MACHINE-LEARNING AND STATISTICAL METHODS FOR DDOS ATTACK DETECTION AND DEFENSE SYSTEM IN SOFTWARE DEFINED NETWORKS

by

Merlin James Rukshan Dennis

Master of Engineering, Anna University, India, 2006

Bachelor of Engineering, Manonmaniam Sundaranar University, India, 2003

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Computer Networks

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION

Machine-Learning and Statistical Methods For DDoS Attack Detection and Defense System in Software Defined Networks

by

Merlin James Rukshan Dennis

Master of Applied Science

Computer Networks

Ryerson University, 2018

Abstract

Distributed Denial of Service (DDoS) attack is a serious threat on today's Internet. As the traffic across the Internet increases day by day, it is a challenge to distinguish between legitimate and malicious traffic. This thesis proposes two different approaches to build an efficient DDoS attack detection system in the Software Defined Networking environment. SDN is the latest networking approach which implements centralized controller, which is programmable. The central control and the programming capability of the controller are used in this thesis to implement the detection and mitigation mechanisms.

In this thesis, two designed approaches, statistical approach and machine-learning approach, are proposed for the DDoS detection. The statistical approach implements entropy computation and flow statistics analysis. It uses the mean and standard deviation of destination entropy, new flow arrival rate, packets per flow and flow duration to compute various thresholds. These thresholds are then used to distinguish normal and attack traffic. The machine learning approach uses Random Forest classifier to detect the DDoS attack. We fine-tune the Random Forest algorithm to make it more accurate in DDoS detection. In particular, we introduce the weighted voting instead of the standard majority voting to improve the accuracy. Our result shows that the proposed machine-learning approach outperforms the statistical approach. Furthermore, it also outperforms other machine-learning approach found in the literature.

Acknowledgements

I wish to express my sincere gratitude to my supervisor Dr. Ngok-Wah Ma for his continuous support in the MASc program. Without him, the completion of this study would not have been possible.

I would like to thank my co-supervisor Dr. Xiaoli Li, who gave lots of ideas to improve my results despite her busy schedule.

Special thanks to the Computer Networks Department and Yeates School of Graduate Studies at Ryerson University, for giving me this great opportunity and their financial support throughout my studies.

Last, but not the least, I would like to thank my parents, husband and my cute little daughter for their love and encouragement. Without their patience and sacrifice, I could not have completed this thesis.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
FN	False Negative
FP	False Positive
FPR	False Positive Rate
IP	Internet Protocol
ML	Machine Learning
OF	OpenFlow
OS	Operating Systems
RF	Random Forest
ROC	Receiver Operating Characteristics
SDN	Software Defined Networking
TN	True Negative
TP	True Positive
TPR	True Positive Rate
UDP	User Datagram Protocol

Chapter 1

1 Introduction

1.1 Problem Statement

Software Defined Networking is an emerging technology, which enables the network to be programmable, centralized and flexible. The SDN architecture has a separate control plane and data plane. System administrators can control the entire network through the centralized control plane (controller). These features of SDN can be used in the construction of intelligent and automated networks. Also, the operational costs in large data centers have been greatly reduced with the implementation of SDN.

However, this centralized feature of the SDN controller makes it an ideal target for the attackers. On the other hand, the same feature can also provide a new and efficient way to detect network attacks. One of the major network attacks is a Denial of Distribution of Service (DDoS) attack. With the immense internet growth, a large number of hosts are vulnerable to the attacks. Most of the DDoS attacks are generated by attacking software which is installed on the vulnerable hosts unknowingly.

This thesis proposes two approaches for the detection of the DDoS attack. The first approach, the statistical approach, uses destination Entropy and Flow statistics measurements to distinguish the normal and attack traffic. The second approach uses a machine-learning algorithm based on Random Forest (RF) classifier to classify the normal and attack traffic.

We will compare the two approaches based on three performance parameters: detection accuracy, false negative and false positive. In addition, we will also compare our Machine Learning approach with other Machine learning approaches in the literature.

1.2 Research Objective and Contribution

The main goal of this research is to develop a detection system to identify Distributed Denial of Service (DDoS) attacks in the SDN environment. In this thesis, we use the traffic parameters of normal traffic, such as payload size and packet per flow, to identify the attack. In the statistic approach, the means and standard deviations of these parameters are measured to compute various thresholds. These thresholds are used to distinguish the normal and attack traffic. Whereas in the machine learning approach, an RF classifier with appropriate modifications is implemented and used to classify the traffic.

Furthermore, a mitigation method is also proposed to mitigate the effect of the attack.

Our contributions in this thesis are,

1 Design and implementation of a DDoS attack detection system based on the statistical approach proposed by Kia [1]. We have modified and improved the approach of [1] by computing the threshold values based on the mean and standard deviations of the normal traffic parameters.

2 Propose an efficient mitigation method based on pushing a drop flow to block the attack traffic, thus, protect the controller and switch.

3 Design and implementation of a DDoS attack detection system based on the machine learning model using the Random Forest algorithm. The RF algorithm is modified in such a way that it uses weighted voting instead of standard majority voting for attack prediction as used by Alphna et al [14], Malik et al [15], Farnaaz et al [16].

4 Compare, analyze and evaluate the proposed detection and mitigation techniques with the approaches found in the literature.

1.3 Thesis Organization

The report consists of 5 chapters. The rest of the thesis is organized as follows.

Chapter 2 introduces SDN architecture and the OpenFlow protocol. It also gives a brief description of different types of DDoS attacks and a survey on DDoS detection methods found in the literature. It also covers the machine learning fundamentals and the RF algorithm in particular. The chapter ends with a summary of the DDoS detection system proposed in [1].

Chapter 3 describes the details of the two proposed approaches.

Chapter 4 discusses the experimental setup for both statistical approach and machine learning approach. It also provides the detailed results, including the detection rate, False Positive (FP) and False Negative (FN) values of the two approaches. The performances of the proposed approaches are compared with each other. We also compare the performance of our approach with other methods found in the literature.

Chapter 5 concludes this thesis and provides a brief description of the further research to make the detection system more efficient.

Chapter 2

2 Background and Related Work

2.1 Introduction to Software Defined Networking

SDN is an emerging network architecture which is dynamic, manageable and cost-effective. It is based on the abstraction of forwarding plane from the control plane. This abstraction makes the network directly programmable and flexible, which is ideal for configuring, managing, securing and optimizing the network resources dynamically and automatically.

In a traditional network, switch’s proprietary protocol tells the switch where to forward the network packet. The switch treats all the packets belonging to the same destination equally. This has been changed with the introduction of SDN technology.

SDN can make decisions about how packets should flow through the network in the forwarding plane. Packet handling rules are sent to the switches from a controller. The controller is a software application running on a server located remotely. The switches seek guidance from the controller for packet handling.

Switches and controller communicate via the controller’s south-bound interface. This communication is achieved by the OpenFlow protocol. Similarly, applications can talk to the controller via the controller’s north-bound interface. The SDN architecture is shown in Fig-2.1.

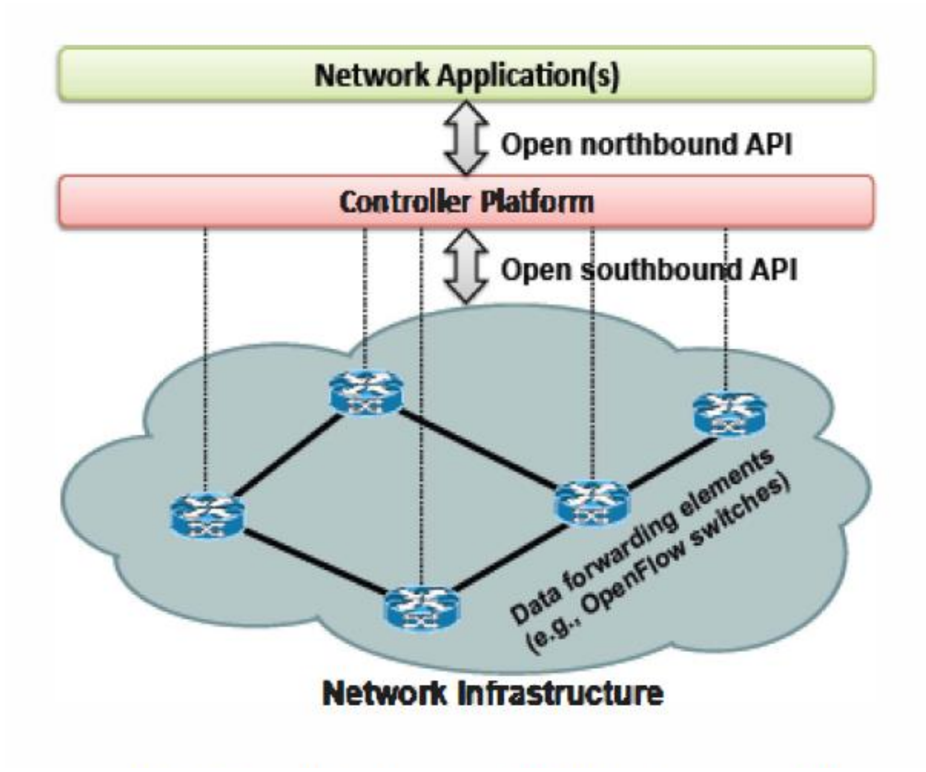


Fig 2.1: SDN Architecture [2]

Fig-2.1 shows the concept of decoupling data plane and control plane in SDN. The function of the control plane is to make decisions on where the traffic should be sent. The control plane consists of one or more SDN controllers. The controller is nothing but a software program. The controller is centralized, and it maintains a global view of the network in the control plane. A single control plane can control a number of forwarding devices such as OpenFlow switches. It defines the forwarding rules of the devices in the data plane and can remotely configure all the devices in the data plane. The network devices in the data plane forward traffic, according to these rules.

2.2 Benefits of SDN

The decoupled nature of the control plane and data plane makes SDN technology programmable. The controller is a logical entity which gives the global view of the network. It can communicate with both the SDN applications and the hardware network devices about the statistics and events happening.

SDN facilitates automated load balancing and it has the ability to scale network resources dynamically. The open standard implementation simplifies the network design and operations. It is ideal for today’s high-bandwidth applications.

Today it's easier to unify cloud resources with SDN. Large data center platforms can be easily managed from SDN controller. It's also used to implement centralized security.

2.3 OpenFlow Protocol

The communication between the controller and the data plane devices needs a suitable standard. OpenFlow is the communication interface defined between the control and forwarding layers of an SDN architecture [3]. OpenFlow manages the switches in the network and allows the controller to manipulate the flow of packets through the network.

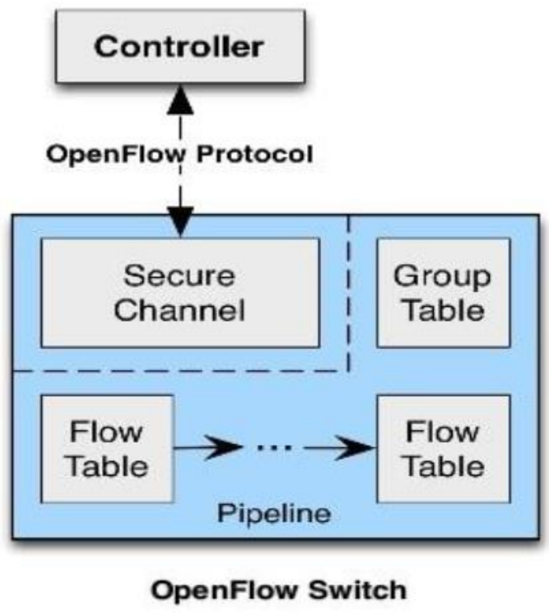


Fig 2.2: OpenFlow switch [3]

Fig-2.2 shows the structure of an OpenFlow switch. An OpenFlow switch consists of one or more flow tables, a group table and a secure channel to an external controller. The switch communicates with the controller, and the controller manages the switch via OpenFlow protocol. Each flow table in the switch contains a set of flow entries. Using the OpenFlow protocol, the controller can add, delete and update flow entries in the flow table.

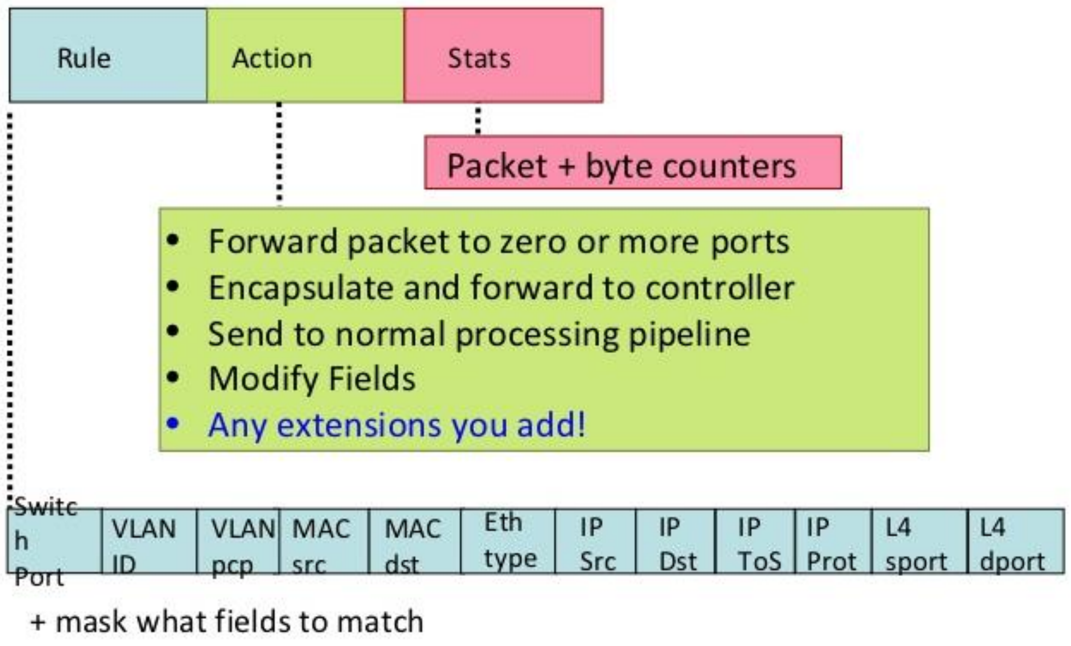
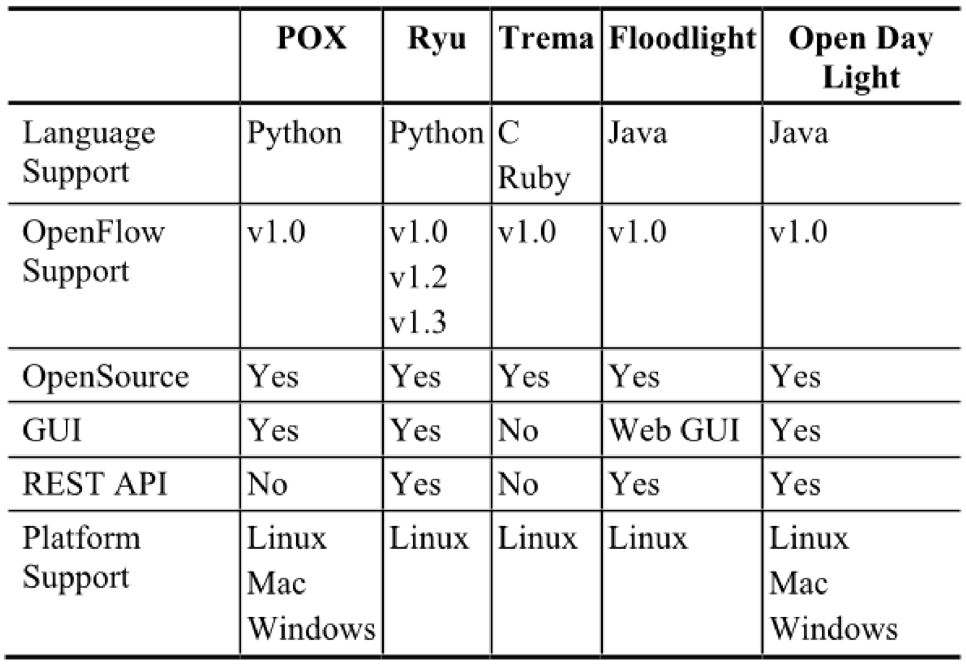


