Georgia State University

Computer Science Master's Project Report

TCP SYN Flood DDoS Attack Detection and Prevention using Machine Learning

Shyama Bhuvanendran Sheela

December 2018

Advisor: Dr. Wei Li

Committee member: Dr. Zhisheng Yan

Table of Contents

1 Abstract

Transmission Control Protocol (TCP) Synchronize packet flood attacks, commonly known as TCP SYN Flood attacks, are one of the most commonly used Distributed Denial of Service (DDoS) attacks. The attacker sends continuous data requests to exhaust the target's resources thus flooding the server. Flood attacks can consume all the server resources, making it unavailable for legitimate requests. The ultimate goal is to crash the target's servers and disrupt the business. In DDoS attacks, the data requests to flood the target server will come from multiple sources. This makes the attack difficult to stop by simply blocking a single source.

The main aim of the proposed project is to detect possible SYN Flood attacks and prevent them using supervised learning classification methods. The incoming packets are continuously monitored and classified using a classifier to determine if they are safe to be forwarded or not. The packets are forwarded only if they are tagged as safe. If the attack is identified to be launched from a single source, the attack is detected and source IP is blocked permanently thus preventing future attacks. In case of attacks launched from multiple sources, the 'unsafe' packets are not forwarded to the server. They are dropped mid-way which keeps the server safe from flooding attacks.

Key words: DDoS, SYN Flood, Classification, Machine Learning

2 Introduction

A Distributed Denial of Service (DDoS) attack is a malicious attempt to take down a target server by overwhelming its resources. The attacker uses compromised machines as botnets or zombies to launch the attack simultaneously from multiple sources. DDoS attacks are difficult to detect and prevent as they appear as normal traffic coming from multiple sources and blocking a single source wouldn't prevent it. These attacks will impact the performance and result in unavailability of services for legitimate users.

One of the most common types of DDoS attack is the Network Layer Attacks. These are DDoS assaults set up to clog the 'pipelines' connecting the network. These includes UDP flood, SYN flood, NTP amplification, and more. The largest network layer assaults can exceed 200 Gbps; however, 20 to 40 Gbps is enough to completely shut down most network infrastructures.

SYN Flood attack is classified as a semantic attack as it exploits the three-way handshake feature of Transmission Control Protocol (TCP). When a client wants to establish a connection with the server, the client does so by using the TCP three-way handshake. The three-way handshake includes three steps – (i) Client requests connection by sending SYN (synchronize) message to the server. (ii) Server acknowledges by sending SYN-ACK (synchronize-acknowledge) message back to the client. (iii) The client responds with an ACK (acknowledge) message, and then the connection is established.

To launch a TCP SYN flood attack, the attacker sends repeated SYN packets to every port on the targeted server. The server responds to each attempt with a SYN-ACK packet from each open port. The malicious client does not send the expected ACK. The server under attack will wait for acknowledgement of its SYN-ACK packet

for some time. During this time, the server cannot close down the connection by sending an RST packet. The server keeps a backlog queue in its system memory to maintain all half-open connections till the connections time out. Before the connection can time out, the attacker sends another SYN packet. Once the backlog queue limit is reached, all future SYN requests will be dropped.

In the proposed method, Machine Learning Classifiers are used to identify ‘safe’ and ‘unsafe’ packets. Statistical features extracted from batches of incoming packets are used to train the model. The model could detect simple SYN Flood attacks as well as spoofed SYN attacks using multiple random source IPs with almost 100% accuracy. All the ‘safe’ packets are forwarded and ‘unsafe’ packets are dropped once an attack has been detected.

3 Dataset Generation

To collect data to train and validate the model, a network topology was created using Mininet, Openflow switches and Floodlight controller deployed in Ubuntu on Virtual Box.

3.1 Network Architecture

The network topology consists of five hosts (Host_1, Host_2, Host_3, Host_4 and Host_5) as shown in Fig 1. Host_5 is the target host which runs an HTTP web server to handle requests. Host_1 and Host_2 generate normal traffic to Host_5. Host_3 and Host_4 are bot hosts which attack the target server Host_5. SYN Flood attack from Host_3 and Host_4 to Host_5 is generated using the attack tool HPING3.

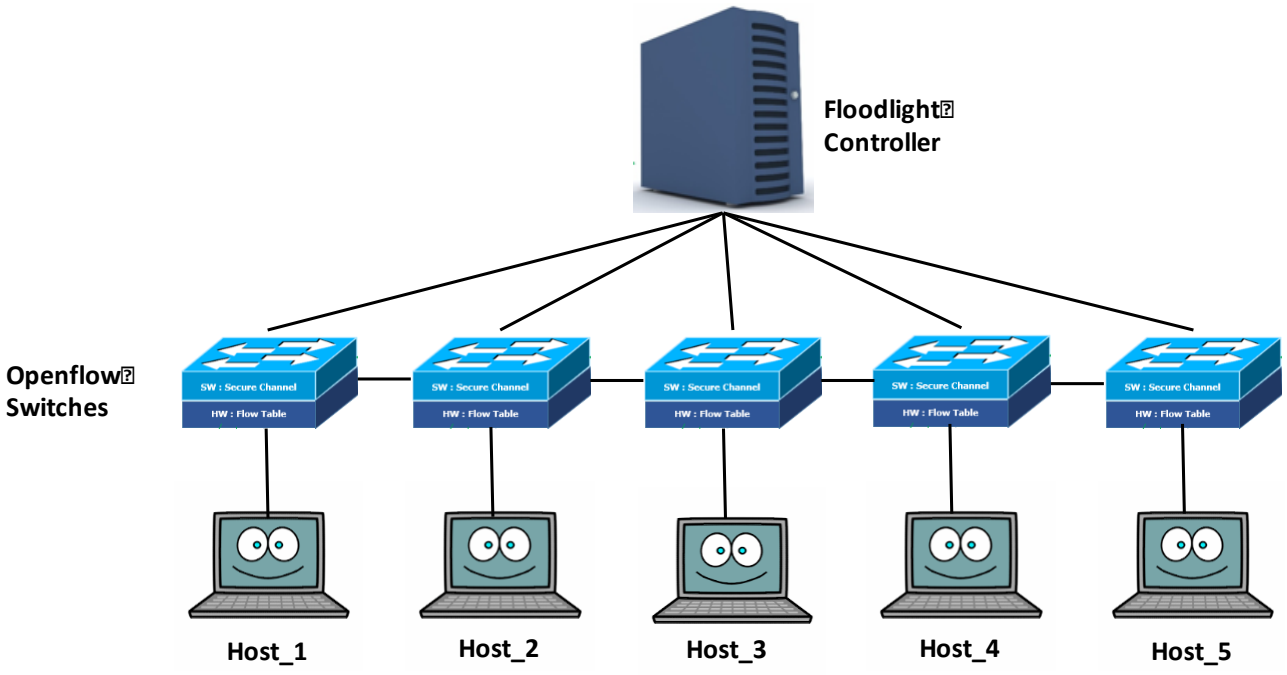


Fig 1. Network topology diagram implemented using Mininet and Floodlight

a. Mininet

Mininet is a network emulator that creates realistic networks of virtual hosts, links, controllers and switches. Mininet is an Open Source project originally created by Bob Lantz and Brandon Heller based on a prototype demonstrated by Bob Lantz. The virtual hosts run on Linux software and switches use Openflow. Mininet supports the development, testing and research on networking by creating a complete virtual network on a single PC or Laptop using a single command. Mininet is written almost entirely on Python and it provides extensible Python API for network customization and experimentation. A mininet host run just like any Linux server. You can SSH into it and run any application on the mininet hosts. Mininet can be used to test the connectivity and performance of a virtual network.

In this project, mininet is used to create a custom topology with five hosts connected to five switches and one remote controller. SSH daemons are started on every host to SSH into the created hosts. Once we SSH into these five virtual servers, normal packets can be sent from Host_1 and Host_2, attacks can be launched from Host_3 and Host_4 and incoming packets to Host_5 can be captured.

b. Openflow Switch

Openflow switch is a hardware device or software program which forwards packets in a network using Openflow communications protocol. Openflow protocol enables a network controller to control the flow of packets in a network of switches. The switch communicates with the controller and the controller can add, delete and update flow entries in the switch. The controller can also add some packet manipulation when changing the flow tables of the switch.

