/128 [==============================] - 125s 975ms/step - loss: 0.6370 - accuracy: 0.9739 - val_loss: 0.6166 -
Epoch 3/3
128/128 [==============================] - 125s 975ms/step - loss: 0.5975 - accuracy: 0.9739 - val_loss: 0.5739 -
<keras.callbacks.History at 0x7f0281490610>
```

```
#y_pred_MLP = model.predict_classes(X_test_20_MLP)
y_pred_MLP=np.argmax(model.predict(X_test_20_MLP), axis=-1)
```

```
y_pred_MLP
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
print("Classification Report for MLP: \n", classification_report(le.inverse_transform(y_test_MLP_20), le.inverse
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precisior
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precisior
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report for MLP:
              precision    recall  f1-score   support

      BENIGN       0.00      0.00      0.00      1375
     Portmap       0.97      1.00      0.99     53194

    accuracy                           0.97     54569
   macro avg       0.49      0.50      0.49     54569
weighted avg       0.95      0.97      0.96     54569
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precisior
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
mlp_conf_mat = confusion_matrix(y_test_20, y_pred_MLP)
print("MLP Confusion: \n", mlp_conf_mat)
```

```
MLP Confusion:
 [[    0  1375]
 [    0 53194]]
```

```
acc_score_mlp = accuracy_score(y_test_20, y_pred_MLP)
print("Accuracy Score for MLP: \n", acc_score_mlp*100)
```

```
Accuracy Score for MLP:
 97.4802543568693
```

VI) LSTM

```
y_train_lstm_20 = np.array(y_train_20)
y_test_lstm_20 = np.array(y_test_20)

y_train_onehot_lstm = to_categorical(y_train_lstm_20)
y_test_one_hot_lstm = to_categorical(y_test_lstm_20)

X_train_lstm_20 = np.array(X_train_std_20)
X_test_lstm_20 = np.array(X_test_std_20)
```

```
X_test_std_20
```

```
array([[-1.04377087, -0.18496365,  0.0995201 , ..., -0.01660308,
        -0.56148747, -0.04620849],
       [ 0.3519855 , -0.1741018 ,  0.49117211, ...,  0.05117229,
        -0.64672949, -0.04620849],
       [-1.40899254, -0.16994194,  0.0995201 , ..., -0.01660308,
         1.19243953, -0.04620849],
       ...,
       [-1.31325075, -0.16994194,  0.0995201 , ..., -0.01660308,
         1.5861149 , -0.04620849],
       [-0.53445045, -0.16994194, -0.29213192, ..., -0.08437845,
         0.73255675, -0.04620849],
       [ 1.08723215, -0.18057269,  0.0995201 , ..., -0.01660308,
        -0.18177772, -0.04620849]])
```

```
X_train_lstm_20.shape[0]
```

```
127325
```

```
X_train_lstm_reshape = np.reshape(X_train_std_20, (X_train_lstm_20.shape[0], 1,  X_train_lstm_20.shape[1]))
X_test_lstm_reshape = np.reshape(X_test_std_20, (X_test_lstm_20.shape[0], 1, X_test_lstm_20.shape[1]))
```

```
batch_size = 1000

# Initialize the network
model_LSTM = Sequential()
model_LSTM.add(LSTM(8,input_dim=20, return_sequences=True))
model_LSTM.add(Dropout(0.1))
model_LSTM.add(LSTM(8,input_dim=20, return_sequences=False))
model_LSTM.add(Dropout(0.1))
model_LSTM.add(Dense(2))
model_LSTM.add(Activation('softmax'))
```

```
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
```

```
#model_LSTM.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model_LSTM.compile(loss="categorical_crossentropy",optimizer='adadelta',metrics=['accuracy'])
model_LSTM.fit(X_train_lstm_reshape, y_train_onehot_lstm, validation_data=(X_test_lstm_reshape, y_test_one_hot_lstm),batch_size=batch_size, epochs=3
```

```
Epoch 1/3
128/128 [==============================] - 5s 12ms/step - loss: 0.6890 - accuracy: 0.5631 - val_loss: 0.6895 - val_accuracy: 0.5359
Epoch 2/3
128/128 [==============================] - 1s 6ms/step - loss: 0.6886 - accuracy: 0.5739 - val_loss: 0.6890 - val_accuracy: 0.5551
Epoch 3/3
128/128 [==============================] - 1s 5ms/step - loss: 0.6880 - accuracy: 0.5891 - val_loss: 0.6884 - val_accuracy: 0.5862
<keras.callbacks.History at 0x7f027ea0c110>
```