Fig 2.3: Flow Table Entries [4]

When a new packet arrives at an OpenFlow switch, it will look into the flow table to find a match. If a match is found, the action assigned to that entry is applied and the counter for the entry will be updated. If there is no match in the table, it is called table miss and the switch sends a Packet_IN message to the controller through the secure channel. The controller processes the packet and sends Packet_OUT and or Flow_MOD message back to the switch. The Packet_OUT message is an instruction to the switch on what to do with the packet. Whereas Flow_MOD message instructs the switch to install a new flow entry in the flow table. Hence, the packet is forwarded according to this new rule.

2.4 SDN controller

The controller is considered as the core of an SDN network. The controller uses protocols like OpenFlow to communicate with networking devices. There are different types of controllers like POX, Ryu, Open Day Light, Beacon, etc. The Fig-2.4 shows the different SDN controllers. In this research, POX controller is used.



	POX	Ryu	Trema	Floodlight	Open Day Light
Language Support	Python	Python	C Ruby	Java	Java
OpenFlow Support	v1.0	v1.0 v1.2 v1.3	v1.0	v1.0	v1.0
OpenSource	Yes	Yes	Yes	Yes	Yes
GUI	Yes	Yes	No	Web GUI	Yes
REST API	No	Yes	No	Yes	Yes
Platform Support	Linux Mac Windows	Linux	Linux	Linux	Linux Mac Windows

Fig 2.4: Types of SDN controller [5]

POX

POX is inherited from NOX controller [5]. It is an open source development platform, used to create an SDN controller using python programming language. POX controller provides an efficient way to handle the OpenFlow devices. Using POX controller, you can run different applications like a hub, switch, load balancer, and firewall. It is a great tool for SDN research works. The proposed algorithm in this research is implemented in the pox controller.

2.5 DDoS Attacks

Ensuring security in SDN is very important to provide secure communication. This research concentrates on Distributed Denial of Service Attack (DDoS) on the data plane. The DDoS attack makes a machine or network resources unavailable to its users [6]. This is achieved by consuming the entire network bandwidth or the resources of the network nodes (such as memory and CPU). Fig-2.5 shows the DDoS attack on an SDN controller.

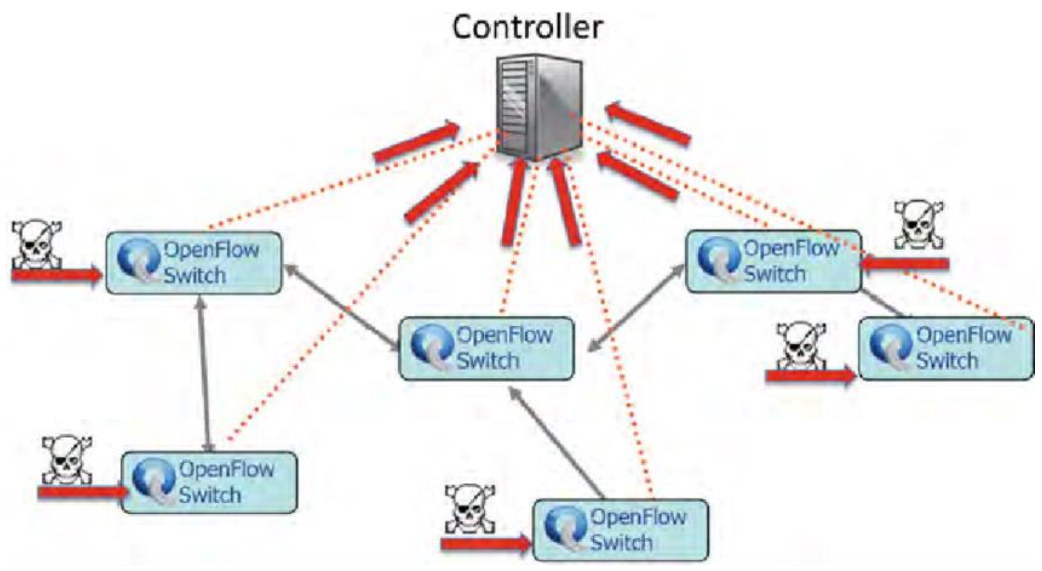


Fig 2.5: DDoS Attack on SDN controller [7]

2.5.1 Types of DDoS attack

UDP Flood [6] is a type of attacks which aims at bringing down the server by sending a large number of UDP packets to random ports on the targeted host. The attackers usually utilize the UDP's connectionless feature to submit a stream of UDP data packet to the victim machine. The victim machine's queue becomes filled and it will not be able to respond legitimate user's request. Usually, in these types of attacks, the attacker spoofs the source IP address of the UDP packets to hide the locations of the attack machines.

SYN Flood is an attack using TCP connection initiation to target the victim's machine. A large number of SYN packets are sent to the victim but no ACK is returned to the victim, causing a large number of resources at the victim's machine and making the machine unavailable to the legitimate users.

The DNS Reflection attack sends DNS request to victim source IP address which causes responses which are much larger than the requests to direct to the victim.

HTTP Flood sends a huge number of requests to a web server and overwhelms it to the point where it cannot respond to legitimate requests.

ICMP Flood is another type of attack that exhausts the resources of the victim by sending a very large number of ICMP pings (echo request), which keeps the server busy in sending responses (echo replies).

2.6 Introduction to Machine Learning

Machine learning is an Artificial Intelligence application which provides the computer program the ability to learn from input data [8]. It allows us to use historical data as the input for the prediction of future data. Thus, the accuracy of the output is solely based on the quality of the historical data.

Nowadays, machine learning techniques are used in various fields to solve different problems. For example, they are used in Email spam filtering, pattern and image recognition, search engines filtering, healthcare applications, etc.

2.6.1 Types of Machine learning algorithms

Machine learning algorithms can be broadly classified as Supervised learning algorithms and Unsupervised learning algorithms.

Supervised Learning algorithms

A Supervised learning algorithm is mainly used to solve classification and regression problems as it makes the detection or decision-making process easier. It uses the past learned data to predict the future events. The input data used to train the learning algorithm is a labeled one. That is, the input data have one or more labels, e.g. in our thesis *attack* and *no attack* are the labels used to classify the traffic data. After appropriate training, the system can classify unknown data. This research uses supervised learning algorithms.

Unsupervised learning algorithms

Unsupervised learning algorithm uses unlabeled input data to train the system. That is, the input data are not tagged with labels. It finds the hidden structure from the unlabeled input, and groups them as clusters showing the similarities. The initial performance of this type of learning algorithm is poor, but the system can tune itself to improve the performance.

2.7 Supervised Machine learning

This is the most commonly used technique in machine learning. Our research problem implements a Supervised machine learning algorithm to classify the network traffic as legitimate traffic and malicious traffic. Here the classifier gets the input which is a set of feature values also called input vector and outputs the predicted value called class. Fig-2.6 shows the supervised learning classifier.

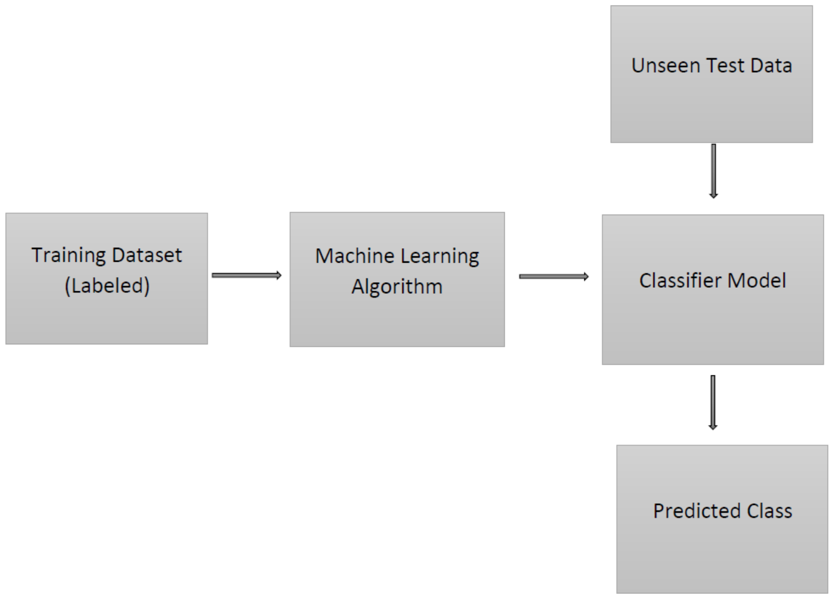


Fig 2.6: Supervised Learning classifier

Here the training data are given as an input to the learning algorithm which results in a classifier model. The performance of the classifier can be evaluated using unseen data.

2.7.1 Random Forest algorithm

Random forest is one of the most powerful algorithms used for predictive modeling [8]. The underlying principle is the construction of multiple decision trees by randomizing the combination of variables. That is, multiple decision trees are constructed from the given data set and the results are combined to make predictions. To construct multiple decision trees, the data set is divided repeatedly into subtrees by changing the combination of variables. The challenge here is to find the best combination of variables which gives the highest accuracy in prediction.

The accuracy of the Random Forest algorithm can be tuned by increasing the number of trees generated. Each individual decision tree generated makes its own prediction. Some may be right and some wrong. The individual trees that produced correct predictions reinforce each other, while wrong predictions get canceled. For this to happen, the individual trees generated must be uncorrelated. Here comes the Bagging technique which helps in generating the decision trees with minimal correlation.

Random Forest is an ensemble classifier which can implement Bootstrap aggregation (Bagging) to improve the accuracy. That is, normally the learning algorithm can choose the split point from all the available features. But the Random Forest algorithm which implements bagging technique, while constructing the individual trees, randomly chooses a node to split on. Here every node is a condition on a single feature to split the dataset. The randomly selected features x is calculated by the following formula:

$$x = \sqrt{p} \qquad (2.1)$$

where p is the total no. of input features.

For example, if a dataset had 16 input variables for a classification problem, then the randomly selected features x is given by,

$$x = \sqrt{16}$$

$$x = 4$$

Thus, the individual decision trees are constructed based on the randomly selected 4 features.

Fig-2.7 shows the process involved in RF algorithm to create decision trees and, to derive the predictions out of them. Here the training dataset D has $d_1, d_2, ...d_N$ variables. Using Bootstrap process, it creates m decision trees. The average value of the results obtained in each decision tree is calculated and the result is predicted.

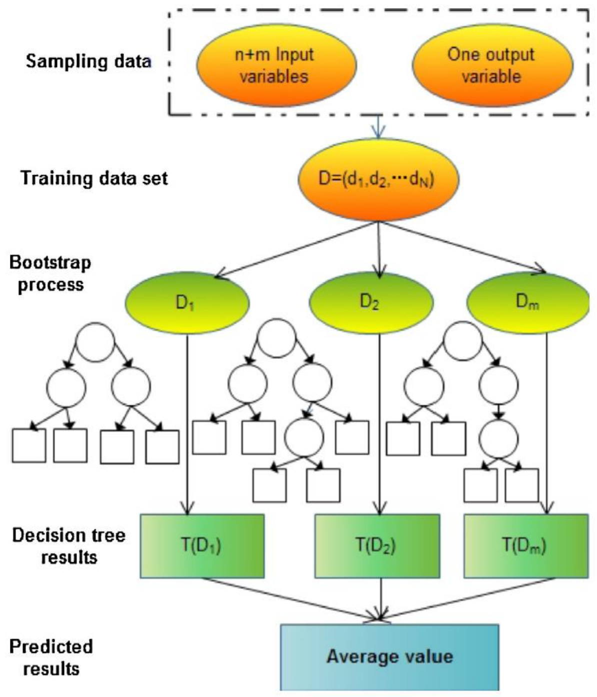
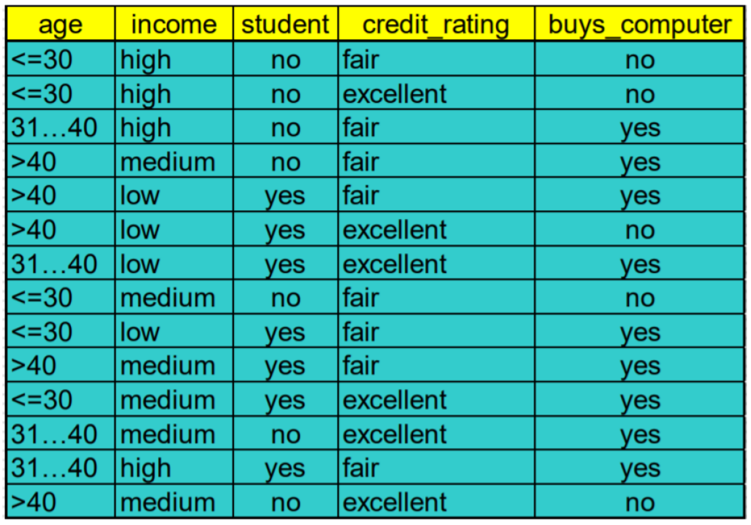


Fig 2.7: Random Forest Algorithm [8]

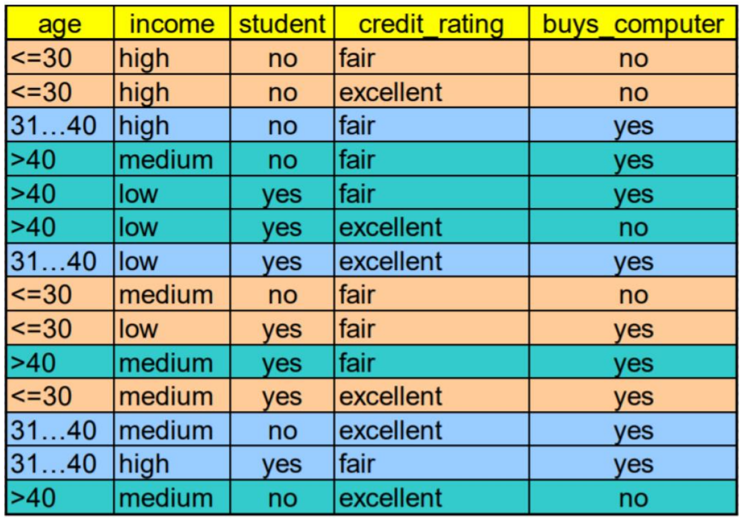
The process of creating a single decision tree can be explained with an example [39]. Fig-2.8 shows a dataset of people who buy a computer.