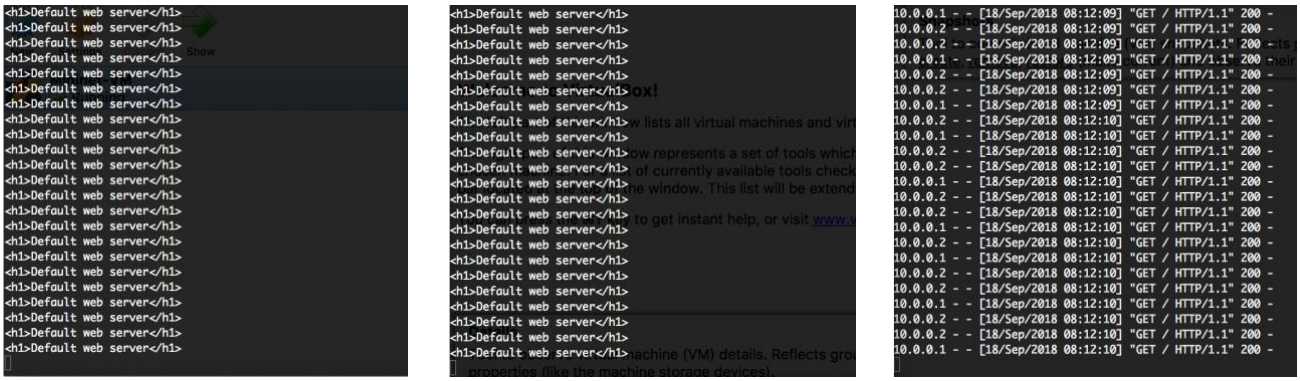
In the topology mentioned above, five Openflow switches are connected to five hosts and one controller. The controller is used to program the switches to drop and forward packets when a flood attack is detected.

c. Floodlight Controller

Floodlight controller is a Software Defined Network (SDN) controller using Openflow communications protocol to control how traffic is handled in an SDN. Floodlight controller is an open source project and is written in Java. The main responsibility of a controller is to maintain all network rules and provide instructions to the underlying switches. Floodlight has REST APIs which makes it easier to program interfaces and the Floodlight websites also has tutorials to help developers to customize the controller according to their needs. In this project, the floodlight controller is used to manage the packets, once an attack has been identified.

3.2 Normal Traffic Generation

To generate normal traffic and to observe the response time of the target host (Host_5), a simple HTTP webserver is run on port 18080 of Host_5. When the webserver receives a request, it replies with a '*Default web server*' message. Host_1 and Host_2 sends HTTP requests to the webserver using curl commands. The webserver replies with the default message as expected (shown in Fig 2) when there is no attack. Tcpdump packet capture tool is used to capture and save the normal traffic in PCAP file format.



a. Host_1 sending http b. Host_2 sending http c. Host_5 sending http

Fig 2. Normal Traffic Generation – Host_1 and Host_2 sends requests to Host_5

3.3 Flood Attack

After running the normal traffic on the simulated network for a few minutes, Host_3 and Host_4 started attacking Host_5 using Hping3 attacking tool.

a. Hping3

Hping is a free command-line packet generator and analyzer. Hping3 is a newer version of hping which is scriptable using Tcl language. It supports TCP, ICMP, UDP and RAW-IP protocols. It is mainly used as a security tool to test Firewall and Network security.

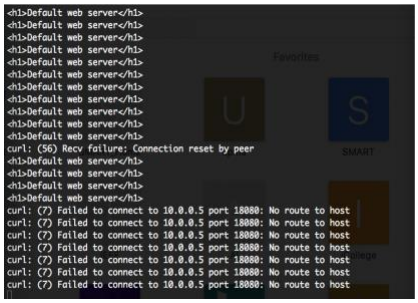
In this project, hping3 is used to launch attack from Host_3 and Host_4 to Host_5 (as shown in Fig 3). The command used to generate SYN Flood attack packets on Host_3 is below:

hping3 -i u1 -S -p 18080 --flood 10.0.0.5

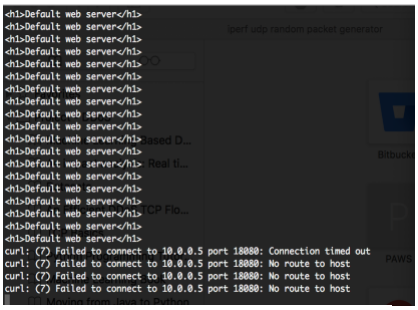
And the below command was used to generate SYN Flood attack packets with IP spoofing on Host_4:

hping3 -S -p 18080 --flood --rand-source 10.0.0.5

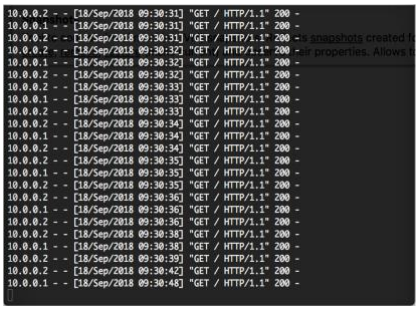
After sending attack packets from Host_3 and Host_4 to Host_5, the Host_5 stopped responding to the requests from Host_1 and Host_2. '*Connection timed out*' and '*No route to host*' errors appeared on Host_1 and Host_2 terminals as seen on fig. 3 below.



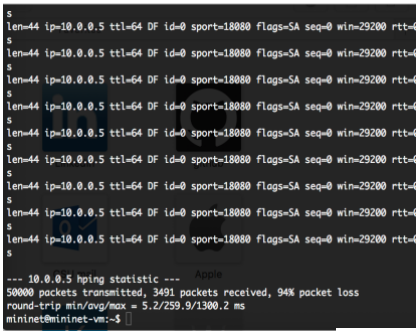
a. Host_1 lost connection



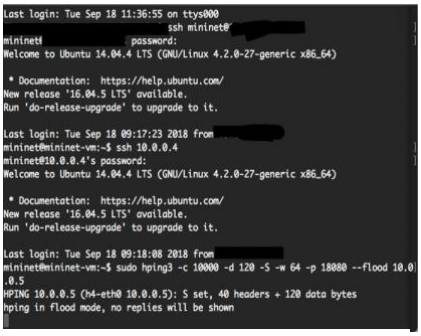
b. Host 2 lost connection



c. Host_5 stopped sending replies



d. Host_3 attacking Host_5



e. Host_4 attacking Host_5

Fig 3. Flood attack launched from Host_3 and Host_4 takes down the target Host_5

3.4 Packet Data Capture

Once the normal traffic packets and attack packets are sent, the next step is to capture these packets. The packets thus captured are used to train and validate the models.

a. Tcpdump

Tcpdump is a command-line packet analyzer. It is used in this project to capture required packets. The captured packets are saved in a pcap file format. Since we only need TCP packets, the tcpdump command is customized to capture only TCP packets.

tcpdump -w data_1.pcap -i h5-eth0 tcp

The above command captures TCP packets and writes them to a file '*data_1.pcap*'.

b. PCAP to CSV conversion using Scapy

The packets captured by tcpdump is saved in a pcap file format. Pcap files can be opened and read using Wireshark. A python script is used to convert the captured packets in PCAP format to CSV format using *rdpcap()* method of Scapy. Scapy is a powerful python library for packet manipulation. It supports a wide number of protocols and can create, send and capture packets.

c. Assigning Labels - safe or unsafe

Now, we have the captured packet data containing normal and attack traffic in a CSV file. A new column *'Attack'* is added to the CSV file which takes values 0 or 1.

Attack values:

0 – Safe/Normal packets

1 – Unsafe/Attack packets

The normal traffic was generated from Host_1 and Host_2 corresponding to IPs 10.0.0.1 and 10.0.0.2. Hence, all the packets with source addresses corresponding to Host_1, Host_2 and Host_5 are given the value 0 for *'Attack'*. All the other packets which has the source IPs corresponding to Host_3 and Host_4 and also other spoofed source IPs are given the value 1 for *'Attack'*.

4 Proposed New Method

Feature Engineering is an essential part of the Machine Learning pipeline. It comes under the Data Preparation part of creating a Machine Learning Model. Feature Engineering is the method of using domain knowledge of the data to create new and meaningful features that helps in improving the accuracy of a machine learning model. It is one of the most time taking process which needs domain as well as mathematical knowledge.

In the proposed new method, new features are extracted from raw packet data to predict and detect SYN Flood DDoS attacks. The main motivation behind extracting new features from the raw packet data is because of the fact that DDoS attacks require strong statistical features to detect with high accuracy. DDoS attacks are harder to detect using just packet level information as the attacker may use different values for the packet data and may attack from multiple sources. This leads to attack packets incoming from multiple sources containing irregular data.