```
#y_perd_lstm = model_LSTM.predict_classes(X_test_lstm_reshape)
y_perd_lstm=np.argmax(model.predict(X_test_std_20), axis=-1)
```

```
#y_perd_lstm = model_LSTM.predict_classes(X_test_lstm_reshape)
y_perd_lstm=np.argmax(model.predict(X_test_std_20), axis=-1)
```

```
print("Classification Report for LSTM: \n", classification_report(le.inverse_transform(y_test_lstm_20),
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report for LSTM:
               precision    recall  f1-score   support

      BENIGN       0.00      0.00      0.00      1375
     Portmap       0.97      1.00      0.99     53194

    accuracy                           0.97     54569
   macro avg       0.49      0.50      0.49     54569
weighted avg       0.95      0.97      0.96     54569
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
  _warn_prf(average, modifier, msg_start, len(result))
```

```
lstm_conf_mat = confusion_matrix(y_test_lstm_20, y_perd_lstm)
print("LSTM Confusion: \n", lstm_conf_mat)
```

```
LSTM Confusion:
 [[    0  1375]
 [    0 53194]]
```

```
acc_score_lstm = accuracy_score(y_test_lstm_20, y_perd_lstm)
print("Accuracy Score for LSTM: \n", acc_score_lstm*100)
```

```
Accuracy Score for LSTM:
 97.4802543568693
```

VII) XGBoost

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
# fit model no training data
gradinet_boost = GradientBoostingClassifier()
gradinet_boost.fit(X_train_std_20, y_train_20)
```

```
GradientBoostingClassifier()
```

```
# Predict the labels
y_pred_xgboost = gradinet_boost.predict(X_test_std_20)
```

```
y_pred_xgboost
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
y_test_20
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
# Accuracy Score
print("Accuracy Score for the XGBoost Classifier is: {0:.3f}%".format(accuracy_score(y_test_20,
```

```
Accuracy Score for the XGBoost Classifier is: 99.218%
```

```
# Classification Report
print("Classification Report for XGBOOST: \n", classification_report(le.inverse_transform(y_tes
```

```
Classification Report for XGBOOST:
               precision    recall  f1-score   support

      BENIGN       0.92      0.75      0.83      1375
     Portmap       0.99      1.00      1.00     53194

    accuracy                           0.99     54569
   macro avg       0.96      0.88      0.91     54569
weighted avg       0.99      0.99      0.99     54569
```

VIII) Ensemble Method of Machine Learning

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
```

```
# ADABOOST
adaboost = AdaBoostClassifier(base_estimator= dt, n_estimators=100)
```

```
adaboost.fit(X_train_std_20, y_train_20)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100)
```

```
y_pred_adaboost = adaboost.predict(X_test_std_20)
```

```
print("Accuracy Score for Adaboost: ", accuracy_score(y_test_20, y_pred_adaboost))
```

```
Accuracy Score for Adaboost:  0.9921750444391504
```

```
print("Classification Report for Adaboost: ",classification_report(le.inverse_transform(y_test_20), le.inverse_
```

```
Classification Report for Adaboost:               precision    recall  f1-score   support

      BENIGN       0.92      0.75      0.83      1375
     Portmap       0.99      1.00      1.00     53194

    accuracy                           0.99     54569
   macro avg       0.96      0.88      0.91     54569
weighted avg       0.99      0.99      0.99     54569
```

```
# Confusion Matrix
adaboost_conf_mat = confusion_matrix(y_test_20, y_pred_adaboost)
print("Adaboost Confusion: \n", adaboost_conf_mat)
```

```
Adaboost Confusion:
 [[ 1037   338]
 [   89 53105]]
```

```
# Generating output for each of the classifier for Comparision with Ensemble learning
clf_labels = ['Decision Tree', 'SVM']
```

Majority Voting Technique for Ensemble classification

```
from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.base import clone
from sklearn.pipeline import _name_estimators
import numpy as np
import operator
class MajorityVoteClassifier(BaseEstimator,
                             ClassifierMixin):
    """ A majority vote ensemble classifier

    Parameters
    ----------
    classifiers : array-like, shape = [n_classifiers]
      Different classifiers for the ensemble

    vote : str, {'classlabel', 'probability'}
      Default: 'classlabel'
      If 'classlabel' the prediction is based on
      the argmax of class labels. Else if
      'probability', the argmax of the sum of
      probabilities is used to predict the class label
      (recommended for calibrated classifiers).

    weights : array-like, shape = [n_classifiers]
      Optional, default: None
      If a list of `int` or `float` values are
      provided, the classifiers are weighted by
      importance; Uses uniform weights if `weights=None`.