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Fig 2.8: Example Dataset of people who buys computer [39]

In Fig-2.9 the dataset is divided into 3 subsets based on the attribute "*age*".



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Fig 2.9: Dataset divided into 3 Subsets [39]

The attribute *age* has three different values: *<=30, 31...40, >40*. From the table, we can learn that all students with *age <=30* buys a computer. The people with *age = 31...40* buys a computer. Also, people with *age >40* and fair credit rating buy a computer. Based on these analyses the decision tree is generated.

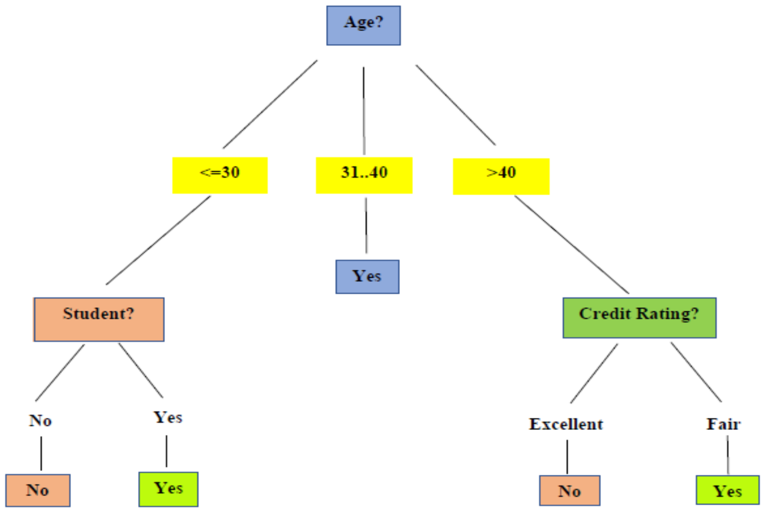


Fig 2.10: Single Decision Tree [39]

Extracting Classification Rules

Each attribute-value pair along a path from the root to leaf forms a rule. The leaf node holds the class prediction. For e.g.

If age = “<=30” and student = “no” then buys_computer = “no”

If age = “<=30” and student = “yes” then buys_computer = “yes”

Consider a new data: (<=30, yes, excellent, ?). To find the class value of new data, the tree is analyzed, and the class value is computed as a *yes*.

2.7.2 Feature Selection

Though there may be numerous features available in the given dataset, only features relevant to the problem to be solved are selected. This is called feature selection. To select the relevant features, we need to find the importance of each feature in predicting the results. This is done by calculating how much the error rate drops for a feature at each split point. Those features with small error rate are considered as more important for the classification problem. This is called a Gini score. The Gini score of a feature can be calculated by the following formula,

$$G(D) = 1 - \sum_{j=1}^{n} p_j^2 \qquad (2.2)$$

where D is the dataset of n classes and p_j is the relative frequency of a feature with class value j. The average of all the Gini score of a feature across all the decision trees gives the importance of that feature. Based on the importance of features, we can select the most relevant features from the given dataset, which gives the accurate prediction result.

2.7.3 Classifier Accuracy Estimation

Estimation of the predictive accuracy of a classifier is done to know how good the prediction will be. Common methods used for accuracy estimation are [40]: Validation set approach and k-fold cross-validation.

In validation set approach, to measure the classifier accuracy, the dataset is divided into a training dataset (50%) and testing dataset (50%). Once the classifier model is built using the training dataset, the accuracy is estimated using the unused testing dataset.

Whereas, k-fold cross-validation [40] can be performed by dividing the entire dataset into k-folds. For each k-folds in the dataset build the model using *k-1* folds of the dataset. Then test the model using the k_{th} fold. Repeat this procedure until all the *k* folds have served as a test data. Fig-2.11 shows the k-fold cross-validation for *k* =10.

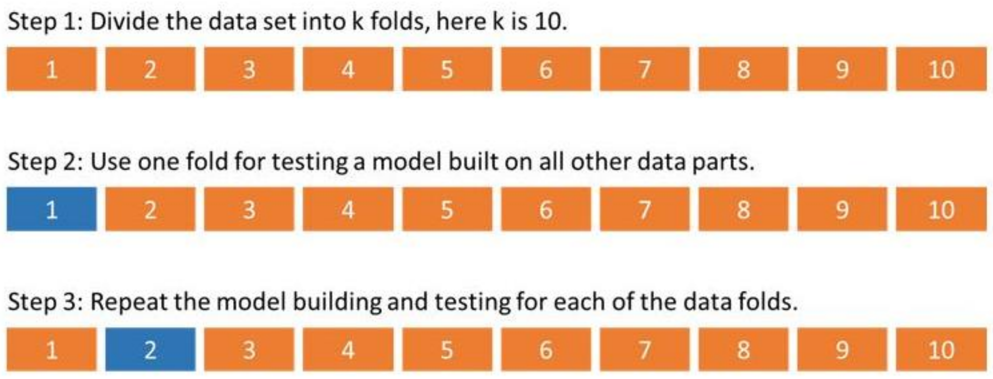

Step 1: Divide the data set into k folds, here k is 10.
1 2 3 4 5 6 7 8 9 10
Step 2: Use one fold for testing a model built on all other data parts.
1 2 3 4 5 6 7 8 9 10
Step 3: Repeat the model building and testing for each of the data folds.
1 2 3 4 5 6 7 8 9 10

Fig 2.11: 10-fold cross-validation [46]

The value of *k* should be chosen carefully because lower *k* value produces more error and higher value of *k* takes large computation time. In our thesis, we have used k-fold cross-validation method with *k* =10.

2.7.4 Advantages of Random Forest

- The Random Forest algorithm can handle large datasets.
- The accuracy is high compared to other machine learning algorithms.
- The implementation is easier and its faster than any other algorithm.
- It overcomes the problem of overfitting (model error due to noise in the training data) if many trees are grown.

2.7.5 Disadvantages of Random Forest

- Utilizes more memory for building a forest.
- Random forest models are black boxes which are hard to interpret.
- Random forest overfits when the number of trees generated is less.

2.8 Related Work in SDN based DDoS attack detection

We discuss the related work based on two groups of detection methods. One group uses statistical approach while the other group uses machine learning approach.

2.8.1 DDoS attack Detection using Statistical Approach

Researchers can find many studies on DDoS attack detection methodologies. The method proposed by Seyed et al. [6] is based on the entropy comparison of consecutive packet samples to identify changes in their randomness. A window of 50 packets is collected, and the entropy is calculated from their destination IP addresses. If the entropy is less than the threshold, an attack is reported.

Surender Singh et al. [9] proposed a distributed framework, which analyzes the behavior of the packet flows. The proposed method uses entropy and a traceback algorithm to distinguish the malicious flows from the legitimate flow.

Jisa David et al. [10] proposed a DDoS attack detection system which is based on fast entropy using flow-based analysis. Their proposed method shows better detection accuracy. They analyze network traffic and compute the fast entropy of request per flow.

SPHINX [11] is a framework proposed to detect attacks in SDN in real-time with low performance overheads. It can detect both known and potentially unknown attacks on network topology. It is mainly based on an approximation of real network into a flow graph. It uses these Flow graphs to detect security threats in the network topology.

Lei et al. [12] proposed a system called FloodGuard. It concentrates on an SDN-specific attack called data-to-control plane saturation attack. It implements two modules proactive flow rule analyzer which preserves network policy enforcement and packet migration protects the controller being overloaded.

Qin et al. [13] proposed a method for intrusion detection with a time window of 0.1 seconds and three levels of threshold. This method tries to reduce false positive and false negative values. It is found that the time and resource consumption of the method is high.

Proposed method extends the recent work done by Kia [1] on Early Detection and Mitigation of DDoS Attacks in Software Defined Networks, which is based on an Entropy variation of the destination IP address, Flow Initiation Rate and Study of Flow Specification. The proposed method is a lightweight DDoS attack detection at its early stage. In our modified method we have implemented moving mean and standard deviation for the computation of adaptive thresholds and a better mitigation module has been introduced.

2.8.2 DDoS attack Detection using Machine Learning Approach

DDoS Attack Detection and Prevention based on Ensemble Classifier (RF) proposed by Alpna et al. [14] uses a combination of classifiers to improve the performance of their model. Experimental results were conducted on UCLA dataset. The results show high accuracy with minimum error.

In [15] [16], the authors have proposed network IDS using Random Forest algorithm. They have classified DDoS attack under network intrusion attacks. They have not considered the enormous volume of attack packets that DDoS detection system has to handle in comparison with intrusion attacks. The method is only suitable to fight against the intrusion attacks. Moreover, their approach cannot be used to mitigate the attacks. Whereas in our thesis, we have classified the attack based on the features like payload size, packet count per flow and the flow duration that are responsible for the breakdown of the server. Our approach can also successfully mitigate the attacks.

Keisuke Kato et al. [17] proposed an intelligent DDoS attack detection system using packet analysis and Support Vector Machine. The detection system used SVM with a Radio Basis Function (RBF) neural networks. Experiments were done using CAIDA DDoS attack 2007 dataset.

Sivatha Sindhu et al. [18] proposed a neural decision tree for feature selection and classification. The proposed method uses six decision tree classifiers namely Decision Stump, C4.5, Naive Baye's Tree, Random Forest, Random Tree and Representative Tree model to detect the anomalous network pattern. They used sensitivity and specificity for the performance evaluation.

Saurav Nanda et al. [19] studied the attack patterns in the network using ML approach. The methodology uses 4 different ML algorithms like C4.5, Naive Bayes, Bayes net & Decision table. The prediction accuracy of the algorithms was compared, and they conclude that Bayesian network has the highest prediction rate.

Zhong et al. [20] proposed a DDoS attack detection method using a fuzzy c-means (FCM) clustering algorithm. To extract the features in network traffic they used Apriori association algorithm.

IDS using RF and SVM proposed by MD Al Mehedi Hasan et al. [21] developed 2 models for IDS using SVM and RF. The performance of these two models was compared based on their detection rate and precision and false negative rate.

The practical drawback in the above-mentioned approaches is that the authors have implemented the RF algorithm using the default prediction method which is the majority voting. Majority voting does not give accurate results out of random predictions. This drawback is eliminated in our approach by replacing majority voting by weighted voting, which gives more accurate results. We have discussed the implementation of weighted voting in detail in chapter 3.

In this thesis, both statistical and a supervised machine learning based DDoS attack detection classifier is proposed for the SDN environment.

2.9 Description of the Original Approach

The proposed statistical approach is based on the work done by Kia [1]. Here we call her work as the original approach. In this section, we will briefly describe the original approach. The approach is designed based on three main concepts: Entropy variation of destination IP address, Flow Initiation Rate and Study of Flow Specifications.

2.9.1 Stage I: Detection based on Entropy Variation

Entropy is a measure of uncertainty or randomness associated with a random variable [1]. This research implements the computation of entropy to detect DDoS attacks in the first stage with less computation time. Here, the entropy is computed based on the measure of traffic randomness with respect to the destinations in a network. The entropy drops considerably when there is a single-victim attack, as all the packets are destined to the same destination address, whereas, in the non-attack scenario the entropy value tends to be larger for the traffic is normally spread out to many destinations.

Entropy computation is done by collecting the incoming packets in a packet window of fixed size n. That is, each window can hold n number of packets. For each window, the incoming traffic is analyzed and classified according to the frequencies of occurrences of the destination IP addresses. The frequency of occurrence (F_i) of destination IP address IP_i is calculated by,

$$F_i = n_i / n \qquad (2.3)$$

where n_i is the number of packets with destination address IP_i.

And the entropy is calculated by

$$H = -\sum_{i=1}^{n} F_i \log_2 F_i \qquad (2.4)$$

since $0 \leq F_i \leq 1 \Rightarrow H \geq 0$, maximum entropy occurs when each packet is destined to exactly one host and minimum entropy occurs when all the packets in a window are destined for a single host. A small entropy is a good indication of a DDoS attack on a single victim. The detection stage here derives an entropy threshold, E_{th}, based on the average entropy of the normal traffic. If $H < E_{th}$, an attack is suspected, and a warning is issued.

2.9.2 Detection based on the number of Flows

Entropy-based DDoS attack detection described in the previous section is best suited for single victim attack detection. For the multiple victim attacks, we cannot rely only on the entropy as the attack is targeted at multiple destinations. Hence, along with entropy variation, the system incorporates another method of DDoS detection based on the flow rate.

Many DDoS attacks, send a large number of packets with spoofed source IP addresses to the switch. The switch, in turn, sends many *packet-in* messages to the controller to set up flows. This process increases the CPU usage of the controller and depletes switch memory and network bandwidth. The consequences can bring down the controller and/or crash the switch.

To detect such attack, the new flow rate is computed in each window by,

$$flow\_rate = n / t \qquad (2.5)$$

where t is the time taken to collect n *packet-in* messages, where n is the window size. The calculated flow rate is compared against the chosen threshold value, which is derived from the average flow rate of the normal traffic. If the current flow rate exceeds the threshold value, an attack is suspected, and the algorithm enters the stage II for the confirmation of the attack. If the flow rate is below the threshold limit, the network is considered as attack free.