The captured data, which is now in CSV file format, is used to extract some meaningful features to train the model. Features are extracted using *pandas* python library.

4.1 Feature Extraction

To extract statistical features from the captured packet data, the packets are treated in batches. The number of packets in a batch is denoted as *batch_packets*. The *batch_packets* can be configured by the Network Administrator based on the traffic load of a host. In the experiment conducted to extract features, the *batch_packets* is set to 100.

One of the main features that could help in detecting a TSP SYN flood attack is tracking the number of SYN and ACK packets received. Similarly, keeping track of the number of FIN packets and RST packets could also help in determining how many connections were successfully closed and how many the server had to reset. The below features were extracted to track the number of packets with each of the TCP flags set.

1. **Number of SYN packets**

   This is the number of packets in a batch that has the SYN flag set. Denoted by *number_of_SYN*. SYN flag stands for Synchronization. Client requests a connection by sending a SYN packet.

2. **Number of ACK packets**

   This is the number of packets in a batch that has the ACK flag set. Denoted by *number_of_ACK*. ACK flag stands for Acknowledgement. The server acknowledges the receipt of a packet by sending an acknowledgement packet with the ACK flag set.

3. **Number of RST packets**

   This is the number of packets in a batch with RST flag set. Denoted by *number_of_RST*.

4. **Number of PA packets**

   This is the number of packets in a batch with PA flag set. Denoted by *number_of_PA.*

5. **Number of FA packets**

This is the number of packets in a batch with FA flag set. Denoted by *number_of_FA.*

6. **Number of SA packets**

   This is the number of packets in a batch with SA flag set. Denoted by *number_of_SA.*

7. **Number of FPA packets**

   This is the number of packets in a batch with FPA flag set. Denoted by *number_of_FPA.*

8. **Number of RA packets**

   This is the number of packets in a batch with RA flag set. Denoted by *number_of_RA.*

In attack conditions, the ratio of number of SYN packets to number of ACK packets is expected to be very high. Keeping a track of ratios of number of these flags could help in detecting an attack. Keeping this in mind, the below ratios were extracted.

9. **Ratio of SYN to ACK**

   This is the ratio of number of SYN packets to number of ACK packets in a batch. Denoted by *ratio_syn_ack*. This ratio is expected to be very high in attack scenarios.

$$ratio\_syn\_ack = \frac{number\_of\_SYN}{number\_of\_ACK}$$

10. Ratio of ACK to RST

This is the ratio of number of ACK packets to number of RST packets in a batch. Denoted by *ratio_ack_rst*. This ratio could help in understanding how many acknowledgements were sent out from the host and how many connections were reset by the host within a particular batch.

$$ratio\_ack\_rst = \frac{number\_of\_ACK}{number\_of\_RST}$$

11. Ratio of SYN to RST

This is the ratio of number of SYN packets to number of RST packets in a batch. Denoted by *ratio_syn_rst*. This ratio could help in understanding how many SYN requests were received and how many connections were reset by the host within a particular batch.

$$ratio\_syn\_rst = \frac{number\_of\_SYN}{number\_of\_RST}$$

Another important set of features that could help in detecting a flood attack is knowing the distribution of the sport / source port and dport / destination port fields of the packets in a batch. The below features were extracted to represent this based on percentage of occurrences. The distribution of the number of sport and dport during an attack is expected to vary from the distribution during normal traffic.

12. Number of source ports constituting 20% of the batch

The number of occurrences of each of the unique source ports in the batch is computed. For each unique port, it is checked if the number of occurrences is greater than or equal to 20% of the *batch_ packets*. A count

of such source ports is extracted from each batch. This number is denoted by *number_sport_20percent*.

Similarly, we take a count of such source ports that constitute different percentages of the batch as discussed below.

13. Number of source ports constituting 35% of the batch

The count of unique source ports whose occurrences are greater than or equal to 35% of the *batch_ packets*. Denoted by *number_sport_35percent.*

14. Number of source ports constituting 50% of the batch

The count of unique source ports whose occurrences are greater than or equal to 50% of the *batch_ packets*. Denoted by *number_sport_50percent.*

15. Number of source ports constituting 75% of the batch

The count of unique source ports whose occurrences are greater than or equal to 75% of the *batch_ packets*. Denoted by *number_sport_75percent.*

The count of unique destination ports that constitute different percentages of the batch is also computed.

16. Number of destination ports constituting 20% of the batch

The count of unique destination ports whose occurrences are greater than or equal to 20% of the *batch_ packets*. Denoted by *number_dport_20percent.*

17. Number of destination ports constituting 35% of the batch

The count of unique destination ports whose occurrences are greater than or equal to 35% of the *batch_ packets*. Denoted by *number_dport_35percent.*

18. Number of destination ports constituting 50% of the batch

The count of unique destination ports whose occurrences are greater than or equal to 50% of the *batch_ packets*. Denoted by *number_dport_50percent.*

19. Number of destination ports constituting 75% of the batch

The count of unique destination ports whose occurrences are greater than or equal to 75% of the *batch_ packets*. Denoted by *number_dport_75percent.*

A distribution of source IP addresses could also be very important in identifying a flood attack. The below set of features represent the count of unique source and destination IPs whose occurrences are greater than 20%, 35%, 50% and 75%.

20. Number of source IPs constituting 20% of the batch

The count of unique source IPs whose occurrences are greater than or equal to 20% of the *batch_ packets*. Denoted by *number_src_20percent.*

21.Number of source IPs constituting 35% of the batch

The count of unique source IPs whose occurrences are greater than or equal to 35% of the *batch_ packets*. Denoted by *number_src_35percent.*

22. Number of source IPs constituting 50% of the batch

The count of unique source IPs whose occurrences are greater than or equal to 50% of the *batch_ packets*. Denoted by *number_src_50percent.*

23. Number of source IPs constituting 75% of the batch

The count of unique source IPs whose occurrences are greater than or equal to 75% of the *batch_ packets*. Denoted by *number_src_75percent.*

The count of unique destination IPs that constitute different percentages of the batch is also computed.

24. Number of destination IPs constituting 20% of the batch

The count of unique destination IPs whose occurrences are greater than or equal to 20% of the *batch_ packets*. Denoted by *number_dst_20percent.*

25. Number of destination IPs constituting 35% of the batch

The count of unique destination IPs whose occurrences are greater than or equal to 35% of the *batch_ packets*. Denoted by *number_dst_35percent.*

26. Number of destination IPs constituting 50% of the batch

The count of unique destination IPs whose occurrences are greater than or equal to 50% of the *batch_ packets*. Denoted by *number_dst_50percent.*

27. Number of destination IPs constituting 75% of the batch

The count of unique destination IPs whose occurrences are greater than or equal to 75% of the *batch_ packets*. Denoted by *number_dst_75percent.*

While launching an attack, it is very difficult to mimic normal SYN packets. The IP bytes of the packets received during an attack could be very different from packets during a normal attack. This can be captured by including a feature that has

the average IP bytes during attack and normal scenarios. The below feature captures this property.

28. Average IP bytes

The average of IP bytes of all the packets in a batch. This is denoted by *average_IP_bytes*.

$$average\_IP\_bytes = \frac{Sum\ of\ IP\ bytes}{batch\_packets}$$

29. Ratio of incoming to outgoing packets

The ratio of number of incoming packets to the number of outgoing packets. This ratio is expected to be very high in attack scenarios. Denoted by *ratio_incom_to_outgo*.

$$ratio\_incom\_to\_outgo = \frac{Number\ of\ incoming\ packets}{Number\ of\ outgoing\ packets}$$

30. Number of unique incoming IP addresses

This feature represents the number of unique incoming IP addresses in a batch of packets. This is expected to be very high in attack scenarios especially in attacks with IP spoofing using random source IPs. Denoted by *number_uniq_ips*.

31. Attack aggregate

The average of values in 'Attack' field of every batch. This feature ranges between 0 to 1. Denoted by *attack_aggregate*.

$$attack\_aggregate = \frac{Sum\ of\ Attack}{batch\_packets}$$

5 Data Analysis and Preprocessing

Exploratory Data Analysis is a data analysis approach that gives us an insight into the data, its structure and relationships mostly using visual methods. It helps to identify outliers and clean up the data if necessary. After doing an exploratory analysis of the data using some visual representations, we preprocess the data and get it ready for machine learning models. In data preprocessing, we remove outliers and features that we find won't help us in the modelling phase.