    """
    def __init__(self, classifiers,
                 vote='classlabel', weights=None):


        self.classifiers = classifiers
        self.named_classifiers = {key: value for
                                  key, value in
                                  _name_estimators(classifiers)}
        self.vote = vote
        self.weights = weights

    def fit(self, X, y):
        """ Fit classifiers.

        Parameters
        ----------
        X : {array-like, sparse matrix},
            shape = [n_examples, n_features]
            Matrix of training examples.

        y : array-like, shape = [n_examples]
```

```
s.
  ConvergenceWarning,
Accuracy Score: 1.00 (+/- 0.00) [Majority Voting]
```

Results and Discussion

The given implementation include classifier algorithms like Random forest, Decision tree, Support Vector Machine, Naive Bayes, Multi-Layer Perceptron(feed-forward artificial neural network), and Long short-term memory model(artificial recurrent neural network) along with Advance ensemble classifier techniques such as XG Boost(Extreme Gradient Boosting), Ada Boost(Adaptive Boosting), & Majority Voting classifier.

Dataset 1: [DDoS 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB]

Result 1: Without Concatenation using DDOS Portmap Dataset

ALGORITHM	Accuracy Score
Random Forest Classifier	99.99083728857043
Decision tree learning	99.21750444391503
Support-vector machines	99.9450237314226
Naive Bayes classifier	99.4465722296542
Multi-Layer Perceptron	97.4802543568693
Long short-term memory	97.4802543568693
XgBoost Extreme Gradient Boosting	99.218
AdaBoost	99.21750444391504

Result 2: With Concatenation using DDOS Portmap & UDPLagDataset

ALGORITHM	Accuracy Score
Random Forest Classifier	99.89685023432513
Decision tree learning	99.69405390256433
Support-vector machines	99.67420243822691
Naive Bayes classifier	71.60773506469242
Multi-Layer Perceptron	98.60689429679107
Long short-term memory	98.60689429679107
XgBoost Extreme Gradient Boosting	99.659
AdaBoost	99.69444314696312

Result 3: With Concatenation using 5% samples of LDAP, MSSQL, NetBIOS, Portmap, Syn, UDP, UDPLag, Dataset 2
https://www.kaggle.com/datasets/manmandes/ddos2019-5percent

ALGORITHM	Accuracy Score
Random Forest Classifier	99.6781990052804
Decision tree learning	99.24390401876263
Support-vector machines	97.54694919703152
Naive Bayes classifier	97.33286972173077
Multi-Layer Perceptron	89.18012333432192
Long short-term memory	89.18012333432192
XgBoost Extreme Gradient Boosting	99.149
AdaBoost	99.23504085577247

Result 4: Using Mixed with 12 types of Dataset 3
https://www.kaggle.com/datasets/pedrohauy/sampledtftpattackcicddos2019

ALGORITHM	Accuracy Score
Random Forest Classifier	99.98294068476092
Decision tree learning	99.93176273904365
Support-vector machines	99.98805847933264
Naive Bayes classifier	99.981234753237
Multi-Layer Perceptron	99.87205513570684
Long short-term memory	99.87205513570684
XgBoost Extreme Gradient Boosting	99.932
AdaBoost	99.93176273904365

Conclusion & Future Enhancement

In this study we have performed DDoS detection using several models like Random forest, Decision tree, SVMe, Naive Bayes, MLP, and LSTM model along with Advance ensemble classifier techniques such as XG Boost, Ada Boost, & Majority Voting classifier. We executed the code for four different dataset based on attack vectors and obtained different results. Random forest has shown highest accuracy for Portmap data but as we concatenate different attack vector data the accuracy of random forest steadily declines.