2.9.3 Stage II: Detection based on Analysis of Flow statistics:

In stage II, the system analyzes the following characteristics: number of packets per flow (P_f), amount of received bytes (B_f) and flow duration (D_f). The controller collects these flow statistics every 10 seconds from the switches.

P_f, B_f and D_f are checked against the packet count thresholds, P_{th}, the payload size threshold, B_{th} and the flow duration threshold, D_{th} respectively. The threshold values P_{th}, B_{th} and D_{th}, on the other hand, are derived based on the averages of P_f, B_f and D_f, respectively.

The packet count, byte count, and duration for each flow are obtained from the default counters of the switch's flow table and checked against the following conditions:

1. Is the packet count of a flow is less than the threshold value, P_{th}
   ($P_f < P_{th}$)
2. Is the payload size is less than the threshold value B_{th}
   ($B_f < B_{th}$)
3. Is the flow duration is less than the threshold value D_{th}
   ($D_f < D_{th}$)

If any two conditions are true, the counter (*fcount*) is increased by one. After all the flows have been examined the attack rate is calculated by dividing the counter (*fcount*) value by the number of flows. If the calculated attack rate exceeds the flow rate threshold value, an alarm is raised confirming the attack.

2.9.4 Mitigation Module

The next goal is to protect the switches and controller under attack. Usually, the controller will not crash easily as they are designed with high capacities. But the switches are not very robust against attacks due to their limited resources. During the attacks, the flow tables of the switches get filled with a large number of short flows which eventually breaks the switch.

Once the attack is detected, to prevent the breakdown of the switch, the default value of flow idle_timer is changed to the mitigating value. The mitigated value which is smaller than the default value makes the short flows timeout quickly and the flows are deleted from the switch flow tables.

Chapter 3

3 Proposed Methods

The focus of our research is on the security challenges in DDoS attack detection in SDN environment. Two DDoS attack detection methods, Statistical Approach and Machine Learning Approach, are proposed.

3.1 Proposed Method - Statistical Approach

Our proposed method is based on the original approach from Kia [1]. Here, we call our approach as the modified approach. Our modified approach differs from [1] in the computation of adaptive threshold values and the implementation of mitigation module.

3.1.1 Computation of Threshold values

In [1], the thresholds are derived from the average traffic parameters only, whereas, in our thesis, the proposed algorithm uses exponential moving mean and standard deviation to calculate the dynamic threshold values. The use of mean and standard deviation provide more accurate measurements of the traffic parameters and thus lead to more appropriate thresholds.

Our proposed statistical detection method requires four threshold values to detect the attack: Entropy threshold E_{th}, Flow rate threshold F_{th}, Packet count threshold P_{th} and Payload threshold B_{th}. The threshold values are computed based on the normal traffic. That is the detection system analyzes the normal behaviour of the network without any attack.

The threshold values used in our thesis are dynamic and are updated based on the current traffic load. Let x_n be the traffic parameter measured in the n^{th} window, then the moving mean, $\bar{x}_n$, and standard deviation, σ_n, calculated at the end of the n^{th} window is given by equation 3.1 and 3.2 respectively.

$$\bar{x}_n = \quad \alpha * x_n + (1-\alpha) * \bar{x}_{n-1} \qquad 3.1$$

$$\sigma_n = \sqrt{\alpha * (x_n - \bar{x}_{n-1})^2 + (1-\alpha) * (\sigma_{n-1})^2} \qquad 3.2$$

where α is a constant whose value can be between 0 and 1.

The adaptive threshold, th_n, calculated in the n^{th} window is given by equation 3.3:

$$th_n = \bar{x}_n + k\sigma_n \tag{3.3}$$

where k is a constant.

In our thesis, the value of k is set to 1 to derive the threshold values used in the first stage of attack detection and the value of k is set to 2 to derive the threshold values in the second stage of attack detection.

By choosing the value of k to be one in the first stage, the system will detect the attack earlier. The false positives that are caused by the use of small k may be eliminated by the second stage of detection. In the second stage, k is set to 2 to prevent the final false positive rate from getting too large.

3.1.2 Mitigation Module

In the previous section, we described the statistical approaches in DDoS attack detection. The next goal of our research is to protect the switches and controller under attack. According to the method proposed in [1], the mitigation module is implemented by replacing the default value of the flow idle timer with a small mitigated value causes the malicious flow timeout quickly and are deleted from the switch flow table. This mechanism is good when the rate of attack is small. But when it is a large attack, the switch breaks down and loses the communication with the controller. Hence, changing the flow idle timer won't be beneficial.

There are different mitigation methods implemented by various authors. Brainard et al [43] mention random early dropping as one of the earliest solutions to handling attacks, but this method has the highest risk of dropping legitimate traffic. Buragohain et al [44] introduce a mitigation method based on how many times a suspicious source address, attempts to attack. That is if the source address attempts to send flows more than a random legitimate counter value then it is blocked. Xu et al [45] propose IP traceback to filter packets during the attack.

In our proposed method, the defense system of the controller is activated once it receives the attack confirmation from the detection system. The mitigation process can be explained as follows.

- For every 3 second window, the number of new flows coming into the controller is counted and checked against the flow rate threshold F_{th} which is calculated using equation 2.5.

- If the count of new flows exceeds the threshold, the packets are dropped until the end of the time window.
- The count will be reset at the beginning of the new window.
- The controller pushes new flows into the flow table of the switch to block the similar malicious flows.

By limiting the number of flows per 3-sec window, the system can defend a large DDoS attack as shown by the results in Chapter 4.

3.2 Machine Learning Approach

There are many machine learning techniques available for DDoS detection. In our research, we have used classification technique using Random Forest (RF) classifier.

3.2.1 Building the Machine Learning Model

Nowadays there are many software tools which can be used to identify the DDoS attack. But malicious traffic can be in any form. The attackers change their attack patterns regularly. So, there is a need to learn from experience. This can be achieved by machine learning. The Machine Learning algorithm can be used to analyze traffic to recognize an attack pattern.

The basic steps of Machine Learning approach can be summarized as follows.

1. Collect the raw data. (UCLA Dataset)
2. Preprocess the dataset to insert missing values and feature extractions, etc.
3. Identify the most important features.
4. Create a sub-dataset with the most relevant features.
5. Train the Random Forest classifier.
6. Calculate the accuracy of the model.
7. Test with unseen data.

8. Evaluate the results.

The above process can be represented by Fig-3.1.

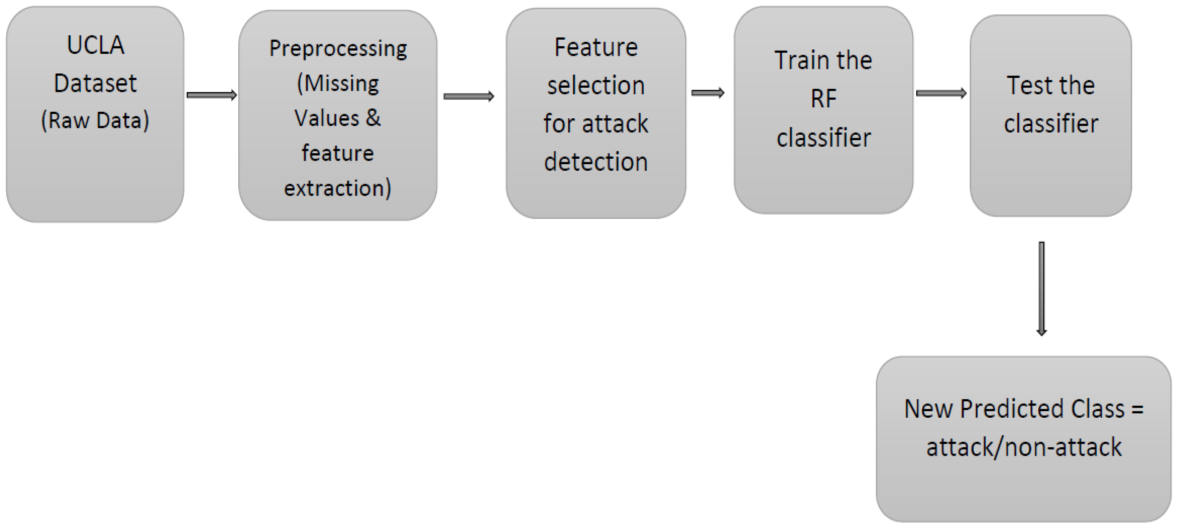

UCLA Dataset (Raw Data)
Preprocessing (Missing Values & feature extraction)
Feature selection for attack detection
Train the RF classifier
Test the classifier
New Predicted Class = attack/non-attack

Fig 3.1: Machine Learning Methodology

3.2.2 Random Forest classifier

The random forest (RF) classifier is constructed by combining the unpruned random decision trees. It uses the power of many decision trees to generate predictive models. In this phase, we have built the random forest classifier using python machine learning library, scikit-learn and the data analysis library, pandas.

How Random Forest Works

Random Forest Creation Algorithm

1. Let the number of training cases in the dataset be *N*.
2. Select randomly *m* features from total *M* features, such that *m* is much less than *M*.
3. Among *m* features calculate the node *d*, using the best split point.
4. Split the *d* node into daughter nodes.

5. Repeat 2 to 4 until a leaf node has been reached.
6. Build the forest by repeating steps 2 to 5 for *i* number of times. The value of *i* is equal to the number of trees to be created.

Random Forest Prediction Algorithm

After the creation of forest using the training dataset, we can perform prediction for the unknown test data using Majority voting.

1. Select the test features and use the rules of each randomly created decision tree to predict the result. Save the result as the target.
2. Calculate the votes for each predicted target.
3. High voted target is declared as final prediction

Internally random forest creates many independent decision trees and sets the rules for each decision tree based on the values of the input variables. There is no need to set the classification rules manually. So the dataset plays an important role here. To get accurate results, our datasets should be error free.

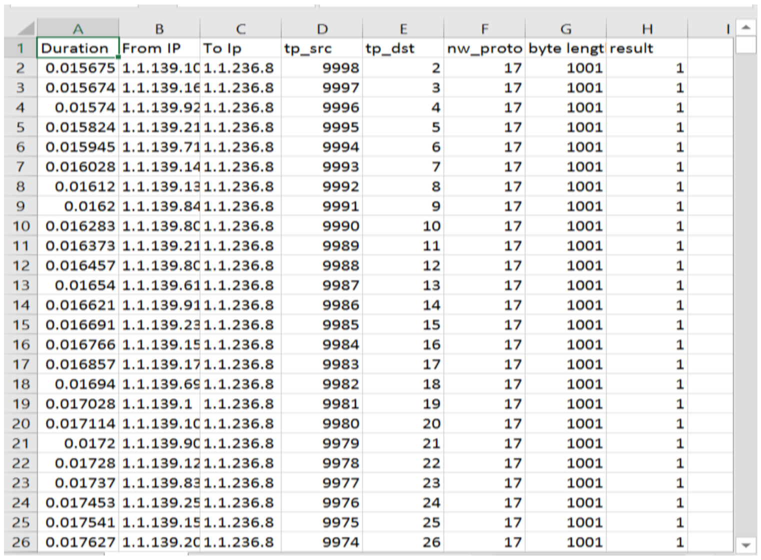
3.2.3 UCLA Dataset

Building the training data is the most important step in the implementation of machine learning approach. We need to select adequate dataset for training our machine learning model. In this research, we have used UCLA dataset [22] to build the training data. The UCLA dataset contains real-time UDP flood attack traces. As our thesis is based on SDN architecture, we chose to modify the UCLA dataset by adding traffic flow entries of the simulated traffic in addition to the real traces. The data from the dataset is downloaded and preprocessed for any missing values and then converted to a comma separated file (.csv) format which can be read by the machine learning module developed in python. The features in UCLA dataset include Packet_TIME, IP_from, IP_to, PORT_from, PORT_to, LENGTH.

Features in UCLA Dataset	
Packet_TIME	Time when the packet was sent
IP_from	Number masking the IP address of packet source
IP_to	Number masking the IP address of the packet destination
PORT_from	Original source port
PORT_to	Original destination port
U	UDP Packet
LENGTH	Length of packet (without header) in Bytes

Table 3.1: Features in UCLA Dataset

The traces of attack data in the UCLA dataset were generated by Tribe Flood Network attack tool. The features used in our training data were altered to match our simulated network setup. Fig-3.2 shows part of the training dataset used in our research.



	A	B	C	D	E	F	G	H
1	Duration	From IP	To Ip	tp_src	tp_dst	nw_proto	byte lengt	result
2	0.015675	1.1.139.1C	1.1.236.8	9998	2	17	1001	1
3	0.015674	1.1.139.16	1.1.236.8	9997	3	17	1001	1
4	0.01574	1.1.139.92	1.1.236.8	9996	4	17	1001	1
5	0.015824	1.1.139.21	1.1.236.8	9995	5	17	1001	1
6	0.015945	1.1.139.71	1.1.236.8	9994	6	17	1001	1
7	0.016028	1.1.139.14	1.1.236.8	9993	7	17	1001	1
8	0.01612	1.1.139.13	1.1.236.8	9992	8	17	1001	1
9	0.0162	1.1.139.84	1.1.236.8	9991	9	17	1001	1
10	0.016283	1.1.139.8C	1.1.236.8	9990	10	17	1001	1
11	0.016373	1.1.139.21	1.1.236.8	9989	11	17	1001	1
12	0.016457	1.1.139.8C	1.1.236.8	9988	12	17	1001	1
13	0.01654	1.1.139.61	1.1.236.8	9987	13	17	1001	1
14	0.016621	1.1.139.91	1.1.236.8	9986	14	17	1001	1
15	0.016691	1.1.139.23	1.1.236.8	9985	15	17	1001	1
16	0.016766	1.1.139.15	1.1.236.8	9984	16	17	1001	1
17	0.016857	1.1.139.17	1.1.236.8	9983	17	17	1001	1
18	0.01694	1.1.139.69	1.1.236.8	9982	18	17	1001	1
19	0.017028	1.1.139.1	1.1.236.8	9981	19	17	1001	1
20	0.017114	1.1.139.1C	1.1.236.8	9980	20	17	1001	1
21	0.0172	1.1.139.9C	1.1.236.8	9979	21	17	1001	1
22	0.01728	1.1.139.12	1.1.236.8	9978	22	17	1001	1
23	0.01737	1.1.139.83	1.1.236.8	9977	23	17	1001	1
24	0.017453	1.1.139.25	1.1.236.8	9976	24	17	1001	1
25	0.017541	1.1.139.15	1.1.236.8	9975	25	17	1001	1
26	0.017627	1.1.139.2C	1.1.236.8	9974	26	17	1001	1

Fig 3.2: Sample UCLA Dataset

3.2.4 Training Phase

With a training dataset and features selected, we can now train the machine learning models. The training phase is shown in Fig-3.3.