5.1 Analyzing the Target variable

Extracting all the required features from the captured data, we end up with the target variable, *attack_aggregate,* and 30 other features. *attack_aggregate* variable is expected to have continuous values ranging from 0 to 1 making this a regression problem. To have a closer look at the distribution of *attack_aggregate* in the extracted dataset, *displot()* method of *scikit-learn* library is used to plot the variable as shown in Fig 4 below.

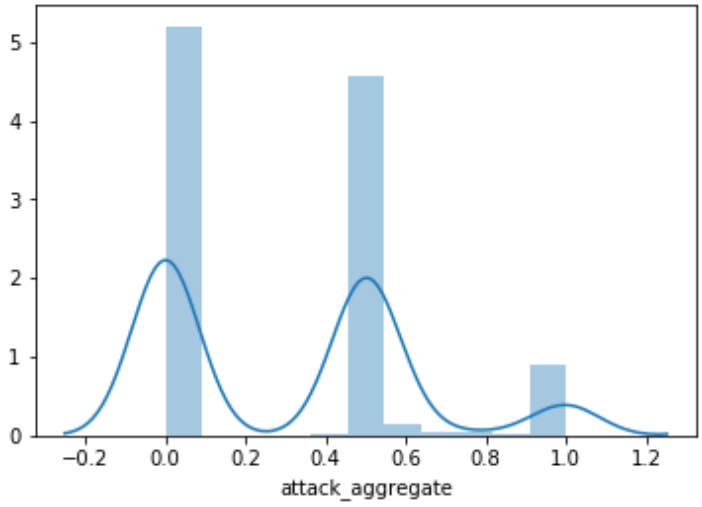


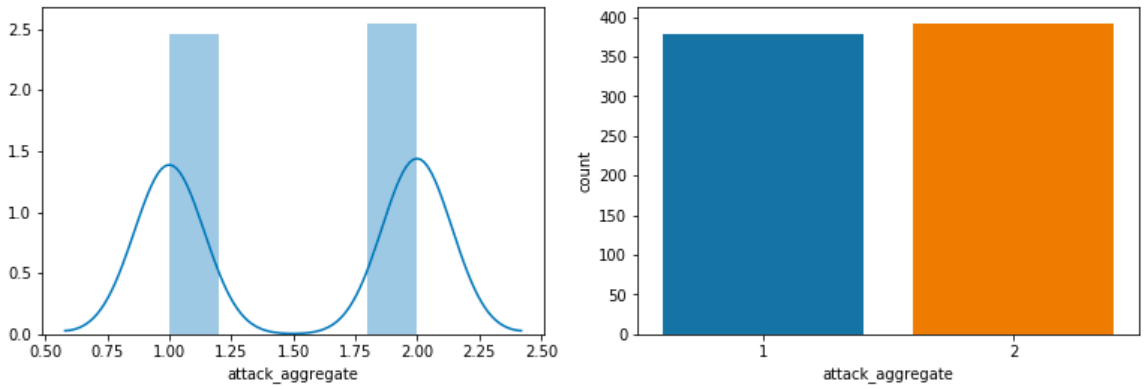
Fig 4. *attack_aggregate* variable distribution plot

attack_aggregate values concentrate mainly on three values – 0, 0.5 and 1. A value of 0.5 implies that it is derived from a batch with 50% safe and 50% attack packets. Looking closer at the dataset, this happens in the initial stages of attack when the server receives a lot of SYN packets and tries to reply with ACK packets. All the SYN packets are tagged as 1 (unsafe/attack) and the ACK packets sent by the server are tagged as 0 (safe/normal). This clearly implies that, values ranging from 0.5 to 1 can be treated as attacking batches and values from 0 to 0.5 can be treated as safe batches. *attack_aggregate* variable can now be mapped into two classes as below.

$0 <= \text{attack\_aggregate} > 0.5 \longrightarrow \text{attack\_aggregate} = \text{class 1}$

$0.5 <= \text{attack\_aggregate} >= 1 \longrightarrow \text{attack\_aggregate} = \text{class 2}$

Class 1 is represented in the dataset with a value of 1 and Class 2 as 2. A value of 1 in *attack_aggregate* thus represents safe/normal batch and value of 2 in *attack_aggregate* represents unsafe/attack batch. The distribution plot and the count plot of the attack variable with the new classes is as shown below (Fig 5). The count plot shows an equals number of instances of both classes thus making it a good dataset for training.



a. Distribution plot of attack_aggregate b. Count plot of attack_aggregate

Fig 5. Distribution and Count plot of *attack_aggregate* variable after mapping

5.2 Analyzing the Features

We now move on to analyzing the features of two datasets – one of TCP SYN Flood attack and the other of TCP SYN Flood attack with IP spoofing using multiple random source IPs. Let us start with plotting the correlation matrix of the first dataset with TCP SYN Flood attack (Fig 6) using *heatmap()* method of *scikit-learn* library.

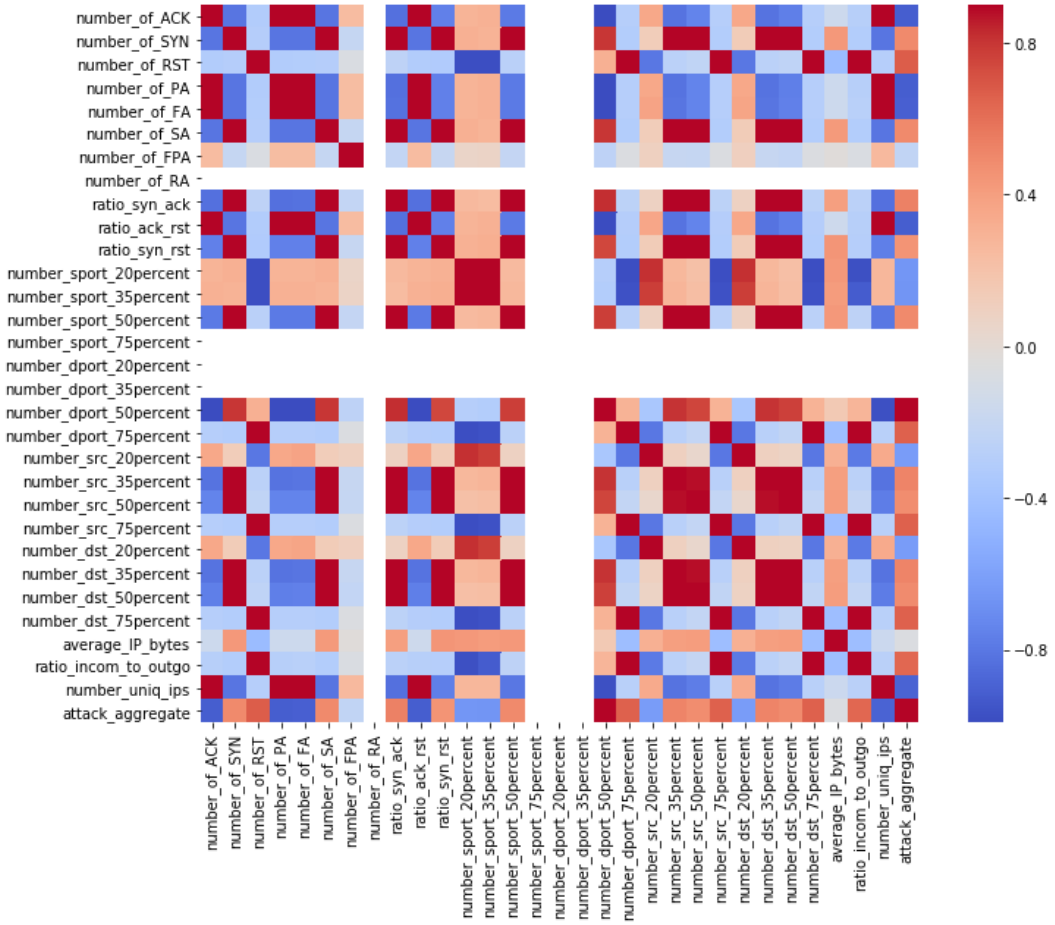


Fig 6. Correlation matrix of SYN Flood training dataset showing the correlation of features

This dataset needs some cleanup. So, removing certain features as shown below.

- *number_sport_75percent*, *number_dport_20percent*, *number_dport_35percent*, *number_of_RA* – Can remove these columns as they appear to have constant values.

- *number_of_FA* and *number_of_PA* – appears to be highly correlated. We need only one of them.
- *number_sport_35percent* and *number_sport_20percent* - appears to be highly correlated. We need only one of them.
- *number_src_35percent* and *number_src_50percent* - appears to be highly correlated. We need only one of them.
- *number_dst_35percent* and *number_dst_50percent* - appears to be highly correlated. We need only one of them.