For the Dataset combining real time data with mixed vector XgBoost Extreme Gradient Boosting has shown highest accuracy. In the future we Aim to develop ready to deploy DDoS Detection System using the algorithm with highest accuracy. We would also like to work on automated DDos Detectors & use new Machine learning techniques to develop better models.

References :

[1] E. Balkanli, J. Alves and A. N. Zincir-Heywood, "Supervised learning to detect DDoS attacks," 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), 2014, pp. 1-8, doi: 10.1109/CICYBS.2014.7013367.

[2] Li L., Zhou J., Xiao N. (2007) DDoS Attack Detection Algorithms Based on Entropy Computing. In: Qing S., Imai H., Wang G. (eds) Information and Communications Security. ICICS 2007. Lecture Notes in Computer Science, vol 4861. Springer, Berlin, Heidelberg.

[4] Indraneel Sreeram, Venkata Praveen Kumar Vuppala, HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm,Applied Computing and Informatics,Volume 15, Issue 1,2019,Pages 59-66,ISSN 2210- 8327,

[5] M. H. Aysa, A. A. Ibrahim and A. H. Mohammed, "IoT Ddos Attack Detection Using Machine Learning," 2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2020, pp. 1-7, doi: 10.1109/ISMSIT50672.2020.9254703.

[6] S. Wankhede and D. Kshirsagar, "DoS Attack Detection Using Machine Learning and Neural Network," 2018 Fourth International Conference on Coputing Communication Control and Automation (ICCUBEA), 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697702.

[7] Pande S., Khamparia A., Gupta D., Thanh D.N.H. (2021) DDOS Detection Using Machine Learning Technique. In: Khanna A., Singh A.K., Swaroop A. (eds) Recent Studies on Computational Intelligence. Studies in Computational Intelligence, vol 921. Springer, Singapore.

[8] Li Q., Meng L., Zhang Y., Yan J. (2019) DDoS Attacks Detection Using Machine Learning Algorithms. In: Zhai G., Zhou J., An P., Yang X. (eds) Digital TV and Multimedia Communication. IFTC 2018. Communications in Computer and Information Science, vol 1009. Springer, Singapore.

[9] M. H. Aysa, A. A. Ibrahim and A. H. Mohammed, "IoT Ddos Attack Detection Using Machine Learning," 2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2020, pp. 1-7, doi:

[10] M. Zekri, S. E. Kafhali, N. Aboutabit and Y. Saadi, "DDoS attack detection using machine learning techniques in cloud computing environments," 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech), 2017, pp. 1-7, doi: 10.1109/CloudTech.2017.8284731.

[11] X. Yuan, C. Li and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," 2017 IEEE International Conference on Smart Computing (SMARTCOMP), 2017, pp. 1-8, doi: 10.1109/SMARTCOMP.2017.7946998.

[12] H. Zhang, P. Cheng, L. Shi and J. Chen, "Optimal DoS Attack Scheduling in Wireless Networked Control System," in IEEE Transactions on Control Systems Technology, vol. 24, no. 3, pp. 843-852, May 2016, doi: 10.1109/TCST.2015.2462741.

[13] Zhu Jian-Qi, Fu Feng, Yin Ke-xin, Liu Yan-Heng,Dynamic entropy based DoS attack detection method,Computers & Electrical Engineering, Volume 39, Issue 7,2013,Pages 2243-2251,ISSN 0045-7906.

[14] A. Agah, K. Basu and S. K. Das, "Preventing DoS attack in sensor networks: a game theoretic approach," IEEE International Conference on Communications, 2005. ICC 2005. 2005, 2005, pp. 3218-3222 Vol. 5, doi: 10.1109/ICC.2005.1495019.

[15] A. A. Y. R. Fares, F. L. de Caldas Filho, W. F. Giozza, E. D. Canedo, F. L. Lopes de Mendonça and G. D. Amvame Nze, "DoS Attack Prevention on IPS SDN Networks," 2019 Workshop on Communication Networks and Power Systems (WCNPS), 2019, pp. 1-7, doi: 10.1109/WCNPS.2019.8896233.

[16] Priyanka Sharma, Rachana Sharma, Emmanuel S. Pilli, and Anand Kumar Mishra. 2015. A Detection Algorithm for DoS Attack in the Cloud Environment. In Proceedings of the 8th Annual ACM India Conference (Compute '15). Association for Computing Machinery, New York, NY, USA, 107–110.