The first step is to get the input dataset (UCLA), then process it. That is, the features or columns that have zero values needs to be edited or removed depending on the importance of the data. Also, the values containing characters must be converted into numeric in order for the data to be processed by the algorithms. The next step is to select the features which are relevant to the attack detection. We then train the models using random forest algorithm from python scikit-learn libraries, and finally, we save the trained model and record the results of cross-validation and use them for future predictions.

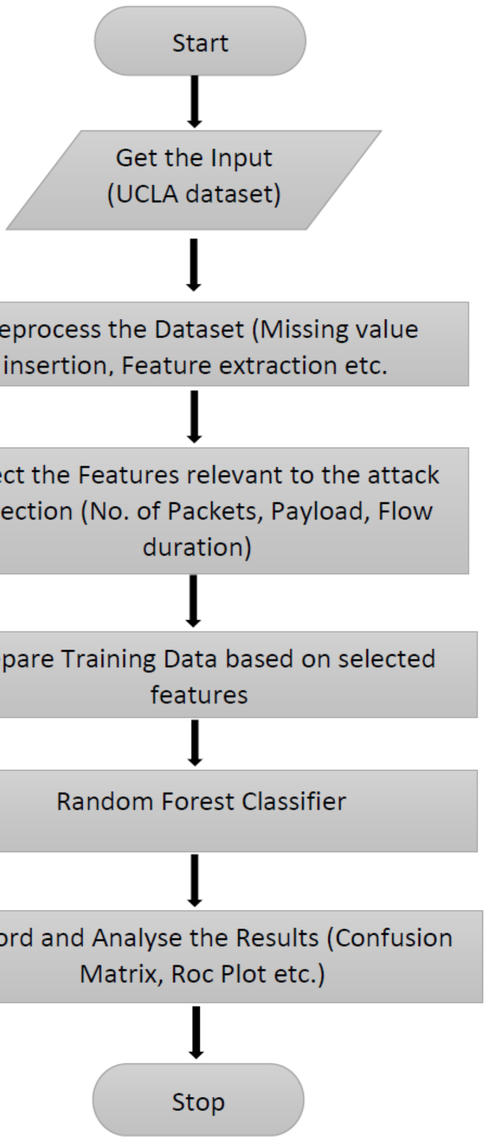

Start
Get the Input (UCLA dataset)
Preprocess the Dataset (Missing value insertion, Feature extraction etc.
Select the Features relevant to the attack detection (No. of Packets, Payload, Flow duration)
Prepare Training Data based on selected features
Random Forest Classifier
Record and Analyse the Results (Confusion Matrix, Roc Plot etc.)
Stop

Fig 3.3: Flowchart for Training Phase

3.2.5 Testing Phase

Fig-3.4 shows the testing phase in the machine learning model of our proposed system. Get the input from the controller every 10 seconds and pass it as the input to the random forest classifier model built in training phase. Check if attacked is detected. If the attack is detected raise the alarm and record the attack to a log file. Otherwise, continue the process of getting the input.

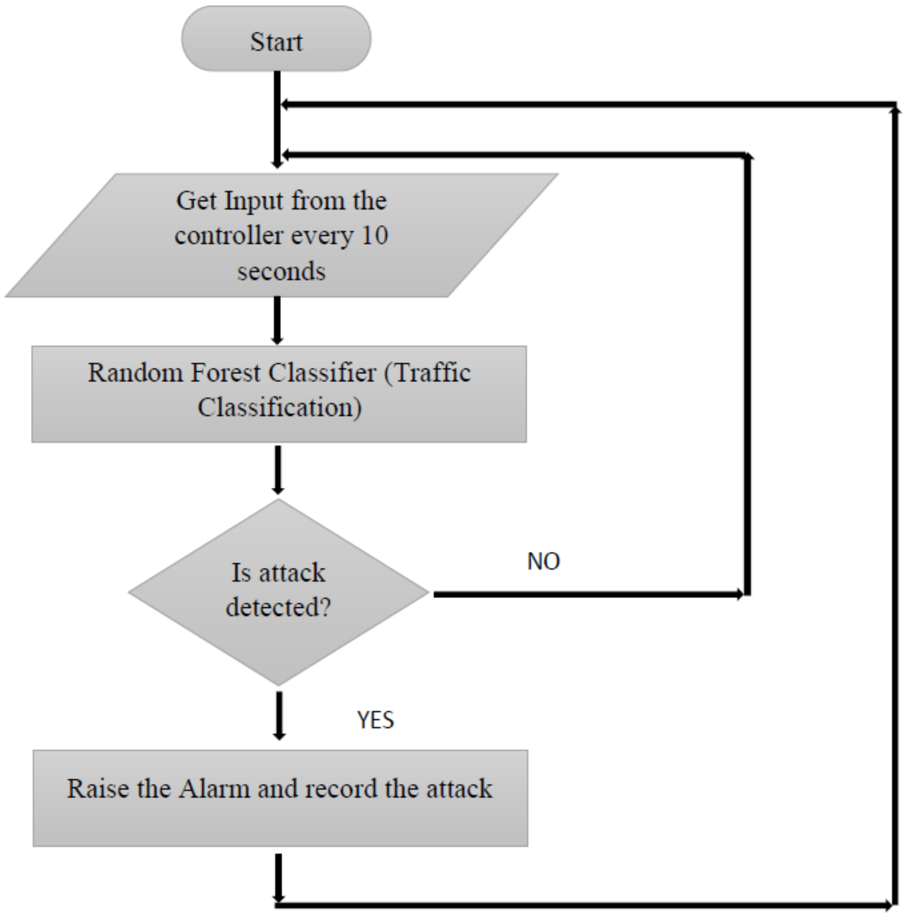

Start
Get Input from the controller every 10 seconds
Random Forest Classifier (Traffic Classification)
Is attack detected?
NO
YES
Raise the Alarm and record the attack

Fig 3.4: Flowchart for Testing Phase

3.2.6 Preparation of Trainingata

The first step in our proposed detection method is the preparation of training data. The dataset downloaded from UCLA website is converted to a *.csv* file and the duplicate values are removed, and the missing parameters are added. After processing the dataset manually, it is loaded by the detection script by using the *read_csv(“filename”)* function. The *.csv* file is converted to pandas *DataFrame* to perform the classification process.

The *DataFrame* consists of two types of data: feature data labeled as *X* and target data labeled as *Y*. Every row in the *X* is a datapoint (i.e. a network traffic flow) and every column in *X* are a feature (e.g. byte length, packet count). For a classification problem, *Y* contains the class value (attack or no attack) of every datapoint.

The *Y* column of our *DataFrame* is removed and saved as a separate numpy array (python data structure) labeled as the *Result*. Now the remaining *DataFrame* has only the feature data labeled *X*. The feature information from the *DataFrame* is saved as a numpy array (*matrix X*) using the pandas function *as_matrix*.

The *Result* array with class value: attack or no attack is replaced with binary values 1 and 0. The 1 represents attack and 0 represents no attack.

3.2.7 Training the classifier

Once the training data is ready, we build an RF classifier and apply the classifier to the training data and use 10-fold cross-validation to compute the accuracy of the classifier. The RF classifier in our proposed method is built using scikit-learn python library. Next step is to generate decision trees. The number of decision trees generated can affect your accuracy. With the increase in the number of decision trees the accuracy also increases. The number of decision trees generated can be specified. In our thesis, we have generated 100 decision trees to provide better detection accuracy without introducing significant processing overhead.

By default, random forest decision trees are generated out of random samples from the training data. Also, splitting on a feature in the decision tree is done by considering a random subset of variables to split on. This randomness may affect the prediction accuracy.

In our research, we have made the following changes to improve the prediction accuracy. First, we have implemented Bootstrap technique. So instead of selecting a random number of features, we select *m* number of features based on equation 2.1. Secondly, to compute the best split, we calculated variable importance based on the Gini Index.

Finally, the RF classifier is applied to the training data and the accuracy is calculated using a 10-fold cross-validation. Our goal is to classify the incoming network traffic as an attack or no attack traffic.

3.2.8 Implementation of RF Classifier in the Controller

Every 10 seconds, the pox controller sends the flow statistics collected from the switches to the RF classifier. The model accepts these as unseen inputs and tries to predict if there is an attack.

In the random forest method, the prediction is done by calling the *predict_proba* method. Once the method is called, it returns the prediction probabilities. For example: at a given point *X* there is a 60% probability that it belongs to class 1 and 40% probability that it belongs to class 0. The classifier's probabilities are converted to predictions. We can visualize the predicted probabilities using the *predict_proba* method.

However, when the classes in the training data are unbalanced (e.g. when the number of attack class is more compared to the no attack class), the predictions calculated by the classifier become inaccurate. This happens because our RF classifier learns the pattern of the training data to predict the unseen output. When the training data itself is unbalanced, the results turn out to be inaccurate.

The default behavior of random forest can be changed by choosing an appropriate threshold value. The analysis of precision rate can be helpful in choosing the appropriate threshold probability.

In our thesis, the value of output probability threshold is tuned so that we get the higher precision rate. More specifically, we have changed this default threshold α to 0.25 based on the analysis of the precision rate of the training set. The precision rate is calculated based on True Positive rate (TP) and False Positive rate (FP):

$$Precision = TP/(TP+FP) \qquad (3.4)$$

The definitions of TP and FP are defined in the subsequent chapter. The implementation of weighted voting instead of majority voting improves the precision rate.

Chapter 4

4 Performances and Analyses

In this chapter, we implement and evaluate the statistical and machine learning DDoS detection approaches presented in chapter 3. The results of the detection system based on statistical approach are compared with the results from the original method [1]. Similarly, the results of the machine learning DDoS detection approach are compared with the results from another existing machine learning approach in literature.

To test the performances of our proposed methods, a virtual network is simulated, using the following technologies:

4.1 Mininet

Mininet is the well-known network emulator for SDN research problems. It uses process-based virtualization to run many hosts and switches on a single OS kernel [23]. Virtual hosts, switches, controllers, and links of Mininet can be used to create any type of network topology. Its hosts run Linux network software. Moreover, its switches support OpenFlow which helps in developing OpenFlow based applications used in SDN environment. It also provides an extensible python API for the creation of the network.

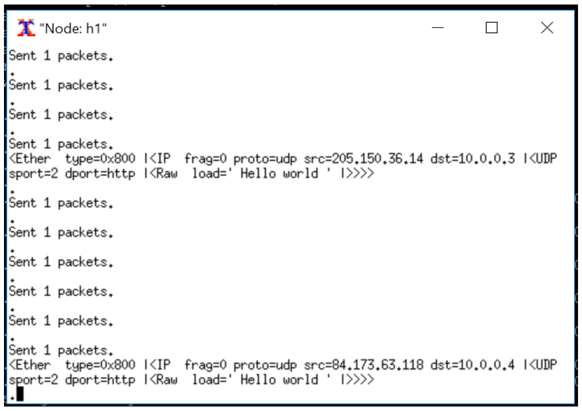
4.2 POX Controller

POX controller comes pre-installed with the Mininet virtual machine. Using POX controller, you can turn dumb OpenFlow devices into the hub, switch, load balancer, firewall devices. The POX controller allows an easy way to implement OpenFlow/SDN experiments. Different parameters can be passed to POX according to real or experimental topologies, thus allowing you to run experiments on real hardware, testbeds or in Mininet emulator.

4.3 Traffic generator

Our thesis is based on UDP flood attack and we use the traffic generator Scapy [24] to generate UDP packets and spoof the source IP address of the packets. Scapy is a powerful interactive packet manipulation program written in python. It can forge packets of different protocols. It can perform tasks like scanning, tracerouting, probing, unit tests, attacks, and network discovery. It is used as a replacement of Hping, Arpspoof, Arping, TCPDUMP, etc.

In the Appendix, we presented the code for generating both the normal and attack traffic. Fig-4.1 shows the traffic generated using Scapy script.



```
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=205.150.36.14 dst=10.0.0.3 |<UDP
sport=2 dport=http |<Raw  load=' Hello world ' |>>>>
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=84.173.63.118 dst=10.0.0.4 |<UDP
sport=2 dport=http |<Raw  load=' Hello world ' |>>>>
.
```

Fig 4.1: Traffic Generation using Scapy

The traffic patterns used in the project are given below:

Normal Traffic Pattern	
Packet Type	UDP
Packet Payload	60 bytes
No. of Packet Sent per Flow	Random between 1 and 8
Packet Inter-Arrival Interval	0.1 Sec
Traffic Rate	10 Packets/Sec

Table 4.1: Normal Traffic Pattern

Attack Traffic Pattern	
Packet Type	UDP
Packet Payload	0
No. of Packet Sent per Flow	1
Packet Inter-Arrival Interval	0.05 Sec
Traffic Rate	20 Packets/Sec

Table 4.2: Attack Traffic Pattern

4.4 Performance Metrics of Machine Learning Approach

The performance of our proposed detection system using the ML approach is evaluated using the parameters, accuracy, error, and precision. We use confusion matrix to calculate these performance metrics.

4.4.1 Confusion Matrix

A confusion matrix is an N × N matrix where N is the number of classes. In this thesis, we have 2 classes (attack and normal). The columns of the matrix represent actual classes and the rows represent the predicted classes. The confusion matrix gives the no. of correctly and incorrectly predicted results by the model. Table 4.3 shows the confusion matrix of our proposed approach.

		Predicted Class	
		Attack	Normal
Actual Class	Attack	TP	FN
	Normal	FP	TN

Table 4.3: Confusion Matrix

where,

TP = True Positive is the number of times the attack traffic was correctly classified.

FN = False Negative is the number of times attack traffic was classified as normal traffic.

TN = True Negative is the number of predictions that were correctly classified

FP = False Positive is the number of times the normal traffic was classified as attack traffic.

From the confusion matrix, we can define the following performance metrics:

Accuracy Rate

Accuracy rate = No. of correct Predictions / Total No. of Predictions.

That is,

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN} \qquad (4.1)$$

Error Rate

Error rate = No. of wrong Predictions / Total No. of Predictions.

That is,

$$Error\ rate = \frac{FP+FN}{TP+FN+FP+TN} \qquad (4.2)$$

Precision

Precision = No. of relevant items selected.

$$Precision = \frac{TP}{TP+FP} \qquad (4.3)$$

In the subsequent sections, we will present the results of different methods based on FP, FN, Accuracy Rate.