After dropping *number_sport_75percent*, *number_dport_20percent*, *number_dport_35percent*, *number_of_RA*, *number_of_PA*, *number_sport_35percent*, *number_src_35percen* and *number_dst_35percent*, we get the correlation matrix in Fig 7.

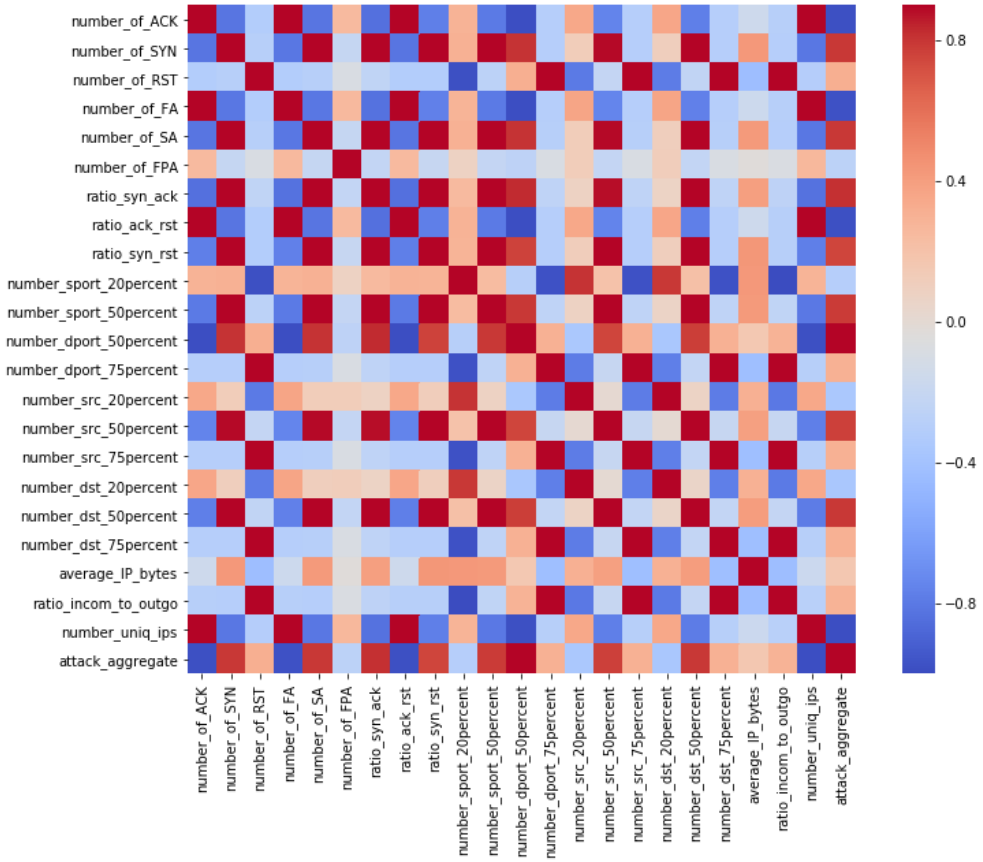
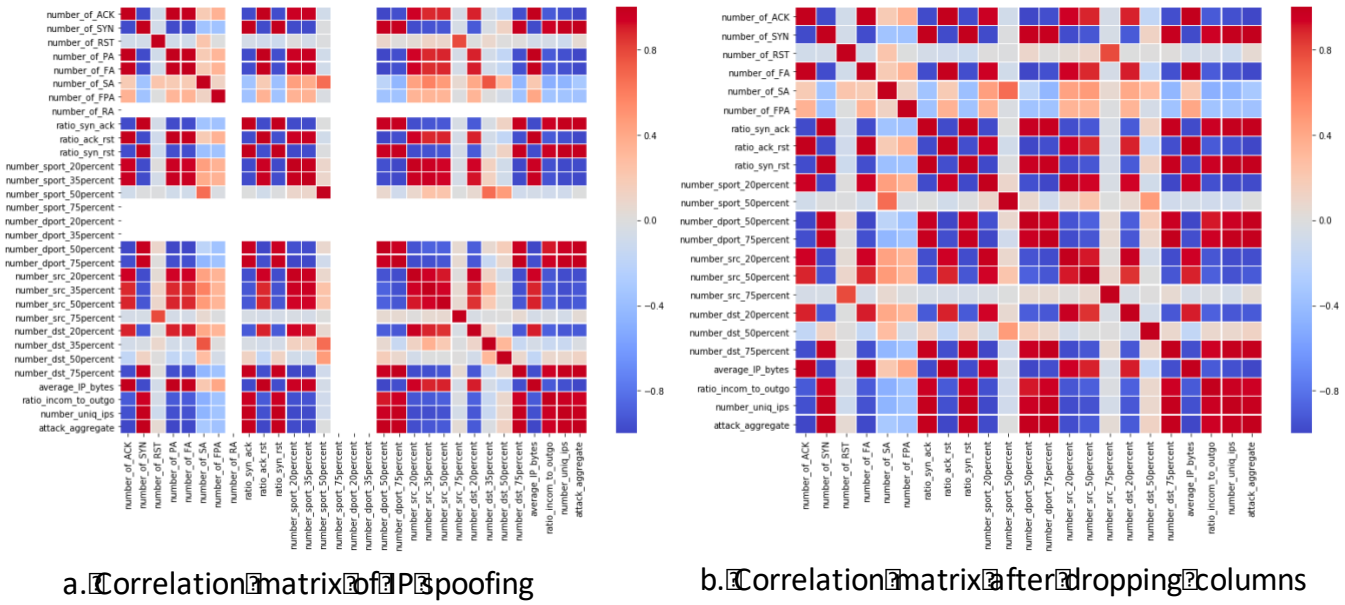


Fig 7. Correlation Matrix of SYN Flood attack training dataset after dropping columns

Similarly, the correlation matrix of dataset with IP spoofing before and after dropping columns is shown in Fig 8. After dropping all the unwanted columns, we are left with 22 features and 1 target variable.



a. Correlation matrix of IP Spoofing b. Correlation matrix after dropping columns

Fig 8. Correlation Matrix of Dataset with IP Spoofing before and after removing columns

In the training data, all the variables, other than target variable, are numeric variables. The target variable is a categorical variable which has two categories – 0 or 1. Hence, there is no need to encode features.

Using the *train_test_split()* from *sklearn.model_selection*, we randomly split the data set into train and test data sets. Looking closely at the range of values of different features, we see that, *number_of_ACK* ranges from 0 to 47, *number_of_FA* ranges from 0 to 1, *average_IP_bytes* from 40 to 74 and *ratio_syn_ack* from 0 to 100. Using *StandardScaler()* from *sklearn.preprocessing*, we standardize all the features such that the models won't use big features as the main predictors.

Towards the end of preprocessing, we are now left with a classification problem with 22 standardized features. The next step is to train and test the models.

6 Training and Analyzing Models

Training models in Machine Learning is the process by which Machine Learning Algorithms learn from training data. In this project, we start by training various models and check the accuracy of prediction. We then check with an ensemble model to see if it improves the prediction accuracy. Finally, we choose a model which can predict the attack with highest accuracy in the lowest possible time.

Using *cross_val_score()* of *sklearn.model_selection*, we first find the cross validation scores. Here, for each iteration, the training data is split into specified number of folds and the estimator is trained using the training set. Scores are computed based on the testing set for each iteration of cross validation. The cross-validation score means and errors using Logistic Regression, Support Vector Classification, K-Nearest Neighbors, Decision Tree Classifier, Random Forest Classifier, Gradient Boosting Classifier, AdaBoost Classifier, Multiple Layer Perceptron and Extra Trees Classifier is shown in Fig 9.

Algorithm	CV Error	CV Mean
Random Forest	0.0	1.0
Gradient Boosting	0.0	1.0
AdaBoost	0.0	1.0
Multiple Layer Perceptron	0.0	1.0
Extra Trees	0.0	1.0
SVC	0.003896	0.998701

a. Accuracy on SYN Flood dataset

Algorithm	CV Error	CV Mean
Random Forest	0.0	1.0
Gradient Boosting	0.0	1.0
AdaBoost	0.0	1.0
Multiple Layer Perceptron	0.0	1.0
Extra Trees	0.0	1.0
SVC	0.0	1.0

b. Accuracy on SYN Flood dataset with IP Spoofing

Fig 9. Accuracy of Classifiers on both the datasets

As seen in fig 9, all the classifiers except SVC could give 100% accuracy for both SYN Flood attack and SYN Flood attack with IP spoofing datasets. SVC could predict with 100% accuracy on SYN Flood with IP spoofing dataset.