[17] A. Elhigazi, S. A. Razak, M. Hamdan, B. Mohammed, I. Abaker and A. Elsafi, "Authentication Flooding DOS Attack Detection and Prevention in 802.11," 2020 IEEE Student Conference on Research and Development (SCOReD), 2020, pp. 325-329, doi: 10.1109/SCOReD50371.2020.9250990

[18] D. S. N. Mary and A. T. Begum, "An Algorithm for Moderating DoS Attack in Web Based Application," 2017 International Conference on Technical Advancements in Computers and Communications (ICTACC), 2017, pp. 26-31, doi: 10.1109/ICTACC.2017.17

[19] C. Barna, M. Shtern, M. Smit, V. Tzerpos and M. Litoiu, "Model-based adaptive DoS attack mitigation," 2012 7th International Symposium on Software Engineering for Adaptive and SelfManaging Systems (SEAMS), 2012, pp. 119-128, doi: 10.1109/SEAMS.2012.6224398.

[20] Enrico Cambiaso, Gianluca Papaleo, Maurizio Aiello,Slowcomm: Design, development and performance evaluation of a new slow DoS attack,Journal of Information Security and Applications,Volume 35,2017,Pages 23-31,ISSN 2214- 2126,https://doi.org/10.1016/j.jisa.2017.05.005.

[21] Meng Wang, Yiqin Lu, Jiancheng Qin,A dynamic MLP-based DDoS attack detection method using feature selection and feedback, Computers & Security,Volume 88, 2020,101645,ISSN 0167-4048,https://doi.org/10.1016/j.cose.2019.101645.

[22] Sree, T. Raja; Bhanu, S. Mary Saira (2017). Detection of HTTP Flooding Attacks in Cloud Using Dynamic Entropy Method. Arabian Journal for Science and Engineering, (), –. doi:10.1007/s13369-017-2939-7 Sree, T. Raja; Bhanu, S. Mary Saira (2017). Detection of HTTP Flooding Attacks in Cloud Using Dynamic Entropy Method. Arabian Journal for Science and Engineering, (), –. doi:10.1007/s13369-017-2939-7

[23] Yusof M.A.M., Ali F.H.M., Darus M.Y. (2018) Detection and Defense Algorithms of Different Types of DDoS Attacks Using Machine Learning. In: Alfred R., Iida H., Ag. Ibrahim A., Lim Y. (eds) Computational Science and Technology. ICCST 2017. Lecture Notes in Electrical Engineering, vol 488. Springer, Singapore.

[24] Zekri, Marwane & El Kafhali, Said & Aboutabit, Noureddine & Saadi, Youssef. (2017). DDoS attack detection using machine learning techniques in cloud computing environments. 1-7. 10.1109/CloudTech.2017.8284731.

[25] B. S. Kiruthika Devi, G. Preetha, G. Selvaram and S. Mercy Shalinie, "An impact analysis: Real time DDoS attack detection and mitigation using machine learning," 2014 International Conference on Recent Trends in Information Technology, 2014, pp. 1-7, doi: 10.1109/ICRTIT.2014.6996133.

[26] Nisha Ahuja, Gaurav Singal, Debajyoti Mukhopadhyay, Neeraj Kumar,Automated DDOS attack detection in software defined networking,Journal of Network and Computer Applications,Volume 187,2021, 103108,ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2021.103108.

[27] M. Hosseinpour, S. A. Hosseini Seno, M. H. Yaghmaee Moghaddam and H. Khosravi Roshkhari, "An anomaly based VoIP DoS attack detection and prevention method using fuzzy logic," 2016 8th International Symposium on Telecommunications (IST), 2016, pp. 713-718, doi: 10.1109/ISTEL.2016.7881916.

[29] Nisha Ahuja, Gaurav Singal, Debajyoti Mukhopadhyay, Neeraj Kumar,Automated DDOS attack detection in software defined networking,Journal of Network and Computer Applications,Volume 187,2021, 103108,ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2021.103108

[30] M. Hosseinpour, S. A. Hosseini Seno, M. H. Yaghmaee Moghaddam and H. Khosravi Roshkhari, "An anomaly based VoIP DoS attack detection and prevention method using fuzzy logic," 2016 8th International Symposium on Telecommunications (IST), 2016, pp. 713-718, doi: 10.1109/ISTEL.2016.7881916