4.5 Simulation Scenario and Results of Our Proposed Method

The experiment was done on a DELL Inspiron 5558 laptop with Intel (R) Core (TM) i5-5250U CPU @1. 60GHz, 1601 MHz, 2 Core(s), 4 Logical Processor(s). Using Mininet, a tree-type network is created. Its depth is two with five switches and 20 hosts. Fig-4.2 shows the network. Open Virtual Switch (OVS) was used for network switches. OVS is a software switch that runs both on hardware and software. In Fig-4.2, all switches refer to OpenFlow enabled switches. The *L2_multi.py* module of POX was used for the controller.

To implement the modified statistical approach, we have added four modules to the controller program: entropy computation module, flow rate computation module, flow statistics collection module and a mitigation module.

The machine learning approach uses the same network set up, however, the controller program is modified by including two machine learning modules: a classifier module which has access to the training dataset and a prediction module which classify the incoming network traffic as an attack or no attack traffic.

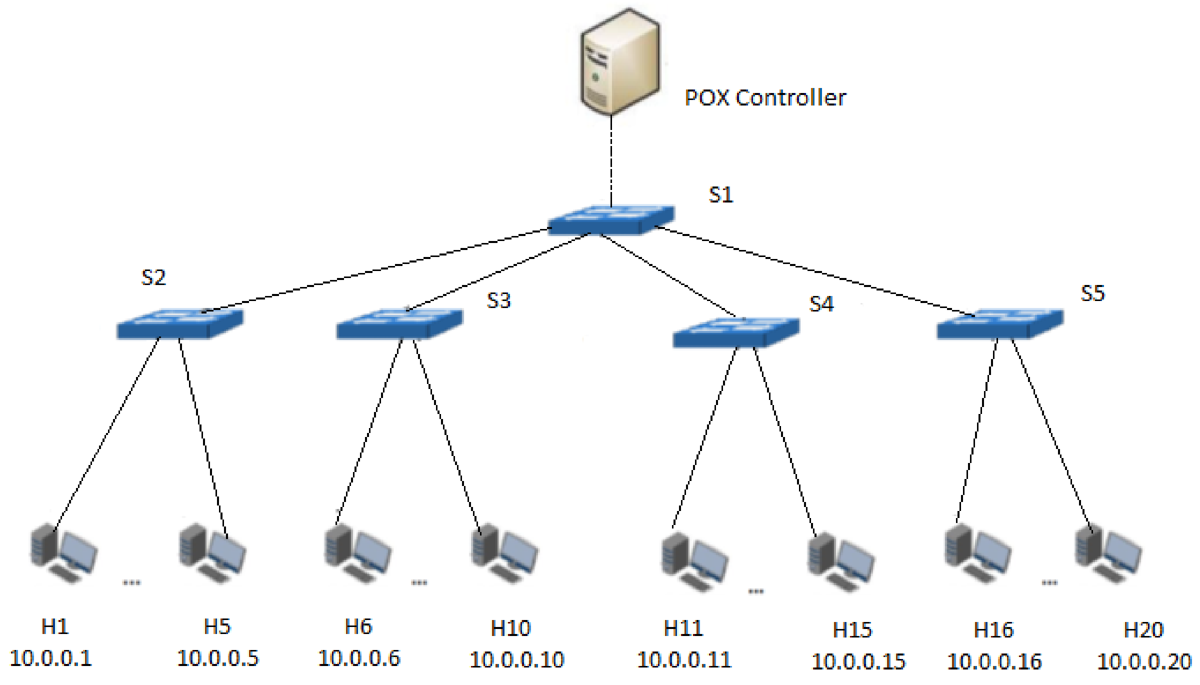


Fig 4.2: Network Setup

The experiment mainly concentrates on multiple victim attacks as the single victim attack can be easily detected. During the simulation, we are running two Scapy programs. The one that is generating the attack sends the packets faster than the one which generates normal traffic. The traffic pattern shown in Table 4.1 and Table 4.2 is used for this purpose. We run attack traffic from randomly chosen 4 hosts to attack four different destinations, consequently remaining 16 hosts generate the legitimate traffic. The IP address in Mininet for all hosts are assigned from 10.0.0.1 to 10.0.0.20.

4.6 Performance of the Statistical Approach

Based on the calculated threshold values, the performance of our detection system is analyzed by running the simulation 50 times. For each run, attack traffic of pattern B is generated on four hosts. Each simulation lasts about 30 minutes. The results were recorded, and the False Positive and False Negative values are summarized in Table 4.4:

FP & FN Values of Multiple Victim Attack	
No. of Attacks	50
FP (Avg.count)	3
FN (Avg.count)	1
Accuracy Rate	92%

Table 4.4: False Positive and False Negative Values

Based on the values in Table 4.4, we can say that there is a possibility for an attack traffic to pass through the detection system as normal traffic, which is harmful to the system.

4.6.1 Comparison of the Basic Approach with Our modified approach

The main difference between the basic approach and our approach is the way of deriving the thresholds. Fig-4.3 shows the comparison of the accuracy rate achieved by the basic approach and our modified approach.

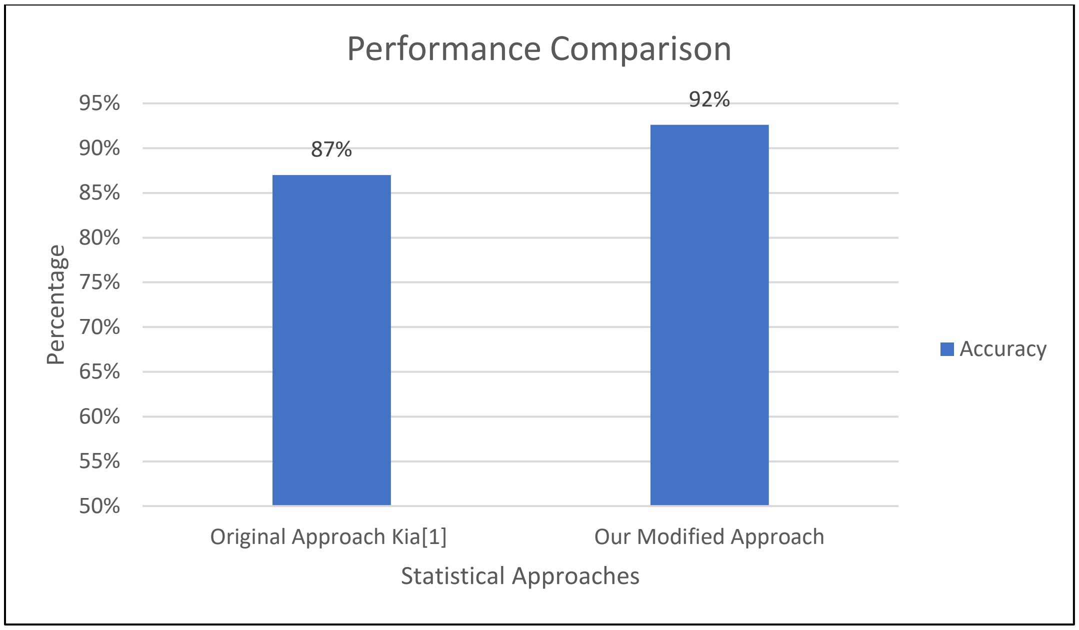
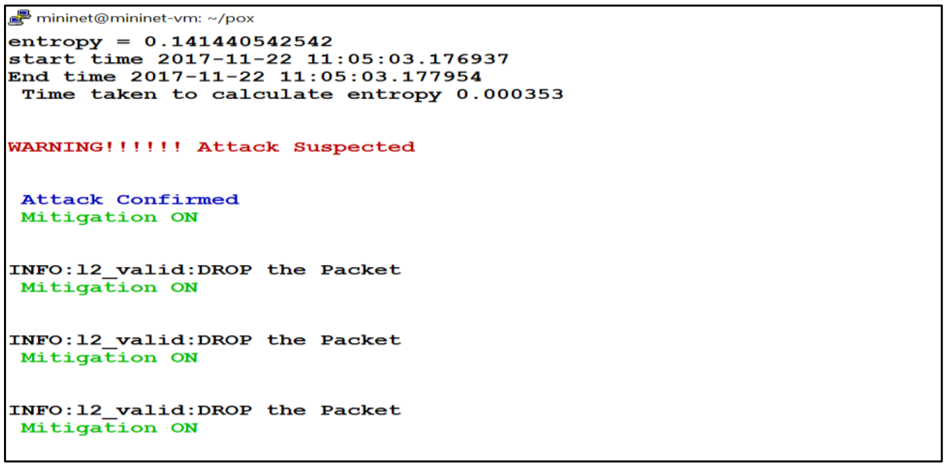


Fig 4.3: Comparison of Performance of Statistical Approaches

From the result, we can say that the accuracy of the detection system has improved by using the thresholds derived based on the mean and standard deviation method.

4.7 Mitigation Results

After the attack has been confirmed by Stage II of our detection system, the mitigation module is called, and the attack traffic is dropped to prevent the breakdown of the switches in the network. Fig-4.4 shows the output of the execution of the mitigation module of the POX controller to drop the attack traffic.



```
mininet@mininet-vm: ~/pox
entropy = 0.141440542542
start time 2017-11-22 11:05:03.176937
End time 2017-11-22 11:05:03.177954
 Time taken to calculate entropy 0.000353

WARNING!!!!!! Attack Suspected

 Attack Confirmed
 Mitigation ON

INFO:l2_valid:DROP the Packet
 Mitigation ON

INFO:l2_valid:DROP the Packet
 Mitigation ON

INFO:l2_valid:DROP the Packet
 Mitigation ON
```

Fig 4.4: Execution of Mitigation Module

Fig-4.5 shows the entire process of attack detection and mitigation of the proposed statistical approach.

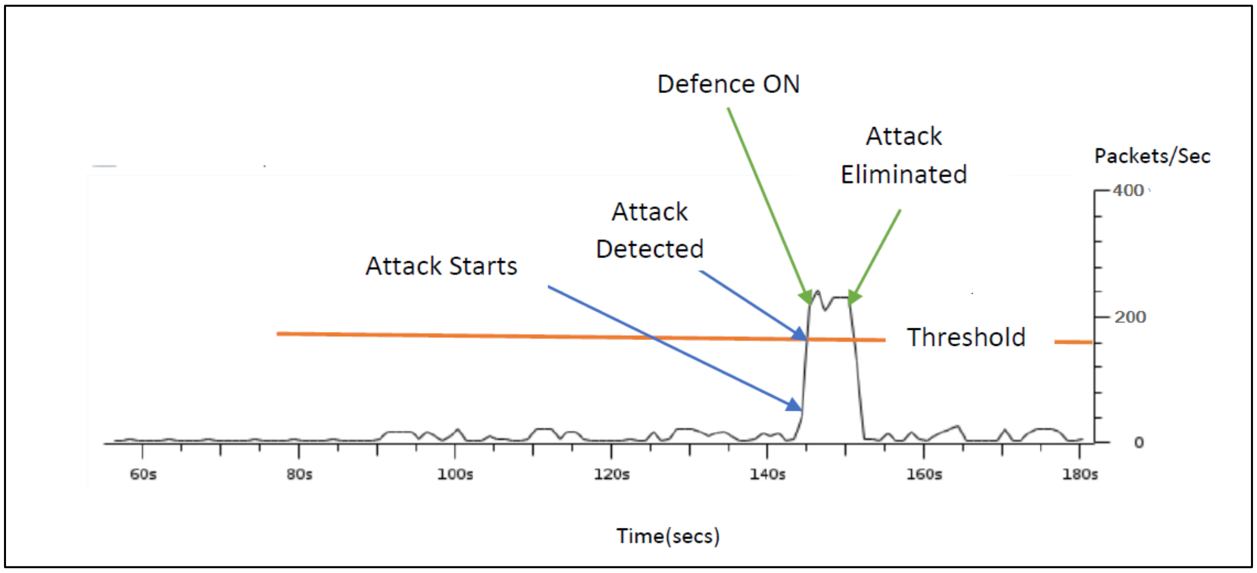


Fig 4.5: Wireshark Output: Attack Traffic Mitigated

4.7.1 Performance Analysis of Mitigation Module

To demonstrate the performance of the defense system in defeating DDoS attacks, we compare results in two cases. In the first case, we start the attack on our simulation network without enabling any DDoS defense mechanisms. The switch at the victim end breaks down due to the large attack traffic. In the second case, we deploy the mitigation technique in the POX controller. After receiving the attack alert, the controller drops the attack traffic in order to protect the switch.

The simulation is run 10 times for each case with different attack rate. Table 4.5 shows the result obtained in each case:

Packets/Sec	**With Defense Window**	**No Defense Window**
10	Operational	Operational
20	Operational	Operational
50	Operational	Operational
100	Operational	Crash
200	Operational	Crash
300	Operational	Crash
400	Operational	Crash
500	Operational	Crash
1000	Operational	Crash

Table 4.5: Performance Analysis of Mitigation Module

Comparing the proposed method with the original method, the obvious difference is that the detection system succeeds in lowering the attack traffic. This demonstrates that our proposed DDoS defense system is able to differentiate between attack and legitimate traffic with high accuracy.

4.8 Performance of the Machine Learning approach

The UCLA dataset is fairly balanced and contains a total of 1200 instances, with 60% (720) attack and 40% (380) no attack instances. A Python script was used for training and testing the classifier. 10-fold cross-validation is used to evaluate the classifier. The accuracy of the resulting classifier is compared with Kato et al [17].