GridSearchCV() of *sklearn.model_selection* helps us to perform parameter tuning on the classifiers. *GridSearchCV()* will methodically build and evaluate the models based on a specified list of parameters. It will also output the best estimator.

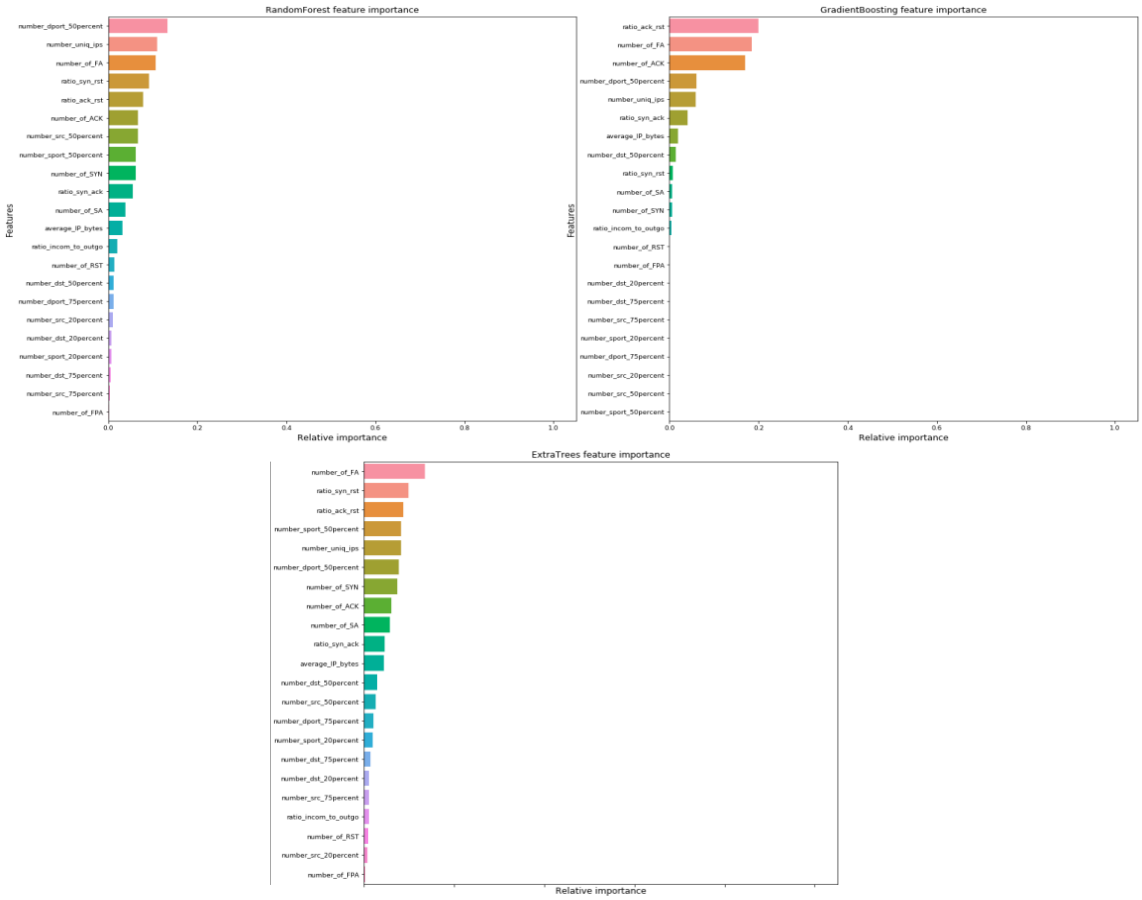


Fig 10. Feature importance of Classifiers on SYN Flood dataset

Fig. 10 and Fig. 11 shows the feature importance of some classifiers on both the datasets. This helps us to understand out of all the features that we extracted from the captured packet data, which features were found to be more important by some of these classifiers.

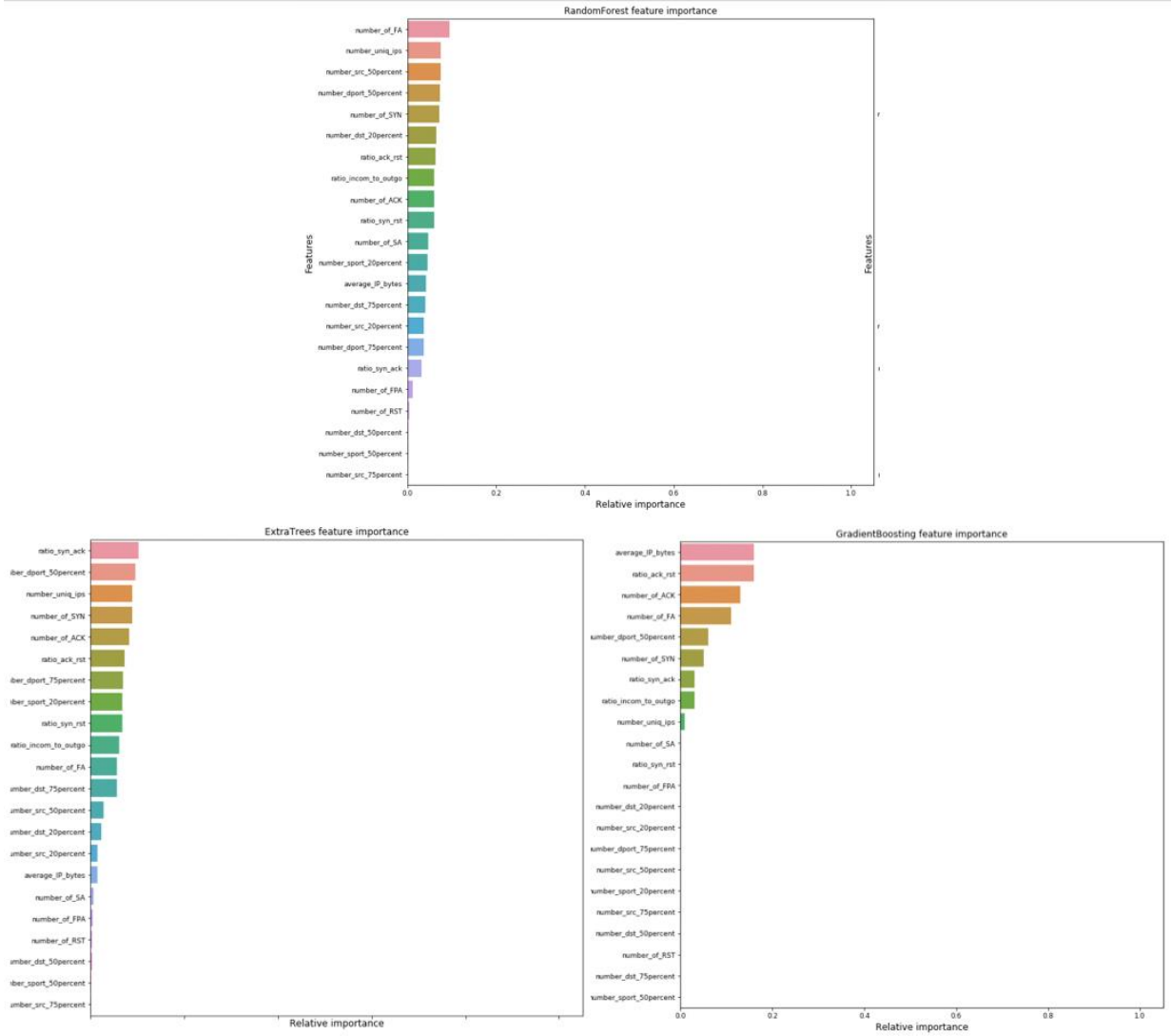


Fig 11. Feature importance of Classifiers on SYN Flood with IP Spoofing dataset

Analyzing the feature importance graphs, we see that, the most important features of SYN Flood without IP Spoofing attack dataset are *number_uniq_ips*, *number_of_FA*, *ratio_ack_rst* and *number_dport_50percent*. While, *number_of_FA*, *number_of_ACK*, *ratio_ack_rst* and *number_of_SYN* are the most important features of SYN Flood with IP Spoofing attack dataset. Most systems currently employ tracking the number of SYN and number of ACK packets as a method to mitigate SYN Flood attacks. Along with those features, keeping track of number of FA

packets, number of unique incoming IPs and number of RST packets could further improve the accuracy of such mitigation systems.