4.8.1 Feature Selection

We need to decide on which features to train on. The UCLA dataset has a variety of features as summarized in Table 3.1. However, all these features are not used in our approach to classify the traffic flow. Hence, we have selected three features: Payload, Packet count and Flow duration from the UCLA dataset and calculated the feature importance based on Gini score. The code to compute a Gini score can be found in the Appendix

Fig-4.6 shows the calculated feature importance. Based on the obtained result, we selected the following features: No. of Packets, Byte length (payload) and Flow Duration, to build our model:

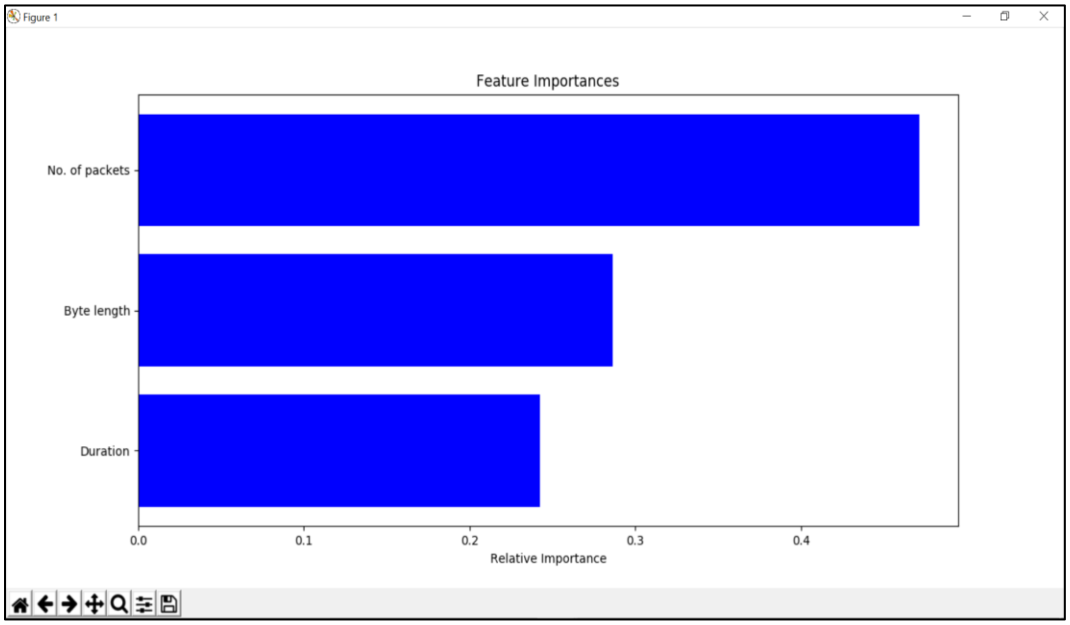


Fig 4.6: Feature Importance Plot

We evaluated the performance of our proposed detection system using parameters such as accuracy rate, error rate, precision and ROC plot. We use confusion matrix to calculate accuracy, error, and precision. Table 4.6 shows the calculated performance measures of our proposed system.

Performance Metrics	
Accuracy	97.70%
Error Rate	2.3%
Precision	95.74%

Table 4.6: Performance Metrics

4.8.2 ROC Plot

Receiver Operating Characteristic (ROC) plot is the graphical way of inspecting the performance of our random forest classifier. It shows the rate at which our classifier is making correct predictions. It is applicable only to binary classification model. It is plotted between the False Positive Rate (FPR) on the x axis and the true positive rate (TPR) on the y axis. Fig-4.7 shows the exemplary ROC plot:

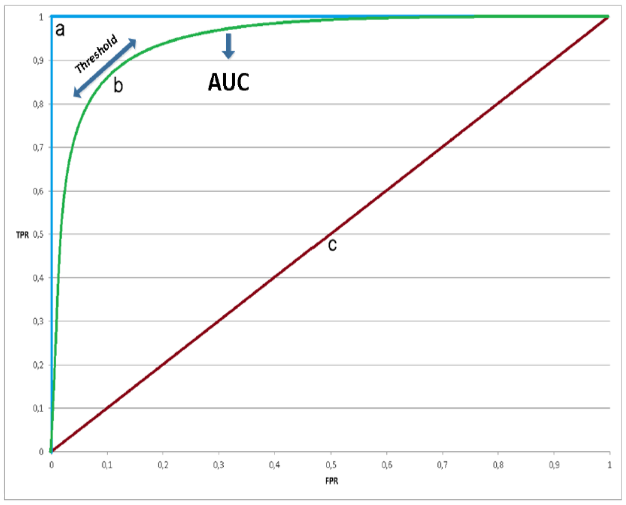


Fig 4.7: Exemplary ROC Plot [8]

AUC or Area Under the Curve is the space underneath the ROC curve. The perfect classifier has the AUC value of 1. AUC value is used to compare different models. It also determines the performance of the classifier.

The ROC Plot of our proposed method is given in Fig-4.8.

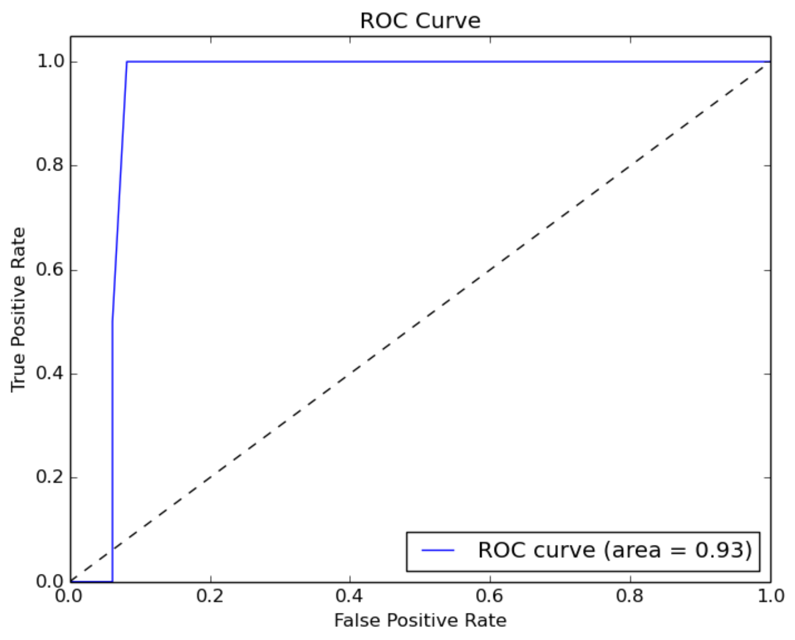


Fig 4.8: ROC Plot of Proposed Method

The AUC value of our classifier is 0.93.

4.9 Comparison of Our Statistical and Machine Learning Approach

In this section, we compare the accuracy of our Machine Learning approach to the solution obtained from the statistical approach.

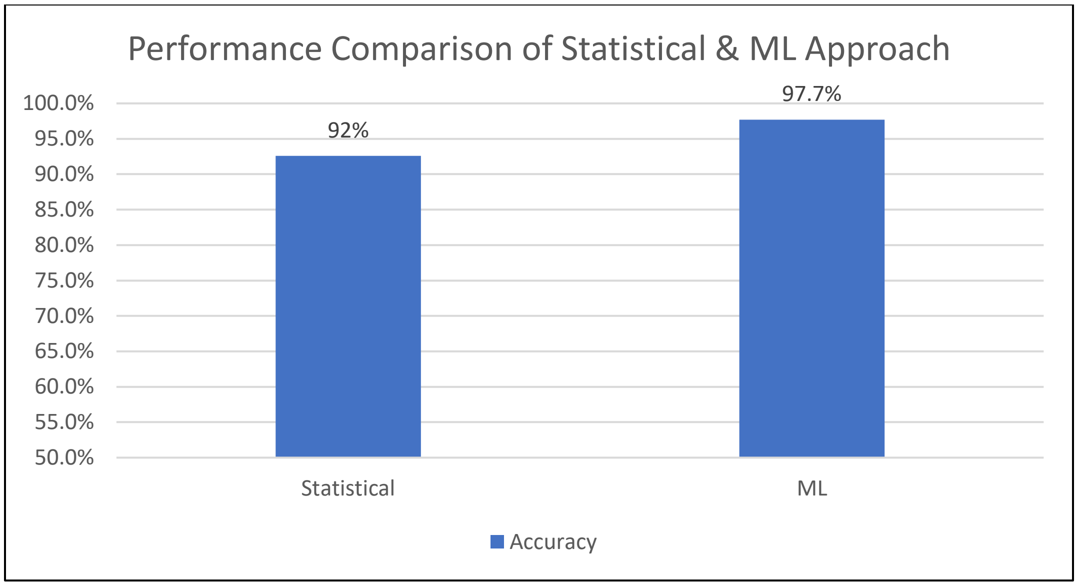


Fig 4.9: Performance Comparison of Statistical & ML Approaches

From the above analysis, we can conclude that the proposed detection system implemented using a Machine Learning approach could successfully detect DDoS attack with higher accuracy.

4.10 Comparison of Machine Learning Approach with Existing ML Approach

In this section, we will compare the proposed machine learning based DDoS Detection method with the approach of Kato et al [17].

In [17] the authors proposed a DDoS detection system using a Support Vector Machine algorithm. The algorithm used CAIDA dataset to analyze the attack pattern. The main idea behind their approach is to perform network packet analysis and to study the patterns of the DDoS attack using the machine learning algorithm. They prepared three different types of the dataset and tested their model. The detection system is 85% accurate with three features.

To compare the performance based on detection accuracy, we implemented the SVM algorithm using the UCLA training dataset prepared in section 3.2.6. The experimental result shows that the detection system could successfully detect the attack traffic with the detection accuracy of 88%.

Fig-4.10 shows the comparison of the performance of our ML approach using the RF classifier with the ML approach using the SVM classifier.

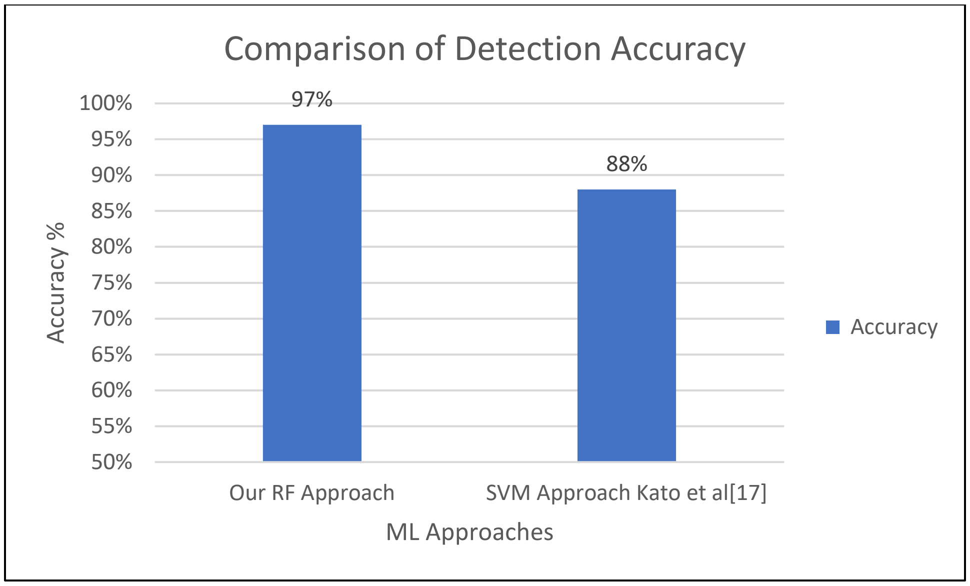


Fig 4.10: Performance Comparison of our ML Approach with other ML Approaches

From the results of the comparison, it can be seen that the proposed ML approach gives the best performance in comparison with other approaches considered in this thesis. It surpasses the other approaches in practical implementation due to the following reasons:

- Random forest classifier which has fast computing times and robustness against noisy data.
- Implementation of the Weighted voting method instead of standard majority voting.

Chapter 5

5 Conclusion and Future Work

5.1 Conclusion

Detecting and defending against DDoS attack is a complex task. Initially, we started the research aiming to improve the DDoS detection system proposed by Kia [1]. We used the mean and standard deviation to compute various thresholds. We also introduced a better mitigation method to protect the controller and the switches. Even though we managed to improve the detection rate performance, we found that it is still not entirely satisfactory. Hence, to improve the detection rate further, we proposed an ML approach. Our proposed approach is based on RF algorithm with weighted voting. Our results show that the proposed approach has the best performance among all the approaches considered in this thesis.

5.2 Future work

As a future work, we recommend developing a controller that can detect any type of network attack and employ deep packet inspection so that the detection accuracy is even higher. Moreover, SDN has also a concept of distributed network security enforcement by making each network element potentially an enforcement node and smart. This can be easily achieved by Machine Learning approach.

Appendix

Entropy Computation Module

```
class Entropy(object):

#Set Counter for Window size = 100

count = 0

# Dictionary to Store Destination IP occurrence

        ipDic = {}

        # List to Store IP addresses

        ipList = []

        # List to store Entropies

        lstEnt = []

        value = 1

        #Function to collect Destination IP

        def colectIP(self, element):

                l = 0

                self.count +=1

                self.ipList.append(element)

                # Check whether window size is reached

                if self.count == 100:

                        for i in self.ipList:

                                l +=1

                                if i not in self.ipDic:
                                        self.ipDic[i] =0
                                self.ipDic[i] +=1
                        # Entropy Function to calculate Entropy
                        self.entropy(self.ipDic)
                        log.info(self.ipDic)
                        self.ipDic = {}
                        self.ipList = []
                        l = 0
                        self.count = 0
        #Entropy computation
        def entropy (self, lists):
                #print "Entropy called"
                l = 50
                elist = []
                for k,p in lists.items():
                        #Probability Of each IP
                        c = p/float(l)
                        c = abs(c)
                        #List of Entropies
                        elist.append(-c * math.log(c, 2))
                        log.info('Entropy = ')
                        log.info(sum(elist))
                        self.lstEnt.append(sum(elist))
                if(len(self.lstEnt)) == 80:
                        #Print to display Entropy
                        print self.lstEnt
                        self.lstEnt = []
                        return(sum(elist))
        def __init__(self):
                pass
```

Flow Statistics Collection

```
#Flow statistics Module
# standard includes
from pox.core import core
from pox.lib.util import dpidToStr
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
# include as part of the betta branch
from pox.openflow.of_json import *
log = core.getLogger()
# handler for timer function that sends the requests to all the
# switches connected to the controller.
def _timer_func ():
  for connection in core.openflow._connections.values():
    connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
  log.debug("Sent %i flow/port stats request(s)", len(core.openflow._connections))
```

```
# handler to display flow statistics received in JSON format
# structure of event.stats is defined by ofp_flow_stats()
def _handle_flowstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.info("flow statistics received from %s",dpidToStr(event.connection.dpid))
  #Dictionary to store the flow details
  flowlist = {}
  #Counter to count the no. of flows
  flow_count = 0
 #Counter for packets
  p_count = 0
 #Counter for bytes
  b_count = 0
  for flow in event.stats:          #for each flow
     if flow.match.dl_type==0x0800:  #Only UDP Packets
            #Collect the Flow statistics
 flowlist = {"flow_Duration": flow.duration_sec, "packet_count": flow.packet_count,
"byte_count": flow.byte_count}
#Increment the counters
             p_count += flow.packet_count
             b_count += flow.byte_count
      if flow.packet_count <> 0:
          flow_count = flow_count+1
  print "Traffic from %s: %s bytes,%s packets and flows",dpidToStr(event.connection.dpid),
p_count, b_count, flow_count
      #print flow_count
     def flow_stat():
     self._timer_func()
# main functiont to launch the module
  def launch ():
  from pox.lib.recoco import Timer
  # attach handsers to listners
  core.openflow.addListenerByName("FlowStatsReceived",  _handle_flowstats_received)
  # timer set to execute every five seconds
  Timer(5, _timer_func, recurring=True)
Mitigation Module:
#Check if Ethernet packet
    if packet.type == ethernet.IP_TYPE:
       #Start the Timer
       self.start = time.time()
        #Set the window interval
       time_interval = 3
       #Increment the Flow Counter
       self.flow_list+=1
       if self.flow_list == 1:
```

```
   self.end = self.start + time_interval

#   #print 'end', self.end

#Check if flow counter exceeds the threshold and if timer expired

if self.flow_list > fth and self.start<self.end:

    # Attack is confirmed , Turn ON mitigation

    cprint(' Mitigation ON ', 'blue', 'on_cyan')

      #Drop the attack packets

      drop()

      self.flow_list = 0

elif self.start>=self.end:

      # If timer expires, reset the flow counter

      self.flow_list = 0

else:

      pass
```