Since, the accuracy of prediction of all the classifiers is close to 100% and the importance of features varies from classifier to classifier, the next step is to use an ensemble model to predict the attack. Ensemble modelling is the method of using multiple models to create a strong model. The *VotingClassifier* of *sklearn.ensemble* is used to create an ensemble model. The ensemble model consists of Random Forest, Extra Trees, Support Vector Machine, Gradient Boosting, Multiple Later Perceptron and AdaBoost classifiers. The ensemble model prediction accuracy on the training dataset is 100% and takes approximately 0.04 seconds to predict a test dataset containing 1,050 rows of extracted data or 1,05,000 packets.

7 Implementation and Results

Next, we need to check how the selected model behaves in a real-time attack scenario. For this, the same topology (shown in Fig 1) that was used to generate training dataset is used to test the model. The topology consists of 5 hosts and 5 switches created using mininet. The hosts and switches are connected to a remote SDN controller. The controller used is Floodlight Controller. Floodlight firewall can be accessed through REST API.

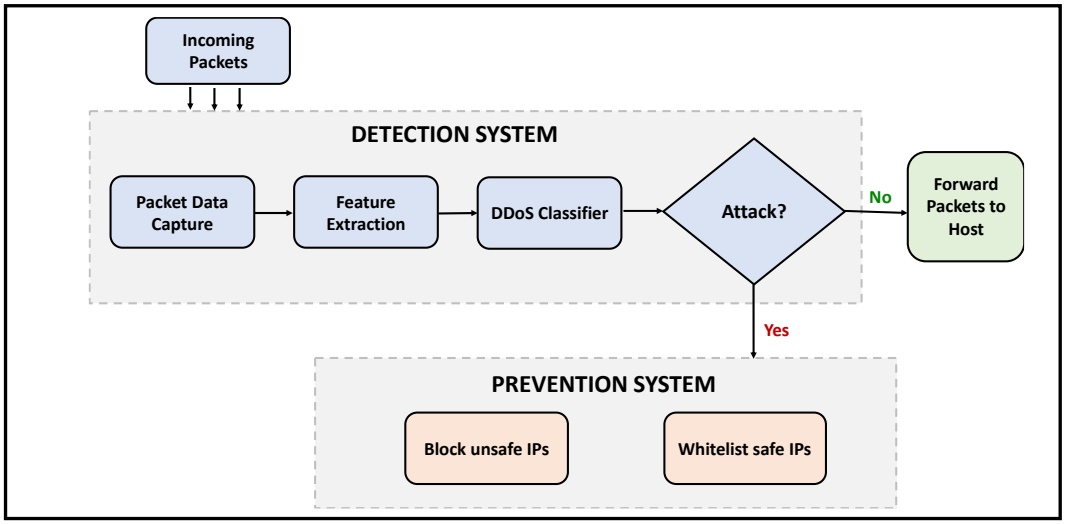


Fig 12. Overall System Design.

The overall system design is shown in Fig 12. The designed protection system has two main components – detection system and prevention system. All the incoming packets are first captured and then the required features are extracted. The extracted features are fed into the DDoS classifier, which is an ensemble model consisting of consists of Random Forest, Extra Trees, Support Vector Machine, Gradient Boosting and AdaBoost classifiers. If the batch of packets is classified as ‘class 1’ or ‘safe’, they are forwarded to the target server without any further processing. If the batch of

packets is classified as ‘class 2’ or ‘unsafe’, they are forwarded to the prevention system. In the prevention system, the IPs are either blocked or whitelisted depending on the type of attack.

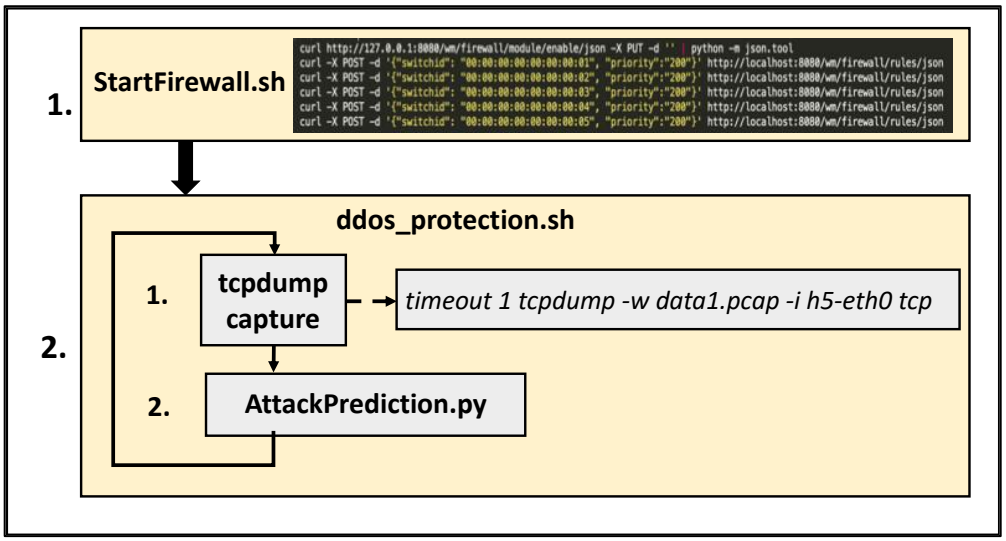


Fig 13. Functions performed by *start_protection.sh* script.

The first step to implement the system to run the script ‘*start_protection.sh*’. The functions performed by this script is summarized in Fig 13. The firewall is enabled using the script ‘*startFirewall.sh*’ which has the commands as shown in Fig 14. The first command enables the Floodlight firewall. By default, all the flows are blocked in floodlight firewall. So, all flows through all the switches are enabled using the curl commands as shown in figure.

```
curl http://127.0.0.1:8080/wm/firewall/module/enable/json -X PUT -d '' | python -m json.tool
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:01", "priority":"200"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:02", "priority":"200"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:03", "priority":"200"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:04", "priority":"200"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:05", "priority":"200"}' http://localhost:8080/wm/firewall/rules/json
```

Fig 14. ‘*startFirewall.sh*’ uses the above commands to enable Floodlight Firewall

The next step is to ssh into the target server (Host_5) and then start the '*ddos_protection.sh*' script. The script uses the below *tcpdump* command to capture incoming packets every second.

timeout 1 tcpdump -w data1.pcap -i h5-eth0 tcp

This command captures TCP packets incoming on Host_5 and saves it in *data1.pcap* file which is later used by another script '*AttackPrediction.py*' to predict the attacks and take necessary preventive measures.

```
#!/bin/bash
export IFS=","
while read var ip
do
    printf "\nBlocking IP: $ip\n"
    curl -X POST -d '{"src-ip":"'$ip'", "dst-ip": "10.0.0.5", "action":"DENY", "priority":"50"}' http://localhost:8080/wm/firewall/rules/json
    printf "\n"
done < Block_Ips.csv
rm Block_Ips.csv
```

Fig 15. Blocking unsafe IPs using Floodlight Firewall REST API

In '*AttackPrediction.py*', we first start by extracting features, as earlier discussed, by setting the *batch_packets* to 100. These extracted features, after performing all the necessary preprocessing, is fed into the selected ensemble model. The trained model is saved as a file and loaded into the script using python *pickle* library. The script then checks if any of the batches are classified as '2'. Class 2 represents attacking batch of packets. If there are any attacking batches in the captured packets, the script then finds out the attacking IP and adds it to the block list. The script shown in Fig 15 is used to block flows from block IPs list to Host_5 using Floodlight Controller Firewall REST API. The functions performed by '*AttackPrediction.py*' is shown in Fig 16.

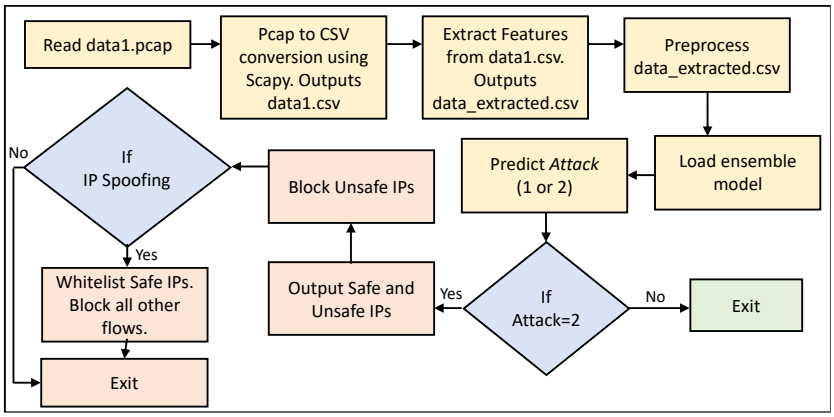


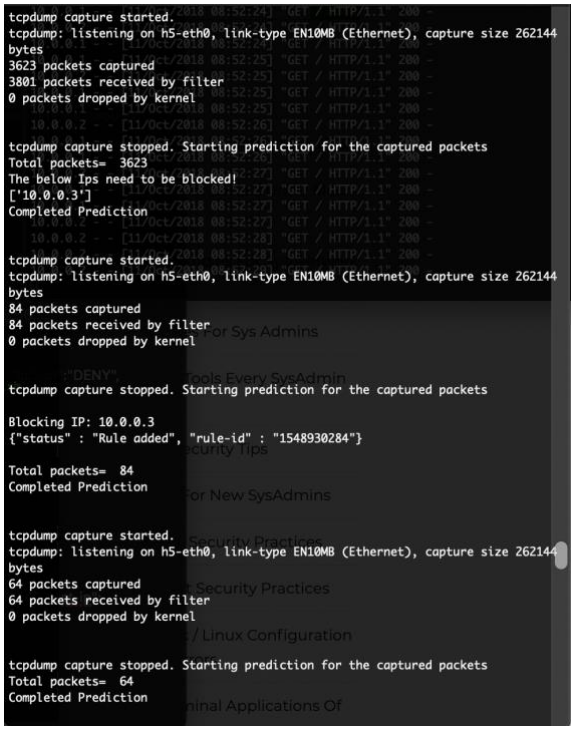
Fig 16. Flow of functions performed by '*AttackPrediction.py*' script.

If the script identifies the attack as an IP spoofing attack, it immediately finds all the batches that are predicted as '1'. Class 1 represents safe batch of packets. It then adds all the safe IPs to a safe list which is later whitelisted by the firewall.

1. Start Floodlight Controller.
2. Start Mininet topology by running '*Topo.py*' script.
3. Start the '*start_protection.sh*' script.
4. Start webserver on Host_5 by running webserver_h5.py.
5. Start normal traffic from Host_1 and Host_2.
6. Start attack from Host_3 and Host_4.

Fig 17. Steps to start and test the protection system

In the cases of flood attacks involving IP spoofing that uses multiple random IPs, blocking a certain number of IPs identified as attacking IPs is not an effective preventive measure. The attacker can continue the attack with another set of random IPs and can take the system down. In such cases, when the classifier identifies a possible flood attack with IP spoofing, the system only allows flow from these whitelisted IPs until the attack has stopped. Once the attack is stopped, the corresponding firewall rules are removed, and all flows are accepted. The steps to start and test the protection system is summarized in Fig 17.



a. Detecting and blocking unsafe IPs

b. Detecting and whitelisting in IP Spoofing attack

Fig 18. Detecting and blocking unsafe flows during SYN Flood attack

The proposed system is found to detect and block attacking IPs with 100% accuracy in a fraction of seconds (as seen in Fig 18 a and b). Host_1 and Host_2 sent normal traffic to Host_5. Host_3 started SYN Flood attack on Host_5 using Hping3. The detection script captured around 3623 TCP packets which were fed into the '*Attack_Prediction.py'* script. This script, using the classifier, was able to predict the attack and detect the source IPs causing the attack (in this case IP of Host_3 which is 10.0.0.3). It then executed necessary firewall curl commands to block the IP to which Floodlight controller replied with a success *"Rule added"* message as seen in Fig 18-(a). Similarly, during an IP spoofing attack, the script detects the attacks and also finds all the safe IPs that need to be whitelisted. The whitelisted IPs include the IPs of Host_1 (with IP 10.0.0.1) and Host_2 (with IP 10.0.0.2) which sends normal traffic to Host_5 (with IP 10.0.0.5) as can be seen in Fig 18-(b). All the requests from IPs

other than the whitelisted IPs will be dropped until the attack is completed. It is now very clear from Fig 19-a, how the connection restored on Host_1 and Host_2 after just a couple of connection failures, in contrast to Fig 19-b, which shows the Denial of Service during an undetected attack where the connection never restored.

Fig a. Connection restored in Host_1 and Host_2

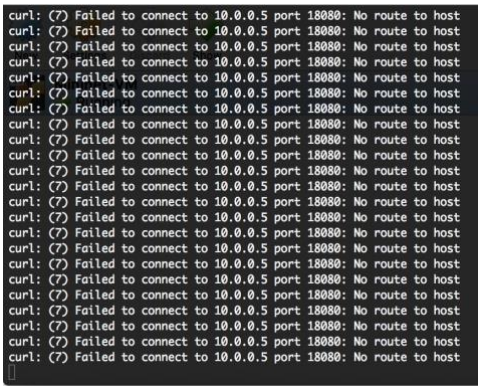


Fig b. Host_1 during an undetected attack

Fig 19. Host_5 during an attack with and without the protection system.

8 Conclusion and Future Works

As discussed in this report, the main aim of this project was to detect and prevent TCP SYN Flood DDoS attacks using machine learning. A new method of using features extracted from batches of incoming packets is used to train the Machine Learning Classifiers. This method detected TCP SYN Flood attacks and TCP SYN Flood attacks with IP spoofing with 100% accuracy. When SYN Flood attack is detected, IPs found as '*unsafe'* by the trained machine learning model are blocked. When SYN Flood attack with IP spoofing is detected, IPs found as '*safe'* IPs are whitelisted. Whitelisting IPs ensures that only flows from these IPs will be allowed, thus blocking all other unsafe flows to the host. Hence, as expected, this method proved to be a very efficient and accurate method for TCP SYN Flood DDoS attack detection.

In this project, when a flood attack with IP spoofing with source IPs randomly generated occurs, the prevention system is designed to whitelist the IPs that were identified by the classifier as safe IPs. The shortcoming of this approach is, new legitimate connection requests from source IPs other than the whitelisted IPs will be dropped. I would like to research more on finding out a better approach to implement the prevention system during such attacks so that even during an attack, legitimate users can connect to the host.

I would also like to expand this project to include detection of more flood attacks like UDP Flood Attack, HTTP Flood Attack, ICMP Flood Attack etc. The approach would involve finding and extracting new features specific to each of the above-mentioned attacks.

9 References

1. Aqeel Sahi; David Lai; Yan Li and Mohammed Diykh, '*An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Enviornment*' (2017).
2. Alptugay Degirmencioglu ; Hasan Tugrul Erdogan ; Mehrdad A. Mizani and Oğuz Yılmaz, '*A classification approach for adaptive mitigation of SYN flood attacks: Preventing performance loss due to SYN flood attacks*' in NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium (2016).
3. T. Subbulakshmi ; K. BalaKrishnan ; S. Mercy Shalinie ; D. AnandKumar and V. GanapathiSubramanian, '*Detection of DDoS attacks using Enhanced Support Vector Machines with real time generated dataset*' in 2011 Third International Conference on Advanced Computing (2011-2012).
4. Muna Sulieman Al-Hawawreh, '*SYN flood attack detection in cloud environment based on TCP/IP header statistical features*' in 2017 8th International Conference on Information Technology (ICIT) (17-18 May 2017).
5. Zerina Mašetic ; Dino Kečo ; Nejdet Doğru and Kemal Hajdarević, '*SYN Flood Attack Detection in Cloud Computing using Support Vector Machine*' (November 2017).
6. Zecheng He ; Tianwei Zhang and Ruby B. Lee, '*Machine Learning Based DDoS Attack Detection from Source Side in Cloud*' in 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) (26-28 June 2017).
7. Gaurav Somani ; Manoj Singh Gaur ; Dheeraj Sanghi ; Mauro Conti ; Muttukrishnan Rajarajan and Rajkumar Buyya, 'Combating DDoS Attacks in the Cloud: Requirements, Trends, and Future Directions' in IEEE Cloud Computing (15 March 2017).
8. *'Python Machine Learning*' – by Sebastian Raschka.
9. *'scikit-learn'*, 11 October 2018. [Online]. Available: http://scikit-learn.org/stable/

10. ‘*Floodlight*’, 11 October 2018. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview
11. ‘*Mininet’*, 11 October 2018. [Online]. Available: http://mininet.org
12. ‘*hping’*, 11 October 2018. [Online]. Available: http://www.hping.org
13. ‘*tcpdump’*, 11 October 2018. [Online]. Available: http://www.tcpdump.org