Traffic Generation Code:

Normal Traffic

```
import sys

import getopt

import time

from os import popen

import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import sendp, IP, UDP, Ether, TCP

from random import randrange

import random

import threading

#this function generates random IP addresses

# these values are not valid for first octet of IP address

def sourceIPgen():

    not_valid = [10,127,254,1,2,169,172,192]

    first = randrange(1,256)

    while first in not_valid:

        first = randrange(1,256)

        ip = ".".join([str(first),str(randrange(1,256)),str(randrange(1,256)),str(randrange(1,256))])

        return ip

#send the generated IPs

def gendest():
    first = 10

    second =0; third =0;

    start = 2

    end = 60

    ip = ".".join([str(first),str(second),str(third),str(randrange(start,end))])

    return ip

def genTraffic():

    m =0

    #a = random.uniform(0.1,0.4)

    # open interface eth0 to send packets

    interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

    for i in xrange(1000):

    # form the packet

    packets = Ether()/IP(dst=gendest(), src=sourceIPgen())/UDP(dport=80,sport=2)

    print(repr(packets))

    while m<=4:

        # send packet with the defined interval (seconds)

        sendp(packets,iface=interface.rstrip(),inter=0.1)

        m+=1

def main():

        #run_event = threading.Event()

        #run_event.set()

        #d1 = 0.1
        timeout = time.time() + 60*1
        #threading.Timer(0.1,genTraffic).start()
        while True:
          genTraffic()
          if time.time()>timeout:
             break
#main
if __name__=="__main__":
  main()
```

Attack Traffic

```
import sys
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange
import time
def sourceIPgen():
#this function generates random IP addresses
# these values are not valid for first octet of IP address
  not_valid = [10,127,254,255,1,2,169,172,192]
  first = randrange(1,256)
  while first in not_valid:
  first = randrange(1,256)
  print first
  ip = ".".join([str(first),str(randrange(1,256)), str(randrange(1,256)),str(randrange(1,256))])
  print ip
  return ip
def main():
  timeout = time.time() + 30*1
  while True:
    mymain()
   if time.time()>timeout:
       break
 #send the generated IPs
 def mymain():
 #getting the ip address to send attack packets
  dstIP1 = sys.argv[1:]
  dstIP2 = sys.argv[1:]
  dstIP3 = sys.argv[1:]
  dstIP4 = sys.argv[1:]
  #print dstIP
  src_port = 80
  dst_port = 1
 # open interface eth0 to send packets
```

```
  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()
  print (repr(interface))
  #for i in xrange(0,2000):
  # form the packet
  packets = Ether()/IP(dst=dstIP1,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
  print(repr(packets))
  packets = Ether()/IP(dst=dstIP2,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
  print(repr(packets))
  packets = Ether()/IP(dst=dstIP3,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
  print(repr(packets))
  packets = Ether()/IP(dst=dstIP4,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
  print(repr(packets))
# send packet with the defined interval (seconds)
  sendp( packets,iface=interface.rstrip(),inter=0.03)
#main
if __name__=="__main__":
  main()
```

Machine Learning code

Classifier Module

```
#Standard Includes
import numpy as np
import pandas as pd
from pandas import read_csv
#The Machine learning alogorithm
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Test train split
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
# Just to switch off pandas warning
pd.options.mode.chained_assignment = None
# Used to write our model to a file
from sklearn.externals import joblib
#Open the data set
data = read_csv("tableset.csv")
print data.head()
print data.columns
#Select the Features
```

```
data_inputs = data ["Duration", "No. of packets", "Byte length"]

print data_inputs.head()

ex_outputs = data[["result"]]

print ex_outputs.head()

#Create the Classifier to create 100 trees

rf = RandomForestClassifier (n_estimators=100)

rf.fit(data_inputs, ex_outputs)

#Accuracy Calculation

accuracy = rf.score(data_inputs, ex_outputs)

print "Accuracy = {}%".format(accuracy * 100)

#Save the ML model

joblib.dump(rf, "test_model1", compress=9)

#Calculate Feature Importance

importances = rf.feature_importances_

indices = np.argsort(importances)

 plt.figure(1)

 plt.title('Feature Importances')

 plt.barh(range(len(indices)), importances[indices], color='b', align='center')

 plt.yticks(range(len(indices)), data_inputs[indices])

 plt.xlabel('Relative Importance')

 plt.show()
```

Prediction Module

```
#Standard Includes

import numpy as np

import pandas as pd

from pandas_ml import ConfusionMatrix

import matplotlib.pyplot as plt

from pandas import read_csv

#The Machine learning alogorithm

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve, auc

# Test train split

#from sklearn.cross_validation import train_test_split

from sklearn.model_selection import train_test_split

# Just to switch off pandas warning

pd.options.mode.chained_assignment = None

# Used to write our model to a file

from sklearn.externals import joblib

#Open Test Data

data = read_csv("data.csv")

test_input = data[["Duration", "packet", "byte length"]]

test_output = data[["result"]]
```

```
#Open the Saved Model

rf = joblib.load("test_model1")

#Predict the Attack

pred = rf.predict_proba(test_input)

print pred

#Calculate the Accuracy

accuracy = rf.score(test_input, test_output)

print "Accuracy = {}%".format(accuracy * 100)

#Calculate Confusion Matrix

results = confusion_matrix(pred, test_output)

print "----Confusion Matrix-----"

print results

plt.matshow(results)

plt.title('Confusion Matrix')

plt.colorbar()

plt.ylabel('Actual')

plt.xlabel('Predicted')

#plt.show()

#calculate True Positive Rate, False Positive Rate

fpr,tpr, _ = roc_curve(test_output, rf.predict_proba(test_input)[:,1])

roc_auc = auc(fpr, tpr)

print 'ROC AUC: %0.2f' % roc_auc
```

```
# Plot of a ROC curve for a specific class

plt.figure()

plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc="lower right")

plt.show()
```

Bibliography

[1]. Maryam Kia, “Early Detection and Mitigation of DDoS Attacks in Software Defined Networks”, Master’s thesis, Ryerson University, Canada 2015.

[2]. Javed Ashraf and Seemab Latif, “Handling Intrusion and DDoS Attacks in Software Defined Networks Using Machine Learning Techniques”, National Software Engineering Conference, 2014.

[3]. opennetworking.org, “OpenFlow”, [Online]. Available: https://www.opennetworking.org. [Accessed February 2016].

[4]. slideshare.net, “OpenFlow Tutorial”, [Online] Available: https://slideshare.net/openflow/. [Accessed September 2016].

[5]. Sukhveer Kaur, Japinder Singh and Navtej Singh Ghumman, “Network Programmability Using POX Controller”, International Conference on Communication, Computing & Systems, pp. 134-138, 2014.

[6]. Seyed Mohammad Mousavi and Marc St. Hilaire, “Early Detection of DDoS Attacks against SDN Controllers”, International Conference on Computing, Networking and Communications, Communications and Information Security Symposium, 2015.

[7]. Duohe Ma1, Zhen Xu, and Dongdai Lin, “Defending Blind DDoS Attack on SDN Based on Moving Target Defense”, researchgate, November 2015.

[8]. Meiling Liu, Xiangnan Liu, Jin Li and Jiale Jiang, “Evaluating total inorganic nitrogen in coastal waters through the fusion of multi-temporal RADARSAT-2 and optical imagery using random forest algorithm”, International Journal of Applied Earth Observation and Geoinformation 33(1), pp. 192-202, December 2014.

[9]. Surender Singh and Sandeep Jain, “A Review of Detection of DDoS Attack Using Entropy-Based Approach”, IJCST Vol 4, Issue 2, April 2013.

[10]. Jisa David and Ciza Thomas, “DDoS Attack Detection using Fast Entropy Approach on Flow-Based Network Traffic”, Procedia Computer Science, pp. 30 – 36, 2015.

[11]. Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan and Vijay Mann, “SPHINX: Detecting Security Attacks in Software-Defined Networks”, IBM Research, 2009.

[12]. Haopei Wang, Lei Xu, Guofei Gu, "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks”, 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15), 2015.

[13]. Z. Qin, L. Ou, J. Liu, A. X. Liu J. Zhang, "An Advanced Entropy-Based DDoS Detection Scheme”, IEEE, 2010.

[14]. Alpna and Sona Malhotra, “DDoS Attack Detection and Prevention Using Ensemble Classifier (RF)”, IJARCSSE, 2016.

[15]. Arif Jamal Malik, Waseem Shahzad, and Farrukh Aslam Khan, “Network Intrusion Detection Using Hybrid Binary PSO and Random Forests Algorithm”, Security and Communication Networks, 2012.

[16]. Nabila Farnaaz and M.A. Jabbar, “Random Forest Modelling for Network Intrusion Detection System”, IMCIP, 2016.

[17]. Keisuke Kato and Vitaly Klyuev, “An Intelligent DDoS attack Detection System Using Packet analysis and Support Vector Machine”, IJICR, Volume 5, Issue 3, September 2014.

[18]. Sivatha Sindhu, Geetha subbiah and Kannan Arputharaj, “Decision tree based lightweight intrusion detection using a wrapper approach”, Expert Systems with Applications, pp. 129-141, January 2012.

[19]. Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa and Baijian Yang, “Predicting Network Attack Patterns in SDN using Machine Learning Approach”, IEEE 2016.

[20]. Zhong-dong Wu, Wei-Xin Xie and Jian-ping Yu, “Fuzzy C-Means Clustering Algorithm Based on Kernel Method”, ICCIMA, 2003

[21]. Md. Al Mehedi Hasan, Mohammed Nasser, Biprodip and Shamim Ahmad, “Support Vector Machine and Random Forest Modeling for IDS”, JILSA, pp. 45-52, 2014.

[22]. lasr.cs.ucla.edu, “UCLA Dataset”, [Online]. Available: https://lasr.cs.ucla.edu/DDoS/traces/. [Accesssed April 2017].

[23]. mininet.org, “Mininet”, [Online]. Available: http://mininet.org/. [Accessed June 2016].

[24]. secdev.org, “Scapy”, [Online]. Available: http://www.secdev.org/. [Accessed January 2016].

[25]. Xin Xu and Xuening Wang, “An adaptive network intrusion detection method based on PCA and support vector machines”, Advanced Data Mining and Applications, Springer, pp. 696-703, 2005.

[26]. Ming-Yang Su, “Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers”, Expert Systems with Applications, PP. 3492-3498, 2011.

[27]. Sindia and Julia Punitha Malar Dhas, “SDN Based DDoS Attack Detection and Mitigation in Cloud”, IJCTA, pp. 39-47, 2017.

[28]. Kuan-yin Chen, Anudeep Reddy Junuthula, Ishant Kumar Siddhrau, Yang Xu, H. Jonathan Chao, “SDNShield: Towards More Comprehensive Defense against DDoS Attacks on SDN Control Plane”, IEEE, 2016.

[29]. Christos Douligeris and Aikaterini Mitrokotsa, “DDoS attacks and defense mechanisms: classification and state-of-the-art”, Computer Networks, pp. 643–666, 2004.

[30]. Mohammed Alenezi and Martin J Reed, “Methodologies for detecting DoS/DDoS attacks against network servers”, The Seventh International Conference on Systems and Networks Communications, 2012.

[31]. Manjula Suresh and R. Anitha, “Evaluating Machine Learning Algorithms for Detecting DDoS Attacks”, Communications in Computer and Information Science, January 2011.

[32]. Stefan Seufert and Darragh O’Brien, “Machine Learning for Automatic Defence against Distributed Denial of Service Attacks”, ICC, 2007.

[33]. Bhatia, Sajal, Schmidt, Desmond, & Mohay, George M, “Ensemble based DDoS detection and mitigation model”, Fifth International Conference on Security of Information and Networks, pp.79-86, 2012.

[34]. Yang Xu and Yong Liu, “DDoS Attack Detection under SDN Context”, IEEE INFOCOM, 2016.

[35]. pythonprogramming.net, “Python for Machine Learning”, [Online]. Available: https://pythonprogramming.net/machine-learning-tutorial-python-introduction. [Accessed June 2017].

[36]. scikit-learn.org, “Sklearn”, [Online]. Available: https://scikit-learn.org/. [Accessed June 2017].

[37]. Przemysław Berezinski, Bartosz Jasiul, and Marcin Szpyrka, “An Entropy-Based Network Anomaly Detection Method”, Entropy, 2015.

[38]. H. Kim and N. Feamster, “Improving network management with software-defined networking,” IEEE Communications Magazine, vol. 51, no. 2, pp. 114–119, February 2013.

[39]. www.cs.iit.edu, “Decision tree classification”, [Online]. Available: http://www.cs.iit.edu/ [Accessed Aug 2017].

[40]. www.analyticsvidhya.com, “Cross-validation in random forest”, [Online]. Available: https://www.analyticsvidhya.com/blog/2015/11 [Accessed Aug 2017].

[41]. Jorge Proenca, Tiago Cruz, Edmundo Monterio and Paul Simoes, “How to use Software Define Networking to improve Security - a Survey”, ECCWS2015-Proceedings of the 14th European Conference on Cyber Warfare and Security, 2015.

[42]. Rodrigo Braga, Edjard Mota, and Alexandre Passito, “Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow”, 35th Annual IEEE Conference on Local Computer Networks, 2010.

[43]. Juels, A. and J. G. Brainard, Client Puzzles, “A Cryptographic countermeasure against connection depletion attacks”, NDSS, 1999.

[44]. Chaitanya Buragohain, Nabajyoti Medhi, “FlowTrApp: An SDN Based Architecture for DDoS Attack Detection and Mitigation in Data Centers”, 3rd International Conference on Signal Processing and Integrated Networks, 2016.

[45]. Sung, M. and J. Xu, “Ip traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks”, ICNP ’02. IEEE Computer Society, Washington, DC, USA, 2002.

[46]. www.rapidminer.com , “k-fold cross-validation”, [Online]. Available: https:// rapidminer.com. [Accessed August 2016].